DSQL - an SQL for structured documents

Extended Abstract

Arijit Sengupta[1] and Mehmet Dalkilic[2]

[1] Department of A&IS, Kelley School of Business
Indiana University, Bloomington, IN 47405, USA
asengupt@indiana.edu
http://www.kelley.iu.edu/asengupt/
[2] School of Informatics
Indiana University, Bloomington, IN 47405, USA
dalkilic@indiana.edu

Abstract. SQL has been the result of years of query language research, and has many desirable properties. We introduce DSQL - a language that is based on a theoretical foundation of a declarative language (document calculus) and an equivalent procedural language (document algebra). The outcome of this design is a language that looks and feels like SQL, yet is powerful enough to perform a vast range of queries on structured documents (currently focused on XML). The design of the language is independent of document formats, so the capability of the language will not change if the underlying markup language were changed. In spite of its familiarity and simplicity, we show that this language has many desirable properties, and is a good candidate for a viable query language for XML. This paper presents a canonical form of DSQL, showing only the properties of the language that affect the complexity of the language. Remarkably, SQL = core DSQL for flat input and outputs.

1 Introduction

We propose an extension of SQL called the Document SQL (DSQL) as a query language for XML. Interestingly, DSQL maintains its "roots" by being able to perform its role as SQL on flat structures that represent tables. DSQL possesses a number of important elements. DSQL, like SQL, gives data independence, freeing users from having to write procedural components. DSQL, like SQL, is simple. And since SQL is the *de facto* standard, learning DSQL requires substantially less effort than learning an entirely new language. DSQL has the potential to be optimized–a benefit of SQL that has made it so commercially popular. Lastly, DSQL is based on formal design principles which means its semantics is well-understood.

There is a precedence for using SQL, *e.g.*, [1], demonstrating various levels of success in their respective fields by relying on the familiarity of SQL. One of the most well-known examples is OQL (Object Query Language) [2], a query language for Object-Oriented database systems that is extension of SQL capable of handling the nuances of the Object-Oriented database model.

There are a number of converging design objectives for an XML query language in this paper. We briefly describe three here. First is a language that is based on a well-known, well-accepted formal semantics. Second is a language that does not rely on the underlying data structure, thus exhibiting data independence. Third, is the applicability of well-known query language design principles like low complexity, closure, and non-Turing completeness.

2 DSQL Specification

W3C (http://www.w3.org) is currently involved in an effort towards the query language XQuery which has been presented through a series of related W3C recommendations. XML has been widely recognized as the next generation of markup language for the Internet. There is necessarily a need for querying XML. We now describe the language characteristics of DSQL. DSQL is different from SQL primarily in DSQL's ability to search along paths and restructure data. Most omitted features are exactly the same as standard SQL. The basic syntax and an example of a DSQL query are shown below.

```
SELECT output_structure
FROM   input_specification
WHERE  conditions
grouping_specs
ordering_specs
```

```
SELECT result<B.title>
FROM   bibdb..book B
WHERE  B.title = 'Extending SQL'
```

As in SQL, only the `SELECT` and the `FROM` clauses are required. The other clauses are optional. Also, multiple `SELECT` queries can be combined using the standard set operations (Union, Intersect, Minus). The `SELECT` clause allows the creation of structures with arbitrary levels of nesting. The `FROM` clause utilizes XPath to retrieve trees, subtrees, *etc.*. The `WHERE` conditions in DSQL are similar to those in SQL. The main difference in semantics is due to paths, *i.e.*, all expressions in DSQL are path expressions, so operators are often set operators.

3 Comparative Examples

The language proposed in this paper is capable of performing most of the queries in the W3C XQuery use case that are independent of XML-specific issues such as namespaces. We show three pairwise equivalent queries to compare the two languages, XQuery (left) and DSQL (right).

Query 1. List books published by Addison-Wesley after 1991, including their year and title.

```
<bib>
{ FOR $b IN
 document("http://www.bn.com")/bib/book
 WHERE $b/publisher = "Addison-Wesley"
   AND $b/@year > 1991
 RETURN
  <book year={$b/@year}>{$b/title}</book>
}</bib>
```

```
SELECT  bib<B.title>
FROM    BN.bib.book  B
WHERE   B.publisher =
          "Addision-Wesley"
  AND   B.attval(year) > 1991
```

Query 2 . Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

```
<results>
{ FOR $b IN
  document("http://www.bn.com")/bib/book,
   $t IN $b/title,                          SELECT results<B.title,
   $a IN $b/author                                 B.author>
  RETURN                                    FROM   BN.bib.book
  <result> { $t } { $a } </result>
} </results>
```

Query 3. For each book in the bibliography, list the title and authors, grouped inside a "result" element.

```
<results>
{ FOR $b IN
  document("http://www.bn.com")/bib/book
 RETURN                                     SELECT   result<B.title,
   <result>                                              B.author>
     { $b/title }                           FROM     BN.bib.book
     { FOR $a IN $b/author                  GROUP BY B.title
       RETURN $a }
   </result>
} </results>
```

4 Important Observations and Contributions

DSQL is derived from a calculus (DC) and has an equivalent algebra (DA) that are analogous to the Relational calculus and algebra for SQL. Among the more important properties of DC and DA are: (i) the calculus and algebra are semantically equivalent; (ii) they are safe and (iii) they are in PTIME.

The primary contribution of this work is based on our results of semantic and syntactic equivalence of DSQL with SQL. This implies any SQL query would run unchanged on the structured equivalent of the tabular data, and thus systems capable of processing SQL can seamlessly integrate DSQL (and hence structured querying) into the existing infrastructure. Although many languages have been proposed for XML, no language to our knowledge has this property. A summary of all the results from the language are presented below.

1. **DSQL properties** Because of the equivalence with the calculus and algebra, DSQL is implicitly safe, closed, and in PTIME.
2. **DSQL Macros** DSQL can be augmented with macros that allow structuring operations that do not change any complexity, such as decision and link traversal. IF-ELSE queries can be implemented using `union`, and finite link traversal can be performed using `exists`.
3. **No ordering ambiguity** The sequential nature of documents is built into DSQL. A single document is always sequential, but a document is fragmented using a path expression creates unordered elements (since the result of a path expression is a set).

4. **Set-Atom ambiguity** All non-atomic expressions (queries and path expressions) in DSQL are set expressions, so there is no ambiguity in comparison of different expressions.
5. **Similarity with Relational domain** Because of the analogous nature of DSQL with SQL, most of the language processing methods work unchanged. For example, all the optimization equalities hold in DSQL, which indicates that the same optimization principles will be applicable in processing DSQL queries.
6. **DSQL=SQL for flat structures** For flat structures as input, and intended flat structures as output, DSQL queries are exactly the same as the corresponding SQL queries.
7. **Merging relational and document databases** DSQL can easily support queries which include local documents, remote documents via a URL, and relational tables all in the same `FROM` clause. This is an easy yet elegant technique for incorporating distributed heterogeneous databases in this query language.
8. **Implementation architecture** We built a prototype implementation of DSQL in DocBase[3] that also demonstrates the applicability of the concepts described here.

5 Future Research Issues

A number of issues need to be handled in order to apply DSQL in XML. Specific XML features (such as namespaces, attributes, etc.) are not included in DSQL because of its markup-independent design. XML-specific features can easily be added on to the language in order to satisfy all the W3C XML Query requirements. DSQL does not support recursion, and we have no intention of supporting direct unbounded recursion in the language. Similar to SQL, DSQL can be embedded in a host language or a 4GL-type language for such language features. We are also in the process of performing an empirical study comparing DSQL with XQuery based on a human-factors analysis involving users. See [4] for a full version of this paper.

References

1. Mendelzon, A., Mihaila, G., Milo, T.: Querying the world wide web. International Journal of Digital Libraries **1** (1997) 68–88
2. Cluet, S.: Designing OQL: Allowing objects to be queried. Information Systems **23** (1998) 279–305
3. Sengupta, A.: DocBase - A Database Environment for Structured Documents. PhD thesis, Indiana University (1997)
4. Sengupta, A., Dalkilic, M.: DSQL - an SQL for structured documents (http://www.kelley.iu.edu/ardennis/wp/tr108-1.pdf). Technical Report TR108-1, Indiana University (2001)