Applying Analogical Reasoning Techniques for Teaching XML Document Querying Skills in Database Classes

Michel Mitri
Computer Information System & Management Science
James Madison University
Harrisonburg, VA 22807, USA
mitrimx@jmu.edu

ABSTRACT

XML has become the most ubiquitous format for exchange of data between applications running on the Internet. Most Web Services provide their information to clients in the form of XML. The ability to process complex XML documents in order to extract relevant information is becoming as important a skill for IS students to master as querying relational databases. But the language for querying XML documents is very different from SQL, which is the query language that IS students typically learn in their database courses. Nevertheless, the database course seems to be the most plausible venue for teaching XML document querying, given the IS 2010 model curriculum. Unfortunately, there are time limitations that may prevent deep coverage of XML in the typical database class. Analogical pedagogy may provide a means to quickly provide significant XML query skills to students who are already familiar with SQL query mechanics. This paper describes a simple but effective way of incorporating XML querying within the broader database course content by making use of analogical reasoning.

Keywords: Extensible markup language (XML), Data structures, Structured Query Language, Instructional pedagogy, Learning styles, Markup languages, Pedagogy, Query language.

1. XML QUERY SKILLS IN THE IS CURRICULUM

The IS2010 model curriculum (Topi et al 2010) does not include any mention of XML, although IS2010 does include extensive coverage of database skills. Students who graduate from a typical IS program will have sufficient understanding of relational technologies and SQL for querying databases, but often lack training in XML structures or the use of XPath/XQuery for querying XML documents.

XML's hierarchical data structure is more suitable than the tabular structure of relational databases for some data storage and representation purposes. Partly for this reason, XML has become ubiquitous, especially as a means of exchanging information between applications over the internet. In addition, more and more database management systems incorporate XML data types and querying functionality into their engines. For example, major database products like Oracle and Microsoft's SQL Server have incorporated data structures and associated functionality for storing and processing XML-formatted data.

There also appears to be an increasing coverage of XML in database textbooks. For example, in Hoffer et al 7th edition of Modern Database Management, only two full pages (422-424) are devoted to XML coverage, and there is no tight integration of this topic with the topic of Web Services. By contrast, the 11th edition devotes 10 pages (360-369) and makes a stronger connection to Web Services.

It makes sense that XML coverage should get greater attention in an IS curriculum, since it has become so prevalent in the real world. Consequently, there have been some advances in XML pedagogy as described in the IT education literature. For example, Olsen et al (2005) discuss integrating XML into database courses, and present a sample database for a medical clinic in SQL Server and queries that make use of SQL Server's XML processing to perform queries on the database and produce results in XML format.

Wagner et al. (2008) outline a set of considerations for incorporating XML into the MIS curriculum, including contrasting XML with HTML, structuring XML coverage using a system model framework, covering the plethora of XML-related technologies, and discussing the meta-language nature of XML. Specific courses that could benefit from XML coverage include database, systems analysis and design, ecommerce, and web development.

A complete coverage of all aspects related to XML including style sheeting, metadata declarations (DTDs and XML Schemas), XML extension frameworks (e.g. XBRL, RSS, SOAP, RDF), and other advanced topics could merit an entire course in itself. But, for the purposes of providing data query and analysis skills relevant for a database or BI course, a much smaller subset of XML-related topics would suffice. In particular, if a database course can provide a thorough understanding of the structure of XML documents(hierarchical, tree-like), along with practical experience with the associated query languages of XPath (XML Path Language) and XQuery, then this would go a long way toward enhancing students' facility with XML in general.

The problem is time. With an already full schedule of topics to cover, database classes are hard-pressed to incorporate new content.

2. ANALOGICAL REASONING AND PEDAGOGY

The prospect of incorporating significant XML querying skills into an already busy database course schedule can be daunting. How can we include these skills without overloading the curriculum? The challenge is to create an avenue for providing deep understanding in a minimal amount of time. In order to do this, we can make use of analogical pedagogy (James 2003, Harrison 1993, Clement 1993), which capitalizes on the promise that students can quickly learn basics by making analogies between a new skill to be learned and an already well-established skill. In our case, the challenge is to bootstrap on the already existing skill of database students for using SQL SELECT statements to query relational databases in order to produce equivalent skills for querying XML documents in a time-efficient fashion.

Research in analogical pedagogy is often credited to seminal work by Dedre Gentner (1983, 1989), whose work also inspired extensive advances in artificial intelligence research (Falkenhainer et al 1989, Forbus and Gentner 1991, Forbus et al 2002). Gentner's study of analogical reasoning and learning is based on her theory of *structure mapping*. In this theory, there is an attempt to map a *base domain* (the already-known) to a *target domain* (which needs to be explained). Each domain is composed of a set of objects; and the objects within these domains contain both attributes (predicates describing the object in itself) and relations (predicates describing associations between objects).

Gentner distinguishes a *literal similarity* between the base and target domain vs. an *analogy* between these domains. Literal similarities involve commonality between the objects of different domains both in terms of their attributes and in terms of their relationships. By contrast, analogies do not include attribute similarities, only relational similarities (Gentner 1983).

For example, when saying the X12 star system is like our solar system, this is (according to Gentner) stating a *literal similarity*. The individual objects in each system (stars and planets) have key object-attribute matches. For example, X12 is a yellow, medium-sized star like our own. In addition, the two systems share common relational features. The planets in the X12 system revolve around X12, just like the planets in our system revolve around our sun.

By contrast, saying that a hydrogen atom is like our solar system is an *analogy*. There is far less in the way of object-attribute correspondence (i.e. the properties of an atom's nucleus are very different from the properties of the sun). However, important relationship predicates are preserved. Electrons revolving around the nucleus correspond with planet revolving around the sun. Also, the nucleus is far more massive than the electron, and thus exerts force to attract electrons just as the massive sun exerts force to attract the far less massive planets. Note that the force of the nucleus attracting electrons (strong force) is not literally the same as the force of the sun attracting planets (gravity). So, in this case we see an analogy between the two systems (similarity of relationships between objects), but not a literal similarity (very little in the way of object-attribute similarities between domains).

Gentner gives another illustration distinguishing between literal similarity and analogy by comparing these two assertions: "milk is like water" and "heat is like water" (Gentner 1989). The first case is *literal similarity*, because, for example, the property of liquidity is held in common by both statements and in addition there is a common causal relation involving the effect of pressure on flow of the substance. The second case is an *analogy* as it is much more difficult to find an inherent commonality in substance between "water" and "heat". There is, however, a relational similarity between the two. Specifically, Gentner associates the causal relation of pressure on water flow with a similar causal relation between temperature difference and heat flow.

Gentner hastens to add that the literal similarity vs. analogy distinction is not a black-and-white dichotomy, but rather a continuum. "Analogy and literal similarity lie on a continuum of degree-of-attribute overlap (Gentner 1989)."

The more there is a successful mapping among object-attributes between domains, the closer the mapping becomes to a literal similarity. To the degree that the similarities are constrained to object-relationships only, the mapping is an analogy. In this paper, I argue that a structure mapping between the base system of querying relational databases and the target system of querying XML documents is more of an analogy than a literal similarity, although some direct mappings of object-attributes between elements of each system are possible, giving a flavor of some degree of "literal similarity" as well.

A key feature of Gentner's theory is what she calls the *systematicity principle*. The gist of this principle is that higher-order predicates (i.e. those that build upon on lower-order ones and therefore give a more comprehensive statement about the system as a whole) will have more influence on the strength of an analogy than lower-order principles, which tend to operate independently in isolated subsystems. For example, consider again the analogy between a solar system and an atom. The distance predicate between the sun and a planet affects the attraction predicate. Similarly, the fact that a sun's mass is greater than the planet's mass causes the planet to revolve around the sun rather than vice versa. The fact that both of these higher-order predicates also hold for an atom's nucleus and electrons adds strength to the analogy, according to Gentner (1983).

From a pedagogical perspective, then, the key to producing useful analogies in order to foster quick learning of one domain (target) based on existing knowledge from another (base) is the ability to (a) demonstrate a wide variety of relational commonalities between the two domains and (b) identify relationships built on higher-order predicates (systematicity principle).

Another important feature of Gentner's theory is that it relies solely on similarity of syntactical structure, and not on similarity of underlying content meaning between the two systems. The implication of this is that it can speed up learning in new domains that bear structural similarities to old domains. In other words, operational competence in the target domain does not require “deep knowledge” for the target domain, but only “surface knowledge”, as long as students have a reasonably deep knowledge in the base domain. In this paper, we will leverage this fact to facilitate learning of XPath for XML queries (target) for students who have a solid knowledge of SQL for relational database queries (base).

Gentner applies the analogical reasoning theory to what she calls spontaneous learning, which is a natural learning process performed by people faced with an unfamiliar domain without the assistance of outside guidance. She describes analogical learning thusly: “Spontaneous analogical learning can be decomposed into subprocesses of (a) accessing the base system; (b) performing the mapping between base and target; (c) evaluating the match; (d) storing inferences in the target; and sometimes, (e) extracting the commonalities (Gentner 1989).” In this paper, we apply Gentner’s ideas on analogical reasoning and learning to the problem of teaching XML queries to students by leveraging on their already existing SQL knowledge.

In recent years, analogical pedagogy has been applied to several educational domains, including geoscience (Gee et al 2010), elementary science education (Guerra-Ramos 2011), physics education (Harrison 1993, Clement 1993), and mathematics education (Peled 2007),.

Harrison (1993) described a pedagogical process for using analogies to facilitate teaching. The process is composed of five steps, and you can see that many of these steps overlap with Gentner’s model of spontaneous learning described above. The steps are as follows (James 2003): (1) introduce the target concept (same as Gentner’s target domain); (2) establish learner’s familiarity with the teacher generated analogy (in other words, verify that students are familiar with the base domain); (3) identify the relevant features of the teacher generated analogy (i.e. point out the relevant concepts of the base domain); (4) map the similarities from source domain to target domain (this was step (b) in Gentner’s spontaneous learning process described above); (5) identify where the analogy breaks down (corresponds with Gentner’s step (c)); and (6) draw conclusions about the target domain.

Clement (1993) elaborated on the mapping process, suggesting that complex mappings in the analogy can be broken up into a chain of *bridging analogies*. In a study on the efficacy of this type of analogy-based teaching in the physics domain, he found that even novice teachers using these approaches can outperform experienced teachers using traditional proof-based and empirical pedagogical methods. In a way these bridging analogies serve a similar purpose as Gentner’s systematicity principle, by increasing the quantity and coherence of the structural edifice making up the analogy, and it appears that analogy-based learning can have a dramatic effect.

Analogical reasoning has also been associated with professional practice in the information technology field. For example, Dawson (2011) “provided evidence that mental modeling based on abstraction and analogous reasoning is used by professional analysts in the development of requirements specifications for system development”, particularly in the area of object oriented design.

It is clear from the above discussion that the relevance of analogical reasoning to both education and information technology has been supported by the literature. In the subsequent sections, we will apply analogical learning theory to the problem of teaching XPath queries to SQL-knowledgeable students. In our case, the base system is the world of relational databases, with its structure of two-dimensional tables related via correspondences between primary and foreign keys. The target system is the world of XML documents, composed of tree-structured hierarchies of elements with associated attributes. We will present several examples of base-to-target mappings, each pertaining to queries returning similar results from the two different structures. We will evaluate each of these mappings, and in the process identify both the strengths of the analogies and their limitations.

3. DESCRIBING THE BASE AND TARGET SYSTEMS: COMPARING DATA STRUCTURES

3.1 The Structure of Relational Databases: the Analogy’s Base System

The framework that students are exposed to in a typical database class is the relational database architecture. This is a model, begun by the work of Codd (1970) which, along with Chen’s (1976) seminal work in entity-relationship modeling, defines the current standard by which databases are designed in most modern-day business environments.

For example, consider a normalized database containing data about books and authors, as shown in figure 1.

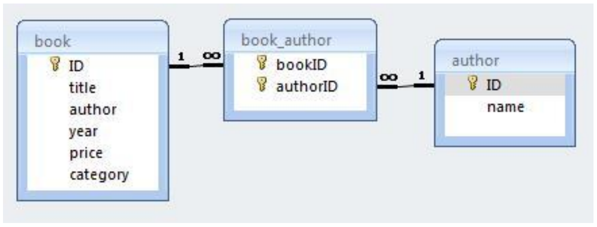
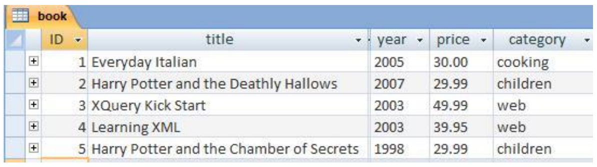


Figure 1: Normalized database structure of books and authors (M:N relationship)

In this database, there is a many-to-many relationship between books and authors, implemented by an intersection table between the two data tables. Database students will typically have a good understanding of such a structure, and will easily recognize the primary key and foreign key associations that make up the relationship between the book and author entities.

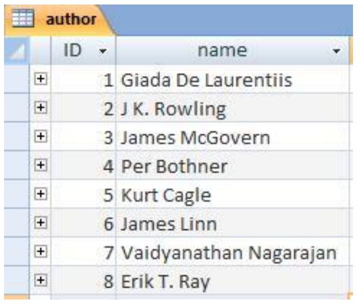
Similarly, students will be very familiar with the tabular structure of the data within tables, as shown in figures 2, 3, and 4. Based on this data layout, students are able to generate many queries to obtain information about books, authors, and their correspondences, as will be shown in subsequent sections.



book

ID	title	year	price	category
1	Everyday Italian	2005	30.00	cooking
2	Harry Potter and the Deathly Hallows	2007	29.99	children
3	XQuery Kick Start	2003	49.99	web
4	Learning XML	2003	39.95	web
5	Harry Potter and the Chamber of Secrets	1998	29.99	children

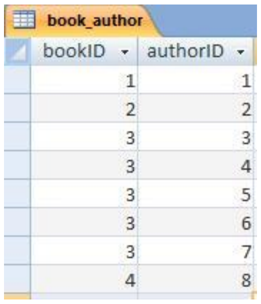
Figure 2: data in the Book table



author

ID	name
1	Giada De Laurentiis
2	J K. Rowling
3	James McGovern
4	Per Bothner
5	Kurt Cagle
6	James Linn
7	Vaidyanathan Nagarajan
8	Erik T. Ray

Figure 3: data in the Author table



book_author

bookID	authorID
1	1
2	2
3	3
3	4
3	5
3	6
3	7
4	8

Figure 4: data in the intersection table

The base system for our analogies, centered on a relational database architecture, involves concepts like tables, rows, columns, primary and foreign keys, etc. (Hoffer et al 2011 ch4). In addition, there are certain design principles that students should recognize, such as the major requirements for well-structured, normalized databases, including the importance of minimized data duplication and prevention of update anomalies. (Hoffer et al 2011 ch5), as well as the syntax and semantics of SQL queries for extracting useful information from relational databases (Hoffer et al 2011 ch6 and 7). These comprise the underlying form and operations of the base system that will be used in the analogical reasoning we will discuss for teaching the target system of XML and XPath.

3.2 The Structure of Markup Languages: the Analogy's Target System

All markup languages are derived from the Standard Generalized Markup Language (SGML) protocol (Coombs et al 1987). This standard defines the structural model and syntax for markup *documents*, which are comprised of a hierarchical arrangement of *elements* (implemented syntactically as *tags*). Elements may or may not contain *attributes*, which are name-value pairs. The hierarchical arrangement of elements in the SGML standard implies a *tree* structure in the underlying data model. In general, the tree data structure is composed of *nodes*, each of which can have a maximum of one parent node, and could contain any number of child nodes. Thus, elements in an SGML document are implemented as nodes in a tree data structure.

As an example, consider HTML's Document Object Model (DOM), as shown in figure 5.

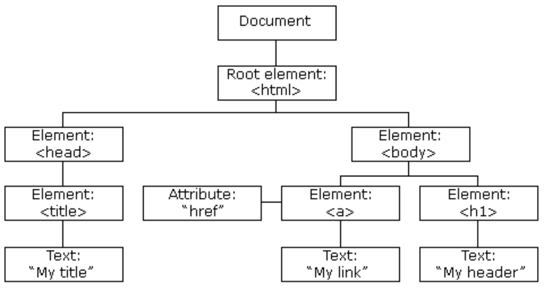


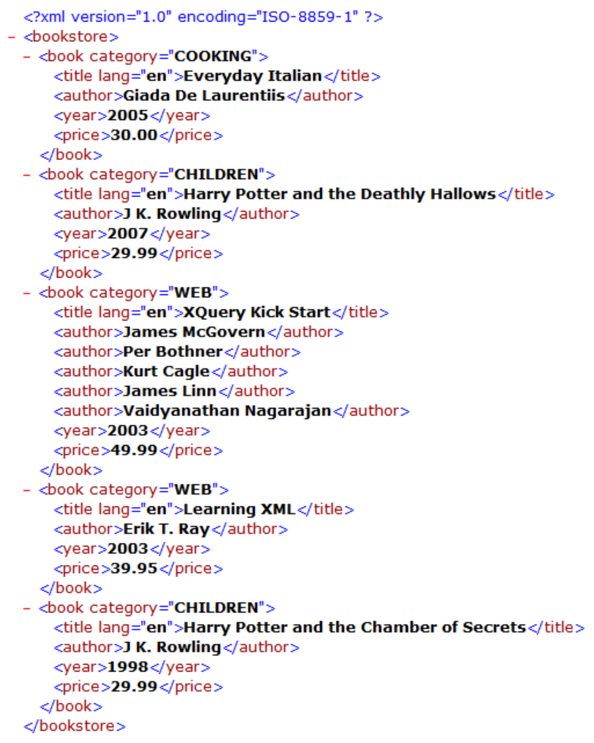
Figure 5: HTML document object model (DOM) (from `http://www.w3schools.com/HTMLDOM/default.asp`)

Anyone familiar with HTML will recognize its hierarchical nature. The root tag in the HTML tree is <html>, which is a parent node for <head> and <body>, each of which can further be parents for a variety of other element types, and so on. In general, markup languages have this hierarchical structure, and XML is no exception. For example, consider the XML document shown in Figure 6, which contains the same data content found in the database described earlier.

The root element for this is denoted by the <bookstore> tag, which is closed at the bottom with </bookstore>. In this document, the <bookstore> element encloses four <book> elements, each of which contains <title>, <author> , <year> and <price> elements, each of which in turn contains text values (atomic values) such as the name of the book, the names of the authors, the year, or the price. In addition, some elements include attributes (e.g. a book's category).

This structure is obviously very different from the relational database structure (involving tables with links via primary and foreign keys) that database students will be familiar with. Although there are some properties shared by both XML structures and relational structures (e.g. both are means of representing information), there are also many differences (e.g. tree vs. tabular structures; recursive vs..iterative search processes; elements, sub-elements, and element-attributes vs. rows, columns, and keys). Perhaps a more "literal similarity" to relational databases could be ascribed to spreadsheets, as both include tabular structures involving rows and columns.

Thus, comparisons between querying an XML document and querying a relational database cannot be done as a literal similarity, which assumes both that structural and attributive features of the compared objects match, and that relationships regarding these objects match as well. Rather, this kind of comparison depends on something more like Gentner's idea of analogy, in which there is little match in the structural features of the objects, but instead the comparison relies mostly or exclusively on conceptual relationships involved in the process of performing the queries.



```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <bookstore>
  - <book category="COOKING">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
  - <book category="CHILDREN">
      <title lang="en">Harry Potter and the Deathly Hallows</title>
      <author>J K. Rowling</author>
      <year>2007</year>
      <price>29.99</price>
    </book>
  - <book category="WEB">
      <title lang="en">XQuery Kick Start</title>
      <author>James McGovern</author>
      <author>Per Bothner</author>
      <author>Kurt Cagle</author>
      <author>James Linn</author>
      <author>Vaidyanathan Nagarajan</author>
      <year>2003</year>
      <price>49.99</price>
    </book>
  - <book category="WEB">
      <title lang="en">Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
    </book>
  - <book category="CHILDREN">
      <title lang="en">Harry Potter and the Chamber of Secrets</title>
      <author>J K. Rowling</author>
      <year>1998</year>
      <price>29.99</price>
    </book>
  </bookstore>
```

Figure 6: XML bookstore example (amended from http://www.w3schools.com/xpath/xpath_examples.asp)

In highlighting the differences between XML and relational databases, an instructor may want to note that the very same term in one context has a different meaning in another. Consider the word *attribute*. In the relational database context, an attribute (i.e. the concept derived from the ER model) is a column or field of a table. This is the lowest level of granularity in a database and refers to one specific datum. By contrast, the term attribute in the XML model refers to a specific (optional) component of an element. While it is true that attributes in the XML model can be thought of as a lowest-level datum, this is not the only possibility. The literal text values of lowest-level elements also contain atomic values (e.g. the values of the title elements in Figure 6). Also, note that the attributes (columns) in a relational database may in fact be implemented as sub-elements in an XML document. We see this with both author and title, comparing figures 1 and 6. But, it is also possible for a relational database attribute to be implemented as an attribute in XML (e.g. book's category).

More generally, when discussing analogies between relational databases and XML documents, caution must be made to prevent students from drawing too absolute of a link between "entity" and "attribute" of the ER model and "element" and "attribute" of the XML structure. Entities in ER models (and rows in relational databases) do not necessarily serve the same purpose as elements in XML structures, although oftentimes they do.

Nevertheless, despite the relative lack of clear-cut structural commonalities, there is a key similarity between XML documents and SQL databases that can be used to foster analogical pedagogy. This is the fact of *queries*, which are actions that users and other information systems agents can employ to glean relevant information from these quite dissimilar data structures. In particular, the process of deciding which subsets of data to show, the conditions under which to show them, the level of aggregation or specificity to return, and the order and format of the desired results, are common requirements that apply to the task of retrieving the most useful information from both types of data structures.

This leads to the possibility of using analogical reasoning to foster quick learning of XML query mechanics by making use of students' already existing knowledge of SQL query mechanics.

4. SQL–TO–XPATH ANALOGIES

Because of XML's hierarchical nature, navigation through an XML document requires the use of tree-processing algorithms, and there are class libraries in Java, PHP, and .NET that could be used to facilitate teaching of XML navigation in programming classes. This is to be contrasted with the iterative (nested looping) nature of searching through the two-dimensional results of a database query result. Although nested looping is a basic programming skill, likely to be learned by most IS students, tree processing (which requires recursion) is often not covered in IS curricula, especially those with a minimum of programming requirements (Topi et al 2010, Saulnier and White 2012).

It is unfortunate that IS students don't receive more detailed instruction of complex data structures like trees, especially in light of the increasing ubiquity of XML. However, a database class can make up for this gap by giving some coverage of tree structures if we contrast trees to table structures, as discussed earlier. Furthermore, the utilization of XPath, a nonprocedural query language for retrieving XML information, can help solidify understanding of tree structures in much the same way that coverage of SQL queries foster students' knowledge of relational database structures.

A good way to convey to students the similarities of task and function between relational database queries and XML document queries is to make this analogy: *An XPath query is to XML documents as an SQL query is to relational databases*. Both XPath and SQL are non-procedural languages whose syntactic and semantic structures reflect the underlying structures of their respective data architectures. And the results of each type of query is of the form consistent with the overall data architecture to which it applies, as we will explore in more detail.

However, keep in mind that the following analogy is far less accurate: *XPath is to XML documents as SQL is to relational databases.* This is because SQL includes data definition language as well as update/insert/delete functionality. None of this is present in XPath. Taking the analogy between XPath and SQL too far may give students the wrong impression about just what can be done with XPath.

We should also consider that, as we'll see later, there are some operations that can be done in SQL which have no corollary capabilities in XPath. For this reason, there may be people who would argue that a better analogy for SQL is XQuery, which includes functionality not available in XPath. But I disagree with this, for two reasons. First, XQuery includes procedural constructs (loops, if-statements, etc.), and is thus a procedural language. In this respect, it is more appropriately associated with procedural SQL extensions (T-SQL or PL-SQL). XPath and SQL share the common distinction of being nonprocedural languages. Secondly, the returned values of both core (non-procedural) SQL queries and XPath queries *exclusively* reflect the structure of their respective data sources. SQL query result sets are always tabular. XPath query results are always node sets (trees). This is not true for their procedural extensions.

Having said this, there are clearly associations that could be made between XQuery (which builds upon XPath) and the procedural SQL extensions. This is beyond the scope of the current paper, but could be fruitful avenue of future research in applying analogical pedagogy to the problem of XML document processing.

4.1 Analogies of Query Output: Result Sets vs. Node Sets
Given the analogies of data structure described in Section 3, a natural follow-up is to relate the structures of query results in the respective data architectures. The first step in this regard it to compare the types of outputs that come from queries of the base and target systems. Whereas a SQL query (SELECT statement) produces a *result set* (i.e. a two-dimensional tabular structure of rows and columns), an XPath query (path expression) produces a *node set* (i.e. a list of nodes, each of which could be the root of a tree), as shown in figure 7.

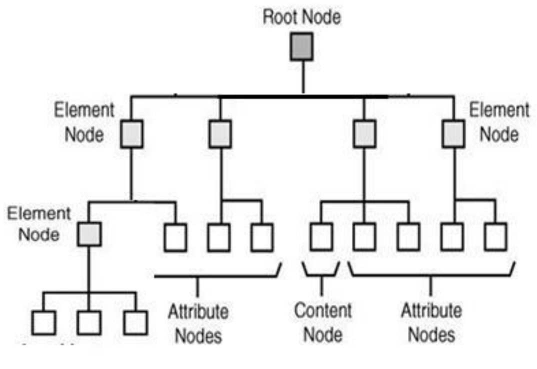


Figure 7: General structure of a node set returned from an XPath query

This figure depicts a tree, with a single root. Below the root is the *node set*, i.e. all of the nodes (elements in this case) that match the criteria of the XPath query. In other words, the level directly underneath the root comprises the set of nodes that satisfied the query, each node of which can be an element (in which case it will be the root of a subtree) or an element's content (an atomic value), or an element's attribute. This is the general structure of a node set returned from an XPath query.

Obviously, the structure of a node set is very different from the structure of a SQL result set. So, in Gentner's terms this is not a literal similarity. The question is, can useful analogies be brought to bear that highlight both the similarities and the differences of the data structures, and thereby foster student learning of XML processing in a time-efficient manner? In order to answer this question, we need to know the purpose of each of the items from the base and target data structures in the context of the overall problem of querying the data structures. In other words, we need to answer these two questions: (a) how are columns and rows and tables used in the syntactical structure of SQL queries, and (b) how are nodes, attributes, and paths used in the syntactical structure of XPath queries?

4.2 Analogies of Syntactical Structure: Select Statement vs. Path Expression
As stated earlier, a SELECT statement in SQL is analogous to an XPath *path expression*. When teaching about SQL SELECT statements, it is typical to identify and describe the major clauses of the query, often expressed as SELECT…FROM…WHERE (Hoffer et al 2010, pp261-263). The SELECT clause determines the order and content of the columns in the result set that returns from the query. The FROM clause specifies the tables and/or views that are used by the query. The WHERE clause specifies conditions under which rows from the tables in the FROM clause will be included in the final result, as well as join conditions if there are multiple tables involved. After students master these primary clauses, they go on to learn about GROUP BY and HAVING, both used in conjunction with aggregation, as well as ORDER BY for sorting results.

Similarly, when teaching about XPath *path expressions*, it is useful to break out and describe their main components. For a database class, making analogies between path expression clauses and SELECT statement clauses is helpful for fostering students' understanding.

Like SQL SELECT statements, XPath path expressions provide the criteria for which to select data from the overall XML document; in the case of XPath the result is formatted as subtrees. A path expression is composed of a series of *location steps*, each of which defines selection criteria for the corresponding level of the XML tree (the data source being queried). A location step consists of an *axis*, a *node-test*, and an optional set of *predicates* that refine the node test. The node test and predicates serve a similar function for path expressions as the column specifications and WHERE clause in a SELECT statement. The axis gives the option to reference a node-set relative to the current node (parent, sibling, child, etc.); in this way path expressions can specify criteria for node relations as well as specifying criteria for the nodes themselves.

When teaching about path expressions, it is useful to start with simple examples, describe their structure, and show their results. For the books document of Figure 6, a good starting example path expression is:

```
/bookstore/book/title
```

This query specifies, in absolute terms, the paths to nodes that will be returned from the XML document. In this expression there are three steps, each with a node test. At the highest level, we focus only on elements named bookstore. At the second level, only those named book. And at the third level, only those named title. What you obtain from this query is a set of elements (nodes) from the XML document that are named title and that are sub-elements of a book element where the book elements must be sub-elements of a root bookstore element. The returned node set is shown below.

```
<root>
  <title lang="en">Everyday Italian</title>
  <title lang="en">Harry Potter</title>
  <title lang="en">XQuery Kick Start</title>
  <title lang="en">Learning XML</title>
</root>
```

What is the analogy between this and a SQL statement on a similar relational table of books (like shown in Figure 2)? A query for this table may look something like this:

select title from book

So, here an analogy is made between *node tests* in XPath and *column specifications* in SQL. In both cases from above, all titles from all books are returned. In particular, note that it is the node test at the end of the path expression that corresponds with the column specification of the SQL query. (Note: although the XPath query is displaying entire elements instead of just the atomic values, this can be done by applying the XPath value() function. For purposes of discussion in this paper, we will not utilize the value function in our queries).

If we want to show both the title and price of each book in the XML document, we can use the following path expression:

/bookstore/book/title | /bookstore/book/price

which produces the following results:

```
<root>
  <title lang="en">Everyday Italian</title>
  <price>30.00</price>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
  <title lang="en">XQuery Kick Start</title>
  <price>49.99</price>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</root>
```

The analogy with SQL would be the following:

select title, price from book

So, an analogy can be made between the *comma* in the SQL statement (which delimits the columns of the SELECT clause) and the *pipe* (vertical bar) symbol in XPath, which similarly delimits paths that will be returned. However, when making this analogy, it is important to also point out the differences. Note that the pipe symbol is allowing retrieval of *multiple paths in the tree*, whereas the comma is retrieving *multiple attributes of the table or join*. Recall our earlier discussion of Gentner's spontaneous analogical learning process: (a) accessing the base system; (b) performing the mapping between base and target; (c) evaluating the match; (d) storing inferences in the target target; and sometimes, (e) extracting the commonalities (Gentner 1989). Step (c) is an important component of the process, and instructors should be sure to critically evaluate analogies as they are presented to the students. Some analogies are stronger than others.

The previous two examples utilized analogies to familiar SELECT clause constructs. For an analogy to WHERE clause constructs (i.e. conditions for returning rows from the table), consider the following path expression, which includes a predicate:

/bookstore/book[price>35]/title

The predicate **[price>35]** restricts the second level of the paths such that only the titles of those book elements whose price sub-elements have values greater than 35 will be returned, as shown below:

```
<root>
  <title lang="en">XQuery Kick Start</title>
  <title lang="en">Learning XML</title>
</root>
```

The simplistic SQL analogy would be the following:

select title from book where price > 35

So, an analogy can be made between *the conditions in a WHERE clause* and the *conditions in a predicate*.

In the above examples, a column in a SQL table was mapped onto a sub-element in an XML document. Recall that the purpose served by columns in a SQL table could also be accomplished using an attribute in an XML document. So, an alternative set of analogies can be made for this mapping. For example, the following two mappings are possible.

1) /bookstore/book[@category="COOKING"]/title

    maps to

    select title from book where category = 'cooking'

2) /bookstore/book[title="Everyday Italian"]/@category

    maps to

    select category from book where title = 'Everyday Italian'

Here, you can point out the lack of clear-cut one-to-one correspondences between concepts in the base domain and concepts in the target domain. Column specifications in

SQL queries can form analogies to either node tests or to element attributes in XPath queries.

4.3 Analogies with Joins

Within the XML tree structure shown in figure 6, there is also the possibility of many authors for a book, and many books for an author as well. Note, however, that unlike with normalized databases, XML hierarchies will often include duplicate data. For example, J. K. Rowling appears twice in the XML document of figure 6..

Nevertheless, there are operations in XML queries that are similar in some ways to join operations in relational databases. For example, in a relational database, you may want to show all the authors for a particular book, using a join query like this:

select a.name from author a, book_author ba, book b where a.id = ba.authorID and b.title = ba.book and b.title = 'XQuery Kick Start'.

An analogous XPath query for returning all the authors for a given book would be:

//bookstore/book[title='XQuery Kick Start']/author

Here we see that a combination of node tests and predicates can be used to gain similar results as a multitable join query Specifically, the predicate is associated with a node test from the step *preceding* the step that contains the node test analogous to the column specification from a SQL query. Note that in a tree structure, there is a one-to-many relationship between the parent node and the child nodes (a parent can have multiple children). This relationship, combines with the allowance of duplicate data (i.e. author names duplicated throughout the document as subelements of books), allows us to make analogies between many-to-many relationships in relational databases and many-to-many relationships in XML documents.

What if we wanted to see all books by a particular author? In this case, the SQL query would look like this:

select b.title from author a, book_author ba, book b where a.id = ba.authorID and b.title = ba.book and a.name = 'J. K. Rowling'.

To perform an analogous operation in the XML document, you can do the following:

//bookstore/book[author='J. K. Rowling']/title

Consider the analogies between the XPath path expressions and the SQL joins. To change from the first SQL query (authors of a book) to the second (books written by an author) involves swapping the column specification in the SELECT clause with the testing field of the WHERE clause. Similarly, performing that same modification in the path expressions involves swapping the lowest-level node test with the predicate test element. In this analogy, SELECT clause items are like node tests, and WHERE clause elements are like predicates.

Here, we have applied Gentner's *systematicity principle*. We build on two individual lower-order analogies: (1) a WHERE clause condition is like an XPath predicate and (2) a SELECT clause column specification is like an XPath node test. We combined the two in order to produce this higher-order analogy: swapping the column specification with the WHERE clause condition is like swapping the predicate with the node test. Another way to look at this case is in terms of Clement's *bridging analogy* concept. The previous (lower order) analogies of node test-for-column specification and predicate-for-WHERE condition served as bridges for the overall analogy of how to modify a query to produce reciprocal results.

4.4 Analogies with Aggregation and Grouping

In both databases and XML documents, there will be the need to get aggregate information such as sum, counts, averages, etc. XPath analogies to SQL aggregate queries are not as direct as the analogies already stated, and aggregation in XML queries often require the use of XQuery or XSL (eXtensible Stylesheet Language), which is beyond the scope of this paper. Nevertheless, some limited aggregation can be done using XPath alone, and learning these can be done through analogy.

For example, suppose you want to know the total number of books in the database. Such a query would look something like this:

select count(*) from book

Here, **count** is an aggregate function, and the query returns a result set of one row consisting of one column, that column containing the number 5, which is the total number of records in the book table (see figure 2).

Similarly, there is a **count** function in XPath, which returns the same sort of information as the count function in SQL. The following XPath expression would also return a 5, the total number of book elements in the XML document of figure 6.

count(/bookstore/book)

However, when evaluating this analogy we see that it is not quite as strong as the previous analogies. Whereas the SELECT statement returns a result set, the XPath expression does not return a node set, but an individual value. This is a weaker analogy than the previous ones because it does not build upon the result-set to node-set mapping. Nevertheless, the two analogous queries both fulfill the same purpose, which is to get the total number of books from the data source.

More problems occur when attempting analogies for aggregation with grouping. This facility is not provided by XPath alone, but requires either XQuery or XSL to complete the operation. For example, the following query has no direct analogy in XPath:

select title, count(*) from book
where category = 'children' group by title

During Gentner's spontaneous learning process, an individual would get stuck at step (b)) performing the mapping between base and target. This is a breakdown in the analogy. However, we learned from Harrison (1993) that from a pedagogical perspective, pointing out the failures of attempted analogies can be as useful as identifying successful ones. We want to impress upon the student than there are many differences between the two data structures as well as their query mechanics and capabilities.

Detailed discussion of the extensions of the base and target query languages is beyond the scope of this paper. But broadening the scope of this study will likely uncover many useful analogies, including the ability to map aggregation with grouping from the SQL to XPath.

5. CONCLUSIONS

Analogy is much less precise and rigorous than teaching from first principles, and is not sufficient by itself for providing deep understanding of a topic or skill. But it can serve a useful role by leveraging students' previous knowledge in one domain in order to speed up learning in another. Considering the time constraints in a typical IS curriculum, and the need to cover a broad scope of distinct standards, protocols, languages, and data formats, it makes sense to use intuitive heuristics such as analogical reasoning where the opportunities arise. In this paper, we looked at one such opportunity, for facilitating XML querying skills by making use of students' existing knowledge in relational database queries. Specifically, we applied Gentner's structure mapping theory and the pedagogical methods that have arisen from it into one particular area in IS education focused on database and XML queries.

There is much promise for future research in this area. Within the context of database concerns, many more analogies can be identified by expanding from the core languages of SQL and XPath to their extensions of PL-SQL, T-SQL, XQuery, and XSL. The use of analogy also brings promise to other areas of information systems education, including programming, design principles, modeling methodologies, case studies, and host of other areas.

A final promising continuation of this research, and one which will be needed in order to validate the theoretical principles outlined in this paper, is to run empirical studies on the practical use of analogical reasoning in information systems courses. Clement's (1993) dramatic results show exciting promise, but these should be replicated in the IS domain.

6. REFERENCES

Chen, Peter Pin-Shan. (1976) "The Entity-Relationship Model-Toward a Unified View of Data" ACM Transactions on Database Systems Vol. 1 No. 1, pp. 9-36

Clement, John (1993) "Using Bridging Analogies and Anchoring Intuitions to Deal with Students' Preconceptions in Physics" Journal of Research in Science Teaching Vol. 30 No. 10, December 1993, pp. 1241-1257

Codd, E.F. (1970) "A Relational Model of Data for Large Shared Data Banks" Communications of the ACM Vol. 13 No. 6, pp. 377-387

Coombs, J.H., Renear, A.H., DeRose, S.J.. (1987) "Markup Systems and the Future of Text Processing" Communications of the ACM, Vol. 30, No. 11, pp. 933-947

Dawson, Linda. (2011) "Cognitive Processes in Object-Oriented Requirements Engineering Practice: Analogical Reasoning and Mental Modelling" 20th International Conference on Information Systems Development (ISD). Edinburgh, UK. Aug. 2011.

Falkenhainer, B., Forbus, K.D., Gentner, D. (1989) "The Structure-Mapping Engine: Algorithm and Examples" Artificial Intelligence Vol. 41, pp. 1-63

Forbus, Kenneth D. and Gentner, Dedre (1991) "Similarity-Based Cognitive Architecture" ACM SigArt Bulletin Vol. 2 No. 4, pp. 66-69

Forbus, K.D., Mostek, T., Ferguson, R., Swart, R. (2002) "An Analogy Ontology for Integrating Analogical Processing and First-Principles Reasoning" Proceedings of the Eighteenth National Conference on Artificial Intelligence pp. 875-885

Gee, B.D., Uttal, D.H., Gentner, D., Manduca C., Shipley T.F., Tikoff B., Ormand, C.J., Sageman, B.. (2010) "Analogical Thinking in Geoscience Education" Journal of Geoscience Education, Vol. 58, No. 1, pp. 2-13

Gentner, Dedre (1983), "Structure Mapping: A Theoretical Framework for Analogy." Cognitive Science Vol. 7, pp155-170.

Gentner, Dedre (1989), The mechanisms of analogical learning, in Similarity and Analogical Reasoning. Cambridge University Press, Cambridge, England.

Harrison, Allan G. and Treagust, David F. (1993) "Teaching with Analogies: A Case Study in Grade-10 Optics" Journal of Research in Science Teaching Vol. 30 No. 10, December 1993, pp. 1291-1307

Hoffer, J.A., Ramesh, V., Topi, H. (2011), Modern Database Management 10th Edition. Pearson – Prentice Hall, Upper Saddle river, NJ

James, Mark Charles (2003), "The Influence of Analogical Reasoning Instruction on the Pedagogical Reasoning Ability of Preservice Elementary Teachers", PhD Dissertation, Kansas State University, ProQuest Information And Learning Company, Ann Arbor, MI.

Olsen, D., Cooney, V., Marshall, B., Swart, R. (2005) "Toward Full Integration of XML and Advanced Database Concepts" The Review of Business Information Systems Vol. 9, N. 4, pp. 13-22

Peled, Irir (2007) "The Role of Analogical Thinking in Designing Tasks for Mathematics Teacher Education: An Example of a Pedagogical Ad Hoc Task" Journal of Mathematics Teacher Education Vol. 10 No. 4, pp. 369-379

Saulnier, Bruce and White, Bruce (2012) "IS 2010 and ABET Accreditation: An Analysis of ABET-Accredited Information Systems Programs" Journal of Information Systems Education Vol. 22 No. 4, pp. 349-355

Topi, H,, Valacich, J.S., Wright, R.T., Kaiser, K.M, Nunamaker, J.F., Sipior, J.C., de Vreede, G.C. (2010) "IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems" Communications of the Association for Information Systems Vol. 26, No. 18

Wagner, W.P., Pant, V., Hilken, R. (2008) "Adding XML to the MIS Curriculum: Lessons from the Classroom" Journal of Information Technology Education Vol. 7, pp. 35-45

AUTHOR BIOGRAPHY

Michel Mitri is a Professor of Computer Information Systems at James Madison University, where he teaches software development, database, and business intelligence courses. His research interests include applications of AI to business and educational domains. Dr. Mitri currently serves as the interim department head of the CIS and Business Analytics department.

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096