Abstract
Recent years have perceived a theatrical increase in the fame and acceptance of XML - eXtensible Markup Language. This impulsive growth is determined by its ability to offer a standardized, extensible means of involving semantic content within documents describing semi-structured data in database management systems. It enables to address the limitations of existing markup languages and support data exchange in relational databases. Most of the information systems that work on relational databases today are to suffer a lack of both performance and inter-operability or migration issues especially when the information systems work on different databases at different nodes of operations. The XML Native Databases also known as NXD cater to this issue, as XML has emerged into the most powerful markup language that can effectively store, carry or even transform data and its unique feature of encoding independent, platform independent nature. Hence, the paper discusses the survey of earlier works and addresses the need for XML databases as compared to relational databases and further, an insight into the document centric feature of the NXD required by present day use case scenarios.

Keywords: Authentication, Markup Language, Relational Databases, Task Time, XML Databases, XML Documentation

1. Introduction
Extensible Markup Language is used to exchange information over the World Wide Web. The first issue that comes in to picture when large amounts of information are exchanged over the internet is security. This includes data level security as well as high level access controls. Thus, XML databases have come into picture. There are a number of challenges and opportunities for research provided by XML with respect to data management. As explained in the World Wide Web has also introduced many standards for accessing, searching, processing and storing XML data. Some of these like XSLT, DOM, Xlink and X Query are well tested in real world applications.

XML finds its place in the hierarchical databases category, that is far more useful and advantageous in maintaining document-centric data as suggested in the earlier works. The survey states that XML could be used as a powerful document-centric hierarchical database for maintaining industry data-intensive information in an efficient way and also this information could be modified or queried using various XML tools similar to relational databases like by using XMLDB APIs, XQuery Language, a Language similar to SQL for relational databases, to modify or retrieve content from XML files. The XMLDB API here is similar to the JDBC or ODBC drivers used for connecting to relational databases.

The document-centric non-structured data cannot be queried or utilized using any of the relational databases that exist today. This problem has to be addressed by someone and that is where XML comes for rescue. XML has various features similar to relational databases like a query language called XQuery to give SQL like commands to XML files for retrieving data in a specific way based on even conditions like where clauses to retrieve a specific set of rows or to modify this set of rows as per the requirement. NXD systems like eXist, sedna exist in the market today that support various tools such as XQuery, XPath 1.0 and Xindice. These tools support XML Based data querying together with client authentications and user privileges and roles. The querying language XQuery supports complex document-centric queries that are not at all possible in relational databases. This creates an interesting affinity towards XML based data storage. In addition to these features, the fact that these XML Native Databases are practically proven to be efficient in data-
intensive organizations such as airlines and manufacturing companies, further motivates the information systems companies to take the alternative of XML databases over conventional relational databases.

2. **Xml Enabled Relational Database and NXD**

XML enabled databases are those where in extensions are added on top of the preexisting relational database to support XML documents. The semi-structured nature of XML does not map well to the current highly structured storage systems. Using XML documents seemed well suited. Thus, the relational database seems to be appropriate for storing XML documents. Native XML databases, on the other hand, are built ground up for storing XML documents. The structure of the data is maintained with in built data structures. These accept and manage XML data in its native form. Hence no mapping is required between the structure and the database.

If XML is stored as XML inside the documents, it is Native XML document otherwise it is called XML enabled relational database. XML databases and XML query language come into picture because of the following differences between relational data and XML data.

- Data structure.
- Depth of nested data.
- Metadata.
- Inapplicable values.
- Ordering.

3. **XML Databases**

XML databases were a solution to a number of relational database problems. These include the following,

- The issue of synchronization between database and non-database components.
- The absence of proper node based querying.
- During the phase of evolving schemas.

XML databases emerged as solution to a lot of them problems. The number of features XML databases are to offer created a spark in the management of large chunks of data.

The following points elaborate the advantages provided by the features of XML databases

- The XML data model that is flexible to handle a varied range of data from technical documentation to health data.
- Another major advantage as required by the shortcomings is the node level updates that is made possible.
- The schema evolution is one another major functionality provided by the XML databases. This is a general feature expected but this moves slowly in case of relational databases but the update is made faster in case of XML databases.
- Native Storage capabilities is one of the strongest points of consideration in XML databases.

4. **NXD’s Evolution**

XML enabled databases are those where in extensions are added on top of the preexisting relational database to support XML documents. The semi structured nature of XML does not map well to the current highly structured storage systems. Using XML documents seemed well suited. Thus, relational database seems to be appropriate for storing XML documents.

5. **Standards for Native XML Databases**

Storage of XML documents is mainly achieved by native XML databases. For XML documents that are document centric, these systems are preferable to XML enabled relational database systems.

The establishment of standard connectivity to NXD is done using a standard API which is XML: DB API. This included methods for searching, querying, and retrieving results. XQuery is used as the query language. This is mainly centered on four concepts: drivers, resources, collections and services.

- Database access logic is encapsulated by drivers.
- XML source can either be binary or XML resource itself which is a resource.
- Collections.
- Services might be used to query a collection.

The various standards for XML are as follows,

- Document type definitions.
- XML schema.
- Namespaces.
- Xpath.
- Xquery.
- Xlink.
- Xinclude
6. Security Analysis

Creating security to XML Databases just like in cases of relational databases is possible by using the XML XUpdate Language\textsuperscript{11}. Here, it is stated that XML could provide security just like relational databases by using the XUpdate language for various privileges and roles for different users like GRANT, REVOKE. The security here is also with respect to the insertions, deletions or updates of data. For example, see the listing below for insertion restrictions,

\begin{verbatim}
xupdate:modifications version="1.0">
  <xupdate:insert-before
    select="/files/record[1]">
    <xupdate:element name="record">
      <name>Franck</name>
      <diagnosis/>
    </xupdate:element>
  </xupdate:insert-before>
</xupdate:modifications>
\end{verbatim}

Listing 1

This document uses an insert-before operation (context node is the element /files/record)\textsuperscript{11} and performs the insertion of the following XML fragment into the document represented in Listing 1.

\begin{verbatim}
<record>
  <name>Franck</name>
  <diagnosis/>
</record>
\end{verbatim}

According to our example of policy, only secretaries (and the owner of the document) may successfully submit this operation to the database. Indeed, only secretaries hold the insert privilege on node /files which is the parent node of the context node\textsuperscript{11}.

7. Use Cases

Most applications in a lot of platforms today need good content management systems. Due to the combined capabilities of standards of XML and XML databases that provide the support enable a uniquely powerful platform for managing the most demanding content applications

Some places where XML is predominantly used is
- Airline Industry.
- Manufacturing Industry.

7.1 Airline Industry

This is the kind of industry where the cost and efficiency mark the difference between success and failure. The information is organized into technical manuals and work cards for specific tasks. The work cards will be specific various problems. This maintenance information is maintained by airline companies in documents. The current maturity in the XML standards and its support from XML databases breaks into an environment where information is reused and integrated. Though airline industry has been using XML for quite some time, the information regarding the older aircrafts might still only be documented. Thus, with the advent and usage of NDB, the document centric nature of this will enable more efficiency. The links between information objects that are required like automatic update are automated or at least tracked by EMC Documentum XML store.

In a nutshell, airline industries need NXD because
- They need to maintain information and retrieve information that was documented decades ago which needs document centric approach.
- Need to avoid being locked into a format that is proprietary which needs a consistent environment.
- Consumers are highly distributed.
- Data is in need for at least three to four decades together.

Considering these requirements and the features that match them, NXD is considered a no opponent match by one of the airline executives.

7.2 Manufacturing Industry

Any manufacturer’s key point to success lies in the understanding of user’s preferences which in turn needs information regarding product. Documents can be shared between various models depending on the components that are common to the products. To make the search effective, the manufactures can take advantage of the XML database’s document centric nature of the native XML databases. The supported standards and the best of breed document component found in the EMC Documentum XML store provided ore scope for improvement.

NXD seemed like the right choice because the major challenge of finding a technology that was compatible with the existing systems and workflows seemed to be effectively solved by the same due to its integration capabilities due to the support provided as already mentioned. From the results, the searches that utilize
Extensible Markup Language Databases: A Study

XQuery are very precise when compared to other XML tools and technologies.

8. Performance Analysis

When analysis is done on the operation and the output, the evidence will support the more efficient form of database to be used. The more efficient systems for specific business such as customer portals, various business transactions can be identified in a systematic manner. Not only does it help us understand NXD’s but also for developing a proper database management system for e-mail filters. As NXD’s for document centric data is relatively a new technology, this analysis will only improve the existing knowledge on the same.

After further understanding of the XML database systems, we further analyze the performance of the NXD’s when compared to the relational database systems. NXD’s definitely open more possibilities for maintaining document data and also comparatively, there is definitely a significant difference in the performance between the Relational databases and NXD’s with respect to storage and management of document centric data.

The data considered for this performance analysis is restricted to a set of e-mail messages. Hence the document centric data is represented by these e-mail messages for the analysis. The PostgreSQL is considered for analysis for the relational database representation whilst eXist is used for representing the NXD’s. The chosen databases are both freely available and widely used. Explaining the key features of eXist,

- Efficient indexing structure based on numerical indexing scheme.
- Quicker querying.
- Provides features for retrieving and interpreting information.

So the performance test is basically done using the time that is taken to perform basic queries. Only searching and retrieving of information is taken into account. Updating of information is not included in the analysis. The basic type of database access includes

- Table scan.
- Exact node access.
- Composite node access.
- Range access.

When compared to RDBMS, NXD’s do an amazing job with respect to document centric data. This can be explained using the following evidences.

- NXD’s can model component sun component data which is what is critical with respect to document centric data.
- Round tripping is one of the best features offered by NXD’s where in the documents are stored and retrieved without any modifications.
- Retrieval of a single document might probably mean accessing multiple tables with respect to RDBMS that requires several joins that increases the complexity.
- To start with the experiment, first it was to make sure that either of the databases do not have an advantage over each other.
- But the PostgreSQL being a relatively older database has an advantage over the recently developed NXD’s. To make the differences clear, queries with both DBMS’s, it was run on an empty database which provided an evidence of the advantage PostgreSQL over NXD’s and the configuration is changed to even out the difference to further carry out the experiment. But it as found that the more mature PostgreSQL had no advantage over the relatively new NXD.
- Caching is also one of the major challenges that had to be taken into account. The data that the non-empty queries returned might make sure the disk is not accessed for repeating queries. To ensure this does not affect the comparison of the database systems, the database was configured to support negligible amount of caching.
- Also, the experiments had to be run on the same machine. PostgreSQL was installed and run on the same machine and it was found that there actually was a two hour difference in the time taken to run the entire query set. To account for this confounding variable, both the databases were run on the same machine one after the other without varying the setup.

8.1 Experimental Design

The experimental setup used included Dell OptiPlex GX110 with an Intel Pentium III 733MHz CPU, 256MB SDRAM and a 10.2GB WD102Western digital hard disk. The operating system used was Fedora. The version of eXist used was eXist 3.1 for Linux. The PostgreSQL used was version 8.0.3. A set of 6432 spam mails were used to populate the databases. These set of emails was stored within eXist as XML documents. These messages were stored as text files initially. The loading of the databases was done using two Perl scripts, both of which disassembled
the emails into the corresponding components. One of these scripts inserted these components into appropriate databases and the other generated XML documents that matched with the original e-mails.

8.2 Process
The connection was established with the database and a query is read from a text file which contained all the queries. The start time was recorded in milliseconds and after the result of the query was established the end time was noted as well. The difference between the start and the end time represented the time used for the query processing. This process is repeated a 100 times before the next query is processed. The only limitation of this is that exact query execution time involves java processing.

8.3 Assumptions
There are quite a number of assumptions to be considered before proceeding with the experiment, major ones including,

- Time for query execution is the representation of the performance.
- The document centric data is adequately represented by the collection of e-mails.
- A valid measurement of performance is done by the selected queries.
- The difference in query languages supported by each database has no impact on performance measurement.

These assumptions were made with the required small assumptions made through the experiment when necessary.

8.4 Results
Each query executed can also be represented as a task. In total, 11 tasks were performed on the databases and the results were obtained to further analyze the performances. Before any critically analyzing the results, the data was graphed and it was observed that there was a significant difference in the performance between RDBMS and NXD’s. For further confirmation, a 2-WAY ANOVA test was performed to check if there was significant difference and it was very well confirmed by the test as well. The following observations can be made,

- In Figure 1, where x-axis is task time while the y axis represents the frequency of runs within the time interval. On an average, PostgreSQL is much faster with mean time of 3923 milliseconds and a median time of 3650ms. But looking at eXist, though the mean time as high as 5880ms, the median time is only 3126 ms which indicated that only because of a few extremes the mean time is high.

- When the time taken for individual task was compared, there was an obvious difference. eXist performs is better for 6 out of 11 tasks. This can be observed Figure 2, where the horizontal axis is represented by the task time in ms and the y axis represents the frequency of time. Looking at task 1, PostgreSQL is outperformed by eXist where the average time crawls towards 0ms whereas the former was near 5000ms. For task 8, the scenario completely switched.

Thus eXist showed better performance when compared to PostgreSQL. Now let us consider the time between tasks. The statistics in Table 1 show that there is a notable difference between the databases about the same. To analyze the performance of each task, we will consider one of the tasks and look into the differences. The first task is retrieval of an e-mail with a certain name. In this case the name was “Gabriel Gilmoore”. eXist visibly outperforms PostgreSQL in this case with the mean time of 503ms over the mean time of 5020 ms. With respect to RDBMS, the query needs a join operation of tables, and
then restricting the sender’s name to the required name. Join operation is a very costly operation. On the other hand, when it is handled using eXist, the search space is reduced a lot as it decomposes the given expression path into separate steps. Figure 2. Retrieving a mail with a particular name.

Now looking at the next case as shown in Figure 2, the retrieving data is done using an unique key. In this case, eXist outperforms PostgreSQL with mean time of 3621 ms over 5028 ms. This is because the next task is range access. In this, the PostgreSQL has outperformed eXist with the former having a mean time of 5027ms and the later with 6333ms as shown in Figure 3. The former uses indexing on the column time_sent hence getting faster response whilst the later directly accesses the column directly. But, it is noticed that the time difference is very less.

Table 1. Analysis of variance in overall time

| Source          | Type II Sum of Squares | df   | Mean Square | F     | Sig.  | Partial Eta Squ |
|-----------------|------------------------|------|-------------|-------|-------|-----------------|
| Corrected Model | 7.133E+011              | 21   | 3.397E+010  | 255521.6 | .000  | .996            |
| Intercept       | 5.286E+011              | 1    | 5.286E+011  | 3976243 | .000  | .995            |
| nDbase          | 2.108E+010              | 1    | 2.108E+010  | 158562.5 | .000  | .878            |
| task            | 2.846E+011              | 10   | 2.646E+010  | 199048.8 | .000  | .989            |
| nDbase*task     | 4.276E+011              | 10   | 4.276E+010  | 321690.2 | .000  | .993            |
| Error           | 2921522849              | 21978| 132929.423  |       |       |                 |
| Total           | 7.162E+011              | 21999|             |       |       |                 |

Now looking at the next case as shown in Figure 2, the retrieving data is done using an unique key. In this case, eXist outperforms PostgreSQL with mean time of 3621ms over 5028ms. This is because the next task is range access. In this, the PostgreSQL has outperformed eXist with the former having a mean time of 5027ms and the later with 6333ms as shown in Figure 3. The former uses indexing on the column time_sent hence getting faster response whilst the later directly accesses the column directly. But, it is noticed that the time difference is very less.

Similarly, the other tasks were also performed and the results were analyzed as shown in Figure 4. The other tasks were just variations of these queries with varied number of condition. It indicated that performance if based on query execution time, it is further dependent on the task. Hence, the performance of NXD’s is inferred to be better than that of RDBMS. This probably is because of the ability of the NXD’s to state a path expression and point out a node to in an entire document. The following observations were particularly made.

- When the number of conditions increase, eXist becomes relatively slower. The search space is reduced due to indexing but PostgreSQL is stuck doing an entire table scan but still provides a better performance.

Figure 4. The queries applied for a range of access.

```
Task2: Query:
for $x$ in collection("spam") where $x$/time-sent > "Fri, 6 Jun 21:50:22
1998 +12:00" and $x$/time-sent < "Fri, 7 Jun 21:50:22 2001 +12:00"
return $x$;
SQL:
SELECT * from tbl_message LEFT OUTER JOIN tbl_header on
tbl_message.message_id = tbl_header.message_id LEFT OUTER JOIN
tbl_recipient on(tbl_message.message_id = tbl_recipient.message_id)
where time_sent > to_date('07-06-2001 - 00:00:00', 'DD-MM-YYYY') and
time_sent < to_date('07-06-2001 - 23:59:59', 'DD-MM-YYYY');
```
• PostgreSQL also performs well in case of range access. This is because of the more efficient range indexing scheme.
• Due to the granular storage of data into tables, PostgreSQL offers better performance.

9. Conclusion

The paper suggests that the XML Databases prove to be future-centric technologies for storing or retrieving information efficiently, supported by the fact that they are effectively used currently by the Airlines and Manufacturing Industries experimentally. The existence of various feature rich data extraction and storage tools like the XQuery, XUpdate and Xpath 1.0, 2.0 or XIndices, proves XML to be a powerful document-centric Hierarchical database as opposed to the less efficient relational databases which also have an overhead of heavy licence fees. The XML Databases that are native are called NXDs and are completely free from licencing issues. This new breed does not only understand the XML structure but also completely utilizes the standards developed by W3C to the complete extent which makes it a powerful tool it is. Moreover with tools like XUpdate, user roles and privileges also could be set in XML Databases, which is another aspect to choose XML as a Database. Thus proving to be a future Database technology, XML sits on the top position leaving behind the relational databases that are not inter-operable and are slow and inefficient. Though, NXD is an excellent option for XML data, most business applications today do not deal only with XML data. There definitely exist a part of relational data as well that need to be dealt with. When it comes to the performance comparison, it is concluded that in cases where restructuring is not necessary, then eXist has performed much better than PostgreSQL. The future scope of comparison for this would be performing the same experiment taking into account the storage space as this experiment took into account only the execution time. Also, the ability of NXD’s to provide text indexing and support full text searches can be taken as a challenge for the future research.

10. References

1. Martinez-Gonzalez MM, Martinez-Prieto MA, Munoz-Nieto M. A study of native XML databases. 5th International Conference on Web Information Systems and Technologies; 2009. p. 89–92.
2. Mary Posonia VL, Jyothi. Context-based classification of XML documents in feature clustering. Indian Journal of Science and Technology. 2014 Jan; 7(9). DOI: 10.17485/ijst/2014/v7i9/50864.
3. Bill T, Waldt D. Using XML and databases; 2008.
4. Noaman AY, Al Mansour AA. A comparative study between two types of database management systems: XML-enabled relational and native XML. World Applied Sciences Journal. 2012; 19(7):972–85.
5. Izadi SK, Haghighj M. A new approach to concurrency control in XML databases. Indian Journal of Science and Technology. 2014 Jan; 7(2). DOI: 10.17485/ijst/2014/v7i2/47678.
6. Li J, Liu C, Zhou R, Yu JX, Quasi SL. CA based keyword query processing over probabilistic XML data. IEEE Transactions on Knowledge and Data Engineering. 2014; 26(4):957–69.
7. Bonifati A, Cuzzocrea A. Synopsis data structures for xml databases: Models, issues, and research perspectives. In DEXA 18th International Workshop on Database and Expert Systems Applications, DEXA ‘07; 2007. p. 20–4.
8. Elmasri R, Navathe SB. Fundamentals of database systems. USA: Pearson Education; 2014.
9. Cheol-Joo C, Kiseok C, Kwang-Nam C. The XML based electronic document image retrieval system. Indian Journal of Science and Technology. 2015 May; 8(59). DOI: 10.17485/ijst/2015/v8i59/68316.
10. Williams A. Performance of relational databases versus native XML databases. Information Science; 2005. p. 76.
11. Gabillon A. A secure XML database. 4th Conference on Security and Network Architectures; France. 2005. p. 1–12.
12. Elisa B, Castano S, Ferrari E, Mesiti M. Specifying and enforcing access control policies for XML document sources. World Wide Web. 2000; 3(3):139–51.
13. Kimelfeld B, Senellart P. Probabilistic XML: Models and complexity. Advances in Probabilistic Databases for Uncertain Information Management. Heidelberg: Springer Berlin; 2013. p. 39–66.
14. Don C. XQuery update facility 1.0. W3C Candidate Recommendation; 2008. p. 1.
15. Haw SC, Lee CS. Data storage practices and query processing in XML databases: A survey. Knowledge-Based Systems. 2011; 24(8):1317–40.