XML Data Management in Compressed Relational Database

Hongzhi Wang, Jianzhong Li, and Hong Gao

Abstract—XML is an important standard of data exchange and representation. As a mature database system, using relational database to support XML data may bring some advantages. But storing XML in relational database has obvious redundancy that wastes disk space, bandwidth and disk I/O when querying XML data. For the efficiency of storage and query XML, it is necessary to use compressed XML database in relational database. In this paper, a compressed relational database technology supporting XML data is presented. Original relational storage structure is adaptive to XPATH query process. The compression method keeps this feature. Besides traditional relational database techniques, additional query process technologies on compressed relations and for special structure for XML are presented. In this paper, technologies for XQuery process in compressed relational database are presented.

Keywords—XML, compression, query processing

I. INTRODUCTION

BECAUSE its extensibility and ability of representation of semistructured data, XML[1] database can represents more complex semantics than traditional database. But the application of XML database needs the support of efficiency storage. The storage strategies of XML includes three types, flat text, OO DBMS and relational DBMS.[2] As a type of mature DBMS, relational database has its special advantages such as efficiency query process, concurrency control and so on. Commercial relational database products are widely used by many users. Using them to support the storage and query of XML data will result in many advantages. There are several research about storing XML in [3-9, 15, 16].

A notable problem of existing storage structure of XML is the redundency of XML. An XML document can be divided into two parts, one is the context of data it represents, the other is structure of XML which is compounded by tags. Structure information is important for the semantics of XML document. However, lots of repetitive tags result in redundency. And the context that XML document represents also has redundency. Storing XML data in relational may result in additional redundency because of the decompositon of the schema. In order to support XML database efficiently, it is necessary to compress XML document in relational database.

Several XML compression technologies have been presented [10-14]. Some of them are only design for the compression the XML document without the requirement of query [10, 11]. The compression methods considering query in [12, 13, 14] are all the methods on original XML document. The query efficiency of them depends on the query process technologies on XML data. In [14], a compression method of the structure of XML is presented without considering of context.

In this paper, the compression technique special for XML document in relational database is presented. At first, a relational structure supporting XML document in various sizes and schemas is presented. This storage structure supports path query very well and adapt for query process on it. In order to make the relational adaptive to be compressed, vertical partition of the relation is used as the structure. And various types of context data are compressed respectively to gain large compress ratio.

The query process technology on compression relations is presented. Without considering compression instance, the query translation technology of XQuery[28] to query plan with relational algebra for common relation database is presented. Since the query optimization and query execution technologies of relational database are quite mature, they needs only a little improvement to support the compression.

The contributions of this paper include:
- The paper presents a relational storage structure of XML data adaptive for both compressing and querying XML data. The path is kept in the relation. Primary part of XPath[29] query can be processed in main memory. Vertical partition based physical storage structure is also designed to support compression.
- Compression technology for XML data in relational database is presented. A practical compression method is designed for the structure information of XML data in relational database.
- Context is separated and compressed based on data type and range of value which are obtained without schema information.
- Query translation technology from XQuery to relational is designed for XML query process on relational database. This technology is independent to the compression structure can be applied for querying XML in common relational database.
- Considering the usage of existing query optimization and execution technologies of relational, a little improvement is done for the special feature of our compress structure for XML data.

The remainder of this paper is organized as follows. In
section 2, we summary related work. In section 3, special relational storage for XML is presented, as well as compression technology on this sturcure. Query processing technology on compressed data is designed in section 4. Section 5 contains the result of our experiments. Finally, in Section 6, we summarize our work.

![Fig. 1 DTD of Example](image)

**Motive Example:** In order to explain our storage and query process strategy, an example of a bookshop is presented. The DTD of the example is shown in fig.1. In the DTD, price float type, number has integer type, name, title, author and publisher are in string type.

### II. RELATED WORK

Data compression is an important field of information theory. Data compression has some advantages such as reduction of storage size, saving network bandwidth and accelerating query processing by reducing times of I/O. With these advantages, compression technology plays an important role in database.

For XML data, there are two types of compression method. One is to separate context and tag. They are coded respectively. The two parts are assembled after coded. This type of compression technology keeps homomorphism between compressed XML document and original XML document. Therefore, most query process technologies directly to XML document are still not efficient to compressed XML document. XGRIND[12] and XPRESS[13] are of this type. These technologies have large compression ratio. But an important problem of them is that in order to support efficient XPath Query, some index is necessary, as will result in additional storage cost. When XML document is large, existing technologies of query processing are inefficient.

The other type of compression is to compress structure and context separately and store them respectively. XMILL[11] belongs to this type. XMILL stores data with same type as a unit and compress each unit using a special compression method. The compression ratio of XMILL is high. But query can not be processed on compressed data without decompressing the whole document. [15] presents a technology that compresses only structure of XML document. Compression of structure has excellent compression ratio. The problem of this technology is that the connection of structure information and context information results in additional storage. And without the compression of context, compression ratio of the whole document is limited.

There are many relational storage structures supporting XML documents. [3] and [4] considers large XML document and store XML document in the relations that are partitioned based on the schema of XML document. [5] and [15] are the extension of this method to self-tuning of the storage structure. These methods are not adaptive to the instance of storing many XML documents with various schemas and bring redundancy. Query process in these storage structure results in many join operations. It is hard to translation XQuery to SQL for various instance of schema. More adaptive storage structure and query translation is presented in [5] and [6]. The method is to store each edge of the XML document. The shortcoming is that the query process efficiency on this storage is low. [7, 16] use interval encoding to represent the ancestor and descendence. XRel[5] is to encode the path and store the code and the extent of each node. But the translation detail of XQuery is not presented in [5]. In [9], a storage structure is presented to converting XQuery to range query. It accelerates XML steps. But none of the above structures consider the problem of compression.

It is noted compression method that can be used to compress database should lossless one. There are much research work about compressed relational database[17-23]. [17] gives many methods to compress database. A block-based compression method is presented in [18]. In [19], an order preserving string compression method is presented, which is useful for the compression of database with text. [20] uses vector quantization to compress the repetitive in recorders. The query optimization strategy is presented in [12]. With the strategy, compressed database can be changed to a self-change one. [22] presents a loss compression method with the usage of data mining technology. The compression method special for 2-ary relation is introduced in [23]. Some technologies of these papers are referenced during the design of our compression method of relational database.

There are also many technologies for lossless compression of general data. Huffman[24] is a classical compression technology. It assigns shorter code to most frequently appearing symbols and longer code to less frequently appearing symbols. BWT[25] is a new technology of compression text. (introduction to BWT) Dictionary coding is effective when only a small number of words exist. The idea of dictionary coding is to assign an integer value to each new word from input data. The differential encoding method, also called delta encoding, replaces a data item with a code value that defines its relationship to a specific data item. Differential encoding is adaptive to compression of numerical value. All these methods can be selected to compress some part of XML data lossless.

### III. COMPRESSION OF RELATIONAL DATABASE WITH XQUERY SUPPORT

In this section, the storage of relational database with XQuery support is presented. Based on the logical structure, a special physical storage with compression technology is designed.
A. Store XML in Relational Database

Beyond existing relational structure storing XML document, a storage structure adapt for XQuery process on XML data. Some redundancy is permitted because compression technology will be applied to eliminate the redundancy without effect to the query process.

Each node of XML document can be uniform identified by its preorder rank. Each path from root to leaf node can be represented as the list of the preorder ranks of nodes in this path. Based on this fact, the storage structure of XML document is to store the preorder ranks of a path with the value. The paths with same path schema $s$ are stored together as a relation, which is called path relation of $s$, $r_{s}$ for short. If $l_s$ is the length of path schema $s$, $r_{s}$ has $l_s+1$ columns, the first is columns are the preorder rank of the path with names the name of corresponding tag and the last column with the name value is the value of the leaf node. It is supposed that for a record a in $r_{s}$, the column i of it is $a(i)$.

The tags in XML document is encoded and stored in a separate relation, $r_{tag}$. Each path schema can be presented as the list of tags. All Path schemas of leaf nodes in XML document are also encoded. The content of path schema with its code is stored in a relation $r_{p}$, in which the content of path schema is represented as a string with each char having ASCII value of the tag nodes’ codes. The schemas of $r_{s}$, $r_{tag}$ and $r_{p}$ are shown in table 1, table2 and table 3.

Two properties of XML document stored in this structure are presented as Theorem1 and Theorem2. Before the two theorems are presented. A property of preorderded rank stored in $r_{s}$ is shown in Lemma1.

**Lemma1** Two records a and b in an $r_{s}$, when $i<ls+1$, if $a(i)=b(i)$, $a(i-1)=b(i-1)$.

**Proof** $a(i)=b(i)$ represents the ith nodes of path corresponding to record a and b are the same. It is supposed to be $n_{i}$. In a XML tree, the parent of a node is uniform. The preorder rank of $n_{i}$’s parent is $preorder(n_{i-1})=a(i-1)=b(i-1)$.

All the records in $r_{s}$ can be sorted by the first $l_s$ columns with the order from 1 to $l_s$. Thus, a property of sorted $r_{s}$ is shown in Therom1

**Theorem1** all records $r_{s}$ which has been sorted by the first $l_s$ columns with the order from 1 to $l_s$ with the same values of the ith ($0<i<l_{s}+1$) column must be coterminous.

**Proof** The theorem is to be proved in mathematic induction. When $i=1$, the 1st column of $r_{s}$ represents the preorder rank of root of XML tree, which is uniform in a XML tree.

If when $i=n$, the theorem holds, it is to be proved that when $i=n+1$, the theorem holds.

By Lemma1, the values of the parents of the ith column with the same value of all the records must be same. By the induction hypothesis and the sorted rules that the records with the same values of the ith column are sorted by the $i+1$ column, records in sorted $r_{s}$ with the same values of the ith ($0<i<l_{s}+1$) column must be coterminous. $\blacksquare$

A property of the order of nodes in the same level is presented in Theorem2.

**Theorem2** In the same column of a sorted relation, the order of the node to the same parent node is kept.

**Proof** Since the model of XML is an ordered labeled the tree, the prerank of a node is smaller than that of its following sibling. Hence in an ordered column, the prerank for the node precedes that of its following sibling. $\blacksquare$

B. Compression of the Relational Database

In this section, the compression method of the relational database is presented. Based on special physical storage for compression, the columns of the path and that of the value are compressed respectively.

1) The Physical Storage adaptive for compression

The data with the same data type and range of value is easy to compress. An observation of $r_{s}$ for various $s$ following the storage schema in 3.1 shows that the redundancy exists in that the some leaves may share the same ancestor, the code of which is stored many times. To solve this problem, the physical storage is the vertical partition of the relation so that the preorder rank of the same type of data can be gathered together. And the values of the leaf nodes with the same path schema often have the same data type and range of value. Storing the values together makes the context information easy to compress. Another advantage of vertical partition is that the acceleration of projection operation, which is often used during the process of XQuery in relations to select special nodes binding to variable in variable forest.

2) The compression of Path property

With the assurance of Therom1, on the physical storage of vertical partition, the value of each path property can be compressed the following compression method. For the storage of the ith column, the difference of each value and the former value to it is obtained. And the value with the number of the number of times the value continues to exist is stored. Although the method is simple, the wide the column can be shortened and the repetitive values can be eliminated.

**Example 3.1:** The array $1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 15$ is converted to $1, 1, 1, 1, 2, 1, 2, 2, 2$ and compressed as $1, 5, 1, 1, 1, 2, 1, 2, 2, 2$.

3) The compression of the value column

The value column of each $r_{s}$ can also be compressed. The

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>int</td>
<td>code of the path</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>name of the tag</td>
</tr>
<tr>
<td>code</td>
<td>int</td>
<td>code of the tag</td>
</tr>
</tbody>
</table>
The type of the column of value is judged by statistics. The data value column is compressed by its data type and value of range. Hence at first, each path type is supposed as enumeration type. The values of the path class are stored. If the number of the values is larger than a threshold, the path type is considered as float. If existing value is all number but the first character is not '0', path type is considered as integer. If existing values is larger than a threshold, the path type is considered as text. The range of value should be recorded with path type integer and float.

Compressors to various path types are as follows:

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Path type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u8</td>
<td>Integer</td>
</tr>
<tr>
<td>u16</td>
<td>Integer</td>
</tr>
<tr>
<td>u32</td>
<td>Integer</td>
</tr>
<tr>
<td>F32</td>
<td>Float</td>
</tr>
<tr>
<td>Suffix encoder</td>
<td>Enumerate typed data</td>
</tr>
<tr>
<td>Dictionary encoder</td>
<td>Enumerate typed data</td>
</tr>
<tr>
<td>u8</td>
<td>Integer</td>
</tr>
<tr>
<td>u16</td>
<td>Integer</td>
</tr>
<tr>
<td>u32</td>
<td>Integer</td>
</tr>
<tr>
<td>Dictionary encoder</td>
<td>Enumerate typed data</td>
</tr>
</tbody>
</table>

IV. QUERY PROCESSING ON COMPRESSED DATABASE

In this section, query process technology on the database compressed with the method of is presented.

A. An Overview of Query Processing

Basic idea of query process is to express query plan with relational operators. XQuery is expressed as relational The steps of query process are shown in fig.2

Before query process, XQuery expression should be parsed and preprocessed. Besides computing expression with only constant and reduce constant restriction,

The first step of query translator is to extract the variables and their relationship from the query. They are represented as a tree, as is defined as variable tree. Each node represents a variable with its path in the query. More than one tree can be extracted from the query. The results of the trees are to be join in semantics. All of variable trees of a query form a forest, as is defined as variable forest.

And then the condition of the query is transformed into the form of DNF. Each clause of DNF is CNF. Each clause is to be generated on forest. For the optimization, some clause should be merged.

The next step is to add each CNF clause of restriction clause to the forest. Thus, several restricted VFs generate. The final result is the unions of these restricted VFs. Additional nodes are added to the forest. Restrictions are added to existing nodes. The generation and adding restriction process of VF is presented in section 4.2.

Query plan is generated from restricted VFs. The query plan generation method is presented in section 4.3. And the query plan is executed to obtain the information from compressed relational database. The query plan is represented by relational operators. The implementations of these operators on compressed storage are quite like those on common relational database. The improvements of the implementation of operators are introduced in section 4.4. The last step is to wrap the final result to the format of XML with the result schema.

B. Generation of Variable Forest

1) Generation of Original Variable Forest

The first step is to convert the query to variable forest containing several variable trees, which represents the relationship between the variables in the query.

Definition 4.1 Variable Tree (VT for short): A Variable Tree of a query Q T_V(r, V, E), where r is the root of the tree, V is the set of nodes of the tree and E is the edge of the tree such that:

Each v  V can be represented as (ID, var, path, res, isresult, agg), where ID is the identify of the v which is uniform in the whole variable forest, var is the name of variable v binding to, path is the XPath expression of the var in Q and res is the restriction of this var. If v is a leaf node, res is the comparison of var and a constant. Otherwise res is the existing restriction of some child of v in T_V or the relationship between the children of v in T_V. property isresult represents if the variable exists in the final result. Property agg represents the aggregation function with id of group by nodes on this node.

Each e  E is labeled pc(for parent child relationship), ad(for ancestor descendant relationship), ad* (for self-or descendant relationship).
Definition 4.2 Variable Forest (VF for short): A set of variable trees $F_Q=(S_I, R)$, where $S_I$ is the set of variable trees in $F_Q$ and $R$ is the set of relationships between the trees in $S_I$ in $F_Q$, $R$ is represented as the restriction to connect the VTs.

Example 4.1 For the query to “for each book published by MIT Press find the number of book published by Prince Hall with the same title as it for $Sa$ in document("bookshop.xml")/book $Q1$

\[
\begin{align*}
&\text{let } S_c:=\text{for } Sb \text{ in document("bookshop.xml")/book} \\
&\text{where } S_a/title=Sb/title \text{ and } Sb/publisher='Prince Hall' \\
&\text{result } <\text{item}>Sb</text/item> \\
&\text{where } S_a/Publisher='MIT Press' \\
&\text{result } <\text{result}> \\
&<\text{book1}>S_a/title</book1> \\
&<\text{number}>count(S_c)</number>$/\text{number}> \\
&<\text{result}>
\end{align*}
\]

Fig. 3 Initial VF of Q1

The VF of Q1 has two VTs. The initial VF of Q1 is shown in fig.3, in which $v1$ represents a virtual variable.

The generation of a VF is to extract the variables from query and construct VTs. At first, for each independent variable that is obtained from for and let clause, a VT is built with ID, var and path expression. It is noted that the path expression is absolute path from root to the node var standing for.

1) Binding Restrictions to VF

Each DNF $r_i=r_1 \wedge r_2 \wedge \ldots \wedge r_m$ of restriction is then added to the VF. The restricted VF generated by $r_i$ is $V_{Fi}$. Internal plan $p_i$ is translated from each VF, respectively. Final result is the union of the result of all the internal plans.

The process of adding $r_i$ to VF is to apply each $r_i$ to VF. The application is processed according to the variables in each $r_i$.

- If $r_i$ contains just one variable or XPath expression that has been bound to some node $n$ in VF, the restriction $r_i$ is added to res property of $n$.
- If $r_i$ contains variables or XPath expressions that have not been bound to any node $n$ in VF, a node $n$ binding to a virtual variable is added to corresponding VT in VF. In this instance, if $r_i$ has only one virtual variable, the restriction is added to res property of $n$ directly.
- If $r_i$ contains more than one variable or XPath expression, all variables or XPath expressions are bound to the VF at first. And then, each variable $v$ is check to determine which VT it belongs to. If all the variables belong to the same VT $T_v$, restriction $r_i$ is added to the nearest ancestor of them in $T_v$. Otherwise, a connection node is built to connect all the VTs that $r_i$ has variable binding to.

Example 4.2: When the restriction in where clause of Q1 is applied to VF in Example 4.1, the VF is converted to the form as fig.4.

C. Construct Query Plan from Restricted Variable Forest

The query plan of compressed relational database is in the same form as that of common relational database[26].

1) Generation of Logical Plan

The generation of logical plan is to traverse bound VF and generation corresponding logical plan for each node. The translation result may have more than one operator, such that the relationships among them are defined. The translated result of one node $n$ in VF can be considered as a box with its inputs the result of the children of $n$ and the output of $n$ is used as the input of $n$’s parent.

The traversal is the hybrid of preorder and postorder. Each internal node in VF is to be accessed two times during the traversal. During the traversal, a stack is used to record the variable to be projected in the path. When first time to access an internal node $n$, if var of $n$ has more than one child or agg property of $n$ is not NULL or it appears in res property of its ancestors, the tag number corresponding to $n$ is pushed in to project stack. At the same time, the set of various ids that all its ancestors need are recorded in $n$.

When a leaf node is met, the names of the relations of leaves of each VT in VF are obtained by matching the path property of the leaf node to the path context column in $r_p$. These relations are represented as $r_{11}$, $r_{12}$, …, $r_{1n}$. A projection operation is added to each $r_i$ with projection columns the union of current project stack $t$, the tag number of the leaf and value. A selection operator is then added above projection operators if there is some restriction binding to the node. A projection precedes selection is because that the cost of projection operator is constant. If more than one relation matches the path expression of this node, an addition union operation is added to the top of the query plan for each single relation.

When the second time visiting an internal node $n$, a join operation is added with the objects the results of the operations corresponding to its children. The join restrictions include not only the res property but also the restriction of the equation of the column corresponding to the tag number of $n$. The selection of the order of the join is left for the generation of physical plan.
A projection operation is added to the top of
A join operator may be generated from the top level
restriction node with the restriction stated in this node. The
projection of final result and aggregation operation is added
to the top level.

Example 4.3: The logical query plan of the VF in
example4.2 is shown in fig.5

2) Generation of Physical Plan
The generation of physical plan is the improvement of that
of common relational database with the data properties in
the storage structure. And the cost estimated also needs a little
improvement. Here only the improvements are presented.

First of all, based on Theorom1, arbitrary column except
value is ordered. Thus, the join with restriction of only columns
of preorder ranks is naturally merge sort join with the cost only
that of merge operation without that of sort. In order to keep the
order during the process, if there is union above the logic plan
according to a node in VF, the physical operator of the union is
multi-way merge union.

Since our operator is on compressed data, the additional
operators of compression and decompression are necessary. In
the scan for selection operator or join operator by the restriction
of value column, decompression is necessary only when the
string matching operation is applied to the column with type
text. Otherwise, the value in restriction is compressed to compare
with the compressed value in value column. The cost of
compression and decompression operator is also depends on
the compress method and the size of compression data. The
cost of scan operation should consider the reduction of I/O. The
reduction depends on the estimate of compression ratio of the
compression method and the statistics of the data.

D. Execution of Query Plan
The implementation of query operators in a query plan is also
quite similar to that of common relational database except
compression operators should be added to the implementation.

The sequence scan operation on compressed data should be
considered. When the operation scan is executing, on the
compressed column of preorder rank in path, the preorder rank
of the node can be extracted from compressed data in a cup time
of adding. And when the operation scan is by the restriction of
value, the value in restriction is compressed in the compression
method of corresponding value column. It is noted that
performing scan on the vertical partitioned table needs to keep
the synchronization of the block of all necessary columns.

V. EXPERIMENTS

A. Experimental Environment
The experiments are performed on a PC with AMD 1G CPU
and 256M main memory. Operation system is Microsoft
windows XP Professional and develop platform is Visual C++
6.0. Relations and temporal tables are stored as files in local
computer.

Data Set: XMark[27] is a famous benchmark of XML data
management. It covers many features of XQuery. We use
XMark as our data set with various factor.

Queries: All the 20 queries presented in XMark are used for
experiments.

B. Performance Result and Analysis
Three measurements are used to represent the performance of
the compression:
RE ratio(relation expand ratio)=relations size/original size-1
RC ratio(relation compression ratio)=1-size of compressed
relations/ size of relations
DC ratio(document compression ratio)=1-size of
compressed relations/size of original XML document

The result of compression is in table5.
The compression ratio of XMark data sets with factor above
to 0.1 is a similar. The reason is that they are automatically
generated and have similar ratios of various data types and tags.

To test the efficiency of query process on compressed
data, the 20 queries in XMark is applied to XMark data set with
factor 1.0. The result of query process is shown in table 6.

From the result, the efficiency of the query processing on compressed
data is acceptance. Q8-Q11 are slower because the
join of XML trees results many joins with many temporal
tables. Especially 14 temporal tables are resulted during
processing of Q10. Q14 is slower because that we do not
optimize for the string matching.

If the compression and query processing technology is
embedded to an existing database system with optimization of
execution of joins and temporal tables, higher efficiency can be

VI. CONCLUSIONS

In this paper, the strategy of storing and querying XML
document in compressed relational compressed database is
presented. On one hand, the strategy has the relation database’s
advantage of easy query process. On the other hand, it reduces
the storage needed to store XML data, as well as the disk I/O
during query processing. A special relational structure for
storing XML is presented. And the compression on it obtains a
good compress ratio. The technique of converting XQuery to logical query plan of relational is designed as the basic of query on the XML data in compressed relations. During query process, most techniques of query process in relational database continue to be used with a little improvement, which is also introduced in this paper.

This compression method adapts to very large XML databases with the application of XML warehouse and to be embedded to existing relational databases for better supporting of XML data.

**REFERENCES**


[21] Hongzhi Wang. Hongzhi Wang received his PhD in computer science from Harbin Institute of Technology in 2008. He is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is XML data management and information integration.

[22] Jiazhong Li is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area includes parallel database, sensor network, data mining, data warehouse, compressed database and XML data management.

[23] Hong Gao received her PHD in computer science from Harbin Institute of Technology in 2004. He is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is graph database and data warehouse.

### Table V Result of Compression

<table>
<thead>
<tr>
<th>Query</th>
<th>DS Factor</th>
<th>Size</th>
<th>Rel. size</th>
<th>RE ratio</th>
<th>Com. size</th>
<th>RC ratio</th>
<th>DC ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMark</td>
<td>0.0001</td>
<td>33.13K</td>
<td>36.532K</td>
<td>10.2%</td>
<td>29.98K</td>
<td>17.9%</td>
<td>9.5%</td>
</tr>
<tr>
<td>XMark</td>
<td>0.1</td>
<td>11.13M</td>
<td>11.71M</td>
<td>5.3%</td>
<td>5.26M</td>
<td>55.0%</td>
<td>52.70%</td>
</tr>
<tr>
<td>XMark</td>
<td>0.2</td>
<td>22.43M</td>
<td>23.63M</td>
<td>5.4%</td>
<td>10.44M</td>
<td>55.84%</td>
<td>53.46%</td>
</tr>
<tr>
<td>XMark</td>
<td>0.4</td>
<td>46.89M</td>
<td>44.53M</td>
<td>5.3%</td>
<td>20.54M</td>
<td>56.19%</td>
<td>53.87%</td>
</tr>
<tr>
<td>XMark</td>
<td>0.8</td>
<td>89.23M</td>
<td>93.95M</td>
<td>5.3%</td>
<td>40.85M</td>
<td>56.48%</td>
<td>54.18%</td>
</tr>
<tr>
<td>XMark</td>
<td>1.0</td>
<td>111.1M</td>
<td>116.9M</td>
<td>5.3%</td>
<td>50.8M</td>
<td>56.51%</td>
<td>54.22%</td>
</tr>
<tr>
<td>Shakes</td>
<td>/</td>
<td>7.50M</td>
<td>7.97M</td>
<td>6.2%</td>
<td>2.98M</td>
<td>62.60%</td>
<td>60.3%</td>
</tr>
</tbody>
</table>

### Table VI The Performance of Query Process on

<table>
<thead>
<tr>
<th>Query</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(s)</td>
<td>3.012</td>
<td>28.352</td>
<td>2.565</td>
<td>97.002</td>
<td>3.546</td>
<td>0.03</td>
<td>0.52</td>
<td>79.691</td>
<td>358.21</td>
<td>1102.35</td>
</tr>
<tr>
<td>Time(s)</td>
<td>411.12</td>
<td>409.55</td>
<td>1.713</td>
<td>475.35</td>
<td>8.152</td>
<td>1.058</td>
<td>0.581</td>
<td>1.465</td>
<td>3.390</td>
<td>0.791</td>
</tr>
</tbody>
</table>