http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (4): 521-528, 2009
ISSN 1812-5638

On-Line Analytical Processing Queries for eXtensible Mark-up Language

Mourad Ykhlef
Department of Information Systems, College of Computer and Information Sciences,
King Saud University, P.O. Box 51178, Riyadh 11543, Kingdom of Saudi Arabia

Abstract: eXtensible Mark-up Language (XML) is emerged as a standard to exchange data over the Web. A large amount of heterogeneous data is now available on the Web in different sources; these data are generally represented or published in XML format. A XML data warehouse is an integrated repository of different XML data sources enabling analysts to gain insight through fast access to a variety of possible views on XML data which are organized in a dimensional model. XML data warehouse can be queried by On-Line Analytical Processing (OLAP) queries. This study proposes XML data cube model which is a well-founded approach to represent OLAP data using XML. XML data cube enables the use of external XML data for selection and grouping. Recently a SQL-like query language XRQL is proposed to query XML data. The XRQL queries capture the flavour of SQL queries while offering constructs for navigation, XML data construction and grouping. This study shows how OLAP queries can be expressed by using XRQL in a natural way and extends XRQL by GROUP BY CUBE and GROUP BY ROLLUP operators to enable analysts to express more complex OLAP queries on XML data cube.

Key words: XML cube, OLAP, aggregate functions, GROUP BY CUBE, GROUP BY ROLLUP, nested queries

INTRODUCTION

On-Line Analytical Processing (OLAP) (Vassiliadis *et al*., 1999; Nathan *et al*., 2002) generally involves highly complex queries that use aggregations. The term OLAP was created as a slight modification of the traditional database term OLTP (Online Transaction Processing). The typical applications of OLAP are business reporting for sales and management reporting. The core of any OLAP system is a concept of an OLAP cube or a multidimensional cube. It consists of numeric facts called measures and dimensions. The cube is created from a star schema or snowflake schema of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

XML data (Buneman *et al*., 1998; Deutsch *et al*., 1998; Abiteboul *et al*., 1999) are data, which are not necessary constrained by a schema. XML data are different from relational data in several important points. Relational data have a fixed schema, which is stored in a separate table known as catalogue, in contrast XML files contain elements and/or attributes tags to describe data and schema is not a must. Relational data are not ordered, the order can be deduced from data values, however XML data are ordered. Relational data model represents missing information by NULL value. XML data, in contrast, can represent missing information simply by the absence of an element. XML data model has become an important standard of information representation and exchange on the Web (Berners-Lee *et al*., 1992).

With the increasing amount of data exchanged, stored or generated by XML, a natural question how to perform OLAP on XML data cube. XML cube is different than relational cube in two points: (1) data of facts and dimensions can be nested however only facts are flat in relational cube, (2) missing information in XML cube are expressed simply by the absence of corresponding element without use of NULL.

The contribution of this study is three fold. First, an XML cube model is presented. Second, we show how XRQL query language (Ykhlef, 2007) can express OLAP queries on XML data cube in a natural way. Third, we extend XRQL by GROUP BY CUBE and GROUP BY ROLLUP operators to express more complex analytical queries.

Many studies for expressing OLAP queries over XML data have been proposed so far, such as XQuery (Beyer *et al*., 2005), Gxaggregation (Wang *et al*., 2005) XML-OLAP (Park Byung *et al*., 2005) and many other studies (Jensen *et al*., 2001; Yin and Torben, 2006; Wiwatwattana *et al*., 2007). The importance of this study resides in the underlying formalism based on XRQL query language. In order to express group by queries, XRQL did not use any programming construct such as FOR used in XQuery language. XQuery tends to be a programming language rather than a query language.

XML CUBE MODELING

An XML cube is modeled by a labeled rooted graph called G-XML where:

```
<salesdoc>
<sales>
 <locid>1</locid>
 <prodid>1</prodid>
 <timeid>1</timeid>
 <quantity>10</quantity>
</sales>
<sales>
 <locid>1</locid>
 <prodid>1</prodid>
 <timeid>2</timeid>
 <quantity>20</quantity>
</sales>
<sales>
 <locid>2</locid>
 <prodid>1</prodid>
 <timeid>1</timeid>
 <quantity>30</quantity>
</sales>
<sales>
 <locid>2</locid>
 <prodid>2</prodid>
 <timeid>2</timeid>
 <quantity>40</quantity>
</sales>
</salesdoc>
```

```
<location>
 <locid>1</locid>
 <city>constantine</city>
 <country>dz</country>
</location>
<location>
 <locid>2</locid>
 <city>riyadh</city>
 <country>sa</country>
</location>

<product>
 <prodid>1</departure>
 <pname>green tea</pname>
 <category>Food</category>
</product>
<product>
 <prodid>2</departure>
 <pname>dates</pname>
 <category>Food</category>
</product>
```

```
<time>
 <timeid>1</timeid>
 <day>11</day>
 <month>6</month>
 <year>2007</year>
</time>
<time>
 <timeid>2</timeid>
 <day>12</day>
 <month>6</month>
 <year>2007</year>
</time>
```

Fig. 1: XML sales data cube

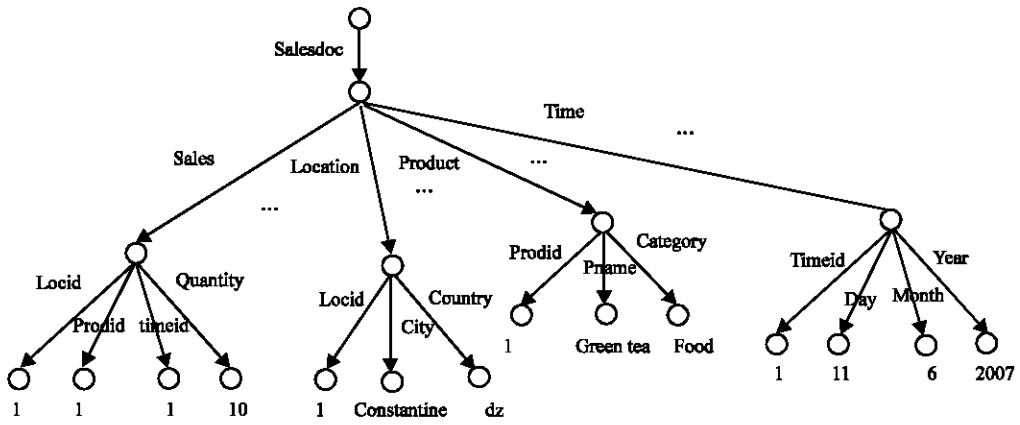

Fig. 2: A G-XML graph for XML sales data cube

- Each edge is labeled by an XML element or XML attribute
- Each leaf node is labeled with a value
- G-XML has a distinguished node called the root

For example let us consider the XML cube of Fig. 1 containing one fact sales with three dimensions: location, product, time and one measure quantity. The corresponding G-XML is given partially in Fig. 2.

QUERYING XML CUBE BY XRQL

The language XRQL is based on path expressions, XRQL is very similar to a multi-sorted first order language. Because in this data model, data are associated to leaves, XRQL path expressions may contain data variables as abstractions of the content of leaves. They may also contain label (resp. path) variables as abstraction of edges (resp. paths) in G-XML graph. The graph variables abstract sub-graphs in a G-XML graph.

Path expression: Here, we assume that four sorts of variable sets are given: a set of graph (resp. path, label and data) variables. Graph and path variables are denoted X, Y, Z, ... Label (resp. data) variables are denoted x, y, z, …

In order to define path expression one needs to define label term and data term. A label term is either an edge label or a label variable. A data term is either leaf value or data variable. A path is simple with respect to edges if all its edges are distinct. In the rest of presentation a path refers always to a simple path. An empty path ϵ is a path with no edges.

Definition 1 (Path expression): A path expression is a sequence of label terms and/or path variables. If s is a path expression and t is a data term then s@t is a path expression.

The path expression s@t means that any path captured by s must have a destination (end) equal to the value associated with the data term t.

Example 1: For example the path expression sales.quantity is intended to represent all paths of length 2 having as edge labels the label sales and then the label quantity. However, the path expression sales.x is intended to represent all paths of length 2 where the first edge label is sales; the label variable x is valuated to one label in the set of labels {locid, prodid, timeid, quantity}. The path expression X.category represents each path having at least one edge where the last edge is labeled by category.

Example 2: The data variable x in the path expression sales.quantity@x captures all leaves values at the destination of paths abstracted by the path expression sales.quantity.

Now some terms are defined. A path (resp. graph) term is either a (simple) path (resp. graph) or path (resp. graph) variable. Remember that a label term is either an edge label or a label variable. A data term is either leaf value or data variable. The origin (resp. destination) of a path is the first (resp. last) node. A node r of a graph G is a source for G if there exists a path from r to any other nodes of G.

Definition 2 (Atom): An atom is an expression having one of the following forms:

- a path expression
- t1 = t2, where, t1 and t2 are terms of the same sort among graph, path, label, data; = is the (polymorphic) equality predicate symbol
- t = s, where, t is a path term, s is a path expression and = is an equality predicate
- t:s, where, t is a graph term and s is a path expression
- s[t], where, t is a graph term and s is a path expression

Intuitively, the atom t = s intends to check whether t is one of the paths captured by the path expression s. The atom t:s tells that s captures at least one path p of the graph t and that the origin of p is the source of t; the intention of the atom s[t] is to check that the graph t has a source which is the destination of a path captured by the path expression s. For example, the atom S:sales.quantity intends to capture all paths p having successively sales and quantity as edge labels in the G-XML graph representing the S valuation; such that the origin of p is the source of S. However, the atom sales[S] captures all graphs S at the destination of paths captured by the path expression sales.

Query: The syntax of XRQL query is SQL-like, it has the form SELECT FROM WHERE; where the clause SELECT contains XML document fragments with variables, the valuation of SELECT yields a XML document. The clause FROM specifies a series of XML documents to be queried; it has the following form: FROM f_1.xml[AS D_1],...,f_n.xml[AS D_n], where, [AS D_i] is a renaming option, the graph variable D_i is valuated to the G-XML graph associated to a file f_i.xml. When there is only one document in a query; one can drop the option [AS D_i].

Example 3: To return the products, we write the following query:

```
SELECT <pname> x </pname> AS <products>
FROM sales.xml
WHERE salesdoc.product.pname@x
```

Intuitively this query selects all product names <pname> x </pname> and environs them by the element <products>. This query posing on the G-XML graph of Fig. 2 returns the following XML data:

```
<products>
    <pname>green tea</pname>
    <pname>dates</pname>
</products>
```

The same query can be expressed as below:

```
SELECT P AS <products>
FROM sales.xml
WHERE salesdoc.product.P and P = pname
```

Note that P is a path variable and the atom P = pname intends to check whether P is one of the paths captured by the path expression pname.

As we have seen the clause SELECT can contain the expression AS <elem>. The element elem environs the last collection of elements returning before AS. Note that in the expression P Q AS <elem>, where, P and Q are path and/or graph variables, <elem> environs only the collection of elements captured by Q.

Example 4: The below query returns all product names sold in Algeria (dz):

```
SELECT <pname> z </pname> AS <products>
FROM sales.xml
WHERE salesdoc.sales[S]
    and S:locid@x and S:prodid@y
    and salesdoc.location[L]
    and L:locid@x and L:country@"dz"
    and salesdoc.product[P]
    and P:prodid@y and P:pname@z
```

GROUPING AND AGGREGATE FUNCTIONS

Here, we present GROUP BY operator that will be used to formulate most of OLAP queries. The GROUP BY statement has the following form:

```
SELECT … var1 [AS <element1>]
          … varn [AS <elementn>] …
FROM …
WHERE …
GROUP BY var1,…,varn
```

where, variables $var_1,\ldots,var_n$ are either path or graph variables and the symbol [] indicates that element is optional. All variables in SELECT clause must appear in GROUP BY in the absence of aggregate functions. In the case of the presence of aggregate function like COUNT, MAX, MIN, AVG, SUM references in the SELECT clause can only made to aggregate functions and to the variables in the GROUP BY clause.

Example 5: For each location, return the list of items sold there:

```
SELECT <locproducts>
            L Pn AS <products>
        </locproducts> AS <result>
FROM sales.xml
WHERE salesdoc.location.L
    and L = locid and L@x
    and salesdoc.sales[S]
    and S:locid@x and S:prodid@y
    and salesdoc.product[P]
    and P:prodid@y and P:Pn and Pn = pname
GROUP BY L,Pn
```

Posing this query on the G-XML graph of Fig. 2 will return, the following XML data:

```
<result>
<locproducts>                <locproducts>
 <locid>1</locid>             <locid>2</locid>
 <products>                   <products>
  <pname>                      <pname>green tea
  green tea                    </pname>
  </pname>                       <pname>
  <pname>                        dates
  green tea                    </pname>
  </pname>                     </products
 </products                   </locproducts>
</locproducts>
</result>
```

Note that for locid = 1 the product green tea appears twice because for the same location, it was sold in two different times. In order to eliminate duplicate, one can introduce in SELECT clause, the word DISTINCT before the path variable Pn as we will do in the next example.

Example 6: This query show the use of GROUP BY with HAVING. It returns for each location its items if number of sold items is above one:

```
SELECT <locproducts>
            L DISTINCT Pn AS <products>
        </locproducts> AS <result>
FROM sales.xml
WHERE salesdoc.location.L
    and L = locid and L@x
    and salesdoc.sales[S]
    and S:locid@x and S:prodid@y
    and salesdoc.product[P]
    and P:prodid@y and P:Pn and Pn=pname
GROUP BY L,Pn
HAVING COUNT(Pn)>1
```

OLAP QUERIES

Here, we present typical OLAP queries include slice, dice, drill down, roll up and pivot. We will show how XRQL query language can express OLAP queries in a natural way.

Slice and dice: Slice query is a OLAP query putting constraints on only one dimension among location, product and time. However, dice query is a query putting constraints on at least two different dimensions. Slicing and dicing reduce the number of dimensions.

Example 7 (Slice): Return the sales in location with locid = 1.

```
SELECT X AS <result>
FROM sales.xml
WHERE salesdoc[X] and X:sales.locid@1
```

In order to impose more constraint on ordering, one can write the above query as follows:

```
SELECT <sales>
            L P T Q
        </sales> AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid@1
      and S:P and P = prodid
      and S:T and T = timeid
      and S:Q and Q = quantity
ORDER BY P, T
```

Note that in WHERE condition, we put constraint on only one dimension L = locid@1. The ORDER BY is used to return the XML data cube ordered according to prodid and timeid.

Example 8 (Dice): Return the sales of product with prodid = 1 sold in location with locid = 1.

```
SELECT <sales> L P T Q </sales>
      AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid@1
      and S:P and P = prodid@1
      and S:T and T = timeid
      and S:Q and Q = quantity
ORDER BY T
```

Note that in WHERE condition, we put constraints on two dimensions L = locid@1 and P = prodid@1. The ORDER BY is used to return the XML data cube ordered by timeid.

Roll up and drill down: The roll up queries summarize data by climbing up hierarchy, if we are given sales per day, we can aggregate data to obtain sales per month. However, drill down is the reverse of roll up, we go from higher level summary to lower level summary. Given total sales by month, we can ask for sales per day. In the following examples we will roll up data from day to month and then we drill down backward.

Example 9 (Roll up): Return the sales by months.

```
SELECT <monthsales>
            Y M L P
            <quantity> SUM(q) </quantity>
        </monthsales> AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid
      and S:P and P = prodid
      and S:T and T = timeid@x
      and S:quantity@q
      and salesdoc.time[Tm] and Tm:timeid@x
      and Tm:M and M = month
      and Tm:Y and Y = year
GROUP BY Y,M,L,P
```

Posing this query on the XML graph of Fig. 2 will return the following XML data:

```
<result>
<monthsales>
 <year>2007</year>
 <month>6<month>
 <locid>1</locid>
 <prodid>1</prodid>
 <quantity>30</quantity>
</monthsales>
<monthsales>
 <year>2007</year>
 <month>6<month>
 <locid>2</locid>
 <prodid>1</prodid>
 <quantity>30</quantity>
</monthsales>
<monthsales>
 <year>2007</year>
```

```
<month>6<month>
<locid>2</locid>
<prodid>2</prodid>
<quantity>40</quantity>
</monthsales>
</result>
```

Now suppose the result of roll up query is materialized in a separate file called roll2month.xml.

Example 10 (Drill down): The following query goes from month sales to day sales. Note that the four lines of WHERE condition, are similar to those found in roll up query. The fifth line is related to XML data of roll2month.xml file. However, the last line imposes the join conditions on month and year between documents D1 and D2.

```
SELECT <sales>
            L P T <quantity> q <quantity>
      </sales> AS <result>
FROM sales.xml AS D1,roll2month.xml AS D2
WHERE D1:salesdoc.sales[S]
      and S:L and L = locid
      and S:P and P = prodid
      and S:T and T = timeid@x
      and S:quantity@q
      and salesdoc.time[Tm] and Tm:timeid@x
      and Tm:M1 and M1 = month
      and Tm:Y1 and Y1 = year
      and D2:result.monthsales[MS]
      and MS:M2 and M2 = month
      and MS:Y2 and Y2 = year
      and M1@m and M2@m and Y1@ye and
      Y2@ye
GROUP BY L, P, T
```

Pivoting: Pivot query aims to reorient data cube, the result of pivoting is called in relational model a cross-tabulation. If we pivot the sales cube on the location and product dimensions, we obtain data shown in the left side of Fig. 3, note that pivoting can be combined with aggregate function SUM (right side of Fig. 3).

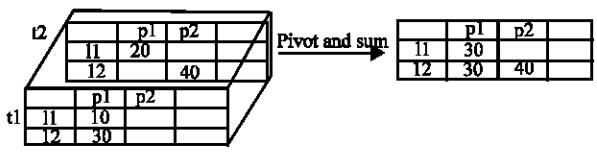

Fig. 3: Pivoting

Example 11 (Pivot): The query reorienting data cube on the location and product dimensions is written as follows:

```
SELECT <sales>
            T L P  <quantity> q </quantity>
      </sales> AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid
      and S:P and P = prodid
      and S:T and T = timeid
      and S:quantity@q
GROUP BY T, L, P
```

The pivoting can be combined with aggregate functions like SUM as follows:

```
SELECT <sales>
                  L P
            <quantity> SUM(q) </quantity>
      </sales> AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid
      and S:P and P = prodid
      and S:quantity@q
GROUP BY L, P
```

EXTENDED AGGREGATION AND NESTING

Extended aggregation: Here, we present GROUP BY CUBE operator, which computes union of GROUP BY's on every subset of the specified elements. The operator GROUP BY ROLLUP is also presented.

Example 12: Consider the query:

```
SELECT <sales>
            L P T
            <quantity> SUM(q) <quantity>
      </sales> AS <result>
FROM sales.xml
WHERE salesdoc.sales[S]
      and S:L and L = locid
      and S:P and P = prodid
      and S:T and T = timeid
      and S:quantity@q
GROUP BY CUBE L, P, T
```

This query computes the union of 8 different grouping ($2^{\text{dimensions}}$) of the sales: (L P T), (L P), (L T), (P T),

```
<salesdoc>
<sales>                    <sales>
 <locid>1</locid>           <locid>1</locid>          <sales>
 <prodid>1</prodid>         <prodid>1</prodid>        <locid>1</locid>
 <timeid>1</timeid>         <quantity>30</quantity>   <quantity>30</quantity>
<quantity>10</quantity>    </sales>                  </sales>
</sales>
<sales>                    <sales>
 <locid>1</locid>           <locid>2</locid>          <sales>
 <prodid>1</prodid>         <prodid>1</prodid>        <locid>2</locid>
 <timeid>2</timeid>         <quantity>30</quantity>   <quantity>70</quantity>
 <quantity>20</quantity>   </sales>                  </sales>
</sales>
<sales>                    <sales>
 <locid>2</locid>           <locid>2</locid>          <sales>
 <prodid>1</prodid>         <prodid>2</prodid>        <quantity>100</quantity>
 <timeid>1</timeid>         <quantity>40</quantity>   </sales>
 <quantity>30</quantity>   </sales>
</sales>
<sales>
 <locid>2</locid>
 <prodid>2</prodid>
 <timeid>2</timeid>
```

Fig. 4: GROUP BY ROLLUP result

(L), (P), (T), () where () denotes the empty GROUP BY list. In relational data cube, for each grouping, the result contains the null value for attribute not present in the grouping, however in XML data cube, we can represent missing information simply the absence of an element. For example the result of grouping (L P T) (resp. (L P)) is given in first (resp. second) column of Fig. 4. The grouping (L) yields the first two sales of the third column. However, the empty grouping (), representing the total sales in all locations of all products in all times, yields the last sales of the third column.

The GROUP BY ROLLUP operator can be directly added to our formalism; this operator generates the union on every prefix of specified list of variables. For example GROUP BY ROLLUP L,P,T generates the union of 4 different grouping of the sales: (L P T), (L P), (L), (). The result of GROUP BY ROLLUP L,P,T is included in Fig. 4.

Nesting for more complex OLAP queries. The result of a XRQL query is a XML document. It seems therefore natural to use the result of one or more queries, which we then call sub-queries, to formulate other queries. We will examine the use of sub-queries, also called nested queries, in the WHERE and HAVING clauses to formulate more complex OLAP queries.

Example 13: List locations with sold products above the average number of sold products in all locations.

```
SELECT AVG (x)
FROM (SELECT
            <locproducts>
               COUNT(P)
             </locproducts> AS <rt>
      FROM sales.xml
      WHERE salesdoc.sales[S]
               and S:L and L = locid
               and S:P and P = prodid
      GROUP BY L
HAVING COUNT(P) > AVGQueryResult
```

where, AVGQueryResult is defined as follows:

```
SELECT AVG(x)
FROM (SELECT
          <locproducts>
             COUNT(P)
           </locproducts> AS <rt>
      FROM sales.xml
      WHERE salesdoc.sales[Sl]
```

```
                and Sl:L and L = locid
                and Sl:P and P = prodid
        GROUP BY L)
    WHERE rt.locproducts@x
```

CONCLUSION

In this study, we have presented a data cube model for XML, we have shown how OLAP queries on XML data cube can be expressed by XRQL in a natural way to enable analysts to gain insight into data through fast access to a variety of possible views on data. An extension for XRQL concerning the use of the operators GROUP BY CUBE and GROUP BY ROLLUP is presented. We have also illustrated how more complex OLAP queries can be formulated by using query nesting.

REFERENCES

Abiteboul, S. and P. Buneman and D. Suciu, 1999. Data on the Web: From Relations to Semi-Structured Data and XML. 1st Edn., Morgan Kaufman Publishers Inc., San Francisco, CA., USA., ISBN: 155860622X.

Berners-Lee, T., R. Cailliau and J.F. Groff, 1992. The world-wide web. Comput. Networks ISDN Syst., 25: 454-459.

Beyer, K.S., D. Donald Chamberlin, S. Latha Colby, O. Fatma, P. Hamid and X. Yu, 2005. Extending XQuery for analytics. Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, Jun. 14-16, ACM New York, USA., pp: 503-514.

Buneman, P., D. Alin, F. Wenfei, L. Hartmut, S. Arnaud and T. Wang Chiew, 1998. Beyond XML query languages. Proceedings of the Query Language Workshop, (QL-98), National Science Foundation, pp: 1-20.

Deutsch, A., F. Mary Fernandez, F. Daniela, Y. Alon Levy and S. Dan, 1998. Xml-QL. A Query Language for XML. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

Jensen, M.R., H. Thomas Moller and P. Torben Bach, 2001. Specifying OLAP Cubes on XML data. J. Intell. Inform. Syst., 17: 255-280.

Nathan, G.C., M. William and R. Berthold, 2002. Relational extensions for OLAP. IBM Syst. J., 41: 714-731.

Park Byung, K., H. Hyoil and S. Il-Yeol, 2005. XML-OLAP: A multidimensional analysis framework for XML warehouses. Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery, Aug. 22-26, DaWaK, Copenhagen, Denmark, pp: 32-42.

Vassiliadis, P. and K. Timos Sellis, 1999. A survey of logical models for OLAP databases. SIGMOD Rec., 28: 64-69.

Wang, H., L. Jianzhong, H. Zhenying and G. Hong, 2005. OLAP for XML data. Proceedings of the 5th International Conference on Computer and Information Technology (CIT 2005), Sept. 21-23, IEEE Computer Society, Shanghai, China, pp: 233-237.

Wiwatwattana, N., H.V. Jagadish, V.S. Laks Lakshmanan and S. Divesh, 2007. X3: A cube operator for XML OLAP. Proceedings of the IEEE 23rd International Conference ICDE, Apr. 15-20, IEEE, pp: 916-925.

Yin, X. and Torben, 2006. Bach pedersen. Evaluating XML-extened OLAP queries based on physical algebra. J. Database Manage., 17: 85-116.

Ykhlef, M., 2007. Recursive SQL-like query language for XML. Proceedings of the 9th International Conference on Information Integration and Web-based Applications and Services (IIWAS 2007), Dec. 3-5, Jakarta, Indonesia, pp: 1-10.