Investigating XQuery for Querying Across Database Object Types

Nancy Wiegand
University of Wisconsin—Madison

Abstract

In addition to facilitating querying over the Web, XML query languages may provide high level constructs for useful facilities in traditional DBMSs that do not currently exist. In particular, current DBMS query languages do not allow querying across database object types to yield heterogeneous results. This paper motivates the usefulness of heterogeneous querying in traditional DBMSs and investigates XQuery, an emerging standard for XML query languages, to express such queries. The usefulness of querying and storing heterogeneous types is also applied to XML data within a Web information system.

1. Introduction

As the Web is being considered to be one huge database of information, DBMS-type query languages are being developed for XML data. XQuery, derived from Quilt [CRF00, RCF00], is an emerging standard for XML query languages [CCF+01]. XQuery is noted for its heterogeneous ability to query both documents and databases. Here, XQuery is explored to query heterogeneous relation types within a traditional database converted to XML. Current DBMS query languages do not allow such querying. In XML, however, there is a looser notion of type creating the possibility of allowing heterogeneous querying and results. The example queries presented in this paper are modeled after queries and information found in the working specifications for XQuery [CCF+01, CFM+01] and the Quilt papers [CRF00, RCF00].

Currently, database organization is by type of object and querying is done by database type. For example, the FROM clause of an SQL query can only specify one relation unless a join is to be performed. Joins are not considered here as a query across types because the result is a set of homogeneous entities. The discussion in this paper applies equally to Relational, Object-Relational, and Object-Oriented DBMSs because they all only allow homogeneous results.

2. Motivating Examples

Querying using a different type of database partitioning than object type is a needed functionality if a user wants to gather information based on descriptive criteria regardless of object type. Examples include retrieving by size, time, color, date, ownership, material composition, and location. As an initial example of querying across database types, consider a database of a museum's historical items organized by type of entity. That is, kitchenware, ornaments, tools, jewelry, etc. each form their own groups (relations or class extents). Suppose, however, a user wants to find all items made of bronze. Currently, this can only be done by posing a separate query on each potential entity type to retrieve items (e.g., SELECT * FROM kitchenware WHERE material="bronze"; SELECT * FROM tools WHERE material="bronze"; etc.). In addition to being tedious, some objects may be left out because the user did not think to query that type. Also, there is no way to hold the separate results as one heterogeneous result. Furthermore, the user might want to pose a subsequent query on the results to restrict on some other criteria. Currently, this again would have to be done separately on each result of the initial queries. What is needed is a FROM construct that allows a list of relations or a * wildcard to denote all relations. Also, a mechanism is needed to hold heterogeneous query results that can be further queried.

A compelling example for DBMSs to allow querying across types and provide for heterogeneous groups is querying by spatial location. That is, a user often wants "all" information associated with a geographic area. Geographic or spatial information is currently being added to information stores. For example, the Alexandria Digital Library project [ADL, FFL+95] uses a "geographic footprint" as a new paradigm for information access [Goo98]. This paradigm allows information to be gathered or organized spatially in ways not now found in libraries. Searching can be done by interacting with maps or gazetteers. An example is being able to find all books on France when "France" is not part of the book title or subject. Also, more undefined criteria can be used such as finding data on the "Bay Area". However, querying

by geographic area may result in different types of data being gathered. For example, [Goo98] mentions a need to retrieve all resources of a city planning department or to gather data from multiple sources such as the Census and image archives that relate to a particular geographic area.

To implement the geographic footprint idea from a DBMS perspective, each tuple or object would need either a value for a geographic footprint-type attribute or actual spatial coordinates. In the first case, a user could then retrieve a variety of information on a specified area, such as the "Bay Area". (Or, if actual spatial coordinates are used, a query can be done over an arbitrary area by supplying a range of coordinate values or using a screen tool to create a rubber band box to specify an area.) To retrieve all relevant data for the Bay Area, the following type of query construct is needed.

```
CREATE BayAreaData AS
SELECT *
FROM *
WHERE footprint = "Bay Area"
```

The FROM * clause would automatically range over all types encountered. SELECT * would retrieve all attributes and is the safest approach because the various object types may have limited attribute identifiers in common. In fact, if the footprint attribute does not occur, it is important that the query processor ignores those objects and not generate an error. Also, the heterogeneous result of this type of query needs to be able to be named, stored, and available for further querying. This is shown by the new group "BayAreaData." Briefly, implementation could be done using iteration over relations and the creation of a composite result data structure that retains type information in subheaders.

Querying across database types is not possible in current DBMSs because of the strict adherence to type. Subtyping in OODBMSs produced discussions of heterogeneous sets and their problems [BBO88]. [Bla95] states heterogeneous collections can only consist of subtypes in OQL[C++]. Also, ODMG 2.0 specifications state "the distinguishing characteristic of a collection is that all elements are of the same type" [CB97, p. 20] except for objects that are part of an inheritance hierarchy. However, we have shown that there are many cases in which the objects of interest do not naturally fit into an inheritance hierarchy.

Parks

name	acres	attraction	footprint
Indian Lake	442	1857 chapel	Madison Area
Fish Lake	252	fishing	Sauk City Area
Badger Pr.	339	softball	Madison Area

Restaurants

name	specialty	MCD	footprint
Imperial Garden	Chinese	Middleton-C	Madison Area
Dorf Haus	German	Roxbury-T	Sauk City Area

Figure 1: RDB relations, sample data: (MCD is Minor Civil Division: city, village, or town).

[KKS92] present an XSQL query without a FROM clause to be able to return objects from all classes that contain a specified attribute and value. Their example is that both people and organizations satisfy a query to find all winners of Nobel prizes. They propose allowing varying degrees of query type correctness to relax adherence to type when needed. [BDH+96] note that a query to find a string value anywhere in a database is not expressible in a general relational algebra statement. They show UnQL can express this and provide a "deep" search, but the result may be heterogeneous. The FROM * notation was mentioned in [NDM+00] but with a different meaning. Although they also did not want to require users to specify the Web XML data files for searches or queries, they were not purposely accommodating a heterogeneous result nor were they suggesting that FROM * be applied to traditional DBMSs.

3. Querying Across Database Types

This paper explores querying across database types in the realm of XML query languages. XML by itself does not have a strong concept of type, and XML query languages do not require a schema or a DTD.

Here, a traditional database is converted into XML and queries are posed in XQuery. Suppose a relational database for Dane County has travel information that includes relations for parks, restaurants, hotels, taxicab companies, and so on. Also, assume each relation has an attribute that contains a value for a geographic footprint (Figure 1).

```
<?xml version="1.0"?>
<!DOCTYPE DaneCounty SYSTEM "travelinfo.dtd">
<DaneCounty>
<park pno="p1">
   <name> Indian Lake County Park </name>
    <acres> 442 </acres>
    <attraction> 1857 Chapel </attraction>
   <footprint> Madison Area </footprint>
</park>
<park pno="p2">
    <name> Fish Lake Park </name>
    <acres> 252 </acres>
    <attraction> fishing </attraction>
    <footprint> Sauk City Area </footprint>
</park>
etc.
<restaurant rno="r1">
   <name> Imperial Garden </name>
    <specialty> Chinese </specialty>
   <MCD> Middleton-C </MCD>
    <footprint> Madison Area </footprint>
</restaurant>
<restaurant rno="r2">
    <name> Dorf Haus </name>
    <specialty> German </specialty>
    <MCD> Roxbury-T </MCD>
    <footprint> "Sauk City Area" </footprint>
</restaurant>
etc.
</DaneCounty>
```

Figure 2: RDB relations converted to XML and stored in "DaneCounty.xml".

The transformed tables are stored in an XML file, "DaneCounty.xml" (Figure 2). In this section, all data are stored in one file, contrary to typical RDB to XML examples such as in [CCF+01, CFM+01], to show the ability of XQuery to query over and retrieve elements of different types within one document.

Needed Functionalities:

To query across database types, a query language is needed that provides high level constructs for retrieving and storing heterogeneous objects. Specifically, the needed functionalities are:

- a high level and user-friendly method to query multiple type extents in one statement,
- a mechanism to name and store the result of the query,
- a method to sort the result by element type,
- an ability to further query the result, and
- the need to not generate errors for non-existing subelement types.

The rest of this section explores queries to meet the above requirements. In addition, other query features are shown relevant to spatial queries such as executing functions within a query.

- **Query multiple type extents and store the result**.

The type of query we want to execute is:

Query: **Find all the travel information for the Madison Area (from the Dane County XML file).**

The desired result of this query will be a collection of all the elements that have "footprint" equal to the "Madison Area". The result is expected to be heterogeneous, that is, consisting of elements representing different concepts. The following expressions in XQuery use an asterisk instead of specifying an element name. This allows the query to range over multiple types. Because XQuery includes syntax from XPath 1.0 [CD99], the queries can be written as path expressions or FLWR expressions.

Path expression:

```
document("DaneCounty.xml")//*[footprint=
        "Madison Area"]
```

FLWR expression:

```
<MadisonResult>
{
    FOR $b IN document("DaneCounty.xml")//*
    WHERE $b/footprint="Madison Area"
    RETURN
        $b
}
</MadisonResult>
```

The query result consists of *all* elements that have a "footprint" subelement equal to "Madison Area". The FLWR expression specifies a named persistent result containing the following heterogeneous elements. The result of the path expression does not have the common root node.

Result:

```
<MadisonResult>
<park pno="p1">
 <name> Indian Lake County Park </name>
 <acres> 442 </acres>
 <attraction> 1857 Chapel </attraction>
 <footprint> Madison Area </footprint>
</park>
```

<**park** pno="p3">
<name> Badger Prairie Park </name>
<acres> 339 </acres>
<attraction> softball </attraction>
<footprint> Madison Area </footprint>
</**park**>

<**restaurant** rno="r1">
<name> Imperial Garden </name>
<specialty> Chinese </specialty>
<MCD> Middleton-C </MCD>
<footprint> Madison Area </footprint>
</**restaurant**>
</**MadisonResult**>

The above query can also be expressed in XML-QL [DFF+98, FSW]. The * below, an abbreviation for $*, matches any sequence of edges.

```
WHERE<*>
    <footprint>Madison Area</footprint>
    </>ELEMENT_AS $b
        IN  document("DaneCounty.xml")
CONSTRUCT <MsnResult> $b </MsnResult>
```

The queries presented so far specify the XML document but do not specify element names. The analogy to queries in a regular DBMS is that only the database itself would need to be specified and not individual relations, class extents, or collections. This idea may be reminiscent of the Universal Database assumption, but the focus here is on a heterogeneous result which is not part of that model.

- **Sort by element types.**

For clarity in presenting heterogeneous results, the XQuery FLWR expression can be written to sort the result by element type. Because functions in the XPath core function library are included in XQuery, the "name" function can be used to return the name of the current node.

```
<MadisonResult>
{
    FOR $b IN document("DaneCounty.xml")//*
    WHERE $b/footprint="Madison Area"
    RETURN
        $b SORTBY (name(.))
}
</MadisonResult>
```

- **Pose further queries on heterogeneous results and not generate errors.**

Another requirement of querying across database types is to be able to pose further queries on the heterogeneous result of an initial query, and, in particular, not generate errors for missing subelements. The beginning "DaneCounty.xml" document already consisted of heterogeneous elements, but the first query below illustrates the case in which all elements in an initial result have an additional attribute in common. The second query presents a consideration when a common attribute does not occur, a condition more likely in a subsequent query.

Common subelement:

Query: Find all the elements in MadisonResult that have "Badger" as part of their "name". In this case, all elements have the specified subelement (i.e., "name") in common.

```
<badgerset>
{
  FOR $b IN document ("MadisonResult.xml")//*
  WHERE contains ($b/name,"Badger")
  RETURN
      $b
}
</badgerset>
```

The result includes the element for Badger Prairie Park and with the full database may also include, for example, an element for Badger Cab Company.

Specified subelement is missing:

Query: Find information specific to the city (MCD) of Middleton. Here, after retrieving information on the "Madison Area", the user wants to refine the result to just get information for Middleton. This second type of subsequent query ranges over elements that do not have an additional common subelement. That is, park elements in MadisonResult do not have an MCD subelement.

```
<MiddletonSubset>
{
  FOR $b IN document ("MadisonResult.xml")//*
  WHERE $b/MCD="Middleton-C"
  RETURN
      $b
}
</MiddletonSubset>
```

If this query were processed by generating subqueries over all main entity types (as could be done in a traditional DBMS), it is important that entities/elements not containing appropriate subelements are "skipped over" without an error message being generated so that the user gets as much information as possible. This situation is

related to Issue 30 (Queries with Invalid Content) in [CCF+01]. We further suggest that, in addition to the query result, a processing option exists for the user to get an indication of the number and/or type of elements that were skipped either because the MCD subelement did not exist or existed but did not have a value equal to "Middleton-C".

- **Functions.**

With a user-defined function capability, the more arbitrary type of spatial query mentioned earlier can be supported. That is, suppose each element has a "spatial_location" subelement containing actual x,y coordinates (instead of text) and a "SPATIALLY_WITHIN" function exists. Then, the following would be XQuery syntax for finding all elements geographically within an area denoted by bounding box parameters.

```
<ItemsInBoundingBox>
{
    FOR $b IN document("DaneCounty.xml")//*
    WHERE
       SPATIALLY_WITHIN($b/spatial_location,
           bounding_box_parameters)
    RETURN
        $b
}
</ItemsInBoundingBox>
```

- **Results consisting of references to objects.**

In object type DBMSs, the result of a query may consist of references to objects rather than the values of the objects. However, element IDs are not defined in XQuery at present due to the complexities of their use in updates and persistence outside the Web.

4. Querying XML Data within a Web-Based Information System

An especially useful application of XML query languages is for a controlled subset of the Web representing a distributed Web-based information system. Current information systems often have limited search and query facilities with typical results just consisting of a list of URLs for data sources. Applying full-power DBMS-type querying would greatly enhance their functionality.

4.1 Ranging over all available data

In XQuery, a query can easily be expressed to range over all available data sources in a Web information system. This is because the document function is not required. Information system data sources would likely have common subelements (here, "footprint") such that a query ranging over element types in unspecified documents would have a useful heterogeneous result.

Query: Find all the travel information for the Madison area searching over <u>all</u> available sources.

Path expression:

```
//*[footprint = "Madison Area"]
```

FLWR expression:

```
<ExtendedResult>
{
    FOR $b IN //*
    WHERE $b/footprint="Madison Area"
    RETURN
        $b
}
</ExtendedResult>
```

Again, as a comparison to other languages, XML-QL does not have the notion of an implicit context node as in XQuery. Instead, a function with a document parameter would need to be applied to multiple documents to achieve the result of the above queries.

4.2 Information integration

Heterogeneous results are also important in a distributed Web-based information system in which locally stored data contain similar concepts that must be integrated over multiple jurisdictions. For example, land parcel data have formats and codes that are unique within most jurisdictions. Suppose a user wants to retrieve all information for parcels that have a land use code of *agriculture*, and the data are available in XML. Using XQuery and master terms, a user can simply pose a single query:

```
//*[landuse = "agriculture"].
```

This query can be processed with query rewrites using lookup tables or ontology information to generate subqueries in native terms for each parcel data source [WZP02]. However, the composite result of this query is a heterogeneous "union" of the diverse, but conceptually similar, parcel elements.

In this paper, we focus on language capabilities and not implementation issues. However, one issue for querying Web data without specifying element types is finding the meaningful element boundary level to return that contains subelements stated in the query. Some additional stored knowledge or interaction with the user may be necessary. Another issue is where to store the result of a query so that it is not confused with the original source when further queries also do not use the document function.

We tested querying across element types in the Niagara XML query engine [NDM+00] and found

heterogeneous results may be returned (although not stored). This was not purposefully designed but occurs because of Niagara's "IN *" capability and its manner of storing elements from all crawled data in the same inverted lists. Niagara's heterogeneous result was an XML data stream, and, by default, Niagara returned an element level that was one level up from the element in the query.

5. Summary and Conclusions

This paper motivated the usefulness of querying across database types in traditional DBMSs and Web-based information systems. Heterogeneous results are particularly useful in gathering all information pertaining to a geographic region. Over the Web, heterogeneous results also occur when querying diverse, but conceptually similar, elements.

It was shown that an XML query language such as XQuery already has the expressive capabilities needed for heterogeneous querying. For example, a complete path does not need to be precisely specified in a query nor is the document function required. As a result, a query expression can range across unspecified and unlimited data types and data sources to retrieve information. Providing appropriate implementation would be an enhancement to existing traditional and Web DBMS query languages.

Acknowledgements

The author would like to thank Don Chamberlin for answering general inquiries regarding Quilt/XQuery and Mary Fernandez for XML-QL examples. This work was partially supported by the Digital Government Program of NSF, Grant No. 091489.

References

[ADL] Alexandria Digital Library Project. http://www.alexandria.ucsb.edu.

[BBO88] Breazu-Tannen, V.; Buneman, P.; and Ohori, A. 1988. "Can Object-Oriented Databases be Statically-Typed?" In Database Programming Languages, Second International Workshop, pp. 226-237.

[BDH+96] Buneman, P.; Davidson, S.; Hillebrand, G.; and Suciu, D. 1996. "A Query Language and Optimization Techniques for Unstructured Data." In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp.505-516.

[Bla95] Blakeley, Jose A. 1995. "OQL[C++]: Extending C++ with an Object Query Capability." In Modern Database Systems, The Object Model, Interoperability, and Beyond, Won Kim (Editor), Addison-Wesley, pp. 69-88.

[CB97] Cattell, R.G.G. and Barry, Douglas K. (Editors). 1997. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, San Francisco, California.

[CCF+01] Chamberlin, D.; Clark, J.; Florescu, D.; Robie, J.; Siméon, J.; and Stefanescu, M. (Editors). 2001. "XQuery 1.0: An XML Query Language." W3C Working Draft 07 June 2001. http://www.w3.org/TR/2001/WD-xquery-20010607.

[CD99] Clark, J. and DeRose, S. 1999. "XML Path Language(XPath) V. 1.0." http://www.w3.org/TR/xpath.html.

[CFM+01] Chamberlin, Don; Fankhauser, Peter; Marchiori, Massimo; and Robie, Jonathan. 2001. "XML Query Use Cases." W3C Working Draft 08 June 2001. http://www.w3c.org/TR/xmlquery-use-cases.

[CRF00] Chamberlin, Don; Robie, Jonathan; and Florescu, Daniela. 2000. "Quilt: An XML Query Language for Heterogeneous Data Sources." http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html.

[DFF+98] Deutsch, A.; Fernandez, M.; Florescu, D.; Levy, A.; and Suciu, D. 1998. "XML-QL: A Query Language for XML." http://www.w3.org/TR/NOTE-xml-ql/.

[FFL+95] Fischer, C.; Frew, J.; Larsgaard, M.; Smith, T.; and Zheng, Q. 1995. "Alexandria Digital Library: Rapid Prototype and Metadata Schema." In Digital Libraries: Research and Technology Advances, pp. 221-240, Springer-Verlag.

[FSW] Fernandez, M.; Siméon, J.; and Wadler, P. "XML Query Languages: Experiences and Exemplars." http://www-db.research.bell-labs.com/user/simeon/xquery.html.

[Goo98] Goodchild, Michael F. 1998. "Workshop on Research and Development Opportunities in Federal Information Services." White Paper, http://www.ncgia.ucsb.edu/~good/WhitePaper.html.

[KKS92] Kifer, M.; Kim, W.; and Sagiv, Y. 1992. "Querying Object-Oriented Databases." In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp. 393-402.

[NDM+00] Naughton, Jeffrey; DeWitt, David; Maier, David; and others. 2000. "The Niagara Internet Query System."http://www.cs.wisc.edu/niagara/Publications.html.

[RCF00] Robie, J.; Chamberlin, D.; and Florescu, D. 2000. "Quilt: an XML Query Language." http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html.

[WZP02] Wiegand, N.; Zhou, N.; and Patterson, D. 2002. "A Domain Space Concept for Semantic Integration in a Web Land Information System." Submitted.