The Extended XQL for Querying and Updating Large XML Databases

Raymond K. Wong
School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
wong@cse.unsw.edu.au

ABSTRACT

XQL has been argued as just a model for asking for specific sets of elements with very limited query capability. This paper proposes several extensions of XQL to address the issues. The extensions include full-text indexed search, path variables, joins, session-based navigations, and updates. Effort has been spent to preserve the conciseness of the language syntax. Its corresponding query processor with optimization mechanism has been prototyped and available online. Finally, implementation issues are discussed.

1. INTRODUCTION

XQL has been argued as just a model for asking for specific sets of elements with very limited query capability. This document describes the extensions of XQL for querying XML documents [3] stored in the SODA2 XML Data Server [9]. The extended XQL is based on the original XQL proposal [7] extended with regular expression matching, path variables, joins and update operators, as well as several other subtle additions.

Several query languages have been recently proposed to query XML documents. They are different from each other in terms of language constructs and expressive power. For instance, [2] provided detailed comparisons of five XML query languages including Lorel [6] , XML-QL, XML-GL, XSL, and XQL. A new query language proposal called XQuery [4] from W3C have drawn lots of attention from the community regarding the future directions of XML query languages. XQuery has a very rich set of language constructs including joins, FLWR expressions, transformations, etc. However, it is far less concise and compact than XQL. With respect to updating XML data, [8] presented an extension to XQuery to support updates. The update constructs for XQL described in this paper are more concise with same expressive power, and they have been available online since early last year (refer to the SODA2 demo link from Robin Cover's XML page at http://www.xml.org or [9, 10]). Finally, [5] contained extensive survey on most of the issues related to semistructured and web data [1], ranging from data models to query languages to database systems.

The organization of this paper is as follows. Section 2 describes the architecture of SODA2 so that readers can relate the extended XQL with the underlying database system. Section 3 gives a detailed description of those query extensions we implemented on XQL and Section 4 describes the new update constructs. In Section 5, some implementation issues of SODA2 to support efficient query processing are presented. Finally, Section 6 concludes the paper. The restaurant example shown in the Appendix is used throughout the paper (unless explicitly specified) to illustrate the query constructs. The example is very similar to the demo example available online.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2001 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

2. ARCHITECTURE

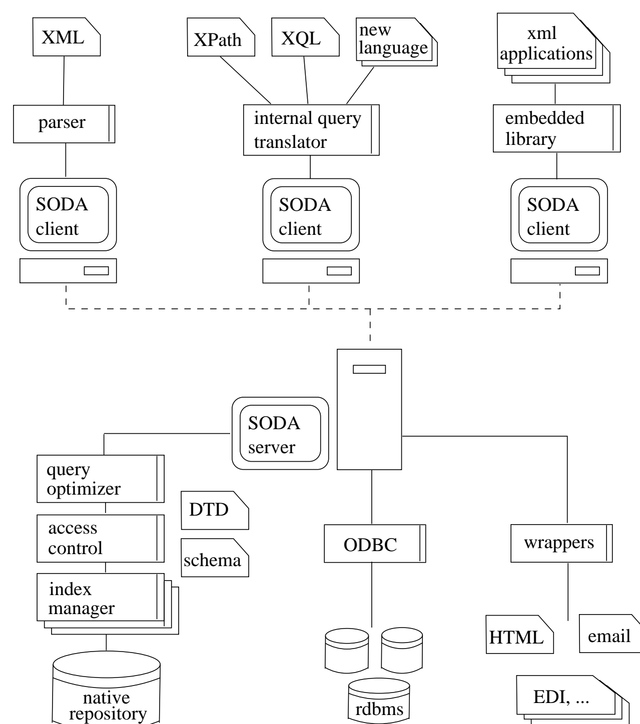

Figure 1: SODA2 Architecture

SODA2 (for Semistructured Object DAtabase, Version 2) is a client-server, semistructured database system which is tailor-made for managing XML information. Query processing and optimization are implemented in and executed by clients while the server is responsible for storing and retrieving XML objects; handling transactions, element locks, garbage collection, database backups and recovery. A lazy XML object conversion approach is used for versioning. Therefore, different clients can simultaneously work with different versions of DTD. The novel SODA2 architecture facilitates several crucial features which are seldom available in other database systems. The SODA2 query processor is mainly located at the client side. Each query processor contains an internal query translator that maps a query from one language into a SODA2 internal micro-query language. Therefore, SODA2 supports multiple query languages which include XPath expressions, XQL and limited XML-QL to date (SODA QL is supported for downward compatibility with SODA version 1). Web and e-commerce applications can be built by linking to the SODA2 client library. The library interface supports embedded query languages such as XPath, XQL, and XML-QL for rapid application development. XML parser or loader is itself a database client so that multiple loaders can be run simultaneously to load multiple documents while the database is being updated concurrently by multiple users at the same time. This feature is a must for large-scale enterprise applications instead of small corporate or personal applications. Advanced wrapper system plays an important role in SODA2, as it provides a bridge between SODA2 database and other different data types or different data sources, for instance, emails, HTML, SGML, RTF, EDI, and so on.

SODA2 server itself consists of a number of components. Each of these components is responsible for its own task and interacts with other components by means of a strictly defined interface. These components include a storage memory manager, an access control manager, a transaction manager, a page pool manager, an object access manager and an index manager. The modular design makes SODA2 possible to choose different implementations for each component and also to fine-tune SODA2 according to the efficiency of various database management algorithms and strategies for specific application requirements. SODA2 server can hook to other relational database systems such as Oracle or Sybase through ODBC interface. The underlying physical repository supports standard DOM, and SOM (SODA Object Model) which provides system-level interface to the SODA2 physical storage. Compression and low-level optimization are supported with meta information such as DTD and XSchema defined by the users or automatically learnt from the XML documents.

The storage model of the SODA2 XML Data Server mimics a well-formed XML document. There is a document root called `_system:root` for the entire repository such that every element in the repository is its predecessor. `_system:root` contains several children including `_system:data`, `_system:workspace`, `_system:default`, `_system:index`, `_system:access`, `_system:user` and `_system:schema`. They are for holding XML data, using as a temporary storage during query processing, holding system catalog and parameters for default settings, holding various kinds of indexes, holding access control and security information, holding individual user profiles and holding the corresponding schemas for some data, respectively.

3. THE EXTENDED XQL

3.1 Starting with basic XQL for SODA2

A 'context' is the set of nodes against which a query operates. This term is defined formally in the original XQL proposal. The extended XQL allows a query to select between using the current context as the input context and using the root context as the input context. The root context is a context containing only the root-most element of the XML repository, i.e., `_system:root`. Note that in XQL, the root context is defined as a context containing only the root-most element of the document. By default, a query uses the current context and assumes it is `_system:data`. A query prefixed with '/' uses the root context. A query may optionally explicitly state that it is using the current context by using the './' prefix. A query may use the '//' operator to indicate recursive descent. When this operator appears at the beginning of the query, the initial '/' causes the recursive decent to perform relative to the root of the document or repository. The prefix './/' allows a query to perform a recursive descent relative to the current context. Here are some examples:

- Find all Restaurant elements within the current context:
  `./Restaurant`
  Note that this is equivalent to:
  `Restaurant`
- Find the root element (`_system:data`) of the repository:
  `/_system:data`
- Find all name elements anywhere within the repository:
  `//name`
- Find all Restaurants where the value of the price attribute on the Drink is equal 2 dollars:
  `//Restaurant[/Drink/@price = 2]`
- However, since the entry point (the root element) for the example shown in the Appendix is Dining, we need to state the entry point explicitly if wildcard is not used.
  `Dining/Restaurant/(Name | @Name)`

3.2 Regular Expression

Regular expression is supported to match elements against the given pattern from the extended XQL query. An element pattern is a quoted string in the path of the query. Text index can be created in SODA2 to support fast regular expression string matching. The following examples illustrate its applications:

- Find all elements starting with a letter 'N' within a Restaurant element. Note that the Restaurant children of the current context are found, and then children beginning with 'N' are found relative to the context of the Restaurant elements:
  `Restaurant/"N.*"`

- Find all 4-letters' elements ending with a letter 'e', one or more levels deep in the Restaurant (arbitrary descendants):
  `Restaurant//"...e"`
  Note that this is different from the following query, which finds all 4-characters' elements ending with 'e' that are grandchildren of Restaurant elements:
  `Restaurant/*/"...e"`
- Find all elements of four characters, one or more levels deep in the current context.
  `.//"...."`

A power query can be produced by combining regular expression in the paths and regular expression embedded in the predicates. This can be achieved by using the Unix-like grep operator in the predicate expression. Similarly, SQL-like like operator is also supported.

- Find all restaurants or cafe located at NSW
  `Dining/(Restaurant | ''Cafe.*'')[.//''.*NSW.*'']`
- Find restaurant with Name containing the word "Bamboo"
  `//Restaurant[Name $grep$ ''Bamboo'']`
- Find an element with tagname being 4 characters starting with a letter 'N'
  `//''N...''`
- Find great entree from the restaurant with Name like `''%Bam%''`
  `//Restaurant[Name $like$ ''%Bam%'']/`
  `Entree[Rating = ''Great'']`

3.3 Cascading Function Invocations

All functions, methods, and paths in the extended XQL can be used more than once in the cascading manner. For instance, multiple `end()` can be used as an example below:

```
Restaurant!end()!end()!end()
```

A more meaningful example would be the following: find a restaurant that has "black bean soup" (in lower or upper case) in its menu. In this query, the current context for `Name` to match is determined by the `ancestor( )` function.

```
//*[* $ieq$ ''blACK bean soUP'']
      !ancestor(Restaurant)/Name
```

Another example is to count the number of text nodes: `//Name!textNode( )!count( )`

3.4 Session-based Context Navigation

Session-based context navigation is useful for interactive, ad hoc query environment such as the one in SODA2. There are two contexts defined per user session, namely root context and current context. The former corresponds to the context for those queries starting with //, and the latter corresponds to the one for those queries starting with ./. The system administrator of SODA2 can set the root context for a user so that the user can never access to any data above the nodes of the root context in the XML data hierarchy. On the other hand, users have authority to change their current context anytime for convenience. For instance, if one is going to ask ten queries interactively about the `Dining/Restaurant/Entree`, instead of repeating this prefix path ten times, she may change the current context temporarily to `Dining/Restaurant/Entree`. Therefore, she does not need to include the prefix in all ten queries. The idea is similar to the current working directory in operating systems. The following is the corresponding command to achieve this in SODA2 using extended XQL:

```
Dining/Restaurant/Entree!setcwd( )
```

Furthermore, the **..** is defined as a shorthand for ancestor(.) which can be used to navigate to the parent context. However, for access control reason, ancestor(.) will return empty when the current context is already the root context.

3.5 Querying Internal versus External Data from Multiple Documents

When documents are loaded to SODA2, information such as file name, owner, creation date would be added as system annotations for the document entry point (the document root node). This information is also maintained as metadata for fast access to a particular document. The corresponding XQL function (similar syntax to those in other languages such as XQuery) to specify the document name is

```
document(''restaurant.xml'')//Name
```

This query will find all Name elements from the document restaurant.xml stored in the database. Since document information is stored as annotation, regular expression matching does apply to this information. For instance, the following will find all Name elements from any XML documents stored in the database with names matching `''rest.*''`.

```
document(''rest.*\.xml'')//Name
```

SODA2 supports querying external data sources. The following query will find all Name elements from an external XML file called `cafe.xml`:

```
file(''cafe.xml'')//Name
```

The external file can be stored under a particular element node before queries are issued:

```
//Dining[0]!file(''cafe.xml'')
```

Similar idea applies to external XML pages from a remote URL site:

```
url(''http://www.unsw.edu/~soda/a.xml'')//Name
//Dining!url(''http://www.unsw.edu/~soda/a.xml'')
```

Instead of storing the whole external document to the database, another more interesting and useful case is to query the external document (whether is a file or a URL), obtain the query result and store the result under a particular element node of the database. The detail of the insert operator is described in the Update Section.

```
//Dining[0]!insert(  file(''cafe.xml'')//Name  )
```

3.6 Joins

The syntax for joins is as follows: path1!join(path2, exp1, exp2, op)

Suppose that context1 and context2 are the results after evaluating path1 and path2 respectively. Here is the join mechanism (a version without any optimization):

Algorithm:

```
for each nodeX in context1;
  result1 = match exp1 from nodeX;
  if result1 != NULL then
    for each nodeY in context2;
      result2 = match exp2 from nodeY;
      if result2 == result1 then
        construct a new element called join
          which contains two children: nodeX and nodeY;
        include the new element in the resultant context;
```

By default, if three arguments are given with op missing, equality operator is assumed. Furthermore, if only one argument, namely path2, is given, joining of all combinations between the nodes from the current contexts of path1 and path2 is resulted.

3.7 Path Variables

Path variables are useful when we need to keep the nodes of the context returned by a query. This can mimic the notion of subquery in query languages. The following query will store the query results to a variable t2:

```
//Restaurant[@ACN]!setvar(t2)
```

Further query results can be added to the variable until the variable is reset of deleted:

```
//Cafe!setvar(t2)
delvar(t2)
```

Content of the variables can be used as an ordinary XQL context, e.g.,

```
var(t2)/Name
```

4. UPDATES

The update statement in SQL plays a crucial role to make the manipulation and transactions of data stored in relational databases convenient and expressive. While the original XQL proposal did not include any update capabilities, the extended XQL supports a complete set of update constructs from create to copy and move. These constructs are implemented as functions in XQL and can be invoked as other standard XQL functions like ancestor() or count() using the bang notation '!'.

4.1 Constructors

New elements, attributes, or texts can be created interactively by the insert function. The function can be invoked as other XQL functions and it accepts one argument. The argument must be a plain path: for example, without filter or subqueries. The quoted string in the path will be treated as the value of a text or an attribute value. For example, the following will create an empty element Name under every Restaurant element:

```
*/Restaurant!insert(Name)
```

The following will create an Entree element under every Restaurant, then create a Name element under Entree, and finally create the text "Black bean soup" under Name:

```
*/Restaurant!insert(Entree/Name/"Black bean soup")
```

Finally the below will create an attribute Note with value "Sunday Only" for the second Entree of each Restaurant:

```
*/Restaurant/Entree[1]!insert(@Note/"Sunday Only")
```

Similarly, there are insertBefore(path) and insertAfter(path) to insert path as a sibling before and after the current reference node, respectively. For example, the following will insert an element Cafe which is as the same level of Restaurant, and it will be placed just before the 2nd Restaurant:

```
Restaurant[1]!insertBefore(Cafe)
```

4.2 Delete

Delete can be done by invoking the function delete() with no argument. It will delete all the nodes (and their predecessors) from the current context. For example, the following will delete all the Names (and their predecessors) from Restaurant elements. Note that the Restaurant elements will not be deleted in this case.

```
Restaurant/Name!delete()
```

Another example below will delete everything from the current context. If the root context is set to the global entry point to the whole database, it will delete everything from `_system:data`, i.e., every documents stored will be deleted. That is why only the administrator can set/assign the root context for a user (as described in the Session-Based Navigation Section above).

```
*!delete()
```

4.3 Copy

You can clone elements, attributes, or texts by using the copy function. It accepts one argument which is the source of the copying. It will copy all the nodes (and their predecessors) from the resultant context set from the evaluated argument, to every node of current context. For example, the following will copy the content of the first Entree of Restaurants in the whole repository to the second Entree of the Restaurants.

```
(Restaurant/Entree)[1]!copy(//(Restaurant/Entree)
          [0]/*)
```

Note that very node in the current context will get a copy, so the following example will make a copy of the content of the first Entree to every Entree including the first Entree itself. The argument path will be evaluated every time against every node in the current context. So there will be more content than you may expect in the subsequent nodes in the current context.

```
Restaurant/Entree!copy(//(Restaurant/Entree)[0]/*)
```

Similarly as Create, there are copyBefore and copyAfter besdies Copy, which will copy the source to be just before or after the reference node and as the same level.

4.4 Move

Move will move the resultant reference nodes from the evaluated argument path to under the nodes in the current context. Similar to Create and Copy, there are moveBefore and moveAfter. For example, the following will move the Ratings of Entrees to just before the Price elments of Entrees:

```
*/Restaurant/Entree/Price!moveAfter(//Restaurant/
            Entree/Rating)
```

Note that before the actual move operation being executed, validity check needs to be done to ensure that ancestor nodes are not being moved to become the predecessor nodes of the nodes in the current context. Otherwise, nodes in the current context would become dangling and no longer accessible from the root entry point(s).

4.5 Update

The value of an element or attribute can be updated by using the update function with the new value as an input argument. For example, the following will update Name and Price of the 2nd Entree of each Restaurant to "Onion soup" and "2.04" respectively.

```
*/Restaurant/Entree[1]/Name!update("Onion soup")
*/Restaurant/Entree[1]/Price!update("2.04")
```

Update can also be used to update the tagname of an element. For example, the following will rename all the Restaurant tagnames to Cafes:

```
*/Restaurant!update("Cafe")
```

4.6 Querying External, Non-XML Data

A module called LSX-T is responsible to map external, non-XML data sources to XML so that they can be queried by SODA2. LSX-T is a data extraction and transformation tool for converting any data stream into XML - for example, text-based bioinformatic data, sequences, e-mails, WML, HTML, EDI, RDBMS output, etc. Different from other related work, the motivation behind LSX-T is to be consistent with the XSLT technology as much as possible, so the applications/users will have consistent interface and access to both technologies. For instance, the LSX (for the reverse of XSL) syntax looks almost the same as XSL with a few extensions. Finally, a generic wrapper that can transform data from (any) one format to another is formed by integrating LSX-T with XSLT.

LSX is a language that is based on XSL. Therefore, a LSX file is also an XML file. The XSL employs XPath as a way of specifying part of an XML document for transforming. However, this does not apply to LSX processor, as the input text file is not in XML format. LSX utilises the pattern matching capability of regular expression to provide text matching for processing input text file. LSX supports the Perl5 regular expression. Conceptually, the input data stream is treated as a long string sequence. The input data and LSX file will then be processed sequentially from the beginning of the file. This is different from XSL where a matched template can be applied recursively. However, LSX template can be repeated as a loop in programming language by using a node called lsx:for-each (to be explained later). String can be extracted from the input and assigned to a variable. This allows it to be output into the resultant XML file. Furthermore, a sub-string in a variable can also be replaced or removed if it matches a given pattern. This allows value of a variable to be modified.

To illustrate the basic LSX idea, consider the following short example. Assume we would like to convert a table in a text file with two columns i.e.,. A and B as shown below to the XML file shown next.

```
Col_A,Col_B
a,b
1,2
c,d
3,4
```

Here is the XML file required:

```
<?xml version=''1.0'' ?>
<TableAB>
  <Row>
    <a>a</a>
    <b>b</b>
  </Row>
  <Row>
    <a>1</a>
    <b>2</b>
  </Row>
  <Row>
    <a>c</a>
    <b>d</b>
  </Row>
  <Row>
    <a>3</a>
    <b>4</b>
  </Row>
</TableAB>
```

The corresponding LSX to perform the above transformation is shown below. lsx:template is used to match the header (Col_A, Col_B). The output will have ¡TableAB¿ as a root element. After that lsx:for-each extracted the values of each row and assigned them to variables (A,B). Then these variables are output with appropriate tags (¡a¿, ¡b¿) via lsx:value-of.

```
<lsx:transform xmlns:lsx="http://www.cse.unsw.edu.au/lsx">
 <lsx:template text-match="Col_A,Col_B\s+">
 <TableAB>
 <lsx:for-each pattern-match="$A$,$B$\s+">
       <lsx:exp A="\w+" B="\w+"/>
       <Row>
       <a><lsx:value-of select="A"/></a>
          <b><lsx:value-of select="B"/></b>
        </Row>
      </lsx:for-each>
    </TableAB>
  </lsx:template>
</lsx:transform>
```

We have successful tested LSX-T on numerous non-XML data sources including a bioinformatic database as shown below.

```
xxx
yyy
zzz
xxx
xxx
yyy
zzz
yyy
zzz
```

5. SOME IMPLEMENTATION ISSUES

5.1 Data Storage

SODA2 extends the idea of DOM further. Information regarding access control, recent query results, schemas, indexes and internal system states are stored together with the XML documents under one XML tree with unified node structure. The whole system can be maintained by the appropriate queries, updates or system commands. Unified structure helps the storage manager for space allocation and defragmentation. Database snapshots and backup can be done easily or in remote without stopping the clients working with the database.

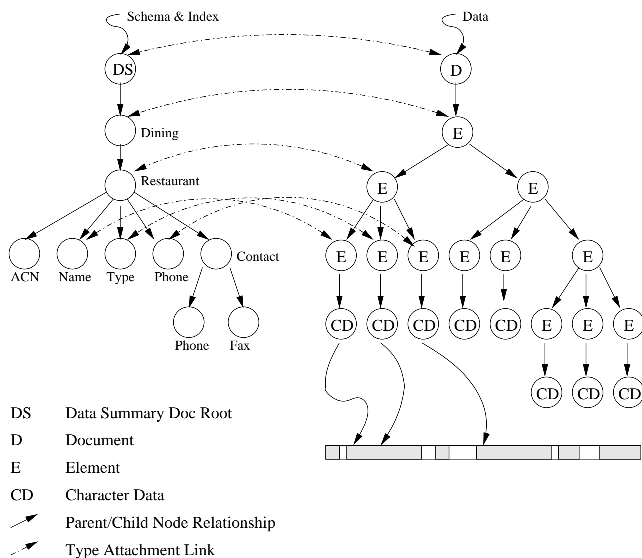

Figure 2: The Physical Storage Model

The XML document below will be stored in SODA2 as shown in Figure 2.

```
<?xml version='1.0'?>
<Dining>
<Restaurant ACN="1012385">
<Name> The Bamboo Restaurant </Name>
<Type>Vegetarian</Type>
<Phone> 92312210 </Phone>
</Restaurant>

<Restaurant ACN="1372358">
<Name> Chen's Seafood Restaurant </Name>
<Type> seafood </Type>
<Contact>
<Phone> 90123210 </Phone>
<Phone> 90123219 </Phone>
<Fax> 90128899 </Fax>
</Contact>
</Restaurant>
</Dining>
```

A Data Summary is built incrementally for each XML document. A Data Summary represents a logical view of a given XML document. It may contain the metadata of that document and some system parameters or statistics. The SODA2 XML parser will make use of any internal or external DTD to help generating the Data Summary, or the parser will generate the DTD itself if none is provided. Element names from XML documents are stored as Data Summary elements. Each XML element node (E) locates its name (i.e., element name) by following its link pointer to the associated Data Summary element node. Elements of the same name will have their link pointers pointing to the same Data Summary element node (that holds the actual name). Since XML documents tend to have element names of more than 6 characters and the same names appearing many times in a document, the above method shall give a good compression ratio in general. Attributes are stored within the Element node itself in the Binary Data field which differs to the Schema. Attribute name-value pairs are encoded using simple offset. Assuming the fact that typical attribute length is small and attributes are less frequently updated than CData or elements. Hence fewer objects and persistent object pointers are required and thus less page fault is resulted. A list of inverse pointers can also be optionally maintained from the element nodes in the Data Summary pointing to their corresponding XML data nodes. This will speed up many queries, especially complicated XQL with lots of regular expressions. It is because regular expression matching can be performed on the Data Summary such that elements with the same name will only be tested once.

5.2 Storage for Character Data

The Character Data for each document is stored in a special structure. This structure has two parts: index and data. The index is a linked list of pointers to the cdata strings in the data part, in the order of its occurrence at the XML document. CData nodes in the document tree keep track of its location at the document tree. Since cdata is not stored within the document tree, traversal of the document tree for operations which only involves cdata is avoided. This certainly improves the performance due to better locality of reference in pure text search.

5.3 Data Summary

Each XML Document Node has a node link to its corresponding Data Summary. Data Summary summarizes the structure, datatype and statistics of XML documents stored in SODA2. A Data Summary is being generated when an XML document is loading into the repository. It will be incrementally maintained and updated every time when its corresponding XML document is updated. In general, Data Summary is significantly smaller than its corresponding XML Document. Therefore, it is usually much faster to acquire the idea of the document from the Data Summary before searching the actual document tree for the query answer. For example, queries with path expressions that do not match any paths encoded in the Data Summary can simply be rejected. However, without Data Summary, this query costs the examination of every node at the Document tree in the worst case. Data Summary also store the occurrence of the descendant nodes, queries containing exists, sequence or position can be speeded up when no possible result can be determined from the Data Summary.

5.4 Query caching

Each query result is cached under the `_system:workspace`, with the statistics including the query processing time and number of requests received, etc. Query result node contains solely links to the satified

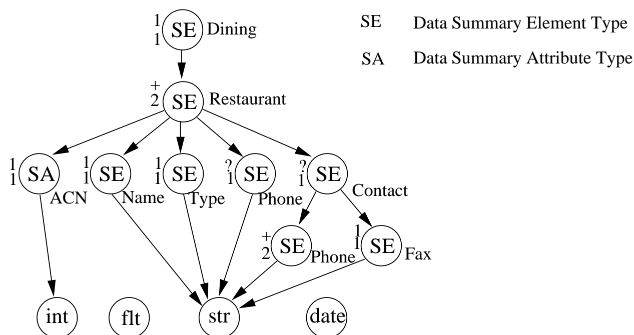

Figure 3: Data Summary Example

result nodes. A result node with all children satifies a query will only be pointed by a single cached query result node otherwise the result node will be expanded in that particular descendent level. For example a simple query that selects the whole XML document will generate only one result node or, otherwise, the whole XML document would be duplicated. This saves storage space and hence query processing time (and re-query time too), but it introduces many technical challenges concurrency control and transaction management. Garbage collector makes use of the query history statistics to determine when the data in the query cache should be freed.

5.5 Queries and Updates

A SODA2 query does retrieval without updating the data, thus it never changes the data involved in the result of another query. However, when an update statement changes the data involved in the result of the previous query, the cached query result would be marked as dirty and will be garbage collected. Alternatively, users may turn a parameter on such that the cached result would still be marked as dirty but will not be garbage collected. Document fragments linked from the cache will still be output but user will get informed. This feature is useful if an user want to know when a particular part of an XML document has been updated. When a node deleted within an XML document, it would be stored at a separate list in order to avoid dangling links at the cached query results. Deleted nodes and cached query results are periodically garbage collected. Frozen version of the query result is supported by materialized views by cloning the query result. Materialized views can be incrementally maintained or static.

6. CONCLUSIONS

The extended XQL query interface and related demos can be found at [9]. The queries and updates used by the online auction demo at the website are built entirely based on the extended XQL constructs described in this paper. We have proposed and implemented several extensions to XQL on a native XML database systems called SODA2. The following are the summarized features of the extensions:

- Still very compact to fit as part of a URL or as attribute value
- Almost the same syntax as XPath which is used as a pattern language in XSL and XPointer
- Updates (insert, delete, update, move, copy) are supported
- Regex supported in tags and texts
- Path variables
- Joins
- Session-based navigations
- Querying internal versus external data
- Richer set of aggregate operators
- Extensible method invocations and functions for XQL

7. REFERENCES

[1] S. Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer Verlag, 1997.

[2] A. Bonifati and S. Ceri, Comparative Analysis of Five XML Query Languages, *Sigmod Record*, March 2000.

[3] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. In *W3C Recommendation, World Wide Web Consortium*, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[4] D. Chamberlin, D. Florescu, J. Robie, J. Simon, and M. Stefanescu (Eds.). XQuery: A Query Language for XML, *W3C Working Draft*, 15 February 2001. http://www.w3.org/TR/2001/WD-xquery - 20010215/.

[5] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World- Wide Web: A survey. *SIGMOD Record*, 27(3):59–74 (1998).

[6] J. McHugh *et al.* Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[7] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In *The XSL Working Group, World Wide Web Consortium*, 1998; http://www.w3.org/TandS/QL/QL98/pp/xql.html

[8] I. Tatarinov, Z.G. Ives, A.Y. Halevy, and D.S. Weld. Updating XML. To appear in *ACM SIGMOD* 2001.

[9] The SODA Research Group. *SODA2: The Semistructured Object Database System, Version 2*, 2000. http://dba.cse.unsw.edu.au

[10] R. Wong. SODA2: An XML Database Management System. *Conference Proceedings of XML Asia*, September 2000. http://www.xmlarena.com/mainpage.asp ?webpageid=61

Appendix

```
<?xml version="1.0"?>
<Dining>
  <Restaurant ACN="1012385">
    <Name> The Bamboo Restaurant </Name>
    <Type>Vegetarian</Type>
    <Phone> 92312210 </Phone>
    <Entree>
      <Name>Black bean soup</Name>
      <Rating>Great</Rating>
      <Price>10</Price>
    </Entree>
    <Entree Note="Sunday Only">
      <Name>Onion soup</Name>
```

```
    <Price>2.04</Price>
    <Rating>Decent</Rating>
  </Entree>
  <Main_Course>
    <Name>Vegetables in oyster sauce w/ rice</Name>
    <Price> Nine bucks </Price>
    <Rating> no comment </Rating>
  </Main_Course>
  <Main_Course>
    <Name>Deep fried Tofu, served with sweet
                      peanut saurce and rice</Name>
    <Price> 9 </Price>
  </Main_Course>
</Restaurant>
<Restaurant ACN="1372358">
  <Name> Chen's Seafood Restaurant </Name>
  <Type> seafood </Type>
  <Phone> 90123210 </Phone>
  <Phone> 90123219 </Phone>
  <Fax> 90128899 </Fax>
  <Address>
    <Street>
      <Number> 118 </Number>
      <Road> City Road </Road>
    </Street>
    <City> Sydney </City>
    <State>NSW</State>
    <Country>Australia</Country>
    <Postcode>2000</Postcode>
  </Address>
  <Entree>
    <Name>Prawn Cutlet</Name>
    <Rating>Good</Rating>
    <Price>11.95</Price>
  </Entree>
  <Entree>
    <Name>Shrimp Salad</Name>
    <Price>$9</Price>
    <Rating>Great and cheap</Rating>
  </Entree>
  <Main_Course>
    <Name>Lobster in creamy sauce</Name>
    <Price> 28.00 </Price>
    <Rating>No comment</Rating>
    <Free> Coffee/Tea </Free>
  </Main_Course>
  <Main_Course>
    <Name>Seafood Laksa</Name>
    <Price> 17.00 </Price>
    <Rating>No comment</Rating>
    <Free> Fruit juice </Free>
  </Main_Course>
  <Drink Name="coffee">
    <Price>3.5</Price>
  </Drink>
  <Drink Name="tea">
    <Price>3.0</Price>
  </Drink>
  <Drink Name="hot chocolate">
    <Price>3.5</Price>
  </Drink>
  <Drink Name="beer">
    <Brand> Foster Light </Brand>
    <Brand>VB</Brand>
    <Brand>Light Ice</Brand>
    <Price>4.5</Price>
  </Drink>
  <Drink Name="orange juice">
    <Price size="Large">3.5</Price>
    <Price size="Small">2.8</Price>
  </Drink>
  <Drink Name="coke">
    <Price>2.0</Price>
  </Drink>
</Restaurant>
<Restaurant Name="Thai Palace">
  <Number_Phone>9191-1234</Number_Phone>
  <Number_Fax>9191-1225</Number_Fax>
  <Number_Home>9232-3883</Number_Home>
  <Type>Thai and Asian</Type>
  <Address>
    <Street> 90-92 Kings Rd </Street>
    <Suburb> Newton </Suburb>
    <City> Sydney </City>
    <State> New South Wales </State>
    <Postcode>2006</Postcode>
  </Address>
  <Entree>
    <Name>Thai salad</Name>
    <Opinion Rating="Great"/>
  </Entree>
  <Entree>
    <Name>Spicy wings</Name>
    <Price>7.85</Price>
    <Rating>Great</Rating>
    <Price>Seven bucks</Price>
  </Entree>
  <Main_course>
    <Name> Tandoori Chicken </Name>
    <Price> 15 </Price>
    <Side_dish> Mango Pickle </Side_dish>
  </Main_course>
</Restaurant>
<Cafeteria>
  <Name>The Restarea 19</Name>
  <Address> 78 Kings Road, Newton </Address>
  <Price_list>
    <coffee> 2.00 </coffee>
    <coffee name="latte"> 2.40 </coffee>
    <coffee name="mocha"> 2.50 </coffee>
    <drink name="soft drink"> 1.90 </drink>
    <pasta> 7.20 </pasta>
    <hotdog> 6.00 </hotdog>
    <sandwich name="club"> 10.50 </sandwich>
    <sandwich name="chicken"> 8.90 </sandwich>
    <salad name="caesar"> 5.60 </salad>
    <salad name="garden"> 5.60 </salad>
    <salad name="octopus"> 8.00 </salad>
  </Price_list>
</Cafeteria>
<Cafe>
  <Name>Java Paradise</Name>
  <Phone> 99112468 </Phone>
  <Price>
    <Coffee> 2.0 </Coffee>
    <Soft_drink> 1.5 </Soft_drink>
    <Hot_choc> 2.0 </Hot_choc>
    <Sandwich name="chicken"> 5.0 </Sandwich>
    <Sandwich name="ham"> 5.0 </Sandwich>
    <Sandwich name="tuna"> 6.0 </Sandwich>
  </Price>
</Cafe>
<Restaurant>
  <Name>Nice Noodles</Name>
  <Type> Chinese noodles </Type>
  <Suburb> Epping </Suburb>
  <Rating> Very good </Rating>
  <Price>
    <noodles price="7"> BBQ chicken </noodles>
    <noodles price="7.5"> Chicken Chow Mein </noodles>
    <noodles price="7.5"> BBQ Pork </noodles>
    <noodles price="9"> Roast Duck </noodles>
    <rice price="6"> BBQ chicken </rice>
    <rice price="8.5"> Roast Duck </rice>
  </Price>
  <Drink price="2">
```

```
      <name> Coke </name>
      <name> Diet coke </name>
      <name> Sprite </name>
      <name> Lemonade </name>
    </Drink>
  </Restaurant>
</Dining>
```