3-2009

Document SQL (DSQL) - An Accessible Yet Comprehensive Ad-hoc Querying Frontend for XQuery

Arijit Sengupta
Wright State University - Main Campus, arijit.sengupta@wright.edu

Document SQL (DSQL) - an Accessible Yet Comprehensive Ad-hoc Querying Frontend for XQuery

By: Arijit Sengupta, Ph.D., Assistant Professor MS, Raj Soin College of Business

This article presents a declarative query language, Document SQL (DSQL) that has the same look and feel as SQL and was designed by updating the semantics of SQL operations in the structured document domain. At the same time, DSQL was designed such that all queries written in it have equivalent counterparts in XQuery. Thus, by using such a language, users can take advantage of their existing SQL knowledge when writing ad-hoc queries without losing the expressive power of XQuery. In addition to describing the syntax and semantics of the language we also present the results of an experiment that investigates whether a language like DSQL make it possible for users to write more accurate and efficient queries than XQuery. This short article explains the basic constructs of DSQL, and explains the capability of the language via examples. More details on the language and its properties can be obtained from [Sengupta and Ramesh, 2009].

DSQL is structured after SQL. Although its operations are based on DA, the syntax of DSQL is essentially the same as the syntax of SQL, with a few small differences to accommodate path expressions and dynamic structure creation, both of which are essential in a query language for XML databases. In this section, we will describe the DSQL language, and demonstrate the properties of this language. DSQL has the same basic structure as SQL, with recognizable SELECT, FROM and WHERE clauses for retrieval, and GROUP BY, HAVING and ORDER BY clauses for grouping and ordering of the results. The GROUP BY clause in DSQL, unlike that of SQL is actually an integral part of the querying method, since GROUP BY turns out to be an elegant way of restructuring the results. Here we briefly mention each of the query clauses of the language.

Every DSQL query has the following basic syntax:

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

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 following sections describe in detail the above constructs of DSQL.

The Select Clause

The SELECT clause in DSQL has the same major structure of SQL, with the main difference that the SELECT clause can create complex structures, and can traverse paths in the items to retrieve. To keep the language simple and close to SQL, generation of attributes has not been included in the base language. In fact, the formal specification of the language uses a form of XML known as ENF (Element Normal Form), which ensures that any XML document with attributes can be re-written without the use of attributes and vice versa. Given these observations, we have the following output structure of the SELECT clause in Backus-Naur Form (BNF).

```
output_structure ::= ['ALL' | 'DISTINCT'] output
output ::= scalar_exp_list | '*'
scalar_exp_list ::= scalar_exp [',' scalar_exp]*
scalar_exp ::= name<scalar_exp_list> | atom | col
col ::= path_exp
```

The above BNF allows select clauses such as:

```
output_structure ::= ['ALL' | 'DISTINCT'] output
output ::= scalar_exp_list | '*'
scalar_exp_list ::= scalar_exp [',' scalar_exp]*
scalar_exp ::= name<scalar_exp_list> | atom | col
col ::= path_exp
```

The only omitted part of the actual BNF for the SELECT clause involves the use of aggregate functions. As is evident from the BNF and the examples above, this extension of the SELECT clause allows the creation of structures with arbitrary levels of nesting. Notice that any grouping is not inherent in this specification and is, instead, the task of the grouping_specs.

The From Clause

The BNF for the FROM clause is as follows:

```
input_specification ::= db_list
db_list ::= db [,db]*
db ::= path_exp [alias]
path_exp ::= SPE
```

The above specification of FROM clause allows the following types of expressions in the FROM clause:

```
FROM books.xml B
FROM books.xml B, authors.xml A
FROM http://www.mycompany.com/docs/invoices.xml V, V..items I
```

The Where Clause

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. For example, consider the following query:

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

(continued on page 15)

(continued from page 14)

The WHERE expression evaluates to true if the path expression yields a singleton set containing an atom identical to `Extending SQL'. Set membership operations such as in and contains are also available in DSQL.

Grouping and Ordering clauses

SQL has several ways of specifying post query formatting and layout generation. The following are the grouping and ordering specifications in DSQL:

- ORDER BY (sorting): DSQL has the same semantics for ORDER BY as SQL. The expressions in the SELECT clause can be ordered by expressions in the ORDER BY clause, regardless of whether or not the ordering expressions appear in the SELECT clause, as long as the expressions are logically related, i.e., a possible ordering is possible using the ordering expression.
- Aggregate functions: DSQL supports the same five basic aggregate functions as SQL (sum, count, min, max, avg).
- GROUP BY: In DSQL, GROUP BY is a restructuring operation but unlike SQL, the aggregate function is optional. The semantics of the restructuring operation is explained using the production that define the created structure, as shown in Table 1.

Table 1. Semantics of GROUP BY

	Before grouping	After grouping by A
Query	SELECT result<A,B,C>	SELECT result<A,B,C> ... GROUP BY A
Structure	+result ::= A, B, C+	+result ::= A, (B,C)*+
Data	<result> <A>a1</A><B>b1</B><C>c1</C> </result> <result> <A>a1</A><B>b2</B><C>c2</C> </result> <result> <A>a1</A><B>b3</B><C>c3</C> </result> <result> <A>a2</A><B>b4</B><C>c4</C> </result>	<result> <A>a1</A> <B>b1</B><C>c1</C> </result><B>b2</B><C>c2</C> <B>b3</B><C>c3</C> </result> <result> <A>a2</A> <B>b4</B><C>c4</C> </result>

There are a few consequences of this type of grouping:

- *Grouping without aggregate functions:* Grouping tasks can be performed with or without aggregate functions. In the case no aggregate functions are specified, grouping is essentially a restructuring operation. In the case aggregate functions are specified, the aggregate functions are performed for each set in the structures produced by the grouping.
- *Group by null:* An interesting effect of the grouping semantics is the possibility of grouping without any grouping clause which would cause the following restructuring operation: result ::= A, B, C transformed to result ::= (A, B, C)* The above is essentially a standard set formation operation caused by nesting on the full relation.
- *Multiple group-by clauses:* A complex structure could be grouped many times. In such cases, the structure to be grouped needs to be specified. For example:

```
SELECT Books<B.Author, Years<B.year, B.title>>
FROM Bib B
GROUP Books by B.Author
Group Years by B.Year
```

The primary contribution of DSQL is the theoretical basis, and the experience of building a language capable of expressing queries on XML data using well-known and well-understood techniques. For flat-to-flat transformations, DSQL has the same semantic and syntactic structure as SQL, enabling DSQL to be easily incorporable into current database systems. The safety and complexity properties of DSQL that we have proved ensure that any query written in DSQL can be guaranteed to be processed using low system resources. Such properties make DSQL highly suitable as a front-end to XQuery for user-centered ad-hoc querying. Through a controlled laboratory experiment, we demonstrated that DSQL clearly surpasses XQuery in terms of user cognition leading to better user efficiency and accuracy, both for flat and structured content. Although the current industry push makes it almost certain that XQuery will become the "standard" XML query language, our paper clearly demonstrates that DSQL is a viable and necessary front-end to systems implementing XQuery, especially for ad-hoc querying, and should prove to be a valuable tool for XML query processing.