International Journal of Multimedia and Ubiquitous Engineering
Vol.12, No.1 (2017), pp.235-244
http://dx.doi.org/10.14257/ijmue.2017.12.1.22

Mapping the Semi-Structured Data to the Structured Data for Inverted Index Compression[i]

B. Usharani

Asst. Professor, Department of CSE
INDIA

Abstract

Semi-structured data is used for representing the data over the Internet. In this paper, an implementation is given for, how to convert XML documents to SQL tables, processing the relational data to get the desired result using SQL queries, and stores the results back to XML and finally displaying the XML data in the web page.

Keywords*: Relational, Semi-Structured data, structured data, XML etc*

1. Introduction

1.1. Semi-Structured Data

"Semi-structured data" refers to the data that has the structure. There is no strict formatting rule for the semi structured data. Semi-Structured is a structured data, but it is not organized in a rational model, like a table.

XML" is an example of a language for representing the semi-structured data.

The XML standard was first published by the W3C in Feb 1998.

Eg.xml file for movies.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<information>
<tuple>
<mid>m1 </mid>
<moviename> toy story (1995)</moviename>
<generes>comedy drama </generes>
<url> http://us.imdb.com/M/title-exact?Toy%20Story%20</url>
</tuple>
…….
</information>
```

In the above example, mid, movie name, generes, url contain the text whereas tuple and information contain the other elements.

1.2. Structured Data

Structured data means there is a structure/format for the data and the format should be the well-defined one. Structured data can be modeled, organized, formed and formatted, easily manipulated and managed. The structured data should have a predefined length. All the data should follow the same order. All the data should be present. In relational databases, information is represented in terms of mathematical relations between items and their properties. A relation is shown in the form of table with rows and columns. Each column in the table represents a property and each row represents item. A cell is the interaction of the row and the column. A cell contains the value of the property for a given item. The structured data is stored in RDBMS. The example is given below

ISSN: 1975-0080 IJMUE

Table 1. Student Information Example

StudentNo	StudentName	Attendance%	Grade	FeeDues
1	usha	98	A	No
2	rani	99	A	No
3	siri	75	C	Yes
4	laxmi	89	B	No

The examples for structured-data are databases, spreadsheets, log-files *etc.*
Structured data is relatively easy to work with.
The following four classes are used for data retrieval in structured data.

- Retrieve by Key
- Retrieve by WHERE clause or its equivalent
- Need Join Queries
- Retrieve by offline queries

1.3. Database

To achieve the goal of this paper, the database should be in the semi-structured form. Here, the XML is taken as the semi-structured data. I have given the paper about XML and its advantages in [1]. In [2], I have given a clear comparison between the structured data and the semi-structured data, the advantages and disadvantages of each data and also discussed in that paper, in which cases the XML data is beneficial.

Users.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<information>
   <tuple>
      <uid>u1 </uid>
      <uname>john</uname>
   </tuple>
   <tuple>
      <uid>u2 </uid>
      <uname>mike </uname>
   </tuple>
……
</information>
```

Movie.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<information>
   <tuple>
      <mid>m1 </mid>
      <moviename> toy story (1995)</moviename>
      <generes> animation childres comedy  </generes>
   </tuple>
    <tuple>
      <mid>m2 </mid>
      <moviename>jumanji (1995) </moviename>
      <generes>adventure childrens fantasy </generes>
   </tuple>
……
……..
```



```
</information>
```

Rating.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<information>
      <tuple>
      <uid>u1 </uid>
      <mid>m1 </mid>
   </tuple>
   <tuple>
      <uid>u1 </uid>
      <mid>m3 </mid>
   </tuple>
   <tuple>
      <uid>u2 </uid>
      <mid>m4 </mid>
   </tuple>
…….
</information>
```

1.4. Index

An index is an alphabetically arranged list of words. Search engine index size is hundreds of millions of web pages. Search engines have to answer tens of millions of queries every day. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours. The additional computer storage required to store the index, as well as the considerable increase in the time required for an update to take place, are traded off for the time saved during information retrieval [6].

The first Google Index in 1998 already has 26 million pages, and by 2000 the Google index reached the one billion mark [7].

1.5. Inverted *Index*

An inverted index (postings file or inverted file) is a data structure used for document retrieval. Inverted index have two variants:1) **record level inverted index** (inverted file index or inverted file) – tells you which document contains that particular word.2)**word level inverted index** (fully inverted index or inverted list)—tells you both which document contains that particular word and the place(position) of the word in that particular document. This can be represented as (document-id**;** no: of occurrences,[position]). [8] For example

D_0– {this is to test a test}
Positions: 0 1 2 3 4 5
D_1—{this is testing to test application}
D_2—{this is a testing application}
The words would store in the inverted index like this:

Table 2. Inverted Index table

words	Inverted file index	inverted list
a	{0,2}	{(D_0;1,[4])(D_2;1,[2])}
application	{1,2}	{(D_1;1,[5])(D_2;1,[4])}
is	{0,1,2}	{(D_0;1,[1])(D_1;1,[1])(D_2;1,[1])}
test	{0,1}	{(D_0;2,[3,5])(D_1;1,[4])}
testing	{1,2}	{(D_1;1,[2])(D_2;1,[3])}
this	{0,1,2}	{(D_0;1,[0])(D_1;1,[0])(D_2;1,[0])}
to	{0,1}	{(D_0;1,[2])(D_1;1,[3])}

2. Implementation

The discussion and implementation given in [3-5] are used in this implementation section to show the final results.

The mapping from the semi-structured data to the structured data is given as programs.

First, store the XML data *i.e*. the whole XML database (given is above section) of the semi-structured data as the relational form in the relational database. Here MYSQL is taken as the relational database.

The program to convert all XML data to the MYSQL data is given below:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page language="java" %>
<%@ page import="org.w3c.dom.*" %>
<%@ page import="javax.xml.parsers.DocumentBuilder" %>
<%@ page import="javax.xml.parsers.DocumentBuilderFactory" %>
<%@ page import="java.sql.*"%>
<%@ page import="java.util.*"%>
<%@ page import ="javax.sql.*" %>
<%!
Connection con;
Statement st;
String user = "root";
String password = "mysql";
%>
<%
   DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
   DocumentBuilder db =dbf.newDocumentBuilder();
   Document doc=db.parse("D:\\compression\\xml\\web\\movies1.xml");
    NodeList nl = doc.getElementsByTagName("tuple");
  %>
<html>
<body>
   <%
   try{
      String url = "jdbc:mysql://localhost:3306/mydb";
       Class.forName("com.mysql.jdbc.Driver");
```



```
            con = DriverManager.getConnection(url, user, password);
             st = con.createStatement();
             st.executeUpdate("delete from movies1;");
          }catch (Exception ex) {}
    for(int i=0;i<nl.getLength();i++)
    {
     NodeList nameNlc=    doc.getElementsByTagName("mid");
     Element nameElements=(Element)nameNlc.item(i);
     String nameTagValue=nameElements.getChildNodes().item(0).getNodeValue();
     NodeList authorNlc=    doc.getElementsByTagName("moviename");
     Element authorElements=(Element)authorNlc.item(i);
     String authorTagValue=authorElements.getChildNodes().item(0).getNodeValue();
      NodeList dateNlc=    doc.getElementsByTagName("generes");
    Element dateElements=(Element)dateNlc.item(i);
     String dateTagValue=dateElements.getChildNodes().item(0).getNodeValue();
     try
     {
      String insert = "INSERT INTO movies1(mid,moviename,generes)" +"VALUES (?,
?,?)";
       PreparedStatement ps = con.prepareStatement(insert);
          ps.setString(1, nameTagValue);
           ps.setString(2, authorTagValue);
           ps.setString(3, dateTagValue);
          ps.executeUpdate();
    }catch (Exception ex) {}
   }
   %>
   <%
   response.sendRedirect("readrating.jsp");
   %>
   </body>
   </html>
```

To store the XML data as a relational form, first there is to need to create the tables in MySQL, and the table data type and column names should match the corresponding XML file. After transforming the XML data to MYSQL data, all the XML data is stored in MYSQL as the table form. Part of the tables is given below:

Table 3. Movie Table

mid	**movie name**	**generes**
m1	toy story (1995)	animation childres comedy
m2	jumanji (1995)	adventure childrens fantasy
m3	grumpier old men (1995)	comedy romance
m4	waiting to exhale (1995)	comedy drama
m5	father of the bride partII(1995)	comedy
m6	heat (1995)	action crime thriller
m7	sabrina (1995)	comedy romance

Table 4. User Table

uid	Uname
u1	John
u2	Mike
u3	Peter
u4	Jimmy
u5	David
u6	Richard

Table 5. Rating Table

uid	mid
u1	m1
u1	m3
u2	m4
u3	m2
u3	m3
u4	m5
u4	m7
u5	m6
u6	m2

Now the data is processed from the MYSQL table's .Run the SQL queries on the SQL tables and store the answers back from the SQL queries in the text file(only for temporary purpose).

The Program for running the SQL queries and store the results back in the text file (for temporary purpose) is given below(only some part):

```
<%@ page language="java" %>
<%@ page import="java.sql.*"%>
<%@ page import="java.util.*"%>
<%@ page import ="javax.sql.*" %>
<%@ page import ="java.io.*" %>
    ----------------------
 Scanner  sc=new Scanner(fis);
      con = DriverManager.getConnection(url, user, password);
          st = con.createStatement();
        rs = st.executeQuery("SELECT COUNT(*)  FROM users");
        rs.next();
        int rowCount = rs.getInt(1);
          out.println();
        while(i<=rowCount)
        {
            con = DriverManager.getConnection(url, user, password);
            pst = con.prepareStatement("select mid from rating where uid=?");
            t=t+i;
            pst.setString(1,t);
            rs= pst.executeQuery();
            v=v+i;
            vd[j][k++]=t;
             while (rs.next())
            {
               count++;
```



```
                vd[j][k++]=rs.getString(1);
              }
              k1=k;
              while(a<k1)
              {
                pst1= con.prepareStatement("select uid from rating where mid = ? ");
                value=vd[j][a];
                pst1.setString(1,value);
                rs1= pst1.executeQuery();
                a++;
                while (rs1.next())
                {
                   value1=rs1.getString(1);
                  if(value1.compareTo(t)==0){}
                  else
                  {
                       if(a>2)
                       System.arraycopy(vd[j], 0, vd[++j], 0, k1);
                       vd[j][k++]=rs1.getString(1);

                   }
                }
                count=0;
              }
              j++;
              i++;
              t="u";
              v="vd";
              k=0;
              a=1;
              count=0;
            }
           rs.close();
           rs1.close();
```

Again map the structured data to the semi-structured data *i.e*. restoring the results back to the XML file. For storing the text data as the XML data, the program is given below:

```
<%@ page language="java" %>
<%@ page import="java.sql.*"%>
<%@ page import="java.util.*"%>
<%@ page import ="javax.sql.*" %>
<%@ page import ="java.io.*" %>
<%
    try
      {
     FileReader fr2 = new FileReader("D:\\compression\\xml\\src\\java\\text\\invertedindex.txt");
       BufferedReader br2 = new BufferedReader(fr2);
      FileWriter fw2=new FileWriter("D:\\compression\\xml\\web\\index.xml");
      String line;
      String tt="<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>";
    fw2.write(tt);
        fw2.write("\n");
       fw2.write("<information>");
```

```
 fw2.write("\n");
 while((line = br2.readLine()) != null)
   {
      line = line.trim();
      if (!line.equals(""))
     {
        fw2.write("<documet>");
        fw2.write("\n");
        fw2.write("<term>");
        fw2.write(st.nextToken());
         fw2.write("</term>");
        fw2.write("\n");
         while(st.hasMoreTokens())
        {
           fw2.write("<docid>");
            fw2.write(st.nextToken());
           fw2.write(st.nextToken());
           fw2.write(st.nextToken());
           fw2.write("</docid>");
           fw2.write("\n");
        }
         fw2.write(line, 0, line.length());
        fw2.write("</term>");
        fw2.write("\n");
        fw2.write("</documet>");
        fw2.write("\n");
     }
      }
  fw2.write("</information>");
    fw2.close();
  }
   catch (IOException ioex) {
  ioex.printStackTrace();
}
   catch (Exception ex1)
   {
        ex1.printStackTrace();
   }
 finally  {     }
 %>
 <%
 response.sendRedirect("homepage.jsp");
 %>
```

After executing the above program the data is stored in the XML file. Part of the XML file is given below:

Invertedindex.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<information>
<documet>
<term>(1995) (d1 2 ) (d2 2 ) (d3 2 ) (d4 1 ) (d5 1 ) (d6 2 )</term>
</documet>
<documet>
```



```
<term>action (d5 1 )</term>
</documet>
<documet>
<term>adventure (d1 1 ) (d3 1 )</term>
</documet>
……
</information>
```

The program to display the above given XML file in the web page is given below:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<HTML>
<HEAD>
      <STYLE>
            table,tr,td {border: 1px solid blue;}
      </style>
</HEAD>
<BODY>
<c:import var="xmlDoc" url="invertedindex.xml"/>
            <x:parse var="parsedDocument" xml="${xmlDoc}"/>
      <table>
            <tr>
                  <th>TERMS</th>
            </tr>
            <x:forEach select="$parsedDocument/information/documet">
            <tr>  <td> <x:out select="term" /> </td> </tr>
            </x:forEach>
      </table>
</body>
</html>
```

After executing the above program the output is:

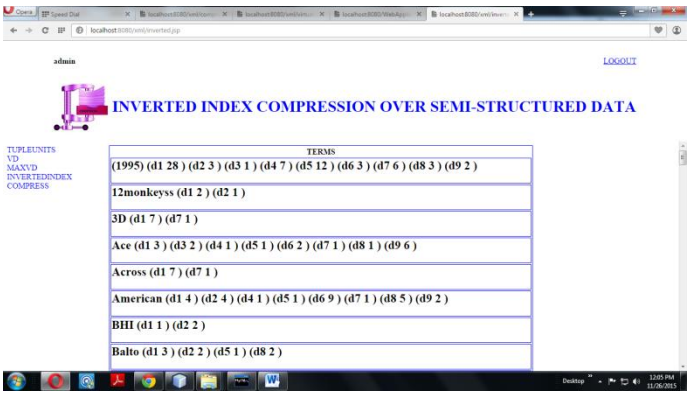

Figure 1. Displaying the XML File in Browser

3. Conclusion

This paper presents a view for storing the information of the XML document in structured databases, processing the structured data and storing the results back in the XML document. In future, different mapping schemes are going to be implemented.

References

[1] B. Usharani, "Survey on Inverted Index Compression using Semi-structured Data", By IJIRS, P. No 97-104.
[2] B.Usharani "Comparison between the structured data and the semi-structured data for Inverted Index Compression" by IJRCSE ,ISSN: 2321-0338(P.No:7-11).
[3] B.Usharani,"Survey on Inverted Index Compression using Structured Data" by IJARCS ,ISSN No. 0976-5697,P.No(57-61).
[4] B.Usharani," Inverted Index Compression using Semi-structured Data by AESSN – 2015, ISSN:2348-8387,P.No(119-124).
[5] Jaime I. Lopez-Veyna, Victor J. Sosa-Sosa , Ivan Lopez-Arevalo "A low redundancy strategy for keyword search in structured and semi-structured data" by Science Direct , p.no(135-142).
[6] http://en.wikipedia.org/wiki/Search_engine_indexing#The_forward_index
[7] http://googleblog.blogspot.in/2008/07/we-knew-web-was-big.html
[8] http://en.wikipedia.org/wiki/Inverted_index

[i] *Corresponding Author: B.Usharani

