Building XML Data Warehouse Based on Frequent Patterns in User Queries

Ji Zhang[1], Tok Wang Ling[1], Robert M. Bruckner[2], A Min Tjoa[2]

[1]Department of Computer Science
National University of Singapore
Singapore 117543
{zhangji, lingtw}@comp.nus.edu.sg

[2]Institute of Software Technology
Vienna University of Technology
Favoritenstr. 9/188, A-1040 Vienna, Austria
{bruckner, tjoa}@ifs.tuwien.ac.at

Abstract. With the proliferation of XML-based data sources available across the Internet, it is increasingly important to provide users with a data warehouse of XML data sources to facilitate decision-making processes. Due to the extremely large amount of XML data available on web, unguided warehousing of XML data turns out to be highly costly and usually cannot well accommodate the users' needs in XML data acquirement. In this paper, we propose an approach to materialize XML data warehouses based on frequent query patterns discovered from historical queries issued by users. The schemas of integrated XML documents in the warehouse are built using these frequent query patterns represented as Frequent Query Pattern Trees (FreqQPTs). Using hierarchical clustering technique, the integration approach in the data warehouse is flexible with respect to obtaining and maintaining XML documents. Experiments show that the overall processing of the same queries issued against the global schema become much efficient by using the XML data warehouse built than by directly searching the multiple data sources.

1. Introduction

A data warehouse (DWH) is a repository of data that has been extracted, transformed, and integrated from multiple and independent data source like operational databases and external systems [1]. A data warehouse system, together with its associated technologies and tools, enables knowledge workers to acquire, integrate, and analyze information from different data sources. Recently, XML has rapidly emerged as a standardized data format to represent and exchange data on the web. The traditional DWH has gradually given way to the XML-based DWH, which becomes the mainstream framework.

Building a XML data warehouse is appealing since it provides users with a collection of semantically consistent, clean, and concrete XML-based data that are suitable for efficient query and analysis purposes. However, the major drawback of building an enterprise wide XML data warehouse system is that it is usually extremely time and cost consuming that is unlikely to be successful [10]. Furthermore, without proper guidance on which information is to be stored, the resulting data warehouse cannot really well accommodate the users' needs in XML data acquirement.

In order to overcome this problem, we propose a novel XML data warehouse approach by taking advantage of the underlying frequent patterns existing in the query

history of users. The historical user queries can ideally provide us with guidance regarding which XML data sources are more frequently accessed by users, compared to others. The general idea of our approach is: Given multiple distributed XML data sources and their globally integrated schema represented as a DTD (data type definition) tree, we will build a XML data warehouse based on the method of revealing frequent query patterns. In doing so, the frequent query patterns, each represented as a Frequent Query Pattern Tree (*FreqQPT*), are discovered by applying a rule-mining algorithm. Then, FreqQPTs are clustered and merged to generate a specified number of integrated XML documents.

Apparently, the schema of integrated XML documents in the warehouse is only a subset of the global schema and the size of this warehouse is usually much smaller than the total size of all distributed data sources. A smaller sized data warehouse can not only save storage space but also enable query processing to be performed more efficiently. Furthermore, this approach is more user-oriented and is better tailored to the user's needs and interests.

There has been some research in the field of building and managing XML data warehouse. The authors of [2] present a semi-automated approach to building a conceptual schema for a data mart starting from XML sources. The work in [3] uses XML to establish an Internet-based data warehouse system to solve the defects of client/server data warehouse systems. [4] presents a framework for supporting interoperability among data warehouse islands for federated environments based on XML. A change-centric method to manage versions in a web warehouse of XML data is published in [5]. Integration strategies and their application to XML Schema integration has been discussed in [6]. The author of [8] introduces a dynamic warehouse, which supports evaluation, change control and data integration of XML data.

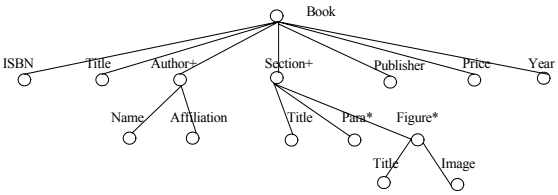
The remainder of this paper is organized as follows. Section 2 discusses the generation of XML data warehouses based on frequent query patterns of users' queries. In Sections 3, query processing using the data warehouse is discussed. Experimental results are repoeted in Section 4. The final section conclude this paper.

2. Building a XML DWH Based on Frequent Query Patterns

2.1. Transforming Users' Queries into Query Path Transactions

XQuery is a flexible language commonly used to query a broad spectrum of XML information sources, including both databases and documents [7]. The following XQuery-formatted query aims to extract the ISBN, Title, Author and Price of books with a price over 20 dollars from a set of XML documents about book-related information. The global DTD tree is shown in Figure 1.

```
FOR $a IN DOCUMENT (book XML documents)/book
      SATIFIES $a/Price/data()>20
   RETURN <QueryResult> <book>{$a/ISBN, $a/Title, $a/Author, $a/Price}</book>
</QueryResult >
```



QP1: Book/ISBN
QP2: Book/Title
QP3: Book/Author/Name
QP4:Book/Author/Affiliation
QP5: Book/Price

Fig. 1. Global DTD Tree of multiple XML documents. **Fig. 2.** QPs of the XQuery sample.

A *Query Path* is a path expression of a DTD tree that starts at the root of tree. QPs can be obtained from the query script expressed using XQuery Statements. The sample query above can be decomposed into five QPs, as shown in Figure 2. The root of a QP is denoted as *Root*(QP) and all QPs in a query have the same root.

Please note that two QPs with different roots are regarded as different QPs, although these two paths may have some common nodes. This is because different roots of paths often indicate dissimilar contexts of the queries. For example, two queries *Author/Name* and *Book/Author/Name* are different because *Root*(*Author/Name*)=Author ≠ *Root*(*Book/Author/Name*)=Book.

A query can be expressed using a set of QPs which includes all the QPs that this query consists. For example, the above sample query, denoted as *Q*, can be expressed using a QP set such as *Q={QP1, QP2, QP3, QP4, QP5}*.

By transforming all the queries into QP sets, we now obtain a database containing all these QP sets of queries, denoted as D_{QPS}. We will then apply a rule-mining techniques to discover significant rules among the users' query patterns.

2.2. Discovering Frequent Query Path Sets in D_{QPS}

The aim of applying a rule mining technique in D_{QPS} is to discover *Frequent Query Path Sets* (FreqQPSs) in D_{QPS}. A FreqQPS contains frequent QPs that jointly occur in D_{QPS}. *Frequent Query Pattern Trees* (FreqQPTs) are built from these FreqQPSs and serve as building blocks of schemas of the integrated XML documents in the data warehouse. Formal definition of FreqQPTs is given as follows.

Definition 1. Frequent Query Path Set (FreqQPS): From all the occurring QPs in D_{QPS} transformed from user's queries, a *Frequent Query Path Set* (FreqQPS) is a set of QPs: $\{QP_1, QP_2, \ldots, QP_n\}$ that satisfies the following two requirements:

(1) Support requirement: $Support(\{QP_1, QP_2, \ldots, QP_n\}) \geq minsup;$

(2) Confidence requirement: *For each* QP_i,

$$Freq(\{QP_1, QP_2, \ldots, QP_n\}) / Freq(QP_i) \geq minconf.$$

where *Freq(s)* counts the occurrence of set *s* in D_{QPS}. In (1), $Support(\{QP_1, QP_2, \ldots, QP_n\}) = freq(\{QP_1, QP_2, \ldots, QP_n\}) / N(D_{QPS})$, where $N(D_{QPS})$ is the total number of QPs in D_{QPS}. The constants *minsup* and *minconf* are the minimum support and confidence thresholds, specified by the user. A FreqQPS that consists of *n* QPs is termed as an *n*-itemset FreqQPS.

The definition of a FreqQPS is similar to that of association rules. The support requirement is identical to the traditional definition of large association rules. The confidence requirement is, however, more rigid than the traditional definition. Setting a more rigid confidence requirement is to ensure the joint occurrence of QPs in a FreqQPS should be significant enough with respect to an individual occurrence of any

QP. Since the number of QPs in the FreqQPS is unknown in advance, we will mine all FreqQPSs containing various numbers of itemsets. The FreqQPS mining algorithm is presented in Figure 3.

The n-itemset QPS candidates are generated by joining (n-1)-itemset FreqQPSs. A pruning mechanism is devised to delete those candidates of the n-itemset QPSs that do not have *n* (n-1)-itemset subsets in the (n-1)-itemset FreqQPS list. The reason is that if one or more (n-1)-subsets of a n-itemset QPS candidate are missing in the (n-1)-itemset FreqQPS list, this n-itemset QPS cannot become a FreqQPS. *This is obviously more rigid than pruning mechanism used in conventional association rule mining.*

For example, if one or more of the 2-itemset QPSs $\{QP_1, QP_2\}$, $\{QP_1, QP_3\}$ and $\{QP_2, QP_3\}$ are not frequent, then the 3-itemset QPS $\{QP_1, QP_2, QP_3\}$ cannot become a frequent QPS. The proof of this pruning mechanism is given below. The pruning the n-itemset QPS candidates are evaluated in terms of the support and confidence requirements to decide whether or not they are a FreqQPS. The (n-1)-itemset FreqQPSs are finally deleted if they are subsets of some n–itemset FreqQPSs. For example, the 2-itemset FreqQPT $\{QP_1, QP_2\}$ will be deleted from 2-itemset FreqQPT list if the 3-itemset $\{QP_1, QP_2, QP_3\}$ exists in the 3-itemset FreqQPT list.

Algorithm MineFreqQPS
Input: D_{QPS}, *minsup*, *minconf*.
Output: FreqQPS of varied number of itemsets.
$FreqQPS^1=\{QP \text{ in } D_{QPS} | \textbf{SatisfySup}(QP)=true\}$;
i=2;
WHILE ($CanFreqQPS^{i-1}$ is not empty) {
  $CanQPS^i=$**CanQPSGen**($FreqQPS^{i-1}$);
  $CanQPS^i= CanQPS^i - \{QPS^i | \textbf{NoSubSet}(QPS^i, FreqQPS^{i-1})<i\}$;
  $FreqQPS^i=\{QPS^i \text{ in } CanQPS^i | \textbf{SatisfySup}(QPS^i)=true \text{ AND } \textbf{SatisfyConf}(QPS^i)=true\}$;
  $FreqQPS^{i-1}= FreqQPS^{i-1} - \{QPS^{i-1} | QPS^{i-1} \subseteq QPS^i, QPS^i \text{ in } FreqQPS^i, QPS^i \text{ in } FreqQPS^i\}$;
  i++; }
MaxItemset=i-2;
IF (MaxItemset≠0) **THEN**
  FOR (i=1;i≤MaxItemset; i++) **Return** ($FreqQPS^i$);

Fig. 3. Algorithm for mining FreqQPSs.

Proof: Suppose a n-itemset QPS has only *p* (n-1)-itemset subsets $QPS^{n-1}_i | 1 \le i \le p$, meaning that there are (n-*p*) subsets of QPS^n are missing in the (n-1)-itemset QPS list. These missing (n-*p*) subsets of QPS^n, denoted as $QPS^{n-1}_i | (p+1) \le i \le n$, are definitely not FreqQPSs and they fail to satisfy the support or the confidence requirement or both. Specifically,

(1) If $QPS^{n-1}_i | (p+1) \le i \le n$ does not satisfy support requirement, then $Support(QP_1, QP_2, ..., QP_{n-1}) < minsup$. Because $Support(QP_1, QP_2, ..., QP_n) \le Support(QP_1, QP_2, ..., QP_{n-1})$, so $Support(QP_1, QP_2, ..., QP_n) < minsup$, meaning QPS^n cannot become a n-itemset FreqQPS;

(2) If $QPS^{n-1}_i | (p+1) \le i \le n$ does not satisfy confidence requirement, then for a certain QP_i, $Freq(QP_1, QP_2, \ldots, QP_{n-1}) / Freq(QP_i) < minconf$. Because $Support(QP_1, QP_2, ..., QP_n) \le Support(QP_1, QP_2, ..., QP_{n-1})$, so for QP_i, $Freq(QP_1, QP_2, \ldots, QP_n) / Freq(QP_i) < minconf$, meaning that QPS^n cannot become a n-itemset FreqQPS. ∎

After we have obtained a number of FreqQPSs, their corresponding Frequent Query Pattern Trees (FreqQPTs) will be built.

Definition 2. Frequent Query Pattern Tree (FreqQPT): Given a FreqQPS, its corresponding *Frequent Query Pattern Tree* (FreqQPT) is a rooted tree FreqQPT=<V, E>, where V and E denote its vertex and edge sets, which are the union of the vertices and edges of QPs in this FreqQPS, respectively. The root of a FreqQPT, denoted as *Root*(FreqQPT), is the root of its constituting QPs.

For example, suppose a FreqQPS has two QPs: *Book/Title* and *Book/Author/Name*. The resulting FreqQPT is shown in the Figure 4.

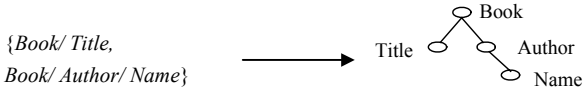


Fig. 4. Build a FreqQPT for a FreqQPS.

2.3. Generating Schemas of Integrated XML Documents

When all FreqQPTs have been mined, the schema of the integrated XML document will be built. We have noticed that a larger integrated XML document usually requires larger space when it is loaded into main memory. In order to solve this problem, we alternatively choose to build a few, rather than only one, integrated XML documents from the FreqQPTs mined, making the integration more flexible. The exact number of integrated XML documents to be obtained is user- specified. The basic idea is to use a clustering technique to find a pre-specified number of clusters of FreqQPTs. The integration of the FreqQPTs is performed within each of the clusters.

Similarity measurement of FreqQPTs

We need to measure the similarity between two FreqQPTs in order to find the closest pair in each step of the clustering process. It is noticed that the complexity of merging two FreqQPTs is dependant on the distance of the roots of the FreqQPTs involved, rather than on the other nodes in the FreqQPTs. Intuitively, the closer the two roots are to each other, the easier the merging can be done and vice versa. To measure the similarity between the roots of two FreqQPTs, we have to first discuss the similarity between two nodes in the hierarchy of a global schema.

In our work, the similarity computation between two nodes in the hierarchy is based on the edge counting method. We measure the similarity of nodes by first computing the distance between two nodes, since the distance can be easily obtained by edge counting. Naturally, the larger the number of edges between two nodes, the further apart the two nodes are. The distance between two nodes n_1 and n_2, denoted as $NodeDist(n_1, n_2)$, is computed as $NodeDist(n_1, n_2)= N_{edge}(n_1, n_2)$, where $N_{edge}()$ returns the number of edges between n_1 and n_2. This distance can be normalized by dividing the maximum possible distance between two nodes in the hierarchy, denoted by *LongestDist*. The normalized distance between n_1 and n_2, denoted as $NodeDist_N(n_1, n_2)$, is computed as follows:

$$NodeDist_N(n_1, n_2)= N_{edge}(n_1, n_2)/LongestDist$$

Thus the similarity between n_1 and n_2 is computed as:

$$NodeSim_N(n_1, n_2)=1- NodeDist_N(n_1, n_2)$$

We now give an example to show how the similarity between two roots of FreqQPTs is computed. Suppose there are two QPs, QP_1: *Book/ Price* and QP_2:

Section/ Figure/ Image as shown in Figure 5. What we should do is to compute the similarity between the roots of these two QPs, namely Book and Section. The maximum length between two nodes in the hierarchy as shown in Figure 1 is 5 (from Name or Affiliation to Title or Image). Thus $NodeSim_N$(Book, Section) = 1 – $NodeDist_N$(Book, Section) = 1–1/5 = 4/5 = 0.8.

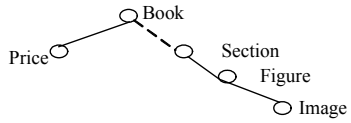


Fig. 5. Similarity between two QPs.

Merging of FreqQPTs

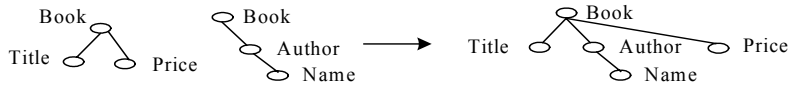
When a nearest pair of FreqQPTs is found in each step of the clustering, merging of these two FreqQPTs is performed. Let $FreqQPT_1$=<V_1, E_1>, $FreqQPT_2$=<V_2, E_2>, Root($FreqQPT_1$)=$root_1$, Root($FreqQPT_2$)=$root_2$, and $FreQPT_M$ be the new FreqQPT merged from $FreqQPT_1$ and $FreqQPT_2$. We will now present the definition of Nearest Common Ancestor Node (NCAN) of two nodes in the DTD tree before we give details of FreqQPT merging.

Definition 3. Nearest Common Ancestor Node (NCAN): The NCAN of root nodes of two FreqQPTs $root_1$ and $root_2$ in the hierarchical structure of a global DTD tree *H*, denoted as $NCAN_H(root_1, root_2)$, is the common ancestor node in *H* that are closest to both $root_1$ and $root_2$.

To merge two closest FreqQPTs, the Nearest Common Ancestor Node (NCAN) of $root_1$ and $root_2$ has to be found, thereby these two FreqQPTs can be connected.

We denote the vertex and edge set of the paths between $NCAN_H(root_1, root_2)$ and $root_1$ as $V_{NCAN\rightarrow root1}$ and $E_{NCAN\rightarrow root1}$, and those between $NCAN_H(root_1, root_2)$ and $root_2$ as $V_{NCAN\rightarrow root2}$ and $E_{NCAN\rightarrow root2}$. The $FreQPT_M$ in this case can be expressed as $FreQPT_M$={Union(V_1, V_2, $V_{NCAN\rightarrow root1}$,$V_{NCAN\rightarrow root2}$), Union ($E_1$, E_2, $E_{NCAN\rightarrow root1}$, $E_{NCAN\rightarrow root2}$)} and *Root*($FreQPT_M$)= $NCAN_H(root_1, root_2)$.

Specifically, there are three scenarios in merging two FreqQPTs, namely, (1) the two FreqQPTs have the same root; (2) The root of one FreqQPT is an ancestor node of another FreqQPT's root; (3) case other than (a) and (b). Figure 6 (a)-(c) give examples for each of the cases of FreqQPT merging discussed above. The dot-lined edges in the integrated schema, if any, are the extra edges that have to be included into the integrated schema in merging the two separate FreqQPTs.



(a) Example for Case 1.

(b) Example for Case 2.

(c) Example for Case 3.

Fig. 6 (a) – (c). Examples of FreqQPT merging.

Clustering of FreqQPTs

The aim of clustering FreqQPTs is to group similar FreqQPTs together for further integration. Merging two closer FreqQPTs is cheaper and requires less re-structuring operations compared to merging two FreqQPTs far apart from each other. In our work, we utilize the agglomerative hierarchical clustering paradigm. The basic idea of agglomerative hierarchical clustering is to begin with each FreqQPT as a distinct cluster and merge the two closest clusters in each subsequent step until a stopping criterion is met. The stop criterion of the clustering is typically either the similarity threshold or the number of clusters to be obtained. We choose to specify the number of clusters since it is more intuitive and easy to specify, compared to the similarity threshold that is typically not known before the clustering process.

Please note that k , the specified number of clusters to be obtained, should not be larger than the number of FreqQPTs, otherwise the error message will be returned. This is because the QPs in the same FreqQPTs are not allowed to be further split. In each step, the two closest FreqQPT pair will be found and merged into one FreqQPT and the number of current clusters will be decreased by 1 accordingly. This clustering process is terminated when k clusters are obtained.

2.4. Acquire Data to Feed the Warehouse

The last step of building the XML data warehouse is to read data from XML data sources when the schemas of the integrated XML documents are ready. Coming from different data source across the Internet, these data may be incomplete, noisy, inconsistent, and duplicate. Processing efforts such as standardization, data cleaning and conflict solving need to be performed to make the data in the warehouse more consistent, clean, and concrete.

3 Processing of Queries Using the Date Warehouse

One of the main purposes of building data warehouse is to facilitate the query processing. When there is no data warehouse, processing of queries use the single mediator architecture (shown in Figure 7), in which all the queries will be processed in this mediator and directed to the multiple XML data sources. When the data warehouse has been built, a dual-mediator architecture is adopted (shown in Figure 8). Mediator 1 processes all the incoming queries from users, and each query will be directed to either the data warehouse or mediator 2 which is responsible for further directing the queries to the XML data sources or both.

Specifically, let QPS_{dwh} be QP set of the integrated XML documents in the data warehouse. QPTra(q) be the QP transaction of the query q.

(i) If $QPTra(q) \subset QPS_{dwh}$, meaning that all the QPs of q can be found in the schemas of integrated XML documents in the data warehouse, and this query can be answered by using the data warehouse alone, then q will only be directed by mediator 1 to the XML data warehouse;

(ii) if $QPTra(q) \not\subset QPS_{dwh}$ and $QPTra(q) \cap QPS_{dwh}$ is not empty, meaning that not all QPs of q can be found in the schemas of integrated XML documents in the data warehouse, and the data warehouse does not contain enough information to answer q, then q will be directed by mediator 1 to both the data warehouse and mediator 2;

(iii) if $QPTra(q) \cap QPS_{dwh}$ is empty, indicating that the information needed to answer q is not contained in the warehouse, thus q will only directed by mediator 1 to mediator 2.

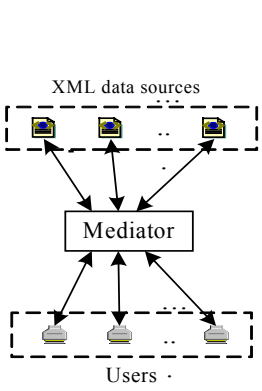


Fig. 7. Query processing without data warehouse

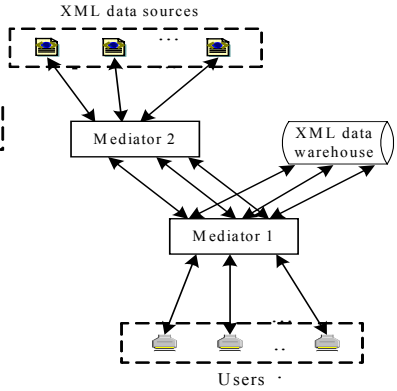


Fig. 8. Query processing with data warehouse

4. Experimental Results

In this section, we will conduct experiments to evaluate the efficiency of the constructing schema of XML data integration and the speedup of query processing by means of the data warehouse we have built. We use a set of 50 XML documents about book information and generate their global DTD tree. Zipfian distribution is employed to produce transaction file of queries, because web queries and surfing patterns typically conform to the Zipf's law [9]. In our work, the query transaction file contains 500 such synthetic queries based on which the data warehouse is built. All these experiments are carried out on the PC of 900 MHz PC with 256 megabytes of main memory running on Windows 2000.

4.1 Construction of the Data Warehouse Schema under Varying Number of Queries

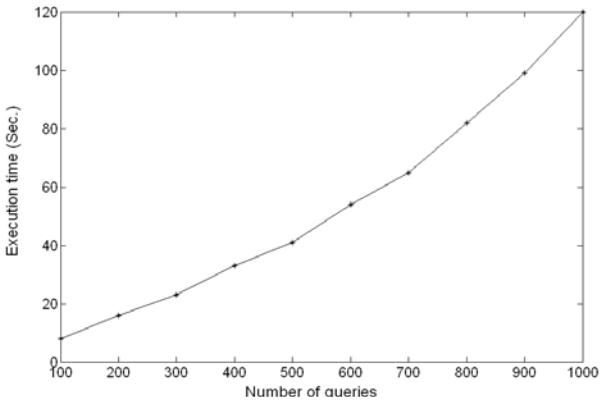


Fig. 9. Efficiency of constructing the data warehouse schema under varying number of queries

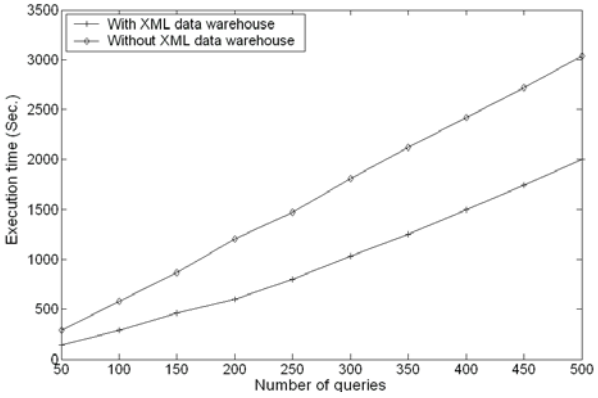


Fig. 10. Comparative study on query answering time

First, we will evaluate the time spent in constructing the schema of XML data integration of the data warehouse under varying number of queries from which frequent query patterns are extracted. The number of the queries used is varied from 100 to 1,000. As shown in Figure 9, the time increases approximately in an exponential rate since the number of candidates of FreqQPSs generated increases exponentially as the number of queries goes up.

4.2 Speedup of Query Processing Using Data Warehouse

The major benefits of building data warehouse system based on frequent query patterns are to not only obtain a smaller but more concrete and clean subset of original XML data sources but also helps speedup the query processing. In this experiment, we measure the response time for answering queries with and without the aid of the data warehouse, respectively. The number of queries to be answered ranges from 100 to 1,000. The results shown in Figure 10 justifies that, by using data warehouse we have built, the query answering is faster than that the case when there is no such a data warehouse. This is because that the potion of information contained in the data

warehouse is smaller in size than that stored in the original data sources, reducing the volume of data needed to scanned in the query answering. In addition, the data has been undergone the processing such as standardization, data cleaning and conflict solving, thus the duplication of data is lower. The smaller size and lower duplication of the data in the warehouse contribute to the higher efficiency in query answering.

5 Conclusions

In this paper, we propose a novel approach to perform XML data warehousing based on the frequent query patterns discovered from historical user's queries. A specific rule mining technique is employed to discover these frequent query patterns, based on which the schemas of integrated XML documents are built. Frequent query patterns are represented using Frequent Pattern Trees (FreqQPTs) that are clustered using a hierarchical clustering technique according to the integration specification to build the schemas of integrated XML documents. Experimental results show that query answering time is reduced when compared to the case when there is no such a data warehouse.

References

[1] H. Garcia-Molina, W. Labio, J. L.Wiener, and Y. Zhuge: Distributed and Parallel Computing Issues in Data Warehousing. In Proc. of *ACM Principles of Distributed Computing Conference (PODS),* Puerto Vallarta, Mexico 1998.

[2] M. Golfarelli, S. Rizzi, and B. Vrdoljak: Data Warehouse Design from XML Sources. In Proc. of *ACM DOLAP'01*, Atlanta, Georgia, USA, Nov. 2001.

[3] S. M. Huang and C.H. Su: The Development of an XML-based Data Warehouse System. In Proc. of *3rd Intl. Conf. of Intelligent Data Engineering and Automated Learning (IDEAL'02)*, Springer LNCS 2412, pp. 206-212, Manchester, UK, Aug. 2002.

[4] O. Mangisengi, J. Huber, C. Hawel and W. Essmayr: A Framework for Supporting Interoperability of Data Warehouse Islands using XML. In Proc. of *3rd Intl. Conf. DaWaK'01*, Springer LNCS 2114, pp. 328-338, Munich, Germany, Sept. 2001.

[5] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet: Change-centric Management of Versions in an XML Warehouse. In Proc. of *Intl. Conf. on Very Large Data Bases (VLDB'01)*, pp. 581-590, Roma, Italy, Sept. 2001.

[6] K. Passi, L. Lane, S. Madria, B.C. Sakamuri, M. Mohania and S. Bhowmick: A Model for XML Schema Integration. In Proc. of *3rd Intl. Conf. EC-Web*, Springer LNCS 2455, pp. 193-202, Aix-en-Provence, France, Sept. 2002.

[7] XQuery Language 1.0. http://www.w3.org/TR/xquery/.

[8] L. Xyleme. A Dynamic Warehouse for XML Data of the Web. *IEEE Data Engineering Bulletin*, Vol. 24(2), pp. 40-47, 2001.

[9] L. H. Yang, M. L. Lee, W. Hsu, S. Acharya. Mining Frequent Query Patterns from XML Queries. In Proc. of *8th Intl. Symp. on Database Systems for Advanced Applications (DASFAA'03)*, Kyoto, Japan, March 2003.

[10] L.Garber. Michael StoneBraker on the Importance of Data Integration. IT Professional, Vol. 1, No.3, pp 80, 77-79, 1999.