Demand more from your SGML database!

Bringing SQL under the SGML limelight

Arijit Sengupta
Computer Science, Indiana University
asengupt@indiana.edu

Abstract

Have you ever been frustrated by how inadequate SGML databases are in terms of searching or querying your documents? With the current state of the art, you will easily be able to search for a word, phrase, or keywords in the whole document. Some systems let you perform approximate searches or regular expression searches. Even fewer systems let you search for keywords or phrases in certain SGML regions. However, there is much more information already in SGML documents that one can utilize cleverly to design a proper SGML database system. The current trend of modeling SGML documents with object-oriented and object-relational databases has certainly brought SGML closer to a complex object database model, but much research and development remains to be done in this area. This article introduces the popular relational database query language SQL (Structured Query Language) and its applicability in the SGML domain.The capability of this query language to express complex queries with a not-so-complex syntax gives relational databases that support SQL an advantage over other similar systems. The ability to use SQL or an SQL-like query language with SGML has the potential of giving much more power to SGML repositories. This article shows how we can pose complex document-related questions easily with SQL. SQL-capable systems will let you solve problems that would otherwise seem impossible, or at least, tedious.

1 Introduction

SQL (Structured Query Language) [SQL86, SQL89, SQL92] is the most widely accepted language for relational databases. It is also gaining popularity as a query language for Object-Oriented (OO) and Object-Relational (OR) databases, and currently most database systems implement some variant of SQL to formulate queries. However, SQL's capability is not limited only to queries. In addition to its use as a data query language (DQL), it can be used as a data definition language (DDL), a data manipulation language (DML) and a data query language (DQL). In this paper, we give a short introduction to SQL in its various flavors and show how we can use the power of SQL to perform complex searches on SGML documents.

1.1 SQL - the standard

Although the original SQL standard dates back to 1986 [SQL86], the language is still evolving with the advances in database theory and practice. The original standard was superseded by an improved version [SQL89]. Because of some programming 1limitations to this standard, a later standard [SQL92, MS92] also known as SQL2 was published. With the advent of the object-oriented and object-relational technologies, SQL users are about to see another major transformation in the form of SQL3 [SQL96], which is still in the works. The main focus of this paper will be based on the most recent SQL standard (SQL2) and some extensions to the standard that we introduce to "make it fit" in the SGML context.

1.2 Relational databases

SQL was designed to work with relational databases. In relational databases, data is stored in the form of flat tables, in which the rows or tuples represent one record of the data, and columns represent a field or property of the data, also called "meta-data" in technical terms. In simple terms, meta-data is a description of the data. For example, if in a table, a field "bookname" has data "The SGML Handbook", then "The SGML Handbook" is the data, and "bookname" is the meta-data". This principle has almost a direct correspondence to SGML, where the generic identifiers (GIs) are meta-data, and the character content in the GIs is the data. Of course, in relational databases, meta-data has more associated information such as data type, data size, index types, etc.

One problem with a relational database is its flat structure. To represent a complex hierarchical structure in a relational database, it needs to be mapped to an equivalent flat structure. The process of doing this involves a data modeling approach, a mapping from the model into tables, and further normalization based on other criteria like functional dependencies. The most commonly used data model to conceptually represent relational databases is the Entity-Relationship (ER) model. In this model, conceptual objects in the data (such as books, authors, publishers, etc.) are treated as "entities" and the relations between the entities (such as "written by", "published by") are treated as "relationships". Relationships involve at least two entities (e.g., book is written by author), that could be the same (e.g., book is cited by book). These entities and relationships are mapped into tables. In most cases, entities become tables, and many-to-many relationships become tables. One-to-one and one-to-many relationships can be mapped into their participating entities. Entities have attributes or properties, that get mapped to corresponding fields in the tables. Multivalued attributes (e.g., a book can have multiple volumes) are separately mapped into tables. (See [EN89] for more information on this modeling approach.)

Because of the way a conceptual database scenario is fragmented into multiple tables, these subdivided tables need to be rejoined as necessary during query processing. In fact, this "join" operation is one of the most important operations in relational databases, although it is not so important for SGML databases, since documents do not need to be broken into flat structures. However, the join operation still gives a great power to SGML database queries, as we will see in section 2.3.2.

2 SQL - a brief introduction

SQL was originally designed to specify queries in a language fairly close to the common English language. In fact the original version of this language, which was developed in IBM's San Jose Research Laboratory was called SEQUEL (Structured English Query Language)[AC75]. The SQL language derives most of its syntax and semantics from this language, and is also sometimes (erroneously) pronounced as "sequel" [MS92]. The original SQL consisted of a core set of operations which could all be performed in polynomial time. This, although seemingly restrictive, was a great advantage for the language because of its tractable nature. Complex queries that go beyond the polynomial complexity could be performed by embedding SQL queries in a common programming language. The proposed SQL3 standard [SQL96], has essentially turned SQL into a complete object-oriented programming language by.

Although SQL stands for "Structured **Query** Language", its use is not restricted to only queries. In fact, as stated earlier, SQL can be used (and is used in most SQL implementations) as a Data Definition Language (DDL) to define the structure of the data in the database, as a Data Manipulation Language (DML) to insert, update and change data in the database, as well as a Data Query Language (DQL) to search for data in the database based on various conditions. The following sections describe briefly how this language is used in these three forms in relational databases.

2.1 DDL properties

SQL can be used to define the data structure information (meta-data information) in relational databases. The top-level structure in SQL is a database, which consists of tables, views, and indices. In addition, many

SQL databases may contain stored procedures, triggers, user-defined data type declarations, temporary tables, etc. In this paper, we only consider the use of tables, views, and indices. Tables are the primary building blocks of relational databases. Views are like virtual tables which normally hold the result of a query that can be used in a different query. Indices are special data structures that are used to speed up the execution of queries. All of these structures can be created or deleted using the DDL statements CREATE and DROP. Some examples DDL statements follow:

```
CREATE TABLE books (ISBN character(20) NOT NULL, name character(30) NOT NULL,
                    pages integer NOT NULL, year integer NOT NULL,
                    price decimal NULL)
CREATE INDEX books.name
CREATE UNIQUE INDEX books.ISBN
CREATE VIEW cheapbooks as (SELECT * FROM books WHERE price < 20.0)
```

The first of the above statements create a table called "books" containing the ISBN, name, number of pages, year of publication, and price. The second one creates a normal index on the book name, and the third one creates an index with unique ISBN values, thus prohibiting multiple books with the same ISBN. The last statement creates a virtual table called "cheapbooks" with only the books whose price is less than \$20.00. The resulting set of books is built using the select statement (see Section 2.3.1). The data definition also includes deleting databases, tables, views and indices using the DROP DATABASE, DROP TABLE, DROP VIEW and DROP INDEX statements respectively, and lets the user make minor modifications to the table structure using the ALTER TABLE statement.

2.2 DML properties

Basic data manipulation includes inserting data in tables, deleting data from the tables, and making changes to the existing data in tables. These operations are performed in SQL using the DML statements INSERT, DELETE and UPDATE, respectively. Some examples using the same schema as above could be:

```
INSERT INTO books values (''1-55860-245-3'', ''Understanding the new SQL'',
                   536, 1993, NULL)
DELETE FROM books where price > 200.0
UPDATE books SET price = 75.0 where ISBN=''1-55860-245-3''
```

The above block of code inserts a row corresponding to a book into the table, deletes all the books priced higher than \$200.00, and updates the price of the book with the given ISBN to \$75.00.

2.3 DQL properties

Although the DDL and DML components of SQL are useful, its primary focus is on querying existing data. This section describes the basic query formulating statements using SQL.

2.3.1 Simple queries

All SQL queries are based on the SELECT-FROM-WHERE structure. Queries are expressed as "SELECT <fields> FROM <tables> WHERE <conditions>". The SELECT clause describes what you want from the query, the FROM clause tells you where you want the data from, and the WHERE clause contains a list of conditions that need to be satisfied. If the FROM clause contains more than one table, the WHERE clause should give a way to link the tables together. Let us suppose we have three tables, books (as above), authors (containing author-id, name, and affiliation) and the joining relation book-auth (containing book-ISBN and author-id). Some simple queries are shown below:

1. Find the name and price of the books that have more than 500 pages.

```
SELECT name, price
FROM books
WHERE pages > 500
```

The above query is quite simple to understand - it simply returns the list of name-price pairs of the books that have more than 500 pages.

2. Find all the books written by Charles Goldfarb that cost less than $50.

```
SELECT books.*
FROM books, authors, book-auth
WHERE books.ISBN = book-auth.book-ISBN
      and authors.id = book-auth.author-id
      and books.price < 50.0
      and authors.name = ``Charles Goldfarb''
```

For the above query, the conditions we need are not contained in one single table, so we need to join the three tables together. Note that in order to join, we need to equate the ISBN fields in the books and book-auth tables, and also the author-id fields in the book-auth and author tables. The two remaining conditions come from the query: *cost less than $50* and *written by Charles Goldfarb.*

2.3.2 Complex queries

The power of SQL becomes more apparent when we include advanced operations like quantification (EXISTS), subqueries (embedding one query inside another), grouping and ordering capability (with GROUP BY - HAVING and ORDER BY), and aggregate functions (such as SUM, COUNT, MIN, MAX, etc.). One moderately complex query follows:

1. Find the name and affiliation of authors all of whose books are more expensive than "The SGML Handbook".

```
SELECT A.name, A.affiliation
FROM authors A, books B, book-auth U
WHERE A.author-id = U.author-id
  AND B.ISBN = U.book-ISBN
  AND NOT EXISTS (
    SELECT * FROM books B1, book-auth U1
    WHERE B1.ISBN = U1.book-ISBN
    AND U1.author-id = U.author-id
    AND B1.price <= (
      SELECT price FROM books B2
      WHERE B2.name = ``The SGML Handbook''))
```

In the above query, the main thing to notice is the slight modification of the English query specification. The above query is the same as saying "find the authors such that there does not exist any book written by them that is less expensive than 'The SGML Handbook' ". This is necessary, since SQL has only one quantifier - the existential quantifier. Once this translation is done, the above query is easy to understand; most of the conditions above are mainly to implement the "joins".

3 Querying in the SGML context

Now that we have seen how we use SQL with relational databases, we can try to apply it in SGML. The most interesting feature of SGML that distinguishes it from relational databases is the schema representation of SGML documents. In SGML, complex structure can be represented quite easily, since there is no need to break documents into pieces in order to represent them. However, basic SQL does not handle complex structures easily. In spite of that, with the core SQL and with a few extensions to handle the hierarchy of the associated data and the complex input/output formulation, we can solve a wide range of queries. If we consider a complete object-oriented language based on SQL, it can express any feasible query. However, it is a good idea to stay within the bounds of a simple and restricted language which is complete with respect to the types of queries that it can handle efficiently. If necessary, we can embed it in another programming language for harder tasks. The core SQL provides this for relational databases.

3.1 Suggested extensions to SQL for use with SGML

The main extensions needed to use SQL with SGML documents involve the navigation of the tree structure, and that of building complex objects from other objects. The three primary extensions that we proposed in an earlier work [SD96] are cascading of the dot ("." or membership) operator, use of the double-dot (".." or descendant) operator, and the ability to specify a form of a DTD in the SELECT clause to build or break complex types. These extensions are not theoretically complete, but still they show how the core SQL with a few simple extensions can give great power to the query language. The main ideas behind these extensions are quite similar to the path expressions proposed by Christophides et. al. [CACS94]. Note that SQL has been adapted and used for object-oriented languages like the Reloop language for the object-oriented database O_2 [BDK92] and the new proposed SQL3 standard [SQL96]

One problem with SQL used in the relational domain is that it is not a good "fulltext" query language. This means it can not solve queries involving just strings - possibly encoded in a regular expression (e.g, "what is known about John in the books database?"). In fact, SQL lacks a way to perform schema-independent queries. The proposed extension SQL remove this drawback to a great extent. However, this SQL still needs meta-data along with data in order to be most efficient.

3.2 General properties of this extended SQL

In standard SQL, it is not necessary to navigate a hierarchy because of the flat structure of relational databases. However, in SGML, navigation is important. It is possible that the same generic identifier could be reached from the same node by two different paths. Using the cascaded "." operator or the ".." operator, one can specify the exact or approximate path that needs to be taken. For example, in the DTD given in Section 5, the expression *book.body.chapter.chtitle* and *book..chtitle* represent the same path. However, there are two paths from *book* to *name*: *book.header.bibl.author.name* denotes the name of the author while *book.bibl.name* denotes the name of the book itself. So *book..name* in this case is ambiguous, and will represent both the paths. In case the source node is omitted, the nodes given in the FROM clause are used as source nodes. So, a query like the following is valid:

```
SELECT * FROM books WHERE chtitle=''Introduction''
```

The above query finds all the books in the database which have a chapter titled "Introduction". In the case where a target node with source node omitted can be reached from multiple nodes in the FROM clause, the query language parser should generate an error, similar to the errors generated by an SQL parser if a field common to more than one table in the FROM clause is not properly referenced.

One important property of SQL is closure. This means that the language is "closed" in its domain. The input to the query is a set of tables, and the output is also a table. SQL adapted for SGML should also have the same property. In other words, every query should generate an SGML document. If discrete fields are specified in the SELECT clause, like Query 1 in Section 2.3.1, they are put under one content model with sequence separators. The closure property is necessary for various reasons, the primary reason being the possibility of cascading SQL statements by using the result of one query as the input to another. Specifying a DTD in the SELECT clause can give the user more control over this process.

It has been mentioned earlier (Section 3.1) that SQL has limited support for fulltext or regular expression queries. The SQL operator LIKE can perform limited regular expression comparisons. A new operator called "containing" can be introduced which can search for strings or regular expressions contained in a particular region. A match is obtained if the search string or expression matches anywhere in the subtree rooted at that region. For example, the expression *book.chapter* **containing** *"query language"* will return all chapters that have the string "query language" anywhere in its content.

4 Types of possible queries

This section can easily take up all of this paper – so we only discuss the major types of queries one can perform with SQL – especially those that are difficult to do in current SGML database systems. Note that some of the high-end systems based on object-oriented and object-relational databases (such as Texcel Information Manager [Hol95]) do claim to have SQL capabilities, and chances are, in the future most of the SGML systems will implement SQL. The following are the broad categories of queries that we consider here:

1. Simple selections using selection conditions combined with boolean operators (e.g., find the books which have more than 200 pages <u>and</u> cost more than $20).
2. Selections involving multiple databases or multiple subtrees of the same database having a common region (e.g., find the books written <u>by an author who</u> has edited another book).
3. Selections involving aggregate functions such as count, minimum, maximum, etc. (e.g., find the <u>total number of</u> chapters in "The SGML Handbook").
4. Selections involving existential and universal quantifiers, such as for all, there exists (e.g., find the authors <u>all</u> of whose books cost more than $20).
5. Selections involving subqueries (e.g., find all the books written by the author of "Practical SGML").
6. Selections involving negations, resulting in non-monotonic queries (e.g., find the authors who have <u>not</u> edited any book).

Detailed SQL formulation of these queries are given in Section 5.1.

5 Example scenario and sample queries

We can demonstrate how queries like the above can be posed in SQL using a sample schema - an extended form of the relational schema described above. In the following DTD, we have added a few additional information to embed the body of the book. Note that to do the same in a relational database, we will have multiple tables (ten tables with a clever design). We will show that we can perform many complex and interesting queries even with a simple design like the following DTD for a generic "books" database.

```
<!DOCTYPE books [
<!ELEMENT books                 O O (book*)>
<!ELEMENT book                  - - (header, body)>
<!ELEMENT header                - - (bibl,other)>
<!ELEMENT bibl                  - O (name,(author)+,(editor)*,year,publisher)>
<!ELEMENT other                 - O (ISBN,price)>
<!ELEMENT (ISBN|price)          - - (#PCDATA)>
<!ELEMENT (name|year)           - - (#PCDATA)>
<!ELEMENT (editor|publisher)    - - (#PCDATA)>
<!ELEMENT author                - O (name,affil)>
<!ELEMENT affil               - - (#PCDATA)>
<!ELEMENT body                - O (chapter+)>
<!ELEMENT chapter             - O (chtitle,(section)*) +(P)>
<!ELEMENT section             - O (sectitle,P+)>
<!ELEMENT (P|chtitle|sectitle) - O (#PCDATA)>
]>
```

5.1 Query formulation

All relational database-like queries still work in this new formulation. In fact, most of the queries described before will run without changes (other than the changes brought in by the new naming conventions and the removal of unnecessary join conditions). For example, the first query, find the name and price of the books that have more than 500 pages, will still be the same as Query 1 in Section 2.3.1. However, Query 2 becomes much simpler, since no joins are necessary:

```
SELECT *
FROM books
WHERE author.name = ''Charles Goldfarb''
  AND price < 50
```

In the above query, the WHERE clauses are internally expanded to (i) book.header.bibl.author.name = "Charles Goldfarb" and (ii) book.header.other.price < 50. Note that in the first clause, the path is partially specified to avoid ambiguity with book.header.bibl.name.

5.2 More advanced query formulation

Let us now look at the queries mentioned in Section 4 and see how to solve them easily using SQL.

1. Find the books which have more than 200 pages and cost more than $20.

   ```
   SELECT * FROM books
   WHERE pages > 200
     AND price > 20
   ```

2. Find the books written by an author who has edited another book.

   ```
   SELECT b1 FROM books.book b1, books.book b2
   WHERE b1..bibl.name <> b2..bibl.name
     AND b1..author.name = b2..editor
   ```

   Note that in the above query, we are joining the database with itself, making sure that the author of one of the books is the editor of the other. The clause **b1 <> b2** is necessary to avoid books where the editor is also an author.

3. Find the total number of chapters in "The SGML Handbook"

   ```
   SELECT COUNT(chapter)
   FROM books
   WHERE bibl.name = ''The SGML Handbook''
   ```

   There is nothing complex about this query other than the use of an aggregate function COUNT.

4. Find the authors all of whose books cost more than $20.

```
SELECT b1..author
FROM books b1
WHERE NOT EXISTS (
  SELECT * FROM books b2
  WHERE b2..author.name = b1.author.name
    AND b2..price < 20)
```

This query is similar to the one described in Section 2.3.2. The query here has been restated as: "find the authors so that there does not exist any book written by him which costs less than $20". Note that not requiring the join conditions has made this query much more readable than the earlier one.

5. Find all books written by the author of "Practical SGML"

```
SELECT * FROM books
WHERE author.name = (
  SELECT author.name
  FROM books
  WHERE bibl.name = ''Practical SGML'')
```

The above query uses a subquery to first find out the name of the author who wrote "Practical SGML", and then uses the result in the condition for the author's name. Note that it may be necessary to change the "=" operator to "IN" if there are multiple books with the name "Practical SGML" in the database - the latter might be safer to use if the user is not aware of the number of "Practical SGML"s in the database.

6. Find the authors who have not edited any book

```
SELECT author.name
FROM books b1
WHERE NOT EXISTS (
  SELECT * from books b2
  WHERE b2..editor = b1..author.name)
```

This one is a simple use of NOT EXISTS.

5.3 Additional query features

The queries, which we discussed above, only use selections based on conditions involving the character content of GIs. Of course, it is also possible to query only on the content model, with queries like "find the books that do not have any appendices" (either using a NOT EXISTS or by doing a COUNT(appendix) = 0). It is also possible to query based on attribute values, using an operator that can retrieve attribute values of a GI. This will be among the minor enhancements to SQL. Using this approach, the query "find all the pages a particular HTML page refers to" can be solved using a query like `SELECT A.attval(HREF) FROM HTML`.

Intermixing all the above techniques can greatly enhance the capacity of the language, while at the same time keeping the general readability of the language quite simple.

6 Conclusion

SQL is the standard language in relational databases, but it is yet to make a major influence in the SGML world. With the advances in SGML database technology, it is easily conceivable that in the future SQL will be a standard language in the SGML domain. SQL already exists in object-oriented and object-relational databases, and in SGML systems built on top of these databases. The queries we saw here are quite generic - but still difficult to solve in the current SGML systems. We would like to see SQL implemented in SGML systems - and that day is not too far away.

References

[AC75] M. M. Astrahan and D. Chamberlin. Implementation of a structured english query language. *Communications of the ACM*, 18(10), October 1975. Also published in/as: 19 ACM SIGMOD Conf. on the Management of Data, King(ed), May.1975.

[BDK92] Francois Bancilhon, Calude Delobel, and Paris Kaneliakis. *Building an object-oriented Database Dystem: The story of O_2*. Morgan Kaufmann Publishers, 1992.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. *SIGMOD RECORD*, 23(2):313–324, June 1994.

[EN89] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummins, 1989.

[Hol95] Sebastian Holst. Database evolution: the view from over here (a document-centric perspective). In Yuri Rubinsky, editor, *Proceedings, SGML '95*. Graphic Communications Association, December 1995.

[MS92] Jim Melton and Alan R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers, 1992.

[SD96] Arijit Sengupta and Andrew Dillon. Extending sgml to accommodate database functions: A methodological overview. Communicated for publication at the JASIS special Issue on structured content, January 1996.

[SQL86] ANSI X3.135-1986, Database Language SQL, 1986.

[SQL89] ANSI X3.135-1989, Database Language SQL, 1989.

[SQL92] ANSI X3.135-1992, Database Language SQL. Also ISO/IEC 9075:1992, 1992.

[SQL96] ANSI X3H2 standards group. Proposed standard for Object Oriented Database Language SQL3, 1996.