# HETEROGENEOUS DATABASE INTEGRATION FOR WEB APPLICATIONS

V. Rajeswari ,
Assistant Professor, Department of Information Technology,
Karpagam College of Engineering, Coimbatore –32,
it_rajeshvari@rediffmail.com

Dr. Dharmishtan K. Varughese ,
Professor, Department of Electronics and Communication Engg,
Karpagam College of Engineering, Coimbatore – 32,
drvarughese@yahoo.com

**Abstract**

**In the contemporary business and industrial environment, the variety of data used by organizations are increasing rapidly. Also, there is an increasing demand for accessing this data. The size, complexity and variety of databases used for data handling cause serious problems in manipulating this distributed information. Integrating all the information from different databases into one database is a challenging problem.**

**XML has been in use in recent times to handle data in web appliccations. XML (eXtensible Markup Language) is a very open way of data communication. XML has become the undisputable standard both for data exchange and content management. XML is supported by the giants of the software industry like IBM, Oracle and Microsoft.**

**The XML markup language should be the lingua franca of data interchange; but it's rate of acceptance has been limited by a mismatch between XML and legacy databases. This in turn, has created a need for a mapping tool to integrate the XML and databases.**

**This paper highlights the merging of heterogeneous database resource. This can be achieved by means of conversion of relational mode to XML schema and vice versa and by adding the semantic constraints to the XML Schema. The developments that the industry has seen in recent times in this field is referred to as the basis.**

***Keywords : XQuery, XML, Schema, XSLT, Heterogeneous databases.***

## I. INTRODUCTION

The most important cause of heterogeneous database is transformation and sharing of data. By using the metadata of participate sites [1], a framework is proposed for integrating heterogeneous database through XML technologies. XML is fully compatible with applications like JAVA and it can be combined with any application which is capable of processing XML irrespective of the platform it is being used on. XML is an extremely portable language to the extent that it can be used on large networks with multiple platforms like the internet and it can be used on handhelds or palm-tops or PDAs. XML is an extendable language meaning that new tags can be created or the tags which have already been created can be used.

The various semantic conflicts occurring among heterogeneous data cubes, the system architecture and related resolution procedures for all kinds of semantic conflicts are discussed[2]. For local data cubes with different schemas, a global XML Schema is defined to integrate the local cube structures [2] and each local cube is transformed into an XML document conforming to the global XML Schema. These transformed XML documents obtained from local cubes will be manipulated by pre-defined XQuery commands to form a unified XML document, which can be regarded as the global cube. The integrated global cube can be easily stored and manipulated in native XML databases.

## II. FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEMS

Relation schema is defined as a set of elements (attributes) connected by a structure (table) to describe something (an entity) in the relational database [3]. This entity should have a certain meaning associated with these elements, and the semantics specify how to interpret data values stored in the schema.

A classification scheme is proposed for various kinds of semantic conflicts, and it is applied to develop a query execution plan which optimizes multi-database queries based on different types of schematic conflicts with the least execution cost. The conflicts are classified as value-to-value conflicts, value-to-attribute conflicts, value-to-table conflicts, attribute-to-attribute conflicts, attribute-to-table conflicts, table-to-table conflicts[2]. These types of conflict classifications were further partitioned into two categories such as conflicts of similar schema structures and conflicts of different schema structures.

The semantic conflicts in a heterogeneous database system are further classified as schema conflicts and data conflicts[2]. There are two causes for schema conflicts. One is the use of different structures for the same concept, and the other is the use of different specifications for the same structure.

**Semantic Conflicts**

**1.** ***Value-to-value conflicts:*** These conflicts occur when databases use different representations for the same data. This type of conflicts can be further distinguished into the following types:

(a) Data representation conflicts: These conflicts occur when semantically related data items are represented in different data types.



ISSN : 0975-3397

(b)Data scaling conflicts: These conflicts occur when semantically related data items are represented in different databases using different units of measure[2].

(c)Inconsistent data: These conflicts occur when semantically related attributes for the same entity have different definite data values in different databases.

2. Value-to-Attribute conflicts: These conflicts occur when the same information is expressed as attribute values in one database and as an attribute name in another database.

3. Value-to-Table conflicts: These conflicts occur when the attribute values in one database [5] are expressed as table names in another database.

4.Attribute-to-Attribute conflicts: This occurs when semantically related data items are named differently or semantically unrelated data items are named equivalently[2]. The former case is also called synonyms and the latter case homonyms.

5.Attribute-to-Table conflicts: These conflicts occur if an attribute name of a table in a database is represented as a table name in another database.

6.Table-to-Table conflicts: These conflicts occur when information of a set of semantically equivalent tables[1] are represented in a different number of tables in another databases. When integrating relations with such conflicts into a global relation, null values are usually generated. Such phenomenon is also called missing data.

The types of conflict are further classified into the following two categories.

a. Conflicts of similar schema structures. This category includes value-to-value conflicts, attribute-to-attribute conflicts, and table-to-table conflicts.

b. Conflicts of different schema structures. This category includes value-to-attribute conflicts, value-to-table conflicts, and attribute-to-table conflicts[2].

For tables with conflicts of similar schema structures, an outerjoin operation is usually employed to integrate the tables into a unified one.

## III. XML AND XQUERY

### A ) XML Introduction

**XML** is a set of rules for encoding documents electronically. As of 2009, hundreds of XML-based languages have been developed[6], including RSS, Atom, SOAP, and XHTML. XML has become the default file format for most office-productivity tools, including Microsoft Office, OpenOffice.org, AbiWord, and Apple's iWork.

XML's design goals emphasize simplicity, generality, and usability over the Internet [5]. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

There are a variety of programming interfaces[6] which software developers may use to access XML data and several schema systems designed to aid in the definition of XML-based languages

### B) XML Declaration

This is a string that is often found at the very beginning of XML documents[5].

<?xml version="1.0" encoding='UTF-8'?>

The following XML document uses all the constructs and concepts.

```
<?xml version="1.0" encoding='UTF-8'?>
                <painting>
  <img src="India.jpg" alt=' Unity in
                Diversity'/>
      <caption>This is painted in
           <date>1511</date>
           <date>1512</date>
              </caption>
             </painting>
```

**Figure 1** XML Program

There are four elements in this example document. They are painting, img, caption, and date. date is a child of caption, which is a child of painting. img has two attributes, src and alt.

### C) Well-formedness and error-handling

The XML specification defines an XML document as a text which is well-formed, i.e., it satisfies a list of syntax rules provided in the specification.

- Every start – tag must have a matching end tag.
- Elements may nest but may not overlap.
- There must be exactly on root element.
- Attribute value must be quoted.
- An element may not have two attribute with the same name.
- Comments and processing instructions may not appear inside the tags.
- None of the special syntax characters such as "<" and "&" appear except when performing their markup-delineation roles.

*D) Programming interfaces*

A variety of APIs[5] for accessing XML have been developed and used, and some have been standardized.

Existing APIs for XML processing tend to fall into these categories:

- Stream-oriented APIs accessible from a programming language, for example SAX and StAX.
- Tree-traversal APIs accessible from a programming language for example DOM and XOM.
- XML data binding which provides an automated translation between an XML document and programming-language objects.
- Declarative transformation languages such as XSLT and XQuery.

Stream-oriented facilities require less memory for certain tasks which are based on a linear traversal of an XML document.


ISSN : 0975-3397

They are faster and simpler than other alternatives. Tree-traversal and data-binding APIs require more memory. They are often found more convenient for use by programmers. Because APIs include declarative retrieval of document components via the use of XPath expressions.

**E) XML on the Web**

XML began as an effort to bring the full power and structure of SGML to the Web in a form that was simple enough for nonexperts to use. Like most great inventions, XML [4] turned out to have uses far beyond what its creators originally envisioned. XML is still a very attractive language in which to write and serve web pages. Since XML documents must be well-formed and parsers must reject malformed documents, XML pages are less likely to have annoying cross-browser incompatibilities. Since XML documents are highly structured, they're much easier for robots to parse. Since XML element and attribute names reflect the nature of the content they hold, search-engine robots can more easily determine the true meaning of a page.

XML on the Web comes in three flavors[7]. The first is XHTML, an XMLized variant of HTML 4.0 that tightens up HTML to match XML's syntax. For instance, XHTML requires that all start-tags correspond to a matching end-tag and that all attribute values be quoted. XHTML also adds a few bits of syntax to HTML, such as the XML declaration and empty-element tags that end with />. Most of XHTML can be displayed quite well in legacy browsers, with a few notable exceptions.

The second flavor of XML on the Web is direct display of XML documents that use arbitrary vocabularies in web browsers. Generally, the formatting of the document is supplied either by a CSS stylesheet or by an XSLT stylesheet that transforms the document into HTML.This flavor requires an XML-aware browser and is not supported by older web browsers such as Netscape 4.0.

A third option is to mix raw XML vocabularies, such as MathML and SVG, with XHTML using Modular XHTML. Modular XHTML lets you embed RDF cataloging information, MathML equations, SVG pictures, and more inside your XHTML documents. Namespaces sort out which elements belong to which applications.

**F) XQUERY**

**XQuery** is a query and functional programming language that is designed to query collections of XML data.XQuery 1.0 [7] was developed by the XML Query working group of the W3C. The work was closely coordinated with the development of XSLT 2.0 by the XSL Working Group, the two groups shared responsibility for XPath 2.0, which is a subset of XQuery 1.0. XQuery 1.0 became a W3C Recommendation on January 23, 2007.

The mission of the XML Query[7] project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web. Therefore it finally provides the interaction between the Web world and the database world. Ultimately collections of XML files will be accessed like databases.

**G ) Features**

XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents. XQuery uses XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR[7] expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN.

The language also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used in other cases, expressions referred to as dynamic node constructors are available. All these constructs are defined as expressions within the language, and can be arbitrarily nested.

The language is based on a tree-structured model of the information content of an XML document, containing seven kinds of node such as document nodes, element node, attribute node, text node, comments, processing instructions and namespaces.

The type system of the language models all values as sequences. The items in a sequence can either be nodes or atomic values. Atomic values may be integers, strings, booleans, and so on. the full list of types is based on the primitive types defined in XML Schema.

Uses of Xquery are as follows :

1. Extracting information from a database [1] for a use in web service.
2. Generating summary reports on [3] data stored in an XML database.
3. Searching textual documents on the Web for relevant information and compiling the results.
4. Selecting and transforming XML data to XHTML to be published on the Web.
5. Pulling data from databases to be used for the application integration.
6. Splitting up an XML document that represents multiple transactions into multiple XML documents.

*A. H) Examples in XML-QL*

The simplest XML-QL queries extract data from an XML document. The example XML input is in the document www.a.b.c/bib.xml, and assume that it contains bibliography entries that conform to the following DTD:

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author+, title, year?, (shortversion|longversion))>
<!ATTLIST article type CDATA>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname)>
```

**Figure 2** DTD

This DTD [6] specifies that a book element contains one or more author elements, one title, and one publisher element and has a year attribute. An article is similar, but its year element is optional, it omits the publisher, and it contains one shortversion or longversion element. An article also contains a type attribute. A publisher contains name and address elements and an author


ISSN : 0975-3397

contains an optional firstname and one required lastname. The name, address, firstname and lastname are all CDATA, i.e., string values.

**I) Matching Data Using Patterns.**

XML-QL uses *element patterns[7]* to match data in an XML document. This example produces all authors of books whose publisher is Mc Graw-Hill in the XML document www.a.b.c/bib.xml. Any URI that represents an XML-data source may appear on the right-hand side of IN.

```
WHERE <book>
<publisher><name>Mc-Graw-Hill
</name></publisher>
<title>  $t </title>
<author> $a </author>
 </book> IN "www.a.b.c/bib.xml"
       CONSTRUCT $a
```

**Figure 3** Element Patterns - I

Informally, this query matches every <book> element in the XML document www.a.b.c/bib.xml that has at least one <title> element one <author> element, and one <publisher> element whose <name> element is equal to Mc Graw-Hill. For each such match, it binds the variables $t and $a to *every* title and author pair. The variable names are preceded by $ to distinguish them from string literals in the XML document (like Mc-Graw-Hill).

For convenience, </element> can be abbreviate by </>. This abbreviation is a relaxation of the XML syntax. Thus the query above can be rewritten as:

```
WHERE <book>
<publisher><name> Mc-Graw-Hill </></>
<title>  $t </>
<author> $a </>
</> IN "www.a.b.c/bib.xml"
CONSTRUCT $a
```

**Figure 3.1** Element Patterns - II

**J)Constructing XML Data.**

The query above produces an XML document that contains a list of (<firstname>,<lastname> ) [5] pairs or (<lastname> ) elements from the input document. Often it is useful to construct new XML data in the result. The following query returns both <author> and <title> and groups them in a new <result> element:

```
WHERE <book>
<publisher> <name>Addison-Wesley </> </>
<title>  $t </>
<author> $a </>
</> IN "www.a.b.c/bib.xml"
CONSTRUCT <result>
<author> $a </>
<title>  $t </>
</>
```

**Figure 3.2** Element Patterns - III

```
<bib><book year="2006">
<!-- A good introductory text -->
<title> Database System Concepts </title>
<author>
<lastname>Silberschatz   Korth   Sudharsan
</lastname>
</author>
<publisher>
<name>Mc-Graw-Hill</name>
</publisher>
</book>
<book year="2003">
<title>Operating Systems </title>
<author>
<lastname>WilliamStallings </lastname>
</author>
<author>
<lastname>Williams</lastname></author>
<publisher>
<name>Mc-Graw-Hill </name >
</publisher>
</book></bib>
```

**Figure 4** Bibliographic Data

When applied to the example data in Figure 4. the example query would produce the following result:

```
                <result>
   <author><lastname>SilberschatzKorth
          sudharsan</lastname>
                </author>
 <title>Database System Concepts </title>
                </result>
                 <result>
    <author><lastname>WilliamStallings
          </lastname></author>
     <title>Operating Systems</title>
                </result>
                 <result>
        <author><lastname>Williams
           </lastname></author>
  <title>Foundation for Object/Relational
            Databases</title>
                </result>
```

**Figure 5** Result of Bibliographic Data

## IV. SYSTEM STRUCTURE

The data warehouse creation [1] module can be regarded as user of the heterogeneous database system. When a user poses a global query on the integrated system, the global site decomposes the global query into sub-queries to request each


ISSN : 0975-3397

participant to return the data in XML format[3]. This can be easily achieved in contemporary commercial database products (e.g., MS SQL Server 2000, IBM DB2 Extender and Oracle). Besides, the DBA in each site should prepare some XSLT[2] format files, together with some necessary template files, in advance to transform local data into the global schema format. Finally, the transformed data are integrated into a consolidated view with the global query applied on it to return global data to the user. The integration process utilizes the semantic knowledge of all participate local schemas. This should be prepared or discovered before integration.

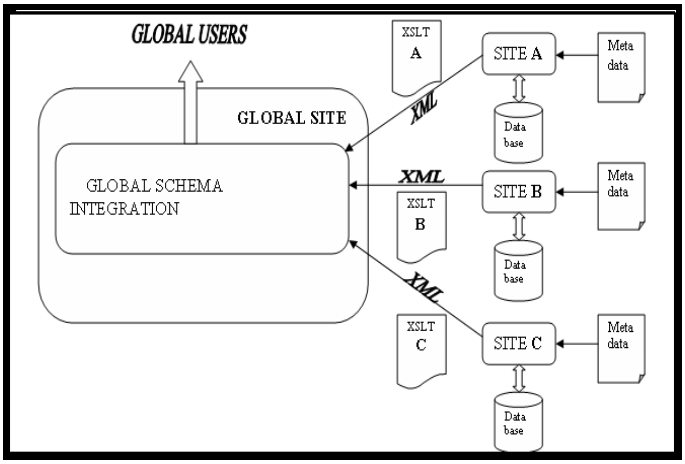


**Figure 6** Heterogenous Database Architectures

**A) Two Databases With Different Semantic Conflicts**

There are two databases containing semantically related information about books but in different formats. Sites X and Y contain tables named products and productslist respectively. There are some semantic discrepancies between these sites. They are listed as follows:

**1.** ***There are two attribute-to-attribute conflicts:*** The attributes products.pno and Products.name in Site X are respectively named productlist.pid and productlist.productname in Site Y.

**2.** ***There are three value-to-value conflicts:*** The products.cost stores the list price of every product, but the productlist.cost stores the discounted price (with 40% off). Besides, The productlist.location stores more detailed data than products.location.

**3.** ***There is a table-to-table conflict:*** The products.component is missing in the relation productlist. Besides, the productlist.manufacturing_year is also missing in the relation products.

**Products (Site X)**

| pno | name | company | cost | dealer | location | components |
|---|---|---|---|---|---|---|
| 1 | MotherBoard | Heiss | 2000 | IBM Company | India | 100 |
| 2 | Micro Controllers | Joe | 2500 | Microns Company | USA | 65 |

**Productslist(Site Y)**

| pid | productname | company | cost | dealer | location | Manufacture_year |
|---|---|---|---|---|---|---|
| 1 | MotherBoard | Heiss | 2000 | IBM | Coimbatore,India | 2000 |
| 2 | Micro Controllers | Joe | 2500 | Microns | Cleveland,USA | 2009 |

**Figure 7** Two Sites with Conflicts of Similar Schema Structures.

The two tables products and prductlists can be integrated as ProductsData(pno, name, company, cost, dealer, location, components, manufacture_year). When the user wishes to show ProductsData.cost by the original list price, ProductsData.dealer by the full names, and ProductsData.location by the concise names two sets of XSLT and template files are used to transform both tables into ProductData, respectively. Figure 8 and Figure 9 show the XSLT and template files for Site X. For Site Y, the XSLT and template files are shown in Figure 10 and Figure 11 respectively. Finally, both tables can be outer-joined into ProductData as Figure 12 illustrates.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Tran
sform" version="1.0">
<xsl:template match = '*'>
<xsl:apply-templates />
</xsl:template>
<xsl:template match = 'products'>
<TR><TD><xsl:value-of select = '@pno'
/></TD>
<TD><B><xsl:value-of select = '@name'
/></B></TD>
<TD><B><xsl:value-of select = '@company'
/></B></TD>
<TD><B><xsl:value-of select = '@cost'
/></B></TD>
<TD><B><xsl:value-of select = '@dealer'
/></B></TD>
<TD><B><xsl:value-of select = '@location'
/></B></TD>
```


ISSN : 0975-3397

```
<TD><B><xsl:value-of select =
'@components' /></B></TD>
</TR>
</xsl:template>
<xsl:template match = '/'>
<HTML>
<HEAD>
<STYLE>th { background-color: #CCCCCC
}</STYLE>
</HEAD>
<BODY>
<TABLE border='1' style='width:600;'>
<TR><TH colspan='7'> ProductsData
</TH></TR>
<TR><TH>pno</TH><TH>Name</TH><TH>compnay</
TH><TH>cost</TH>
<TH>dealer</TH><TH>location</TH><TH>compon
ents</TH></TR>
<xsl:apply-templates select = 'ROOT' />
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

**Figure 8** The XSLT for Site X - Products.xsl

```
<ROOT xmlns:sql="urn:schemas-microsoft-
com:xml-sql" sql:xsl="Products.xsl">
<sql:query>
SELECT pid, name, company, cost, dealer,
location, components FROM products
FOR XML AUTO
</sql:query>
</ROOT>
```

**Figure 9** The Template file for Site X - Products.xml

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Tran
sform" version="1.0">
<xsl:template match = '*'>
<xsl:apply-templates />
</xsl:template>
<xsl:template match = 'productlist'>
<TR>
<TD><xsl:value-of select = '@bid' /></TD>
<TD><B><xsl:value-of select =
'@productname' /></B></TD>
<TD><B><xsl:value-of select = '@company'
/></B></TD>
<TD><B><xsl:value-of select = '@cost'
/></B></TD>
<TD><B><xsl:value-of select = '@dealer'
/></B></TD>
<TD><B><xsl:value-of select = '@location'
/></B></TD>
```

```
<TD><B><xsl:value-of select =
'@manufactureyear' /></B></TD>
</TR>
</xsl:template>
<xsl:template match = '/'>
<HTML>
<HEAD>
<STYLE>th { background-color: #CCCCCC
}</STYLE>
</HEAD>
<BODY>
<TABLE border='1' style='width:600;'>
<TR><TH colspan='7'>Productdata</TH></TR>
<TR><TH>pno</TH><TH>name</TH><TH>company</
TH><TH>cost</TH>
<TH>dealer</TH><TH>location</TH><TH>manufa
ctureyear</TH></TR>
<xsl:apply-templates select = 'ROOT' />
</TABLE></BODY></HTML>
</xsl:template>
</xsl:stylesheet>
```

**Figure 10** The XSLT for Site Y - productlist.xsl

```
<ROOT xmlns:sql="urn:schemas-microsoft-
com:xml-sql" sql:xsl="productlist.xsl">
<sql:query>
SELECT pid, productname, dealer, cost/0.4
as cost,
rtrim(publisher) + ' Company' as dealer,
right(rtrim(location), 2) as location,
manuafactureyear
FROM productlist
FOR XML AUTO
</sql:query>
</ROOT>
```

**Fig 11** The Template file for Site Y. productlist.xml

| pid | productname | company | cost | dealer | location | components | manufactureyear |
|---|---|---|---|---|---|---|---|
| 1 | MotherBoard | Heiss | 2000 | IBM Company | India | 100 | 2000 |
| 2 | Micro Controllers | Joe | 2500 | Microns Company | USA | 65 | 2009 |

**Fig 12** Integrated Relation ProductData in the Global site.

## V. INTEGRATING DATA FROM MULTIPLE XML SOURCES

In XML-QL, several sources may be queried simultaneously and produce an integrated view of their data. In this example, we


ISSN : 0975-3397

produce all pairs of names and social-security numbers by querying the sources www.a.b.c/data.xml and www.irs.gov/taxpayers.xml[7]. The two sources are joined on the social-security number, which is bound to SSN in both expressions. The result contains only those elements that have both a name element in the first source and an income element in the second source.

```
WHERE <person>
<name></> ELEMENT_AS $n
<ssn> $ssn </>
</> IN "www.a.b.c/data.xml",
<taxpayer>
<ssn> $ssn </>
<income></> ELEMENT_AS $i
</>IN "www.irs.gov/taxpayers.xml"
CONSTRUCT <result> <ssn> $ssn </>$n $i </>
```

**Figure 13** Integrating data from multiple XML sources

Alternatively, Skolem functions can be used and a related, but not equivalent, query is written in two fragments:

```
{ WHERE <person>
<name> </> ELEMENT_AS $n
<ssn> $ssn </>
</> IN "www.a.b.c/data.xml"
CONSTRUCT <result ID=SSNID($ssn)> <ssn>
$ssn </> $n </>}
{ WHERE <taxpayer>
<ssn> $ssn </>
<income> </> ELEMENT_AS $i
</> IN "www.irs.gov/taxpayers.xml"
CONSTRUCT <result ID=SSNID($ssn)> ssn>
$ssn </> $i </>}
```

**Figure 14** Fragmented Queries

This query contains, in addition to the previous query, all persons that are not taxpayers, and vice versa. This is the *outer join* of the entities "www.a.b.c/data.xml" and "www.irs.gov/taxpayers.xml". XML-QL queries are structured into *blocks* and *sub-blocks*: the latter are enclosed in braces ({... }), and their semantics is related to that of outer joins in relational databases. In this example, the root block is empty and there are two sub blocks[7]. In the first sub block, all persons' names are produced. Each result has a unique OID given by that person's SSN. In the second sub block, persons and their incomes are produced.

Wherever there is a match with a previously generated OID, the <income> will be appended to the object rather than included in a new <person>. Query blocks are powerful. The following query retrieves all titles published in 2007 from the bibliography database, and it also retrieves the publication month for journal articles and the publisher for books.

```
WHERE <$e> <title> $t </><year> 2007 </>
</> CONTENT_A $p IN "www.a.b.c/bib.xml"
CONSTRUCT <result ID=ResultID($p)> <title>
$t </> </>
{WHERE $e = "journal-paper", <month> $m
</> IN $p
CONSTRUCT <result ID=ResultID($p)>
<month> $m </> </>
}
{
WHERE $e = "book",
<publisher>$q </> IN $p
CONSTRUCT
<result ID=ResultID($p)> <publisher>$q
</> </>
}
```

**Figure 15** Output from Bibliography database

The outer block runs over all publications in 2007, and produces a result element with their title. The first sub block checks if ***e*** is journal-paper and has a month tag. If so, it adds the month element to the *same* publication. The second sub block checks if e is a book and has a publisher, and adds that publisher element

## VI. CONCLUSION

XML – based heterogonous data integration and query has been a hot technology nowadays. Through the Intranet and XML Technology, transforming the information system of enterprise inside, we can realize integrating, sharing and utilizing of the hetrerogenous data source differently and effectively, which has improved the whole efficiency of the Information System.The mapping from heterogenous database integration schemas to XML documents can be prepared according the proposed procedures.

## REFERENCES


[1] Frank S.C. Tseng, Heterogeneous database Integration Using XML.

[2] Frank S.C. XML based heterogeneous Database Integration for Data warehouse Creation.

[3] Wei-Jung Shiang, Minng-Ying Ho, “An Interactive Tool Based on XML Technology for Data Exchange between Heterogeneous ERP systems”, Journal of CIIE, Volume 22, No.4, pp.273-278(2005).

[4] Yanxinwang, Kuiheyang “Research and Realization of XML Based Heterogeneous Databases Integration”

[5] www.xml.com/pub/a

[6] Giovanni Guardalven “ Integrating XML and Relational Database Technologies” Dec., 2004.

[7] *www.w3schools.com/xquery*




ISSN : 0975-3397

**Author's Biography**

Ms. V. Rajeswari is Assistant Professor in the department of Information Technology, Karpagam College of Engineering, Coimbatore. Her areas of interest are J2EE and Database Management Systems.

Dr. Dharmishtan K. Varughese is Professor in the department of Electronics and Communication Engg, Karpagam College of Engineering, Coimbatore. His areas of interest are Database Management Systems & Antenna and wave propagation.



ISSN : 0975-3397