1-1-2008

Xml-based implementation of a bibliographic database and recursive queries

Kirankumar Jayakumar
University of Nevada, Las Vegas

XML-BASED IMPLEMENTATION OF A BIBLIOGRAPHIC DATABASE AND RECURSIVE QUERIES

by

Kirankumar Jayakumar

Bachelor of Computer Science & Engineering
University of Madras, India
2004

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
August 2008

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

MAY 29TH, 2008

The Thesis prepared by

KIRANKUMAR JAYAKUMAR

Entitled

XML BASED IMPLEMENTATION OF A BIBLIOGRAPHIC DATABASE AND RECURSIVE QUERIES

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

1017-53

ABSTRACT

XML-based Implementation of a Bibliographic Database and Recursive Queries

by

Kirankumar Jayakumar

Dr. Kazem Taghva, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Structured Query Language (SQL) of relational database model does not have the expressive power to implement recursive queries. Consequently, recursive queries are implemented as an application program in the host language. The newly developed XML schema provides a different setting for database design and query implementation.

In this thesis, we design and implement an XML schema and a set of associated queries for a bibliographic database. We will investigate and demonstrate the shortcomings of both Xpath and Xquery as standard query languages for XML-based databases. We then show an efficient implementation of the recursive queries in XSLT programming language

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my mother, father, brother and sister for their continuous support, encouragement and love. I'm forever grateful to my spiritual Guru for everything in life.

I'm thankful to Dr. Taghva for supporting me throughout my academic program and thesis research and also for the in-depth knowledge I gained under his tutelage. I'm thankful to Dr. Datta, Dr. Gewali and Dr. Minor who have supported me throughout my graduate assistantship program. I'm grateful to Dr. Kim, Dr. Nartker and Dr. Muthukumar for participating in my thesis committee.

I would like to thank the Computer Science department staff Sharon Achamire, Mario Martin and Carmen Willis for being very helpful and supportive.

I'm thankful to my friends Swamynathan Sambamurthy and Kunal Metkar for their support and encouragement.

CHAPTER 1

INTRODUCTION

1.1 Relational model

Relational model is based on predicate logic and set theory. It was first introduced by Edgar Codd in 1969. In mathematics, the term "relation" is used to refer to a table. Contrary to the popular belief, the term "relation" does not refer to the idea of links between tables. The relational model allows the creation of a consistent, logical representation of information. The consistency of the data is achieved through constraints.

A relation (or table) consists of tuples (or rows). Each tuple is an unordered set of data values. A data value is associated with an attribute (or column name). A relation (table) with tuples of n values (n number of columns) has arity of n.

1.1.1 Data domain

The set of possible values for a given attribute is called "data domain". In mathematical terms, it means that the attribute value must be an element of the specified set.

1.1.2 Constraints

Constraints are a way of restricting the values an attribute can take. It allows a finer degree of control over the attribute's values. Using constraints it is possible to enforce business rules on the attribute values, whereas with the domain, it's only possible to enforce set-membership. For instance, for the year field, in addition to enforcing the year to be a number, it's possible to set a lower and upper limit (such as 1900 – 2100).

1.1.3 Keys

A key is an attribute value which can uniquely identify a tuple. A key can be composed of more than one attribute – in which case, it is called a "compound key".

1.1.4 Foreign key

A foreign key is a *referential constraint* imposed between two tables. One or more columns in the child table can refer to a primary key or a candidate key column(s) in the master table. All the values present in the foreign key column must map to a value present in the primary key column or can be NULL. Contrary to the primary key, which enforces that all the values in the column must be unique, the foreign key column can have more than one row having the same value. Foreign key represents the many-to-one relationship between tables. When a foreign key makes reference to the primary key of its own table, then it is called as "self-referencing" or "recursive" foreign key.

1.2 Query processing

A typical query in a relational database involves retrieving data from one or more tables, specifying a condition on which to limit the number of records and optionally sort the record set. In addition, aggregation can also be performed and additional conditions

can be applied on aggregated attributes as well. Joins can be performed using the conditions. Aggregate functions can be used to compute values on an aggregated set of data (E.g. average value for a set of records grouped by a particular column). The limitation with relational query processing is that complex queries, which require loops or recursive function calls, cannot be implemented. To achieve this, an external programming language such as Java or PL/SQL must be used in addition to the SQL.

1.3 Recursion

A graph or tree relationship can be established between rows of the table using the recursive foreign key. Typically, this can be achieved by having a "Parent Node" foreign key column. This value will point to the primary key value of the parent row. As this is a graph structure, it is only possible to query using recursive approach rather than iterative approach. Graph traversal algorithms such as *Depth First Search* or *Breadth First Search* can be used for querying. An example is shown in figure 1.1 and table 1.1

In our implementation of the database, the bibliographic reference between the articles is represented using recursive foreign keys. So, the problem of identifying whether an article *A* implicitly refers to another article *B* can be interpreted as a directed graph traversal problem, where the source node is *A* and the destination node is *B*. The presence of a path between the two nodes implies that the article *A* refers article *B*. This was achieved by using depth first search starting from node *A*, recursing until node *B* is reached. If node B is encountered, then the article *A* refers to article *B*. If all the paths are exhausted without success, then the article *A* does not refer to article *B*.

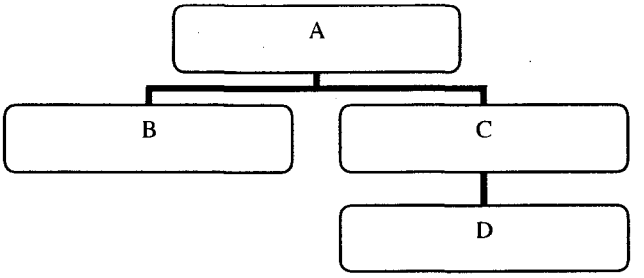

Figure 1.1

Node ID	Node Name	Parent Node ID
1	A	NULL
2	B	1
3	C	1
4	D	3

Table 1.1

CHAPTER 2

XML TECHNOLOGIES

2.1 Introduction

Extensible Markup Language (XML) is a standard for creating custom markup languages. Its primary purpose is for creating a standardized representation of data, which can be accessed across diverse platforms. XML is very flexible and allows the user to create custom structures. It is an open standard and can be used free of cost.

2.2 XML

An XML document is considered "well formed" when it conforms to the following semantic rules:

- Every tag opened must be closed.
- The elements must be properly nested – they cannot overlap.
- A document can have only one root element.
- An optional XML declaration tag can precede the root element. It is used for mentioning the XML version and the character encoding.
- Attribute values must be present within quotes

A "valid" XML document is one which conforms to the standard set by an XML Schema. *Document Type Definition* (DTD) and *XML Schema Definition* (XSD) are the popular XML schema languages. XSD is more powerful than DTD.

2.3 XML schema definition (XSD)

XML Schema is a *World Wide Web Consortium* (W3C) recommended standard schema language, which can enforce rules, in addition to the semantic rules, to an XML document. An instance of the XML Schema is called as *XML Schema Definition.* It usually has the file extension *.xsd.* An XML document can be associated with an XSD. The document must conform to the standards set by XSD to be considered "valid". XML Schema has 19 built in primitive data types and 25 derived data types. They represent commonly used data types such as string, integer, date, boolean etc.

2.3.1 Simple type

Simple types are derived from built in types or other simple types. A simple type can be defined by using one of the following methods

- Restriction – can be used to apply restrictions on a base simple type. Several types of rules, such as enumeration, minimum value, maximum value, length, regular expression etc. can be applied to narrow down the range of values of the base domain.
- List – can be used to define a simple type which contains list of white space separated simple type values.
- Union – can be used to define a simple type which is chosen from two or more simple types.

An element's type can be mapped to a simple type. Examples of how to construct a simple type using different methods is shown in the figure 2.1, 2.2 and 2.3

Simple type using restriction

```
<xsd:simpleType name="Phone7Digits">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1000000"/>
    <xsd:maxInclusive value="9999999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 2.1

Simple type using list

```
<xsd:simpleType name="importantDates">
  <xsd:list itemType="xsd:date"/>
</xsd:simpleType>
```

Figure 2.2

Simple type using Union

```
<xsd:simpleType name="PhoneNumber">
  <xsd:union memberTypes="Phone7Digits Phone10Digits"/>
</xsd:simpleType>
```

Figure 2.3

2.3.2 Complex type

The Simple Type is limited to applying the rules only on the content of an element. Using complex type, it is possible to define a nested structure and define attributes. As with simple type, an element can also be mapped to a complex type.

The complex type can be constructed by using one of the child elements described in the table 2.1. Generally, <SEQUENCE> and <ALL> are commonly used to construct the complex type.

<SEQUENCE> can be used to specify a sequence of elements. It also allows repetition of elements. It is possible to define a lower bound and an upper bound for the number of allowed child elements of the same type within the element.

<ALL> can be used when the order of elements is not of particular importance. No element within <ALL> can be repeated.

An example involving the above concepts is shown in the figure 2.4

Element	Description
SimpleContent	The complex type has character data or a SimpleType as content and contains no elements, but may contain attributes.
ComplexContent	The complex type contains only elements or no element content (empty).
Group	The complex type contains the elements defined in the referenced group.
Sequence	The complex type contains the elements defined in the specified sequence
Choice	The complex type allows one of the elements specified in the choice element.
All	The complex type allows any or all of the elements specified in the all element to appear once.

Table 2.1[1]

Example of Complex Type involving Simple Type elements, Sequence and All

```
<xsd:simpleType name="ST_PublisherName">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z|a-z|0-9|\.|\-| ]+"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- Represents a row -->
  <xsd:complexType name="CT_Publisher">
    <xsd:all>
      <xsd:element name="PublisherID" type="xsd:integer"/>
      <xsd:element name="PublisherName" type="ST_PublisherName"/>
      <xsd:element name="Address" type=" xsd:string"/>
    </xsd:all>
  </xsd:complexType>

<!-- Represents a table -->
<xsd:complexType name="CT_Publishers">
  <xsd:sequence>
   <xsd:element name="Publisher" type="CT_Publisher" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 2.4

2.3.3 Element

The <Element> tag in the XSD defines the actual tag which can occur within an XML document. It can be mapped to either a simple type, complex type or built-in data type. The <Element> tag can occur directly under the <Schema> tag (global scope) or within <ComplexType> tag.

2.3.4 Integrity constraints

Integrity constraints for an XML document can be defined within the XSD.

- Primary Key: The primary key can be defined using <Key> tag. Its scope is within the containing element of the instance document. <Selector> and <Field> tags are used to locate the particular key element.
- Candidate Key: <Unique> tag can be used to define candidate keys.
- Foreign Key: Foreign key to a primary key or a candidate key can be defined using the <KeyRef> tag. The name of the primary key or candidate key must be specified in the "refer" attribute.

2.4 XPath

XPath is a language for selecting nodes from an XML document based on certain condition. It can also be used for computing values from the retrieved nodes. The syntax is similar to file path in UNIX. In addition, XPath provides several in-built functions and operators.

The operators available are

- Path expressions
- Union
- Boolean
- Arithmetic
- Comparison

The functions available are

- Node set functions
- String functions
- Boolean functions

- Number functions

2.5 XQuery

XQuery is a querying language which can be used to extract data from XML documents. Its semantics is similar to SQL. Most queries which can be implemented in SQL for a relational database can be implemented in XQuery for an equivalent XML database. XQuery has "FLWOR" structure, which is similar to SQL. A "FLWOR" expression has the following clauses

- FOR: To iterate through a set of values (can be nodes, or the result of a function)
- LET: Assignment operation
- WHERE: To specify a condition on which to execute the expression
- ORDER BY: To sort the record set
- RETURN: Output expression

2.6 XSLT

Extensible Stylesheet Language Transformation (XSLT) is a language for transforming XML documents from one structure into another. XSLT is Turing complete and hence has more capabilities than XQuery. XSLT is considered to be a template processor. The language structure of XSLT is influenced by functional programming languages. XQuery is restricted by FLWOR structure. Hence, more complex querying, such as recursion, cannot be achieved. Since XSLT has the full capabilities of a programming language, it is possible to implement complex queries which involve recursion.

CHAPTER 3

BIBLIOGRAPHIC XML DATABASE

3.1 Introduction

A bibliographic database is a database containing bibliographic records. It is designed with the intent of capturing all the bibliographic information. It holds information about the material, organized into their respective material type categories such as Articles, Books, Conferences, Proceedings, Journals etc. All the reference information is also captured.

3.2 Commonly used simple types

There are many cases which require a common custom simple type. All these simple types were created and placed in a common name space, so that they can be consistently referred to by the many complex types, which represent different columns belonging to different tables. Further, it allows imposing global restrictions on the database. For e.g., by forcing all the date fields to be of the custom date type rather than the default date type, it is possible to set a lower and upper bound (e.g. 1900-2100)

The table 3.1 lists all the custom types used

Simple Type	Base Type	Description
ST_PKID	Integer	Non negative 5 digit number. Used for primary key fields
ST_Address	String	String restricted with regular expression to allow only valid addresses.
ST_genstring	String	String type for use within the database. Currently restricted to 20 characters
ST_gennumber	Integer	Number type for use within the database. Currently restricted to 5 digit non negative number.
ST_gendate	Date	Date type for use within the database. Can be restricted in future for a specific date range (E.g. 1/1/1900-.12/31/2099)

Table 3.1

3.3 Elements

In XML, the database can be represented as a hierarchy of elements. The column can be represented by an element - lets call it "column element". The attribute value can be represented as the contents of the column element. A group of related column elements is contained within a "row element". A group of related "row elements" is contained within a "table element". Note that we use the attribute "maxoccurs = unbounded" on the row element, so that we can repeat any number of row elements within the table element. A group of the table elements can be composed within a single "database element". Here in this case, we use "maxOccurs=1" for each table element because of the fact that there cannot be more than one table with the same name. The primary key and foreign key relationships can be enforced within the "database element". Thus, the whole database can be represented using this hierarchy of elements.
This concept is illustrated in the sample XML structure in figure 3.1.

```
<Database>
  <Table1>
    <Table1_Row>
      <Table1_Col1></Table1_Col1>
      <Table1_Col2></Table1_Col2>
      <Table1_Col3></Table1_Col3>
    </Table1_Row>
    <Table1_Row>
      <Table1_Col1></Table1_Col1>
      <Table1_Col2></Table1_Col2>
      <Table1_Col3></Table1_Col3>
    </Table1_Row>
  </Table1>
  <Table2>
    <Table2_Row>
      <Table2_Col1></Table2_Col1>
      <Table2_Col2></Table2_Col2>
    </Table2_Row>
    <Table2_Row>
      <Table2_Col1></Table2_Col1>
      <Table2_Col2></Table2_Col2>
    </Table2_Row>
  </Table2>
</Database>
```

Figure 3.1

In actual implementation, a hierarchy of simple type, complex type and element was used to achieve this structure.

3.3.1 Annual reports

The <AnnualReports> element represents a relation. It contains the <AnnualReport> recurring element, which is a tuple. The purpose of this element is to store the bibliographic information related to the annual reports. <AnnualReport> tuple

contains the following elements. The elements cannot occur more than 1 time (ie) no two attribute with the same name are allowed.

Element	Type	Description
<AnnualReportID>	ST_PKID	Primary key
<Editor>	ST_genstring	Editor name
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	
<Pages>	ST_genstring	

Table 3.2

3.3.2 Articles

The bibliographic entries which are articles are stored under the <Article> element. <Articles> represent a relation and contain one or more <Article> element. Each <Article> element represents a tuple. It has the elements described in the table 3.3.

3.3.3 Authors

Information related to an author or a list of authors is stored in the <Author> element. It contains the attribute elements shown in table 3.4. It contains references to the material which the author(s) has written.

3.3.4 Books on CD

Bibliographic information related to a book which is in CD format is stored in the <BookOnCD> element. It contains the elements listed in table 3.5.

Element	Type	Description
ArticleID	ST_PKID	Primary key
Title	ST_genstring	
Journal	ST_genstring	
Volume	ST_genstring	
Number	ST_gennumber	
Pages	ST_genstring	
Edition	ST_genstring	
PublisherID	ST_PKID	Foreign key referencing <Publisher>
MonthYearID	ST_PKID	Foreign key (referencing <MonthYear>)
Note	ST_genstring	

Table 3.3

Element	Type	Description
<OnlineSourceID>	ST_PKID	Foreign key (referencing <OnlineSource>)
<BookOnVCDID>	ST_PKID	Foreign key (referencing <BookOnVCD>)
<BookOnTapeID>	ST_PKID	Foreign key (referencing <BookOnTape>)
<BookOnCDID>	ST_PKID	Foreign key (referencing <BookOnCD>)
<PeriodicalID>	ST_PKID	Foreign key (referencing <Periodical>)
<AnnualReportID>	ST_PKID	Foreign key (referencing <AnnualReport>)
<ConferenceID>	ST_PKID	Foreign key (referencing <Conference>)
<ManualID>	ST_PKID	Foreign key (referencing <Manual>)
<TechReportID>	ST_PKID	Foreign key (referencing <TechReport>)
<MagazineID>	ST_PKID	Foreign key (referencing <Magazine>)
<JournalID>	ST_PKID	Foreign key (referencing <Journal>)
<InProceedingID>	ST_PKID	Foreign key (referencing <InProceeding>)
<InBookID>	ST_PKID	Foreign key (referencing <InBook>)
<ProceedingID>	ST_PKID	Foreign key (referencing <Proceeding>)
<BookID>	ST_PKID	Foreign key (referencing <Book>)
<ArticleID>	ST_PKID	Foreign key (referencing <Article>)
<ID>	ST_PKID	Primary Key
<AuthorList>	ST_genstring	List of authors

Table 3.4

Element	Type	Description
<BookOnCDID>	ST_PKID	Primary key
<CDName>	ST_genstring	
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Volume>	ST_genstring	

Table 3.5

3.3.5 Books on tape

Bibliographic information related to a book which is in tape format is stored in the <BookOnTape> element. It contains the elements listed in table 3.6.

Element	Type	Description
<BookOnTapeID>	ST_PKID	Primary key
<TapeName>	ST_genstring	
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Volume>	ST_genstring	

Table 3.6

3.3.6 Books on VCD

Bibliographic information related to a book which is in VCD format is stored in the <BookOnVCD> element. It contains the elements listed in table 3.7.

Element	Type	Description
<BookOnVCDID>	ST_PKID	Primary key
<VCDName>	ST_genstring	
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Volume>	ST_genstring	

Table 3.7

3.3.7 Books

Bibliographic information related to a book which is in traditional print format is stored in the <Book> element. It contains the elements listed in table 3.8.

Element	Type	Description
<BookID>	ST_PKID	Primary key
<Editor>	ST_genstring	
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Pages>	ST_genstring	
<Volume>	ST_genstring	
<Edition>	ST_genstring	
<Series>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.8

3.3.8 Conferences

Bibliographic information related to a conference is stored in the <Conference> element. It contains the elements listed in table 3.9.

Element	Type	Description
<ConferenceID>	ST_PKID	Primary key
<Name>	ST_genstring	
<Title>	ST_genstring	
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)

Table 3.9

3.3.9 In books

Bibliographic information which is classified as "in book" is stored in the <InBook> element. It contains the elements listed in table 3.10.

Element	Type	Description
<InBookID>	ST_PKID	Primary key
<Editor>	ST_genstring	
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Volume>	ST_genstring	
<Edition>	ST_genstring	
<Series>	ST_genstring	
<Chapter>	ST_genstring	
<Pages>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.10

Element	Type	Description
<InProceedingID>	ST_PKID	Primary key
<Editor>	ST_genstring	
<Title>	ST_genstring	
<BookTitle>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Pages>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.11

3.3.10 In proceedings

Bibliographic information which is classified as "in- proceeding" is stored in the <InProceeding> element. It contains the elements listed in table 3.11.

3.3.11 Journals

Bibliographic information related to a journal is stored in the <Journal> element. It contains the elements listed in table 3.12.

Element	Type	Description
<JournalID>	ST_PKID	Primary key
<Title>	ST_genstring	
<Volume>	ST_genstring	
<Number>	ST_gennumber	
<Pages>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Edition>	ST_genstring	

Table 3.12

3.3.12 Magazines

Bibliographic information related to a magazine is stored in the <Magazine> element. It contains the elements listed in table 3.13.

3.3.13 Manuals

Bibliographic information related to a manual stored in the <Manual> element. It contains the elements listed in table 3.14

3.3.14 MonthYear

The purpose of <MonthYear> element is to keep track of the month and year in which a material was published.

Element	Type	Description
<MagazineID>	ST_PKID	Primary key
<Name>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Pages>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.13

Element	Type	Description
<ManualID>	ST_PKID	Primary key
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<Pages>	ST_gennumber	
<Volume>	ST_genstring	
<Edition>	ST_genstring	
<Series>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.14

Element	Type	Description
<MonthYearID>	ST_PKID	Primary key
<Month>	ST_Month	Custom month simple type defined within the local namespace
<Year>	ST_Year	Custom year simple type defined within the local namespace

Table 3.15

3.3.15 Online sources

Bibliographic information related to an online source is stored in the < OnlineSource> element. It contains the elements listed in table 3.16

Element	Type	Description
<OnlineSourceID>	ST_PKID	Primary key
<TopicName>	ST_genstring	
<PostedDate>	ST_gendate	
<RetrievedDate>	ST_gendate	
<URL>	ST_URL	Valid URL string
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<EntryType>	ST_genstring	
<Type2>	ST_genstring	

Table 3.16

3.3.16 Organizations

Information related to an organization is stored in the <Organization> element. It contains the elements listed in table 3.17

Element	Type	Description
<OrganizationID>	ST_PKID	Primary key
<OrganizationName>	ST_OrganizationName	Valid organization name
<Address>	ST_Address	

Table 3.17

3.3.17 Periodicals

Bibliographic information related to a periodical is stored in the <Periodical> element. It contains the elements listed in table 3.1

3.3.18 Proceedings

Bibliographic information related to a proceeding is stored in the <Proceeding> element. It contains the elements listed in table 3.19

Element	Type	Description
<PeriodicalID>	ST_PKID	Primary key
<Editor>	ST_genstring	
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	
<Pages>	ST_genstring	

Table 3.18

Element	Type	Description
<ProceedingID>	ST_PKID	Primary key
<Editor>	ST_genstring	
<Title>	ST_genstring	
<PublisherID>	ST_PKID	Foreign key (referencing <Publisher>)
<OrganizationID>	ST_PKID	Foreign key (referencing <Organization>)
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Note>	ST_genstring	

Table 3.19

3.3.19 Publishers

Information related to a publisher is stored in the <Publisher> element. It contains the elements listed in table 3.20

Element	Type	Description
<PublisherID>	ST_PKID	Primary key
<PublisherName>	ST_PublisherName	Valid publisher name
<Address>	ST_Address	

Table 3.20

3.3.20 Required fields

The required fields of the database can be tracked by storing in the <RequiredField> element. It contains the elements listed in table 3.21

Element	Type	Description
<Field>	xsd:string	XPath of the mandatory field

Table 3.21

3.3.21 Tech reports

Bibliographic information related to a tech report is stored in the <TechReport> element. It contains the elements listed in table 3.22

Element	Type	Description
<TechReportID>	ST_PKID	Primary key
<Title>	ST_genstring	
<MonthYearID>	ST_PKID	Foreign key (referencing <MonthYear>)
<Subtitle>	ST_genstring	
<Pages>	ST_genstring	
<Note>	ST_genstring	

Table 3.22

3.3.22 Relationships

The bibliographic reference relationship between materials can be tracked using <Relationship> element. The reference relationship is a graph & it can be represented

using a self referencing foreign key. This relation is useful in determining explicit and implicit references between articles.

Element	Type	Description
<ID>	ST_PKID	Primary key
<NodeXPath>	xsd:string	XPath of a particular material
<ParentID>	ST_PKID	Self referencing foreign key

Table 3.23

3.3.23 BibliographicDB

The <BibliographicDB> element represents the database. It contains all other table elements mentioned above. One only instance of a table element can occur. The primary key and foreign key relationships are enforced from this context.

Element	Type	Description
<ID>	ST_PKID	Primary key
<NodeXPath>	xsd:string	XPath of a particular material
<ParentID>	ST_PKID	Self referencing foreign key

Table 3.24

3.4 Queries

The queries for the XML database range from simple value extraction from the Document Object Model (DOM) tree, to SQL styled non-recursive relational queries, to complex recursive queries.

The following are some of the queries which typically occur in a database of this kind.

- All papers referred to by X explicitly
- All papers referred to by X implicitly
- To get the title of the material written by a particular Author
- To list the number of books written by each Author
- To get the list of organizations along with the address of Proceedings
- To get the list of all the Publications published by X
- To get the lists of proceedings based on Title and Book Title based on a particular organization
- To get the pairs of organization ids who have the same organization name
- To get the number of entry types (like books, articles etc) a publisher 'A' published in a particular month-year
- To get the total number of publications for each publisher for all the entry types
- To get the highest number of publications for all the entry types

CHAPTER 4

IMPLEMENTATION OF QUERIES

4.1 Introduction

Querying in an XML database environment can range from extracting the content or attribute value of a particular node, to conditional comparison & selection of a set of nodes, to recursive operation of nodes based on a certain condition. In this research, the following approach was adopted.

- XPath will be used for locating a particular node or a set of nodes satisfying a common criteria within the XML tree
- XQuery will be used in conjunction with XPath for relational non recursive queries
- XSLT will be used in conjunction with XPath for recursive queries

4.2 XPath queries

XPath is a language for selecting nodes in an XML document. The querying capabilities of XPath, used as such, are very limited. XPath is usually used in conjunction with XQuery or XSLT for more complex queries. The following are simple queries where XPath can be used

- *"What is the Organization ID of the organization whose name is 'X'?"*

For this query, we have to locate the <Organization> node, where its child element <OrganizationName> has the content value "X" and return the child element <OrganizationID>. In XPath it can be framed as follows

/ BibliographicDB/Organizations/Organization[OrganizationName="X"]/OrganizationID

- *"Get all the articles published by a given publisher in a particular month year?"*

This query cannot be achieved in a single step, as it is a JOIN operation, which involves pulling up the data from multiple relations. This query can only be solved using multiple steps. We have to first determine the publisher ID for the given publisher name (E.g.: "ABC Publisher") using the following query. Let's call this result as X.

/ BibliographicDB/Publishers/Publisher[PublisherName="ABC publisher"]/ PublisherID

The next step is to determine the MonthYearID for the given the given month year, for e.g april 2008. Let's call this result as Y

/ BibliographicDB/MonthYears/MonthYear[Month="4" and Year="2008"]/MonthYearID

Using the above two sub results X and Y, we can now determine the given query using the following XPath query

/ BibliographicDB/Articles/Article[PublisherID=X and MonthYearID=Y]

As shown from the above queries, XPath is very useful for locating a particular node or a set of nodes for a given condition. However complex queries such as JOINS or recursive queries cannot be achieved through XPath.

4.3 Non recursive relation query implementation using XQuery

XQuery is an XML query language whose semantics is very similar to SQL. In general, any query which can be implemented in SQL can be implemented in XQuery. XQuery has more capabilities than XPath. XQuery is a good choice for queries which involve joins & aggregation. The following queries were implemented in XSLT

- *"Get the list of all the Publications published by X"*

Algorithm:

Step 1: Determine the PublisherID of X by using an appropriate XPath expression.

Assign it to the variable $PublisherID

Step 2: Use XPath to select any relation node, any tuple node, which has <PublisherID>

node whose value is $PublisherID

The XQuery implementation is shown in the figure 4.1

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")
let $db := $src/p1:BibliographicDB
let $PublisherID := $src/p1:BibliographicDB/Publishers/Publisher[PublisherName =
"ABC Publisher"]/PublisherID
return
<Result>
  {$db/node()/node()[PublisherID=$PublisherID]}
</Result>
```

Figure 4.1

- *"Get the list of organizations along with the address of proceedings"*

Algorithm:

Step 1: FOR each <Organization> node in the path

/BibliographicDB/Organizations/Organization

Step 2: FOR each <Proceeding> node in the path

/BibliographicDB/Proceedings/Proceeding

Step 3: If OrganizationID of the <Organization> node matches the OrganizationID of the <Proceeding> node then output the Proceeding title, Organization name and the address

Step 4: End FOR

Step 5: End FOR

The XQuery implementation is shown in the figure 4.2

- *"Get the pairs of organization ids who have the same organization name"*

Algorithm:

Step 1: FOR $o1 in each distinct organization name

Step 2: If the count of nodes which have the organization name $o1 under <Organizations> element is more than 1, then output the organization name and all the <OrganizationID> elements which match the criteria

Step 3: End FOR

The XQuery implementation is shown in the figure 4.3

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")
for $Organization in ($src/p1:BibliographicDB/Organizations/Organization)
for $Proceeding in ($src/p1:BibliographicDB/Proceedings/Proceeding)
where $Organization/OrganizationID = $Proceeding/OrganizationID
return
<Result>
  <Proceeding>
    {$Proceeding/Title/text()}
  </Proceeding>
  <Organization>
    {$Organization/OrganizationName/text()}
  </Organization>
  <Address>
    {$Organization/Address/text()}
  </Address>
</Result>
```

Figure 4.2

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")
for $o1 in distinct-values(
$src/p1:BibliographicDB/Organizations/Organization/OrganizationName)
where count(
$src/p1:BibliographicDB/Organizations/Organization[OrganizationName = $o1] ) > 1
return
<Result>
  <OrganizationName>{$o1}</OrganizationName>
  {$src/p1:BibliographicDB/Organizations/Organization[OrganizationName =
$o1]/OrganizationID}
</Result>
```

Figure 4.3

- *"Get the number of entry types (like books, articles etc) a publisher 'A' published in a particular month-year"*

Algorithm:

Step 1: Determine the PublisherID and MonthYearID for the given input data using XPath

Step 2: FOR $db in each relation node under the database node

Step 3: If the count of tuple nodes which has both matching PublisherID and MonthYear ID, then output the relation node name (which gives the entry type) and the count

Step 4: End FOR

The XQuery implementation is shown in the figure 4.4

- *"Get the total number of publications for each publisher for all the entry types"*

Algorithm:

Step 1: FOR $Publisher in each <Publisher> node

Step 2: Print the name of the publisher

Step 3: Print the count of nodes of any tuple, under any relation, whose PublisherID matches the current node's PublisherID

Step 4: End FOR

The XQuery implementation is shown in the figure 4.5

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")

let $PublisherID := $src/p1:BibliographicDB/Publishers/Publisher[PublisherName =
"ABC Publishers"]/PublisherID

let $MonthYearID := $src/p1:BibliographicDB/MonthYears/MonthYear[Month2 =
"4" and Year2 = 2008]/MonthYearID

for $db in ($src/p1:BibliographicDB/node())
where
count($db/node()[PublisherID=$PublisherID and MonthYearID=$MonthYearID]) > 0
return
<Result>
   <EntryType>
      {node-name($db)}
   </EntryType>
   <Count>
   {count($db/node()[PublisherID=$PublisherID and MonthYearID=$MonthYearID])}
   </Count>
</Result>
```

Figure 4.4

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")

for $Publisher in ($src/p1:BibliographicDB/Publishers/Publisher)
return
<Result>
   <Publisher>
      {$Publisher/PublisherName/text()}
   </Publisher>
   <Total>
{
count($src/p1:BibliographicDB/node()/node()[PublisherID=$Publisher/PublisherID])
}
   </Total>
</Result>
```

Figure 4.5

- *"List the number of books written by each Author"*

Algorithm:

Step 1: FOR each distinct value of Author names

Step 2: Print the count of <Author> nodes whose author matches the current author

Step 3: End FOR

The XQuery implementation is shown in the figure 4.6

```
declare namespace p1 = "http://drtaghva.edu/XML/BibliographicDB";
let $src := doc("file:///C:/Thesis/code/test/test_instance1.xml")
for $uniqueAuthors in distinct-values($src/p1:BibliographicDB/Authors/Author/AuthorList)
return
<Result>
  <Author>
    {$uniqueAuthors}
  </Author>
  <BooksWritten>
    {count($src/p1:BibliographicDB/Authors/Author[AuthorList=$uniqueAuthors])}
  </BooksWritten>
</Result>
```

Figure 4.6

4.4 Recursive query implementation using XSLT

XSLT is a Turing complete language. XSLT is not limited by the "FLWOR" structure of XQuery and is a good choice for the implementation of recursive queries.

- *"Find all the materials which are both explicitly and implicitly referenced by X"*

The references are stored in the <Relationships> element with recursive foreign key. When X references Y, then X explicitly references Y. When X references Y and Y references Z, then X implicitly references Z. This query can be interpreted as a directed

graph traversal problem where each material is a vertex and the relationship is a directed edge. The following algorithm uses Depth First Search.

Algorithm:

Function: DFS(Node, Path)

Step 1: Display the current node, which is an XPath to the material (books, articles, journals etc.)

Step 2: Find all the materials which refer the current node (children).

Step 3: If there are more than one material then

Step 4: FOR each child node

Step 5: If child node is not already in the path, then add child node to the Path variable and call DFS with the child node and the new Path variable

Step 6: End FOR

Step 7: End IF

The DFS() function must be invoked with start node and no value should be passed to the path parameter.

The XSLT implementation is shown in the figure 4.7

- *"Find if the material X implicitly references Y"*

In this query, if a path from X to Y exists, then X implicitly references Y.

Algorithm:

Function: DFS(Source node, target node, Path)

Step 1: If the current node equals target node then the implict reference exists. Display the path

Step 2: Find all the materials which refer the current node (children).

Step 3: If there are more than one material then

Step 4: FOR each child node

Step 5: If child node equals target node then the implict reference exists. Display the path and quit the loop

Step 6: If child node is not already in the path, then add child node to the Path variable and call DFS with the child node as source node, target node and the new Path variable

Step 7: End FOR

Step 8: End IF

The DFS() function must be invoked with start node, target node and no value should be passed to the path parameter. The XSLT implementation is shown in the figure 4.8

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:p1="http://drtaghva.edu/XML/BibliographicDB">

<xsl:template name="search">
    <xsl:param name="node"/>
    <xsl:param name="table"/>
    <!-- add path variable to keep track of cycles -->
    <xsl:param name="path" select="concat('->',concat($node,'->'))"/>
    <!-- Display the current node -->
    <xsl:text> </xsl:text>
    <xsl:value-of select="$table/Relationship[ID=$node]/NodeXPath"/>
    <!-- depth first graph algorithm (eliminates cycles) -->
    <!-- recurse if the current node has children -->
    <xsl:if test="count($table/Relationship[ParentID=$node]) > 0">
     <xsl:for-each select="$table/Relationship[ParentID=$node]/ID">
        <!-- make sure that current text() is not already present in the path
        (eliminate cycles) -->
        <xsl:if test="not(contains($path,concat(concat('->',text()),'->')))">
           <xsl:call-template name="search">
            <xsl:with-param name="node" select="text()"/>
            <xsl:with-param name="table" select="$table"/>
            <xsl:with-param name="path" select="concat(concat($path,text()),'->')"/>
           </xsl:call-template>
        </xsl:if>
     </xsl:for-each>
    </xsl:if>
 </xsl:template>

<xsl:template match="/">
   <xsl:call-template name="search">
    <xsl:with-param name="node"
      select="p1:BibliographicDB/Relationships/Relationship[NodeXPath='X']/ID"/>
    <xsl:with-param name="table" select="p1:BibliographicDB/Relationships"/>
   </xsl:call-template>
   <xsl:text> Completed</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Figure 4.7

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:p1="http://drtaghva.edu/XML/BibliographicDB">

<xsl:template name="search">
    <xsl:param name="node"/>
    <xsl:param name="table"/>
    <!-- add path variable to keep track of cycles -->
    <xsl:param name="path" select="concat('->',concat($node,'->'))"/>
    <xsl:param name="displayPath" select="$table/Relationship[ID=$node]/NodeXPath"/>
    <xsl:param name="target"/>
    <!-- depth first graph algorithm (eliminates cycles) -->
    <!-- recurse if the current node has children -->
    <xsl:if test="count($table/Relationship[ParentID=$node]) > 0">
     <xsl:for-each select="$table/Relationship[ParentID=$node]/ID">
        <!-- assign current loop value to $id (else displayPath not working properly) -->
        <xsl:variable name="id" select="text()"/>
        <xsl:choose>
          <xsl:when test="text()=$target">
            <xsl:text>Found </xsl:text>
            <xsl:value-of select="concat(concat($displayPath,'-
>'),$table/Relationship[ID=$id]/NodeXPath)"/>
          </xsl:when>
          <xsl:otherwise>
            <!-- make sure that current text() is not already present in the path (eliminate cycles) -->
            <xsl:if test="not(contains($path,concat(concat('->',text()),'->')))">
              <xsl:call-template name="search">
                <xsl:with-param name="node" select="text()"/>
                <xsl:with-param name="table" select="$table"/>
                <xsl:with-param name="path" select="concat(concat($path,text()),'->')"/>
                <xsl:with-param name="displayPath" select="concat(concat($displayPath,'-
>'),$table/Relationship[ID=$id]/NodeXPath)"/>
                <xsl:with-param name="target" select="$target"/>
              </xsl:call-template>
            </xsl:if>
          </xsl:otherwise>
        </xsl:choose>
     </xsl:for-each>
    </xsl:if>
 </xsl:template>

<xsl:template match="/">
   <xsl:call-template name="search">
    <xsl:with-param name="node"
select="p1:BibliographicDB/Relationships/Relationship[NodeXPath='A']/ID"/>
    <xsl:with-param name="target"
select="p1:BibliographicDB/Relationships/Relationship[NodeXPath='G']/ID"/>
    <xsl:with-param name="table" select="p1:BibliographicDB/Relationships"/>
   </xsl:call-template>
   <xsl:text> Completed</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Figure 4.8

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

It is demonstrated that for a relational database, an equivalent XML database can be created. Complex recursive queries cannot be implemented in SQL. XML querying technologies such as XPath and XQuery are limited in their querying capabilities. XSLT being "Turing complete" language is an ideal choice for the implementation of complex queries.

Specific scenarios pertaining to the Bibliographic database were identified.

- *"Find all the materials which are both explicitly and implicitly referenced by X"*
- *"Find if the material X implicitly references Y"*

The above two scenarios are examples where neither XPath nor XQuery can be used. XSLT was successfully used to solve the above two problems.

5.2 Future work

The XML technologies are evolving. More capabilities are being added with each release. It is expected that XQuery will have more querying capability. Currently, XQuery does not include "Group By" or "Having" clauses in its structure. It is expected that these features will be present in the newer version. Though XQuery has the ability to

define functions, it does not have the power of a complete programming language. Hence, it is not possible to implement graph traversal & searching algorithms. XSLT despite having the ability to define custom templates, which can be used as substitute for functions, cannot accurately simulate the functions. It is not possible to return a value from the template. If these features are available in the future versions, the current algorithms can be refined to be more elegant.

BIBLIOGRAPHY

[1] MSDN article on XML Schema Elements (http://msdn2.microsoft.com/en-us/library/ms256067(VS.85).aspx)

[2] Wikipedia article on XML (http://en.wikipedia.org/wiki/Xml)

[3] Wikipedia article on XPath (http://en.wikipedia.org/wiki/XPath)

[4] Wikipedia article on XQuery (http://en.wikipedia.org/wiki/XQuery)

[5] Wikipedia article on XSLT (http://en.wikipedia.org/wiki/XSLT)

[6] Wikipedia article on Relational Database (http://en.wikipedia.org/wiki/Relational_database)

[7] Breadth First Traversal in XSLT (http://www.tkachenko.com/blog/archives/000268.html)

[8] Recursion in XSLT (http://www.ibm.com/developerworks/xml/library/x-tiploop.html)

[9] Extending XQuery for Grouping, Duplicate Elimination and Outer Joins (http://www.idealliance.org/proceedings/xml04/papers/229/XQueryExtensionsFinal.html#S3.1)

[10] Philip M. Lewis, Arthur Bernstein & Michael Kifer; *Database and Transaction Processing.* Addison-Wesley, 2002.

VITA

Graduate College
University of Nevada, Las Vegas

Kirankumar Jayakumar

Address:
4340, Escondido Street, # 3
Las Vegas, Nevada 89119

Degrees:
Bachelor of Computer Science & Engineering (B.E), 2004
University of Madras, India

Thesis Title:
XML-based Implementation of a Bibliographic Database and Recursive Queries

Thesis Examination Committee:
Chairperson, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Thomas Nartker , Ph.D.
Committee Member, Dr. Yoohwan Kim, Ph.D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.