Storage Techniques for Heterogeneous Reference Type Data in an XML Schema

Hwan-Seung Yong, Kyoung-Hye Lee†, Wol Young Lee †† and Minsoo Lee†
hsyong@ewha.ac.kr, you-rhee@hanmail.net wylee@scu.ac.kr, mlee@ewha.ac.kr,
Department of Computer Science and Engineering, Ewha Womans University , Seoul, Korea†
Department of Global Business & Information, Seoul Christian University, Seoul, Korea††

Summary
An XML Schema is an XML-based alternative to DTDs which describes the structure of an XML document. Although various storage techniques that can reflect the XML schemas are actively being researched, new techniques need to be developed due to the fact that IDREFS type elements can refer to several different types of data. For example, the PDTNet XML schema data, an international standard for the PDM data model, is an actual example document which includes references to all sorts of elements. However, existing storage techniques only handle homogeneous reference data. In this paper, we have proposed efficient storage techniques for such complex heterogeneous reference type data, and showed that the proposed techniques significantly improve the query performance

Key words:
PDTNet XML schema, heterogeneous reference type, IDREFS.

1. Introduction

The XML (eXensible Markup Language) schema was proposed to improve upon XML DTD (Document Type Definition), which has very weak support for data types[1]. XML schema also supports namespaces which enables the definition of various data types to be used in different application domains. Among the basic built-in data types supported by the XML schema is the reference type data which can refer to arbitrary data. That is, an element of the IDREFS (definition of REFerences to unique IDentifiers) type can refer to several data elements which are heterogeneous types. For example, the PDTNet XML schema, an international standard for the PDM (Product Data Management) data model [2], is used by an information system that manages all the data for products, including planning, designing, manufacturing, validating, and marketing. Here, IDREFS type elements usually refer to heterogeneous type data as shown in Fig. 1. In Fig. 1, the 'reference' element, which is IDREFS type, can reference both the 'item' elements and 'i_ver' elements.

Previous storage techniques have focused mainly on referencing homogeneous type data as shown in Fig. 2. In Fig. 2, the 'refer_item' element, which is IDREFS type, will only reference 'item' elements, and the 'refer_i_ver' element, which is also IDREFS type, will only reference 'i_ver' elements. In these storage techniques, the set attributes or repeating recursive elements are usually stored in a separate table. When the IDREFS references only a specific type of data, this approach is feasible. This is because a single table only needs to be scanned. However, if the IDREFS references several different types of data in different situations, the table scans may need to be extensively carried out on almost all the tables in the database to find the appropriate type data that is referenced. Thus, if an element refers to heterogeneous type data, the processing time will be slower as many tables must be scanned to retrieve the information. Therefore, in order to query complex reference type data such as the PDTNet XML schema data, this issue becomes very important.

In this paper, we have developed storage techniques which will query complex reference type data efficiently. They are based on the relational database storage model. We do not store the information of referenced data in a separate table, but in the same table. That is, we store the attribute values of referenced elements together in the same table in one field as a concatenated string. Moreover, we have developed an application which processes the string to query reference type data. Our techniques can improve the performance of query speed for complex reference type data because they minimize data redundancy and table joins.

The paper is organized as follows. Related work is presented in Section 2. Section 3 explains our proposed storage techniques for heterogeneous reference type data. The performance evaluation for the storage techniques is given in Section 4. And finally, we provide our conclusion and remarks in Section 5.

Manuscript received August 5, 2008
Manuscript revised August 20, 2008

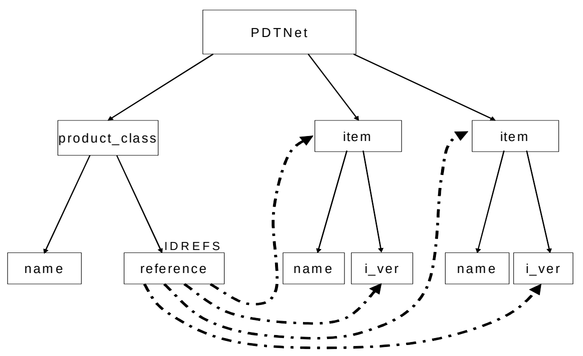


Fig. 1 heterogeneous reference type

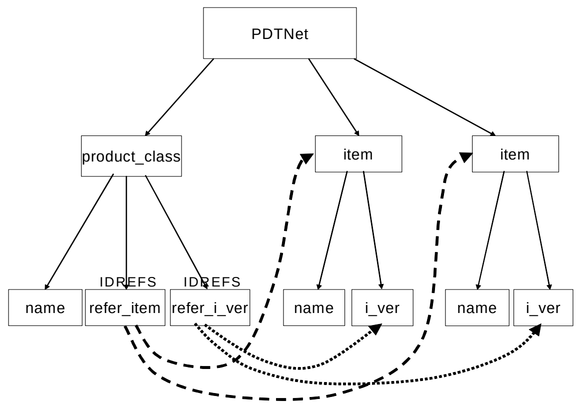


Fig. 2 homogeneous reference type

2. Related Works

As has been mentioned in [3], the XML-to-SQL query translation method remains an open problem. The research on storage techniques for XML currently focuses on how to store elements of various types [4]. Most studies make use of relational databases as storage media because these can easily integrate applications in the Web environment. The storage techniques using relational databases include AT&T's STORED [5], the attribute inlining technique of Daniela Florescu and Donald Kossmann [6], XML-DBMS [7], the hybrid inlining technique developed at the University of Wisconsin [8], a decomposition schema technique [9] and so on. The technique described in [8] was expanded to [10] by considering ordered XML. Our solution can combine the flexibility of the XML model with the performance of the relational model.

In addition, [11] presented an algorithm that translates path expression queries to SQL in the presence of recursion in the schema in the context of the schema-based XML storage shredding of XML into relations. Here, they referred to path expression queries having the descendant axis (//) as recursive XML queries, and to techniques that store XML data into an RDBMS based on an XML schema (or DTD) as schema-based XML storage techniques [12]. In [12], the researchers developed an algorithm to overcome the incompleteness and redundancies caused by the shared element of the shared-inlining algorithm [8].

We have analyzed these storage techniques based on relational databases. As a result, we recognized that reference type data are usually stored into separate overflow tables. In addition, most of the researches do not discuss the referencing of heterogeneous type data. An element in XML can refer to all sorts of elements. In the PDTNet XML schema, an element can actually refer to 50 or more sorts of elements. If the reference type data are stored in overflow tables, query time increases because the query processor has to refer to the separate tables. Furthermore, if the documents reference heterogeneous type data, the query speed becomes even slower. Thus, efficient storage techniques are necessary to handle complex reference type data in the XML schema. In this paper, we do not include reference type data in separate overflow tables, but rather the data are presented through inlining in a table. Our storage technique makes use of table-based mapping for simplicity. Because it matches the structure of tables as well as result sets in a relational database, it is easy to write code based on this mapping; this code is fast, scales well, and is quite useful for certain applications such as transferring data between databases one table at a time.

3. Storage Techniques for Heterogeneous Reference Types

In this paper, we store not only homogeneous but also heterogeneous reference type data in the same table. Therefore, it is possible to process all the data at once and minimize the data redundancy.

3.1 XML Data Example

Fig. 3 is an example XML document that is based on PDTNet XML. Here, two underlined 'Assigned_product' elements respectively refer to different element types. The first one containing 'id:1 id:2 id:3' references 'Product_class' elements, and the second one containing 'idv:1 idv:2' references 'Product_version' elements. Fig. 4 models the XML document shown in Fig. 3 as a graph. An element is represented as a rectangle with a solid line. An attribute belonging to an element is represented as a rectangle with a dotted line and is positioned at the right side of the element. The solid arrows represent parent-child relationships among the elements. And the dotted bold arrows indicate the elements referenced by an IDREFS type element. It is clearly shown that the two different 'Assigned_product' elements are each referencing different types, the 'Product_class' elements and the 'Product_version' elements.

3.2 FKey-Search storage technique

In the PDTNet XML schema, one IDREFS type element can refer to various elements of different types. This makes it hard to store the XML document into relational databases, and incurs significantly increased query processing time.

In the FKey-Search (i.e., Foreign Key-Search) storage technique, each element is mapped to its own dedicated table which contains all the attributes as fields of the table. The bounded(i.e., limited number of) simple type subelements are also stored as fields of the table. The unbounded(i.e., unlimited number) or complex subelements are stored in separate tables and the parent-child relationships are maintained via foreign keys. However, IDREFS subelements with the referencing id values encoded in a specific format are stored together in the parent table. The IDREFS id values are encoded in a format that contains the type information and sequence number so that heterogeneous types can be referenced by IDREFS. For example, in Fig. 5, the 'Item', 'Product class', and 'Document assignment' elements are stored in the 'Item', 'Product class', and 'Document assignment' tables, respectively. The parent-child relationships are stored in the tables by using the 'a_id' and 'f_id' fields in the table to store the element id and the parent id of the element. For example, the 'Product class' element with id value 'id:1' is a child of the 'Pdtnet_schema' with id value 'ps:1'.

```
<Pdtnet_schema version_id="1.8" id="ps:1"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
               xsi:noNamespaceSchemaLocation="Pdtxml
schema.xsd">
  <Product_class id="id:1">
    <Name>Google</Name>
    <Product_id> Product id 1< Product_id>
  </ Product_class>
  < Product_class id="id:2">
    <Name>MSN</Name>
    < Product_id> Product id 2< Product_id>
    < Product_version id="idv:1">
     <Id> Product v1</Id>
    </ Product_version>
    < Product_version id="idv:2">
     <Id> Product v2</Id>
    </ Product_version>
  </ Product_class >
  < Product_class id="id:3">
    <Name>HWP</Name>
    < Product_id> Product id 3</ Product_id>
  </ Product_class>
  <Item id="ii:1">
   <Id> i1</Id>
   <Name>supplied_item</Name>
   <Item_version id= "iiv:1">
     <Id>item_version_id1</Id>
     <Document_assignment id= "ida:1">
       <Assigned_product >id:1 id:2 id:3</Assigned_product >
       <Role>Latest document description</Role>
     </Document_assignment>
        <Document_assignment id= "ida:2">
       <Assigned_product >idv:1 idv:2</Assigned_product >
       <Role>Latest version</Role>
     </Document_assignment>
   </Item_version>
  <Item>
  </Pdtnet_schema>
```

Fig. 3 Example of PDTNet XML schema

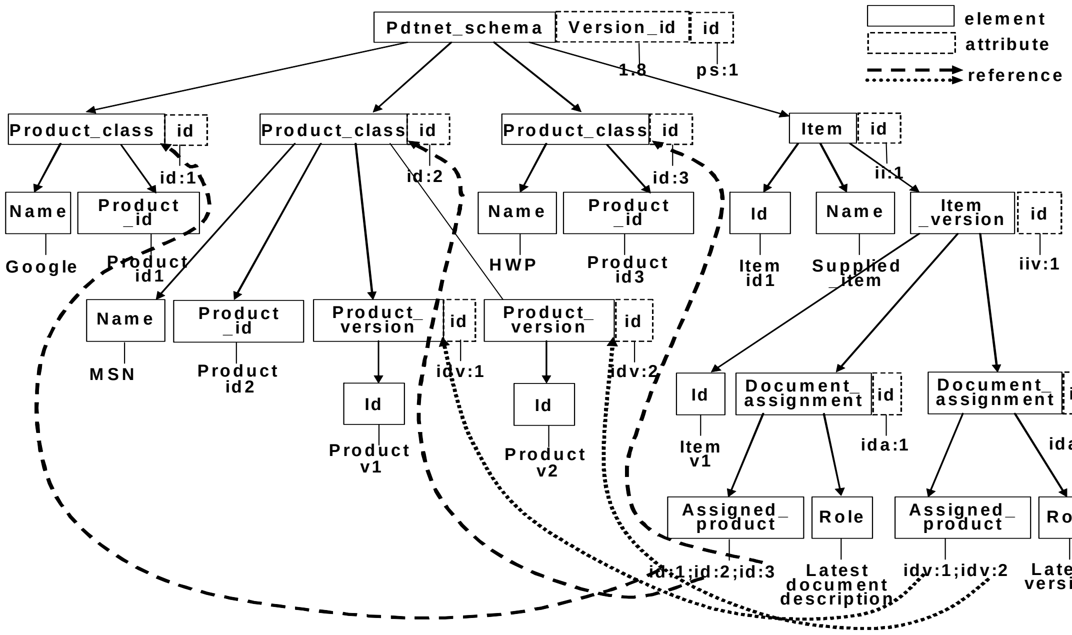


Fig. 4 XML Graph for Fig. 3

When there are a limited number (or bounded number) of simple type subelements, all of the subelements are stored together with the parent element. In Fig. 5, as there exists only a single 'Name' subelement, it is stored together in its parent 'Product class' element table. When unbounded or complex subelements exist, the subelements are stored in separate tables. As an example, in Fig. 5, the 'Item' and 'Product class' elements are not stored together with the parent element in the 'Pdtnet_schema' table. The parent-child relationships are maintained as previously discussed using the 'a_id' and 'f_id' fields**.** Finally, the IDREFS subelement is stored together with the parent element. In Fig. 5, the IDREFS type element 'Assigned_product' is stored together in the parent element table 'Document_assignment' as a field named 'Assigned_product(REF)'. The format of the IDREFS is a string of id values. Each of the id values are formed using a designated format. The id values contain a 'type' value and 'sequence number' value, separated by a ':' delimiter. For example, in Fig. 5 the 'Document_assignment' element with id value of 'ida:2' has a subelement 'Assigned_product' which references id values 'idv:1' and 'idv:2'. These id values can be translated into useful information where the prefix 'idv' means 'Product_version' type and the following number after the ':' delimiter represents the sequence number of the 'Product_version' element to be referenced in the table. Similarly, the 'ida' prefix in the element identifiers mean the 'Document_assignment' type, and the 'ii' prefix means the 'Item' element type, etc.

This approach can keep the database size at a minimum by maintaining the complex parent-child relationships in the child element. As an exception, only the IDREFS type is kept in the parent element by using a special encoding of the id values. Using this storage approach, heterogeneous elements can be referenced by the IDREFS. This approach keeps the storage structure to be simple but requires a parsing mechanism to identify the types from the id values in the IDREFS field. Because the exact types can be identified from the id values, the target tables are immediately identified and efficient query processing is possible. In contrast, existing mechanisms would require extensive table scans to find all heterogeneous matching reference id values.

3.3 Subelement-Inlining storage technique

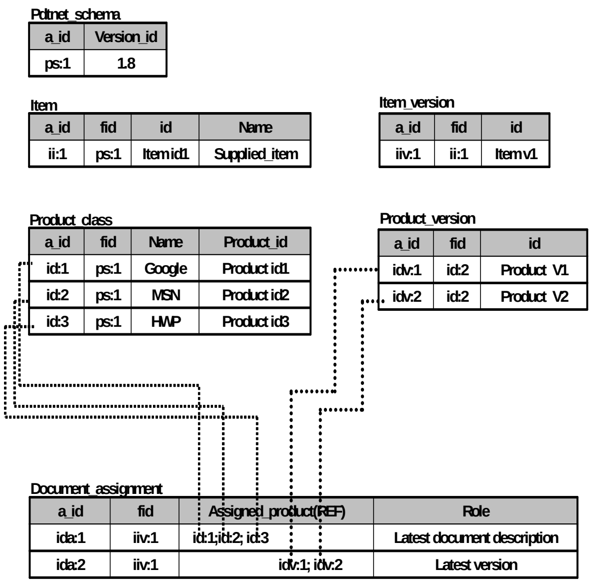
In the Subelement-Inlining technique, the identifiers of the subelements and the reference information of an element are stored all together in one table. That is, the technique stores all the parent-child relationship information as well as the reference id values in the parent table, as shown in Fig. 6. In other words, the subelements are inlined into its parent element, hence the name of the technique is subelement-inlining. This technique stores the id values of the subelements in the table in the same way as the heterogeneous reference types are stored in the FKey-Search storage technique. For each type of subelement there exists one field in the table, and this field is encoded with a string of multiple subelement id values. The reference id values are stored in the same way as was discussed in the FKey-Search storage technique. The encoding of the id values as strings are identical to the FKey-Search storage technique as well.

For example, in Fig. 6, the 'Pdtnet_schema' element table contains a single field 'Product class(REF)' which stores the id values of the 'Product class' subelements as a concatenated string such as 'id:1,id:2,id:3'. The dotted lines in the figure show the parent-child relationship among the 'Pdtnet_schema' element and the 'Product class' subelements. The IDREFS subelements are stored in the same way as the subelements are stored. In Fig. 3-4, the 'Document_assignment' element table contains an 'Assigned_product(REF)' field which is used for IDREFS and stores the reference id values for heterogeneous type elements. The 'Document_assignment' element with id value of 'ida:1' has an 'Assigned_product(REF)' field containing id values 'id:1,id:2,id:3' which are referencing 'Product class' elements. In contrast, the 'Document_assignment' element with id value of 'ida:2' has an 'Assigned_product(REF)' field containing id values 'idv:1,idv:2,idv:3' which are referencing 'Product version' elements, which is a different type referenced by the previous 'Assigned_product(REF)' field. The dotted lines in the figure show the referencing of such elements among the 'Document_assignment' element and the 'Product class' and 'Product version' elements.

This technique further simplifies the storage structure compared to the FKey-Search technique by inlining all subelements into the parent elements as a concatenated string field. The IDREFS storage method is identical to the FKey-Search technique. Therefore, all processing related to finding parent-child relationships and references are treated in the same manner. In this technique, the query processor can also easily find the table containing the subelements of the same type. It will also check all the reference strings related to elements of the IDREFS type and search different tables according to the different element types identified by the prefix of the identifiers.

4. Performance Evaluation

Both the FKey-Search and Subelement-Inlining techniques store a string for the reference information in one field, such as 'id:1;id:2;id:3' and 'idv:1;idv:2'. In order to recognize the string, we have developed an application which divides the string and searches for the referenced data. The operation time for the application is negligible compared to the total query time.


Pdtnet_schema

a_id	Version_id
ps:1	1.8

Item

a_id	fid	id	Name
ii:1	ps:1	Item id1	Supplied_item

Item_version

a_id	fid	id
iiv:1	ii:1	Item v1

Product_class

a_id	fid	Name	Product_id
id:1	ps:1	Google	Product id1
id:2	ps:1	MSN	Product id2
id:3	ps:1	HWP	Product id3

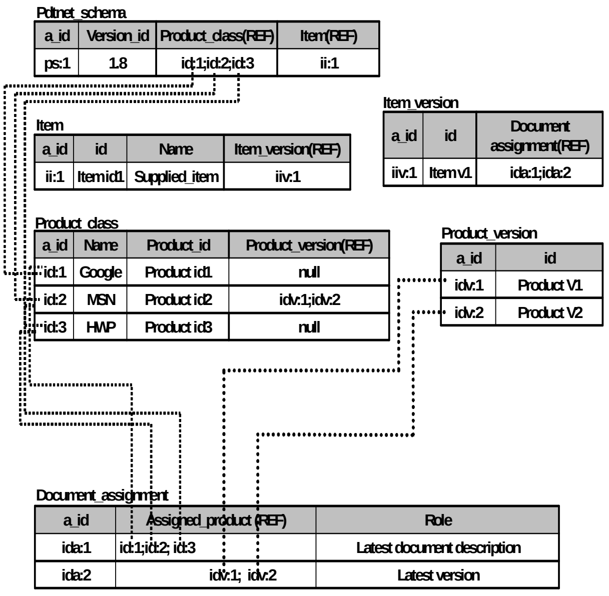
Product_version

a_id	fid	id
idv:1	id:2	Product V1
idv:2	id:2	Product V2

Document_assignment

a_id	fid	Assigned_product(REF)	Role
ida:1	iiv:1	id:1;id:2; id:3	Latest document description
ida:2	iiv:1	idv:1; idv:2	Latest version

Fig. 5 FKey-Search storage technique


Pdtnet_schema

a_id	Version_id	Product_class(REF)	Item(REF)
ps:1	1.8	id:1;id:2;id:3	ii:1

Item

a_id	id	Name	Item_version(REF)
ii:1	Item id1	Supplied_item	iiv:1

Item_version

a_id	id	Document assignment(REF)
iiv:1	Item v1	ida:1;ida:2

Product_class

a_id	Name	Product_id	Product_version(REF)
id:1	Google	Product id1	null
id:2	MSN	Product id2	idv:1;idv:2
id:3	HWP	Product id3	null

Product_version

a_id	id
idv:1	Product V1
idv:2	Product V2

Document_assignment

a_id	Assigned_product (REF)	Role
ida:1	id:1;id:2; id:3	Latest document description
ida:2	idv:1; idv:2	Latest version

Fig. 6 Subelement-Inlining storage technique

4.1 Query processing time according to query types

We have implemented a PDTNet query processor to search for PDM data and create XML documents as query results. The implementation environment that was used is Microsoft Windows XP with SQL Server 2000, and the

system was developed with C++. Users can access the database and search by query expressions. Our processor can return the element specified by a query expression; it can also return every subelement of the element. The query processor can automatically convert the searched data into an XML document.
The query types used for performance evaluation are as shown in Table 4-1. Basically, we make use of the query type classification of P. Griffiths Selinger [13]. In Table 1, a traversal-based query is used to compare the retrieval time for subelements.

Table 1 Query types for performance evaluation

Query types	Example queries
Insert	Q1. Insert description field if Name of Product_class is 'Google'.
Update	Q2. Replace the value with 'MS_WORD' if Name of Product_class is 'HWP'.
Delete	Q3. Delete if Id of Item is 'Item id2'.
Join	Q4. Search Product_id if Name of Product_class is 'MSN'.
Grouping Query	Q5. Count if Name of Item is 'Supplied_item'.
Sort	Q6. Sort by Id if Name of Item is 'Supplied_item'.
Traversal -based Query	Q7. Print the Item table if Id of Item is 'Item id2'.
	Q8. Print the Item table and all the subelement tables if Id of Item is 'Item id2'.

In general, the query execution time of the FKey-Search and Subelement-Inlining techniques are similar, as shown in Fig. 7. However, the insertion time of FKey-Search is better than that of Subelement-Inlining. In addition, in the case of joining, the query time of FKey_Search is faster than that of Subelement-Inlining because the technique does not require parsing of all id values. Furthermore, we have compared the query time for a specified element and all of its subelements. That is, in Table 1, the query Q7 requests that the query processor return only the data of the Item table. However, the query Q8 prints all the subelements of the Item satisfying the condition.

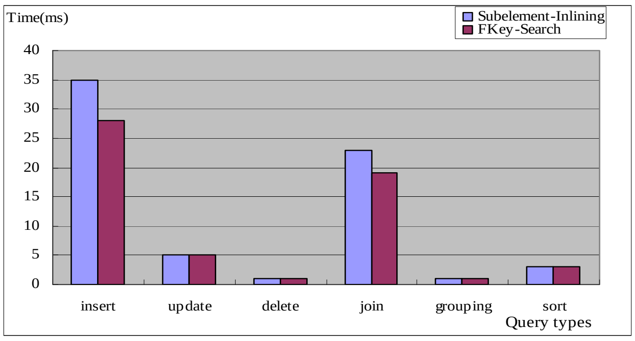


Fig. 7 Query time for basic query

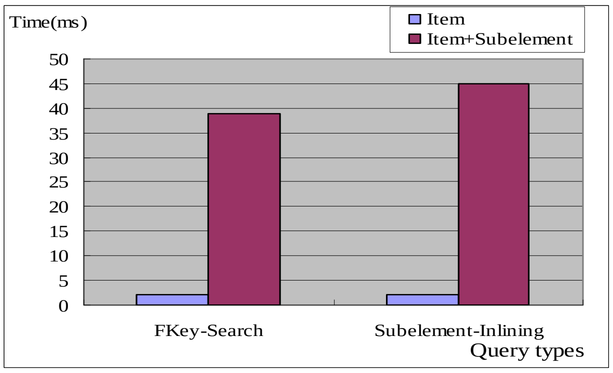


Fig. 8 Query time for traversal-based query

As shown in Fig. 8, in the case of returning only the specified item, there is not a major difference in the query times of the two storage techniques. However, in the case of returning all the subelements of the item element, the FKey-Search technique is better than the Subelement-Inlining because the FKey-Search technique does not require the parsing time for retrieving the subelement identifiers.

4.2 Query processing time for IDREF/IDREFS type data

We have compared the FKey-Search and Subelement-Inlining techniques with XML-DBMS in terms of handling heterogeneous reference type data. This experiment can evaluate the performance based on how reference type data is stored. The FKey-Search and Subelement-Inlining techniques store an element of IDREFS type and the reference information in the same table. In contrast, XML-DBMS store the referenced data in another separate table as, in general, prior methods have done. Therefore, in terms of the two storage techniques, we have evaluated the query time depending on the amount of reference data and the types of reference data.

4.2.1 Homogeneous reference types

In an XML-DBMS, when IDREFS type elements refer to homogeneous type data, the query time increases dramatically, as shown in Fig. 9, depending on the quantity of referenced data.

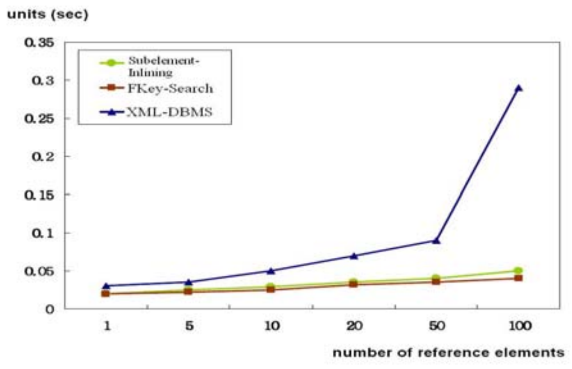


Fig. 9 Homogenous reference type elements

As the number of reference elements increase from 1 to 100, the Subelement-Inlining and FKey-Search techniques show much better performance than the XML-DBMS. The XML-DBMS stores referenced data in another table separately. Therefore, the processor has to join the tables whenever a reference element refers to data. This is why the query time of previous methods is usually slow when it comes to reference type data.

4.2.2 Heterogeneous reference types

Fig. 10 shows the query time measured as the quantity of the heterogeneous reference data of one element varies. The types of reference elements vary from 1 to 10, and each element refers to only a single data.

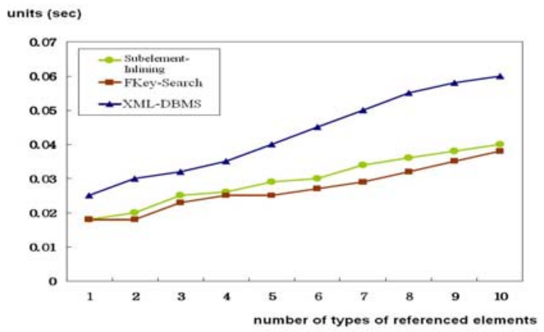


Fig. 10 Heterogeneous reference type elements

As is clear from the figure, the Subelement-Inlining and FKey-Search techniques are always faster than the XML-DBMS. The more types of reference data there are, the greater the difference between the XML-DBMS Subelement-Inlining/FKey-Search becomes. This is because previous XML-DBMS methods store referenced data in separate tables, whereas our technique stores it in one table as one string. Earlier methods increase the number of table joins. If an element refers to 50 or more subelements, as can happen in the PDTNet XML schema, these storage techniques cause not only a waste of storage space, but also increase the time overhead during querying.

To summarize the experiemental results, the Subelement-Inlining and FKey-Search techniques always show much better performance than XML-DBMS when accessing reference type data.

The FKey-Search storage technique uses a primary key and foreign key concept to model parent-child relationships in XML. It uses complex query expressions, due to the upper and lower relationship of tables, to query all subelements. Thus, the optimization of query expressions is required. The index size of FKey-Search is larger than that of Subelement-Inlining due to the use of both primary keys and foreign keys. Moreover, the time it takes to insert a new element is greater than that of Subelement-Inlining due to the integrity checking of the relationship between primary keys and foreign keys.

The Subelement-Inlining storage technique stores the concatenated string of the attribute values of subelements as subelement information. This is the same as the way in which the information of referenced data is stored. The time it takes to insert a new element is less than that of FKey-Search. The Subelement-Inlining is slower than the FKey-Search in querying all subelements because a query processor has to create many query expressions for the subelement information. It also has to solve the reference integrity problem using an extra program.

5. Conclusions

In this paper, we have proposed storage methods to support efficient processing of queries on an XML-based document containing complex reference types. We have developed the FKey-Search and Subelement-Inlining storage techniques and evaluated their performance in terms of the PDTNet XML schema data. This data model is an international standard for PDM (Product Data Management). The two storage techniques show significant performance benefits for such semi-structured XML documents. These techniques minimize data redundancy and improve the performance of query speed for complex reference type data.

Acknowledgments. This work was supported by grant No. (R01-2006-000-10609-0) from the Basic Research Program of the Korea Science and Engineering Foundation.

References

[1] XML schema, http://www.w3.org/XML/Schema, W3C, 2004.

[2] PDTnet Project – Product Data Technology, http://www.prostep.de/en/services/projekte/pdtnet/, 2006.

[3] Krishnamurthy, R., Kaushik, R., and Naughton, J.F., "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems," In *XML Database Symposium*, 2003.

[4] Abiteboul, S., Buneman, P., Sucui, D., "Data on the Web – From Relations to Semistructured Data and XML," *Morgan Kaufmann*, October, 1999.

[5] Suciu, D., Deutsch, A., Fernandex, M., "Storing Semi-structured Data Using {STORED}," In *Proc. of ACM SIGMOD*, pp431-442, 1999.

[6] Florescu, D., Kossmann, D., "A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database," *INRIA Technical Report, INRIA, Mo.3680*, May, 1999.

[7] Boirret, R., "XML-DBMS Version 1.01," http://www.rpbourret.com/xmldbms/ readme.htm, 2000.

[8] Shanmuhasundaram, J., Tyfte, K., He, G., He, C., Zhang, C., DeWitt, D., Naughton, J., "Relational Databases for Querying XML Documents: Limitations and Opportunities," In *Proc. of the 25th VLDB Conference*, pp.302-304, 1999.

[9] Schmidt, A., Kersten, M., Windhouwer, M., Waas, F., "Efficient Relational Storage and Retrieval of XML Documents," In *WebDB*, 2000.

[10] Tatarinov, I., Viglas, S., Beyer, K. S., Shanmugasundaram, J., Shekita, E. J., Zhang, C., "Storing and Querying Ordered XML Using a Relational Database System," In *SIGMOD Conference*, pages 204–215, 2002.

[11] Krishnamurthy, R., Chakaravarthy, V. T., Kaushik, R., Naughton, J., "Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation," In *Proc. of the 20th International Conference on Data Engineering*, pages 42–53, Boston, Massachusetts, USA, March 2004.

[12] Lu, S., Sun, Y., Atay, M., Fotouhi, F., "A New Inlining Algorithm for Mapping XML DTDs to Relational Schemas," In *Proc. of the First International Workshop on XML Schema and Data Management, in Conjunction with the 22nd ACM International Conference on Conceptual Modeling (ER2003)*, Chicago, Illinois, October 2003.

[13] Selinger, P. G., Astrahan, M., Chamberlin, D., Lorie, R., Price, T., "Access Path Selection in a Relational Database Management System," In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp23-34, 1979.

Hwan-Seung Yong received the B.S., M.S. and Ph.D degrees in Computer Engineering from Seoul National University in 1983, 1985 and 1994, respectively. During 1985-1989, he was a member of research staff in Eletronics and Communications Research Institute (ETRI. Since 1995, He is a professor of Ewha Womans University.

Kyoung-Hye Lee received the B.A. degree in Christian Studies and M.S. degree in Computer Science from Ewha Womans University in 2004, 2006 respectively. She is a member of technical staff at Samsung Electronics Corp.

Wol Young Lee received the B.S., M.S. and Ph.D degrees in Computer Engineering from Ewha Woman's University in 1988, 2000 and 2005, respectively. During 1988-1998, she was a CEO of Daegi Computer Institute. During 2005-2007, She was also a professor of Seoul Christian University.

Minsoo Lee received the B.S., M.S. degrees in Computer Engineering from Seoul National University in 1992, 1995 respectively. And he received Ph.D degree in Computer and Information Sciences and Engineering from University of Florida 2000. During 2000-2002, he was a member of technical staff at ORACLE Corp, U.S.A. Since 2002, he is a associate professor of Ewha Womans University.