DocBase – The INEX Evaluation Experience

Sriram Mohan[1] and Arijit Sengupta[2]

[1] Computer Science Department, Indiana University,
Bloomington, IN 47405, USA
srmohan@cs.indiana.edu

[2] Information Systems Department, Kelley School of Business,
Indiana University, Bloomington, IN 47405, USA
asengupt@indiana.edu
http://www.kelley.iu.edu/asengupt

Abstract. Can a system designed primarily for the purpose of database-type storage and retrieval be used for information-retrieval tasks? This was one of the questions that led us to participate in the INEX 2004 initiative. DocBase, a prototype database system developed initially for SGML, and adapted to work with XML, was used for the purpose of answering the queries. DocBase uses DSQL, an adaptation of SQL to provide a mechanism for querying XML using existing database and indexing technologies. The INEX evaluation experience was encouraging - although it did show the limitations of database query languages for classic information retrieval tasks, it also demonstrated that several interesting results can be obtained by using database query languages for information retrieval, especially for queries involving both content and structure. Our results demonstrate the adaptability and scalability of a database system for processing IR queries.

1 Introduction

Database management systems (DBMS) are designed for the purpose of efficient management of data in low-level storage devices. DBMS technology excels in the processing of transactions in multi-user environments, ensuring the quality and integrity of data in the presence of adverse conditions such as concurrent access and modification, as well as unpredicted system failure. DBMS provides high level languages and models to design, understand, and manipulate the data. On the other hand, the process of information retrieval is concerned with the extraction of the most relevant information from a data repository, with very little assistance from users. The classic information retrieval method is keyword search, where the objective is to retrieve information using just a few keywords. A critical question is whether these two apparently diverse technologies can be brought together for the purpose of retrieving information from future document repositories? What can each field learn from the other? What can each field use to better achieve its goals with knowledge from the other field? Such questions drove us to test DocBase, a system primarily designed for the purpose of SQL-like query processing on documents, in the INEX framework.

XML is fast becoming one of the most commonly used document representation formats. In less than ten years of its conception, it has become the leading technology for document representation for the next generation of applications. One of the primary differences between XML and other document formats (*e.g.*, word processing formats, HTML) is that XML incorporates logical structure in the documents. XML embeds additional structural information (meta-data) with the text content of the documents. XML applications need to appropriately utilize this structure for the purpose of information retrieval. This provides a potential for improving the recall and relevance of retrieved information when content is mixed with structure in the query. However, when a user searches only for content without the structure, such structural information may not be immediately useful. The retrieval application however can use such structure to improve recall by utilizing past searches and other statistical information gathered over time.

1.1 Differences Between DBMS and IR

Databases have always used structure in searches. In fact, a database search that looks for keywords anywhere in the database is an extremely complex query, not immediately supported in most current database management systems. Such searches, even if implemented, are a challenge to execute efficiently. However, since underneath the document structures in XML, the data is predominantly textual, one can potentially improve the types of retrieval operations that are possible, by using database-type searches. We intend to determine the types of functionality that we gain when a database query language is used for the purpose of information retrieval of both structure and content.

Databases make efficient use of low level storage indexes such as B+ trees to retrieve data quickly. Indexes are used in information retrieval as well (*e.g.*, "Inverted tree index" [1, 2], Patricia tree index [3]). The main difference between index use in databases and IR lies in the fact the IR indexes are used for full-text retrieval, whereas database indexes are used for speeding up retrieval in specific structures.

The rest of the paper is organized as follows. In the rest of this section, we motivate our participation in the INEX initiative. Section 2, introduces the DocBase system, and subsequently Section 3, introduces DSQL, the query language used by DocBase, and Section 4 describes how it can be used for the purpose of information retrieval. Section 5 describes how the data was prepared for indexing, and Section 6 describes how DocBase was adapted to work as an IR engine for the data. Section 7 presents the relevance and timing results and we conclude with a discussion on the lessons learned from this experience in Section 8.

1.2 Motivation

The power of a DBMS comes from its ability to perform data manipulation and management. But if the underlying data is essentially static, i.e., changes are not frequently made to the data, then the use of databases for retrieval purposes

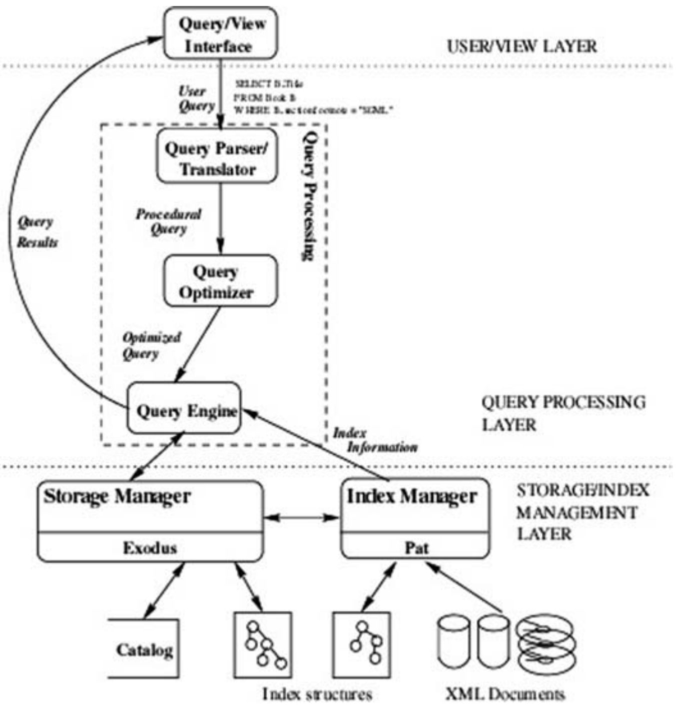

Fig. 1. The DocBase Architecture

might seemingly introduce more overhead than efficiency. However, the fact that a database query language potentially has the capability of restructuring and creating new structures while retrieving information, can still be useful for the purpose of information retrieval. Such creation and restructuring of information in ad hoc queries is not typically available in IR. Because of the efficient use of storage, indexes, and corresponding retrieval methods, database retrieval is typically highly efficient and scalable to very high volumes of data.

The motivation behind this work can be briefly stated as: (i) to identify areas where a database language could augment and potentially improve information retrieval methods, and (ii) to determine areas where a database query language can itself be altered and augmented to allow the possibility of information retrieval. This paper presents our findings from the INEX evaluation initiative from the perspective of the above objectives, and to show how some of the advantages and disadvantages of database technologies for information retrieval applied to the use of DocBase as an IR engine.

2 DocBase – A Database for Documents

DocBase is a research prototype system, developed in 1997 as part of a Ph.D. dissertation [4]. Although originally developed for the purpose of querying SGML documents, DocBase was modified to allow XML indexing and retrieval as well. The primary intent of DocBase is the storage and retrieval of structured data. The storage is handled by one of several possible storage manager components, while the retrieval is handled by an internal query engine, that uses an external indexing system if available. The DocBase architecture block diagram is shown in Figure 1. Users specify queries using a command-line interface, or by using QBT (Query By Templates), a specialized graphical interface for the purpose of querying structured data using the shape of the information [5], or by specifying the query in DSQL - DocBase's internal query language [6].

DocBase uses a 3-layer architecture similar to most DBMS. At the top layer (user/view layer) sits the query interface which allows users to specify queries using DSQL or QBT, and provides the user with a view of the result of the query. In the middle is the query processing layer, which includes a query parser, a rudimentary optimizer, and the query engine which evaluates the optimized query. At the lowest level is the physical storage and index management layer. There are two components in this layer. The storage manager component deals with keeping track of the data in the database, and provides safe access, modification, and management of the data. The index management component includes one or more indexes that can be created based on the data. The current implementation of DocBase uses Exodus, a storage management system developed in Wisconsin [7] and OpenText's Pat [8] software to create indexes. The query processing in DocBase is performed by (i) parsing the query and decomposing it into algerbaic atomic operations, (ii) optimizing the query to separate operations that can be performed using the created indexes, and the operations that require external processing. The individual operations are performed and composed as much as possible, and the intermediate results are combined in memory for the purpose of final packaging to the view layer.

3 DSQL – An SQL for Documents

DSQL is the query language used in DocBase and is entirely based on SQL, and has all the properties of SQL that make SQL the most popular database language today. DSQL uses the same syntax as SQL, and in fact, we can prove that DSQL queries are syntactically equivalent to the corresponding SQL queries, when the underlying data structure is flat. For flat structures, DSQL does not introduce any additional complexity over SQL - and has the same PTIME and LOGSPACE complexity as SQL. However, when the embedded structure is available and relevant, DSQL allows the creation and traversal of such structure, and hence DSQL queries can be used with advantages in the information retrieval context.

DSQL has the same basic structure as SQL, with recognizable `SELECT, FROM` and `WHERE` clauses for retrieval, and `GROUP BY, HAVING` and `ORDER BY` clauses for grouping and ordering the results. Moreover, queries can be nested using the standard `IN`, `EXISTS` or `NOT EXISTS` sub queries. As in SQL, only the `SELECT` and the `FROM` clauses are required. The other clauses are optional. Also, multiple

SELECT queries can be combined using standard set operations (Union, Intersect, Minus). The following section describes in detail the above constructs of DSQL.

3.1 The SELECT Clause

The SELECT clause in DSQL has the same major structure as SQL. The main difference is that it can create complex structures, and can traverse paths. To keep the language simple and close to SQL, generation of attributes has not been included in the base language. In fact, the formal specification of the language uses a form of XML known as ENF (Element Normal Form) [9], that ensures that any XML document with attributes can be re-written without the use of attributes and vice versa. DSQL provides a rudimentary element construction technique by specifying the element name and its contents within <>. Some examples of the select clause are given below:

```
SELECT *
SELECT output<*>
SELECT result<B.title, B.author>
SELECT booklist<B.author, books<B.title, B.year>>
```

The SELECT clause allows the creation of structures with arbitrary levels of nesting. Notice that grouping is not inherent in this specification and is, instead, the task of the GROUP BY clause.

3.2 The FROM Clause

The FROM clause allows the specification of the source of the information. The FROM clause can use URL sources, as well as aliased sources within the database. Some examples of the FROM clause are given below:

```
FROM books.xml B
FROM books.xml B, authors.xml A
FROM http://www.mycompany.com/docs/invoices.xml V, V..items I
```

3.3 The WHERE Clause

WHERE conditions in DSQL are similar to those in SQL. The main difference in semantics is due to paths, *i.e.*, all expressions in DSQL are path expressions, so operators are often set operators. For example, consider the following query:

```
SELECT result<B.title>
FROM bibdb..book B
WHERE  B.title = ‘Extending SQL’
```

The WHERE expression evaluates to true if the path expression yields a singleton set containing an atom identical to ‘Extending SQL’. Set membership operations such as in and contains are also available in DSQL.

3.4 Grouping and Ordering Clauses

SQL has several ways of specifying post query formatting and layout generation. The following are the grouping and ordering specifications in DSQL:

- `ORDER BY` (sorting): DSQL has the same semantics for ORDER BY as SQL. The expressions in the `SELECT` clause can be ordered by expressions in the ORDER BY clause, regardless of whether or not the ordering expressions appear in the `SELECT` clause, as long as the expressions are logically related, i.e., an ordering is possible using the ordering expression.
- Aggregate functions: DSQL supports the same five basic aggregate functions as SQL (`sum, count, min, max, avg`).
- `GROUP BY`: In DSQL, `GROUP BY` is a restructuring operation but unlike SQL, the aggregate function is optional. Moreover, multiple GROUP BY clauses can be specified for a single SELECT query.
- `IN/EXISTS/NOT EXISTS`: As in SQL, queries can be nested using sub queries. The semantics of sub queries remains the same in DSQL.

4 Information Retrieval Tasks Using DSQL

Section 6 describes how the INEX CO and CAS queries were performed in DocBase. Queries specified using the INEX format need to be first translated to DSQL before they can be evaluated in DocBase. Here we quickly motivate the translation process of the queries, and the reasoning for such alteration of the queries. We also discuss how such alteration potentially alters the semantics of the queries.

4.1 Keyword-Based or Content-Only (CO) Queries

DocBase technically, does not allow completely structure-free keyword based queries. Structure-free queries are difficult, and are not supported in standard database query languages such as SQL. With an immediate glance, it may be construed that DSQL also is unable to perform keyword-based queries. DSQL queries must specify where the keywords are being searched for, and what to retrieve as the result of the search.This implies that along with keywords, the queries should also include the structure regions of the documents where these keywords must be found. In the case keywords may appear anywhere in the document, the query must specify the bounding structure as the top level of the document, which is often termed the "root node" or the "root element" of the document. This is made possible by the hierarchical structure of XML. Of course, in the case of multiple documents, this implicitly assumes that all the documents where information is sought from have the same structure, or at the least, the same root element. In the case the repository contains documents with different root elements, a disjunction of these root elements must be specified in the query, so a completely structure-independent query may not be possible. Note that XPath uses the concept of a structure-independent root element (the '/' element) which can be used for this purpose. However, DocBase does not support this "super-root" element.

DSQL and other database query languages do not have a mechanism to specify the actual "point of match" of a search keyword. For example, if the query is a search for the keyword "XML" in a document repository, the database search might retrieve the documents that contain the keyword "XML", whereas an IR technique will retrieve all points of match of the keyword. Obviously, the number of matches retrieved by the database retrieval will be less than the number of matches retrieved by the IR method, although the documents retrieved by the database method will still contain all the matches returned by IR.

If the content-only query includes multiple keywords, potentially combined using boolean expressions, the actual "point of match" may not be easily identifiable, since one match might span several match points within same document and possibly across different structural regions. For example, a search on "XML or Information retrieval" may result in some documents where both the keywords appear in potentially different regions of the document. In such cases, it is not immediately apparent which of these match points is the actual result of the search. In the database search, the return point is always specified in the query, and hence always well-defined.

CO queries are translated to DSQL by simply searching for all the keywords bounded by the root element, and returning the root element as the retrieved element as well. So, for example, for a search in a poetry database (root element "poem") if the CO search is *'love' AND 'heart' AND NOT 'hate'*, then the equivalent DSQL Query is:

```
Select poem
From   poetry
Where  poem = 'love' and poem = 'heart' and not poem = 'hate'
```

The return region can be varied depending on what level of detail should be shown in the result, although the semantics of the query might change slightly based on what is returned. Actual translation of selected INEX queries will be shown in Section 6.

4.2 Content and Structure (CAS) Queries

CAS queries are more natural in databases. In such queries, the content that is being searched for is augmented with the logical region of the document where the content should be found. For example, Searching for "XML" in "document title" should result in documents where the the keyword "XML" appears in the document title. DSQL is better designed for the purpose of performing such queries. However, the same limitations regarding retrieving the actual point of match still apply. For CAS queries, a "point of match" is potentially more problematic, since the semantics of the language must determine whether the match should return the position of the keyword or the structure where the match was obtained (the position of the document title or the position of the keyword "XML" in the above example). The requirement in DSQL of always specifying the region to retrieve leads to more concrete semantics for such queries.CAS Queries are translated to DSQL using a logical translation of the CAS conditions into conditions in the WHERE clause. Actual translations of selected INEX CAS queries are shown in Section 6.

4.3 Approximation

Like any other database query language, DSQL queries are very rigid. If the query asks for "XML", the retrieved results would only have "XML". There is never any approximation, never any semantics associated with the search keywords. For example, the result will not contain documents with the phrase "Extensible Markup Language", although semantically they are the same. Several information retrieval methods incorporate such semantics in searches, so that the results not only contain the documents using the exact search, but also documents containing approximate searches. Moreover, if the search condition is long, such as "complexity analysis of computationally complete query languages", information retrieval techniques may use a disjunction of keywords and phrases from the search phrase to retrieve the result. DSQL does not have such functionality, so for searches involving such long search phrases, we often divided up the search phrase into multiple keywords, and used a conjunction of these keywords, instead of a disjunction, so that only documents containing all the chosen keywords would be matched. Searches in DSQL are, however, always partial searches, so a search for "XML" would retrieve "XML database systems" as well.

4.4 Ranking

Ranking is another aspect where database languages are lacking. In database terminology, a data instance either matches a query or it doesn't, and the query simply returns the items in the database that match the query conditions. However, ranking of the results is an important aspect of information retrieval. DocBase has no way of ranking the retrieved results. This is definitely an area where database systems and languages need to improve in order to provide enough value in information retrieval.

5 Preparing and Indexing the INEX Data

DocBase is capable of using any index structure that is appropriate for text and tree-based searches. The current implementation of DocBase uses OpenText 5 "PAT" indexing engine [8]. The INEX data was parsed by the built-in tagged document parser of OpenText Parser, and the required catalog entries were created in order to enable querying of the INEX data using DocBase. Because of the nature of the Pat indexes, most of the tree traversals can be performed by DocBase except those of an immediate child or immediate parent. Since the version of Open Text used was somewhat dated and did not have direct support for XML, the INEX data needed some preprocessing to enable it to be queried using DocBase. The following section will briefly explain the processes that were carried out on the INEX data collection to reach the "Query - Capability" state.

5.1 Indexing the Data Using Patricia Trees

The Design of DocBase focused on query processing capabilities, so incorporating new data in DocBase usually involves specifying a number of required

Table 1. A Sample Data Dictionary Control File

```
<Text>
  <MfsFiles>
    <FileMap>inex</FileMap>
    <FilterChain>
      <FileGroup>
        <MfsDir>./xml/an/1995</MfsDir>
        <MfsFile>*.xml</MfsFile>
        <MfsExpand>file</MfsExpand>
      </FileGroup>
    </FilterChain>
  </MfsFiles>
</Text>
```

configuration parameters. These parameters have to be specified using several predefined control files created specifically for this purpose. Patricia trees can handle different kinds of database formats with differing indexing schemes available to best suit the chosen data format. DocBase assumes that all the available data is in ASCII format and needs the following 3 control files:

1. Data Dictionary Control File: The Data dictionary control file is used to define the location of the database files and other database specific information. The control file consists of a number of segments each of which is specific to a different property of the associated database. Since we are dealing with ASCII format or in other words textual data, the only segment that needs to be modified is the Text control segment delineated by the `<Text> </Text>` tags in the data dictionary. Within the text control segment, one has to specify the source directories containing the actual data files and the format for the same. A sample Data Dictionary control file is shown in Table 1.
2. Region Tags Control File: The region tags control file is used in cases where the source data has tags delimiting various regions in the text on which source queries will be performed. This is required for indexing efficiency, since not all XML tags used in the document would be suitable for searching, and hence providing a small set of indexable tags would improve the efficiency of the searches. The tags specified in the region tags file are used to generate region indices that can be used to facilitate text queries that search for phrases within specified regions. Although tags not specified in the region index would still be searchable, region searches are much more efficient when a region index is created. In order to automatically generate this region file, we used a small XSL script that runs through the DTD/Schema associated with the source data and automatically determines the tags that are present in the document.
3. Regions Configuration Control File: The regions configuration file consists of one or more region segments, one for each tag file in use. This control file is used to specify the output format from the indexing structure(For Example

ASCII) and also the name of the tag file whose contents are used to build the data region indices.

It is easy to generate the Pat Indices for the source data once the above configuration files have been generated. DocBase makes use of the predefined "dbbuild" command to generate the indices. The current implementation of DocBase also creates a detailed catalog of objects in the database, including a binary representation of the document structure and a list of the different types of objects (*e.g.,* documents, DTDs, stored queries, auxiliary join indices and temporary structures) etc. The catalog has to be updated to reflect the newly indexed INEX data collection. Once the index has been created and cataloged the INEX data collection reaches the "Query Capability" state.

6 The INEX Retrieval Evaluation Process

As described in Section 4, INEX CO and CAS queries need to be translated for them to be executed in DocBase using DSQL. The translation process is fairly straight forward, as described earlier. For CO queries, the WHERE clause contain the keywords bounded by the root level element of the documents (article in the case of INEX). The returned element is typically the root element as well, which unfortunately returns the complete article. We altered the returned element in some of the queries to find more specific items in the article (the title of the article, for example). See Table 2 for a selection of the queries in INEX format, along with their DSQL equivalent.

CAS queries are translated using a logical translation of the Structural conditions. In the case all conditions are on the same structure, a single alias is created in the FROM clause, and conditions for all of the searched query words are used in the WHERE clause. However, when nested structures are used, multiple aliases need to be declared in the FROM clause to capture the nesting, and

Table 2. CO Topics and Equivalent DSQL Queries

	Candidate Topic	**DSQL Equivalent**
162	Text and Index "Compression Algorithms"	Select i From inex.article i Where i="text" and i="index" and i="compression algorithm"
166	+"tree edit distance" + XML	Select i From inex.article i Where i="edit distance" and (i ="XML" or i="xml") and (i ="tree")
178	"Multimedia Retrieval"	Select i.fm.tig.atl From inex.article i Where i="multimedia retrieval"

Table 3. CAS Topics and Equivalent DSQL Queries

	Candidate Topic	DSQL Equivalent
127	//sec//(p|fgc)[about(., Godel Lukasiewicz and other fuzzy implication definitions)]	Select s From inex.sec s Where (s.fgc =“Godel Lukasiewicz” and s.fgc=“fuzzy logic”) or (s.p =“Godel Lukasiewicz” and s.p=“fuzzy logic)”
133	//article[about(.//fm//tig//atl, Query) and about(.//st, optimization)]	Select i From inex.article i Where i.fm.tig.atl=“Query” and i..st=“optimization”
137	//article [about(.//abs, ”digital library”) or about(.//ip1, ”digital library”)]	Select i From inex.article i Where i..abs=“digital library” and i..ipl=“digital library”

the WHERE clause needs to appropriately use the alias for the specific search condition. Once again, long search phrases can be split into several small keywords or phrases. Some selected CAS queries from INEX and their translations are shown in Table 3.

7 INEX Evaluation Results

Using the translation process described in Section 6 the INEX queries were converted to equivalent queries in DocBase. We were able to convert all the 75 Queries (40 CO and 35 CAS) successfully into DSQL revealing the capability and applicability of DSQL and DocBase as a querying platform for XML databases. This section details the relevance and timing results obtained while executing the above queries on DocBase.

7.1 Relevance Results

The INEX initiative has some restrictions on the format of each run submission. Each run submission must consist of the file from which that particular result was obtained and the XPath expression that can be evaluated to obtain that specific result instance. DocBase relies heavily on its indexing mechanism to perform data retrieval. The INEX restrictions demanded changes in the code to obtain the results in the desired format. The indexes were retraced back to the source data file for each result instance that matched the query conditions. But because of the design of the system the XPath corresponding to matched components cannot be determined (in many queries such an XPath is even impossible - most XQuery results cannot be evaluated with an XPath). So the runs for each candidate topic reveals the filename, and a system specific unique offset generated using the Patricia trees indexing mechanism instead of the XPath expression for each result

Table 4. Modified Query Output to meet INEX Requirements

```
<topic-id="127"><result>
 <file>./xml/an/2001/a1057.xml</file>
 <path><![CDATA[12421643<st>LECTURES AT
THE HISTORY CENTER<st>...]]> </path> </result><result>
 <file>./xml/an/2001/a1057.xml</file>
 <path><![CDATA[12421967<st> consultant,
author, and "technomad"<st>...]]></path> </result> </topic-id>
```

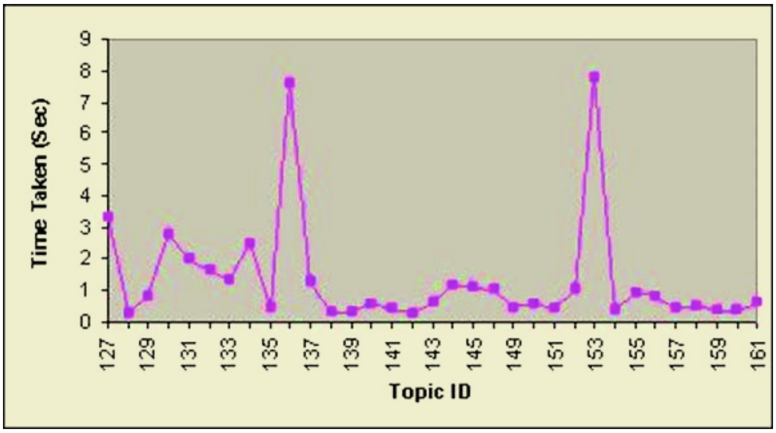

Fig. 2. INEX Content and Structure Queries on DocBase

instance. This was necessary because database queries use indexes that enable the query engine to access the data directly at the index points, without having to explicitly navigate the tree structure. So although the result still contains the relevant document regions, the actual path of the tree used for navigating to that region is not returned by the query. In addition, the INEX initiative requires that only the first 1500 results to be generated and that for each result instance, only the relevant portion be returned. The query evaluation engine was appropriately updated to meet the conditions.See Table 4 for details.

As mentioned earlier, relevance ranking is not a forte of database query languages. The retrieval engine ensures that all retrieved results correspond to the query and since no semantic alterations were performed on the query, all results are treated as equally relevant (a drawback with using database techniques).

7.2 Timing Results

The queries were executed on our test server platform - a Sun Enterprise 250 (2 * Ultrasparc II 400 Mhz, 4x34GB RAID disk with RAID disabled and 2048MB RAM). The primary indexing engine used for the tests was Open Text Pat, and an installation of the Sybase database was used as an auxiliary storage manager.

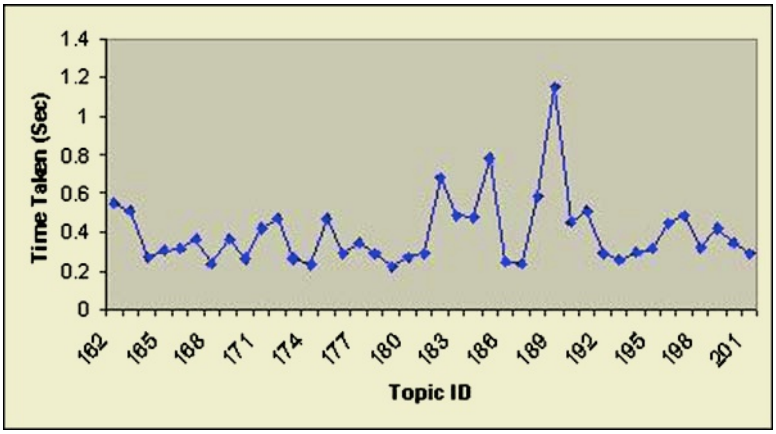

Fig. 3. INEX Content Only Queries on DocBase

On average CAS queries took 1.33 seconds while CO queries took 0.395 seconds. The standard deviations for the same were 1.80 and 0.179 seconds respectively. Most of the queries performed within the acceptable range except for two queries which took approximately 7 seconds to execute. These queries are also responsible for the slightly elevated mean of CAS queries. Careful analysis of the topics in question reveal that those queries had a conjunction of very specific search conditions on widely different structures.

Figures 3 and 2 demonstrate that most queries, whether involving only content, or both content and structure, can be efficiently executed by DocBase. Because of the indexing technique, DocBase achieves a very high retrieval speed, and hence, potentially multiple queries can be executed to create more relevant and ranked results.

7.3 Scalability Results Using XMark

The timing results on the INEX data clearly show the efficiency of DocBase. However, without a discussion of scalability, the real impact of a database management system cannot be felt. Although not part of the INEX process, we evaluated DocBase for scalability against the XMark Benchmark Suite [10]. XMark provides a data generator called "xmlgen" which can generate documents of differing sizes (controlled by a scaling factor) modeling a real world auction web site. We evaluated DocBase on standard XMark Benchmark queries using different scaling factors ranging from 0.1(Document size is 10 MB) to 1.0 (Document Size is .1GB). The time (See Figure 4) taken by DocBase to evaluate the queries compares favorably to the 4 systems reported in [10]. Further DocBase scales very well as the size of the database increases from 10 MB. The results show that even with a ten-fold increase in data size, the query time only increased linearly, a fact that clearly demonstrates the scalability of DocBase.

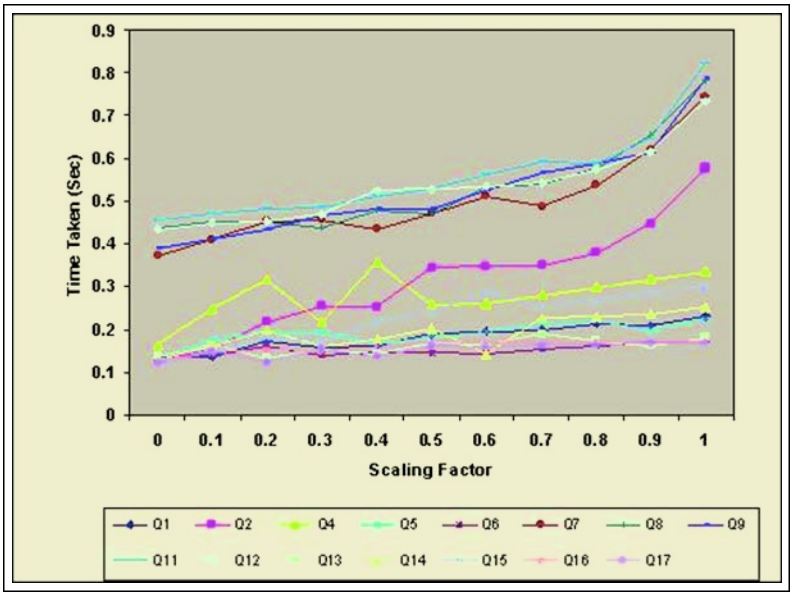

Fig. 4. XMark Benchmark Queries on DocBase

8 Discussion and Conclusion

The INEX evaluation experience with DocBase was an extremely useful learning process and helped demonstrated that with XML, it is possible for database systems and languages to come closer to fulfilling traditional IR tasks. Similarly IR engines can be augmented with the security, stability, and scalability of databases by taking advantage of the transaction processing principles seen in DBMS. Databases and information retrieval are not orthogonal principles, but can coexist and use the lessons learned from decades of research to improve the fields. With the advent of XML, databases and IR have come a step closer. The new XML full-text recommendation from W3C clearly is a step in that direction. The results from our INEX evaluation process does show that there are still limitations in a database language to provide ranking and relevance, but this is certainly a place where database languages can be augmented to achieve such effects. Similarly, IR languages can be augmented to be more specific about the retrieval scope, and become more first-order complete by allowing constructs like sub queries. We believe that XML databases will provide the opportunity for both database and information retrieval applications to co-exist, processing the same data repository, and providing value-added functionality to both methods.

References

1. Salton, G.: Developments in automatic text retrieval. Science **253** (1991) 974–980
2. Tomasic, A., Garcia-Molina, H., Shoens, K.: Incremental updates of inverted lists for text document retrieval. SIGMOD RECORD **23** (1994) 289–300

3. Gonnet, G.H., Baeza-Yates, R.: Lexicographical indices for text: Inverted files vs pat trees. Technical Report TR-OED-91-01, University of Waterloo (1991)
4. Sengupta, A.: DocBase - A Database Environment for Structured Documents. PhD thesis, Indiana University (1997)
5. Sengupta, A., Dillon, A.: Query by templates: A generalized approach for visual query formulation for text dominated databases. In Aho, A., ed.: Proceedings: Symposium on Advanced Digital Libraries, Library of Congress, Washington, DC, IEEE/CESDIS, IEEE Computer Scociety Press (1997) 36–47
6. Sengupta, A., Dalkilic, M.: DSQL - an SQL for structured documents. Lecture Notes in Computer Science **2348** (2002) 757–760 Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAISE'02), Toronto, Canada.
7. Carey, M.J., DeWitt, D.J., Frank, D., Graefe, G., Muralikrishna, M., Richardson, J.E., Shikita, E.J.: The architecture of the EXODUS extensible DBMS. In Dittrich, K.R., Dayal, U., eds.: Proceedings, 1996 International Workshop on Object-Oriented Database Ssytems, Pacific Grove, California, USA, IEEE-CS (1986) 52–65
8. Open Text Corporation Waterloo, Ontario, Canada: Open Text 5.0. (1994)
9. Layman, A.: Element-normal form for serializing graphs of data in XML. Based in part on an earlier paper, Serializing Graphs of Data in XML, by A. Bosworth, A. Layman, M. Rys, in XML Europe '99, Granada, April 1999 (1999)
10. Schmidt, A.R., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In: Proceedings of the International Conference on Very Large Data Bases (VLDB). (2002)