Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Elina Hazaran

The Support of XML Technologies in Oracle DB, IBM DB2 and Microsoft SQL Server

Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. Irena Mlýnková, Ph.D.

Study programme: Computer Science

Specialization: Administration of Computer Systems

Prague 2012

I would like to thank my supervisor RNDr. Irena Mlýnková, Ph.D. for leading my work. I would also like to thank my sister, my parents and my friends for supporting me during my studies.

Název práce: Podpora XML technologií v Oracle DB, IBM DB2 a Microsoft SQL Server

Autor: Elina Hazaran

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.

Abstrakt: Cílem této práce je provedení podrobné srovnávací analýzy podpory XML technologií ve třech databázových systémech: Oracle DB, IBM DB2 a SQL Server. Obsahuje teoretický popis podporovaných technologií a jejich srovnávací analýzu, spolu s implementovaným programem, který je určen k provádění experimentů dotazování nad XML daty pomocí XQuery, SQL/XML technologií a XMark benchmarku. Umožňuje srovnání výkonu databází na základě různých metod ukládání, vytváření statistik pro časové srovnání, export výsledků do textového souboru a ukládání historii předchozích výsledků. Hlavní částí práce je demonstrace provedení několika experimentů a detailní analýza výkonů jednotlivých databází založená na získaných výsledcích.

Klíčová slova: XML technologie, XML-enabled databáze

Title: The Support of XML Technologies in Oracle DB, IBM DB2 and Microsoft SQL Server

Author: Elina Hazaran

Department / Institute: Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. Irena Mlýnková, Ph.D.

Abstract: The aim of this thesis is to perform an all-round comparison analysis of XML technologies support in three database systems: Oracle DB, IBM DB2 and SQL Server. It includes a theoretical technologies description and their comparison analysis, along with implemented software which is designed to run experiments by querying XML data using XQuery, SQL/XML technologies and XMark benchmark. It provides the comparison of database performances based on different storage methods, create time comparison statistics, export results to a text file and store the history of previous results. The main part of the work is the demonstration of several experiments run and a detailed analysis of the database performances based on the obtained results.

Keywords: XML technologies, XML-enabled databases

Contents

Introduction

XML [1] is a commonly used format when sharing data. For that reason most of modern database systems develop tools and technologies to support this format. When creating an application which has to perform operations upon XML data, it is necessary to determine the most suitable database system to meet the application requirements. In order to do so a program that allows to compare database systems on basic criteria of processing XML data is needed. Most of the available testing technologies do not introduce the possibility to represent the results of testing in a graphical user-friendly way, nor do they allow comparing the results of different database systems.

The aim of this thesis is to create the application that will allow for representing and comparing of XML data experiments results and perform a general comparison analysis of XML technologies support in three database systems: Oracle DB, IBM DB2 and SQL Server, using created application. The program will be able to store XML documents in databases and perform querying operations upon it. A user will be able to choose the method of storing XML documents in database, the way the experiment will be processing it, and specify the systems to compare. The application will also be able to create the statistics of database performances based on the most important criteria of XML data querying.

This work consists of two main parts: theoretical and practical. The first chapter describes existing XML technologies and introduces basic examples of working with them. The second chapter includes the description of three database systems: Oracle DB, IBM DB2 and SQL Server. It also describes which of described XML technologies are introduced in each database system and the way it is implemented there. The third chapter describes the goals and requirements of created application. The fourth chapter includes user documentation. The fifth chapter contains the description of the application implementation along with the program architecture and used algorithms. The sixth chapter includes the demonstration of a set of experiments upon the databases and a detailed analysis of the obtained results. The seventh chapter includes a summarizing comparison table. The conclusion summarizes the work. The list of tables includes all the tables presented in this thesis. Appendix A lists the contents of the attached CD.

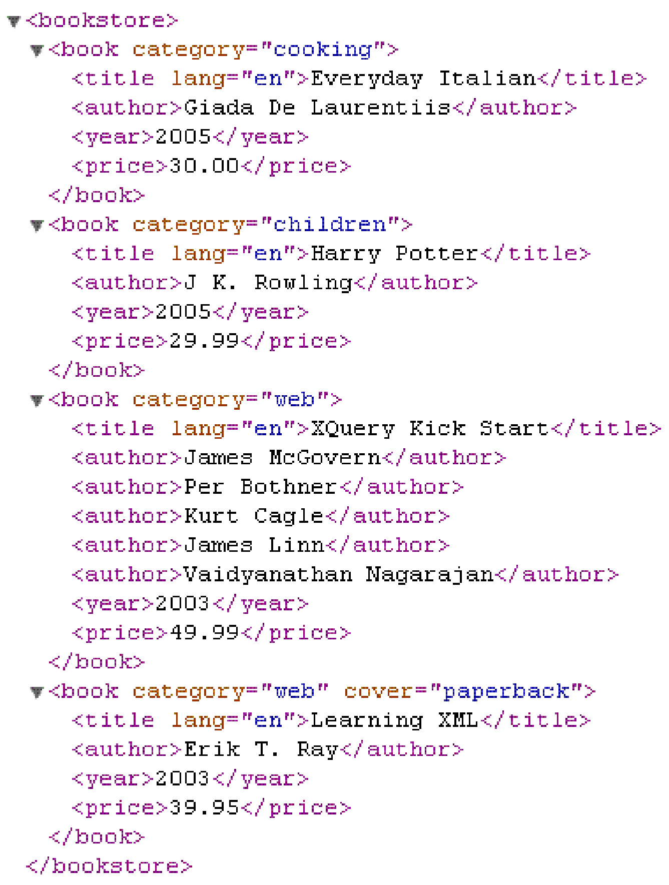
1. XML Technologies

The eXtensible Markup Language (XML) is a format for sharing and exchanging data. It is a subset of SGML (Standard Generalized Markup Language), its goal is to enable SGML to be served, received, and processed on the Web [1]. It was designed for simplicity, generality and usability over the Internet.

Unlike other markup languages, XML is a flexible and extensible language. The reason for this is that it allows us to create new markup elements along with the specification for a browser to interpret these elements.

The popularity of XML constantly increases. There are several main advantages that distinguish this particular format from the similar ones, such as:

- Simplification of the interaction between different programs
- Flexibility of designing markup languages based on XML
- Data structure
- Data integration



```
▼<bookstore>
  ▼<book category="cooking">
     <title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author>
     <year>2005</year>
     <price>30.00</price>
   </book>
  ▼<book category="children">
     <title lang="en">Harry Potter</title>
     <author>J K. Rowling</author>
     <year>2005</year>
     <price>29.99</price>
   </book>
  ▼<book category="web">
     <title lang="en">XQuery Kick Start</title>
     <author>James McGovern</author>
     <author>Per Bothner</author>
     <author>Kurt Cagle</author>
     <author>James Linn</author>
     <author>Vaidyanathan Nagarajan</author>
     <year>2003</year>
     <price>49.99</price>
   </book>
  ▼<book category="web" cover="paperback">
     <title lang="en">Learning XML</title>
     <author>Erik T. Ray</author>
     <year>2003</year>
     <price>39.95</price>
   </book>
 </bookstore>
```

Listing 1: Sample XML document

XML document has a tree structure with a root element and sub elements. In the example the root element is <bookstore> which contains three <book> elements. Every <book> element has four sub elements: <title>, <author>, <year>, and <prize>. An element can contain other elements, text, and attributes.

Each element within XML document must have opening (e.g. <bookstore>) and closing (e.g. </bookstore>) tags that must be written with the same case and properly nested within each other (e.g. <b><i> text </i></b>). Element in XML documents can have attributes that have to be quoted (in the example <book category="cooking">). All elements within XML document can be extended to carry more information.

To perform with the XML format data a set of related technologies is needed. These XML technologies will be described below.

1.1. A Brief Introduction to XML Technologies

Following chapters describes the main XML technologies to process XML data.

1.1.1. Interface for Working with XML Data

One of the interfaces for working with XML data *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document"* [11]. The XML DOM is a standard object model that defines objects and properties of all XML elements and a standard programming interface that defines the methods to access XML elements.

The XML DOM represents an XML document in a tree structure, where elements, text, attributes, comments within document and entire document are nodes. In XML DOM there is a set of properties that are defined and can be used inside methods to help retrieving information from nodes. The most typical properties are: x.nodeName, x.nodeValue, x.parentNode, x.childNodes, x.attributes where x is a node in an XML document. The XML DOM contains methods to access, insert and delete nodes.

There are three possibilities of accessing nodes in XML documents with XML DOM: using the getElementsByTagName() method, traversing the nodes tree using documentElement.childNodes() method, and navigating the node tree by node relationships using e.g. parentNode.nodeName() method.

To add or insert the node we can use appendChild() and insertBefore() methods respectively.

```
<html>
<head>
<script type="text/javascript" src="loadxmldoc.js">
</script>
</head>
<body>

<script type="text/javascript">
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);

document.write(x.getElementsByTagName("edition")[0].nodeName);
</script>
</body>
</html>
```

Listing 2: Sample XML DOM add node document

In Listing 2 the node is added after the first child node. First using the method createElement() the node "edition" is created then it is appended to the first <book> element.

The removeChild() method allows us to delete the node.

```
<html>
<head>
<script type="text/javascript" src="loadxmldoc.js">
</script>
</head>
<body>
<script type="text/javascript">
xmlDoc=loadXMLDoc("books.xml");

y=xmlDoc.getElementsByTagName("book")[0];
xmlDoc.documentElement.removeChild(y);

</script>
</body>
</html>
```

Listing 3: Sample XML DOM remove node document

In Listing 3 we first set which element is to be removed and then remove the child element node by using the method from the parent node.

The above described examples use sample XML document introduced in the Listing 1.

1.1.2. The Definition of Document Type and Structure

As an example XML Schema is one of several XML schema languages to *"provide a means for defining the structure, content and semantics of XML documents"* [4]. XML Schema is a W3C recommendation. In order to avoid the confusion between the specific language and schema languages in general it is also referred to as XML Schema Definition (XSD). XML Schema is extensible to additions, written in XML and it supports data types and namespaces.

XSD allows us to define the elements and attributes that can appear in XML document, their data types, default and fixed values; to define which elements are child elements, its order and the number; to define whether an element is empty or it can include text [13]. XML documents are validated against its XML schema to be checked on well-formedness. Well-formed documents whereas conforming to the XML syntax rules can contain further errors that can be caught by a validating software.

The most common built-in data types in XML Schema are: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time. XML Schema distinguishes two types of elements: a complex type is the element that can contain other elements and a simple type that contains only text and can not contain other elements or attributes.

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Listing 4: XML Schema definition for simple XML elements

In Listing 4 there are three XML simple elements of different data types and corresponding definition in XSD using built-in data types.

If an element in XML document has attributes it is a complex element, but the attribute itself is always declared as a simple type. The definition of an attribute can contain default (automatically assigned value when no other value is specified) of fixed (automatically assigned with no possibility of specifying another value) values.

```
<lastname lang="EN">Smith</lastname>
<xs:attribute name="lang" type="xs:string"/>
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Listing 5: XML Schema definitions for the attributes of different types

In Listing 5 three possibilities of defining attributes are introduced: the simple attribute definition, the definition of an attribute with a default value, and the definition of an attribute with a fixed value.

The following example demonstrates the definition of a complex element.

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 6: XML element and corresponding XML Schema definition

In Listing 6 the XML element contains both text and other elements. In XSD the attribute mixed is set to "true" to enable the character data to appear between the child elements.

1.1.3. Querying Collections of XML Data

As an example XQuery is a query and functional programming language that is *"designed to be broadly applicable across many types of XML data sources"* [14]. It is flexible enough to query a broad spectrum of XML information sources, including structured and semi-structured documents, relational databases, and object repositories. XQuery is defined in terms of the data model and the expression context.

XQuery operates on the abstract, logical structure of an XML document; this logical structure defines its data model. Unlike other XML languages XQuery includes two new features in its data model such as: support for XML Schema simple and complex data types, and representation of collections of documents and complex elements.

XQuery is a language in which queries are concise and easily understood. The basic building block of XQuery is the expression, which is a string of Unicode characters. All the information within an expression that can affect its result is called expression context, which is of two types: static context and dynamic context. The static context of the expression is all the information available during static analysis, prior to expression evaluation. The dynamic context, contrariwise, is defined as information that is available at the time the expression is evaluated.

XQuery allows us to find and extract elements and attributes from XML documents, to generate summary reports and transform XML data. The set of tools designed to perform the operations includes: comparison (eq, =, is), conditional (if) logical (and, or), and quantified (some, every) expressions, order by and where clauses in FLWOR expressions. This chapter provides simple examples to illustrate these tools operation.

There are three types of comparison included in XQuery language: value, general and node comparison.

```
$book1/author eq "Kennedy"
```

Listing 7: Sample XQuery clause "eq" comparison expression

```
$book1/author = "Kennedy"
```

Listing 8: Sample XQuery "=" comparison expression

```
/books/book[isbn="1558604820"] is /books/book[call="QA76.9 C3845"]
```

Listing 9: Sample XQuery "is" comparison expression

Listing 7 expression atomizes the node that are returned by the expression $book/author. The comparison is true only if the result of atomization is the value "Kennedy".

In Listing 8 the comparison is true of the value of any <author> subelement of $book1 is "Kennedy".

Comparison in Listing 9 is true only if the left and the right sides each evaluate to exactly the same single node.

A logical expression in XQuery is either an and-expression or an or-expression.

```
1 eq 1 and 2 eq 2

1 eq 1 or 2 eq 3
```

Listing 10: Sample XQuery "and or" expression

The sample code in Listing 10 returns true value.

XQuery supports a conditional expression based on the keywords if, then, and else.

```
if ($part/@discounted)
  then $part/wholesale
  else $part/retail
```

Listing 11: Sample XQuery condition expression

In Listing 11 the condition expression tests for the existence of an attribute named discounted, independently of its value.

Quantified expressions in XQuery support existential and universal quantification.

```
every $part in /parts/part satisfies $part/@discounted
```

Listing 12: Sample XQuery "every" expression

```
some $emp in /emps/employee satisfies
     ($emp/bonus > 0.25 * $emp/salary)
```

Listing 13: Sample XQuery "some" expression

In Listing 12 the expression is true if every <part> element has a discounted attribute.

In Listing 13 the expression is true if at least one <employee> element satisfies the given comparison expression.

XQuery provides a feature called a FLWOR expression that supports iteration and binding of variables to intermediate results. It is defined by keywords for, where, order by, and return.

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Listing 14: Sample XQuery FLWOR expression

In Listing 14 "for" expression first assigns to $x variable each <book> subelement in <bookstore> element of "books.xml" document (see Listing 1). From received subelement it then chooses the ones which have <price> element value greater than 30. The expression return <title> subelement of the previously collected subelements, ordered by <title>.

The result of XQuery statement can be represented in HTML list.

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{data($x)}</li>
}
</ul>
```

Listing 15: Sample XQuery HTML expression

In Listing 15 in order to represent all <title> elements under the <book> elements inside the HTML list the tags <ul> and <li> are added. To eliminate the title element and show only data the keyword "data" is added.

Although XQuery is a native XML programming language based purely on XML and itself has no concept of relational data, several products and projects provide ways to query relational data using an XML view of the database.

The list of XQuery built-in functions can be found at the W3C webpage [16].

Another tool that allows us to query XML data is SQL/XML [38], an extension to the SQL specification, which defines the use of XML in conjunction with SQL. Despite the fact that SQL/XML is complementary to XQuery, SQL/XML is SQL-centric (it is based on relational structure of the data) unlike XQuery which is XML-centric.

The SQL/XML extension includes a set of functions to provide the implementation of XML publishing, the XML Data type and the mapping rules to determine how SQL data or metadata is represented as XML. In order to produce XML content from SQL data using a SELECT statement several XML publishing functions are available in SQL/XML.

Xmlelement() function creates an XML element with a name specified which allows us to wrap the result of a particular SELECT statement (rows and columns) into XML elements. In order to specify attributes to a particular XML element xmlattributes() function can be used.

```
SELECT xmlelement(name foo);
```

Listing 16: Sample SQL/XML expression

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar));
```

Listing 17: Sample SQL/XML expression

In Listing 16 the SQL/XML expression creates an XML element named "foo" with a following result <foo/>.

In Listing 17 the SQL/XML expression creates an XML element named "foo" specifying an attribute "bar" with a following result –

<foo bar='xyz'/>.

The examples shown above imply the possibility of constructing a corresponding XML structure to any SQL SELECT statement by creating an XML element to each row of the result.

In order to create a sequence of XML elements an xmlforest() function can be used. It can be implemented using the given name and content or a column reference, in which case the column name is used by default.

```
SELECT xmlforest('abc' AS foo, 123 AS bar);
```

Listing 18: Sample SQL/XML expression

```
SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
```

Listing 19: Sample SQL/XML expression

In Listing 18 the expression creates two XML elements named "foo" and "bar" with the following result – <foo>abc</foo><bar>123</bar>.

In Listing 19 the expression creates a set of XML elements named by default by a column name with a following result:

```
<table_name>pg_authid</table_name><column_name>rolname</column_name>
<table_name>pg_authid</table_name><column_name>rolsuper</column_name>
...
```

Listing 20: Sample result of SQL/XML expression

Listing 20 represents an XML forest which is not a valid XML document (it consists of more than one element). For that reason it might be useful to wrap xmlforest expression in xmlelement.

More on SQL/XML functions can be found in Oracle documentation [5].

1.1.4. Updating XML Data

XQuery Update facility is an extension to XQuery. *"The XQuery Update Facility provides expressions that can be used to make persistent changes to the instances of the XQuery 1.0 Data Model"* [17]. The XQuery Update Facility provides five basic operations upon the nodes within XML document such as inserting, deleting, replacing a node, replacing the contents or the value of the node, renaming a node.

```
insert node <number>3</number>
   after fn:doc("books.xml")/bookstore/book[1]/price
```

Listing 21: Sample XQuery Update insert expression

In Listing 21 a number element is inserted after the price of the first book in the XML document "books.xml".

```
delete node
fn:doc("books.xml")/bookstore/book[3]/author[last()]
```

Listing 22: Sample XQuery Update delete expression

In Listing 22 the node that represents the last author of the third book is deleted. The last() function returns the last element in the array of authors.

```
replace node fn:doc("books.xml")/bookstore/book[1]/price
with fn:doc("books.xml")/bookstore/book[2]/price
```

Listing 23: Sample XQuery Update replace expression

In Listing 23 the price of the first book is replaced with the price of the second book.

```
rename node fn:doc("books.xml")/bookstore/book[3]/author[1]
as "principal-author"
```

Listing 24: Sample XQuery Update rename expression

In Listing 24 the first author element of the third book is renamed to "principal-author".

The above described examples use sample XML document introduced in Listing 1.

1.1.5. Transformation of XML Documents

XSLT (eXtensible Stylesheet Language Transformations) is a *"language for transforming XML documents into other XML documents"* [18]. XSLT also provides tools to transform an XML document into another type of document usually by transforming each XML element into an (X)HTML element. XSLT allows us to add or remove elements and attributes to or from the output file; rearrange and sort elements; perform tests and make decisions about which elements to hide and display; etc. A common way to describe the transformation process is to say that XSLT transforms an XML source-tree into an XML result-tree or a plain text document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

Listing 25: Sample XML document with linked XSLT file

In order to transform an XML document into an HTML table first the link to the transformation file should be added as it is shown in Listing 25.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Listing 26: XSL Style Sheet with a transformation template

In Listing 26 the XSLT document which transforms the XML document introduced in Listing 25 into an HTML table using a template represented by <xsl:template> tag. The "match" attribute in the example associates the template with the root of the XML source document. The <xsl:value-of> tag is used to extract the value of an XML document and add it to the output stream of the result-tree. The <xsl:for-each> tag is used to select every XML element of a specified node-set. The value of the select attribute is an XQuery expression and works similar to navigating a file system (a forward slash selects subdirectories).

Due to the fact that XML is the most commonly used format for data representation there is a great amount of XML data nowadays. That leads us to the development of the demand for modern databases to perform and manipulate effectively with XML documents and also to query XML data.

1.2. XML Technologies in Databases

The main reason for XML technologies to pervade in modern database systems is the demand for fast and effective manipulation with XML data. Based on this demand there are several aims to be accomplished in this prospect such as:

- Processing of external data like Web pages, text database, structured data
- E-commerce like product lists, personalized views etc.
- Integration of heterogeneous information sources specifically the ability to manipulate with web and tables simultaneously

To solve the tasks described above in modern databases there should be implemented several mechanisms to store XML data, manage a big sets of XML documents effectively, update and change XML documents (for example: insert nodes, delete nodes, modify the maintenance, modify the copy etc.).

1.2.1. The Advantages of Bringing XML Technologies to Modern Databases

The mechanisms in modern databases to introduce XML technologies allow us to estimate the main points that they are different from each other.

It is necessary to describe the main criteria based on which we can choose one database over another in prospect of processing XML data. Knowing these criteria we will be able to perform a comparative analysis between three commercial leaders in modern database: Oracle DB, IBM DB2, and Microsoft SQL Server.

2. XML Technologies Support in Databases

In modern database systems there exist two main approaches in storing and processing XML format data: having a fixed schema for XML documents and using XML model directly to map XML instances into database; or mapping XML schema to database schema and having an original data model to map XML instances into database.

Databases that support the first method are called native XML databases (NXD). A native XML database *"defines a (logical) model for an XML document and stores and retrieves documents according to that model"* [20]. In other words, it indexes XML documents directly and stores the entire XML document including related elements. Usually, a logical model for an XML document maintains the hierarchical structure of the document, thus is must include elements, attributes, PCDATA, and document order. Opposed to a relational database which basic logical storage unit is a row in a table, native XML database has an XML document as its fundamental logical storage unit. Whereas NXD defines a logical model it does not require any particular physical model and can be built on a relational or object-oriented database. For that reason a NXD may not be a standalone database, it can be used as a tool to provide storage and manipulation of XML documents. NSD can usually be used in the situation when it is more effective to store data as XML and not to convert them into a relational schema for example when a big collection of data already exists in XML format outside the database.

Databases that support the second method are called XML-enabled databases (XED). An XML-enabled database is the relational database which functionality is extended in order to include XML data management capabilities. In XED an XML document shredded into relations and after the data from the document are stored in a database the XML document is discarded. For that reason the original document that was stored in database may not be reconstructed. Whereas the data from XML document are transferred through a relational schema XED allows us to publish the results of required operations as XML document. Usually all the tools for shredding, constructing and publishing XML documents are integrated into a database. The basic use of XML-enabled databases is to publish existing relational data (regardless of its source) as XML.

In this chapter there are introduced three commercial leaders in XML-enabled databases: Oracle Database, Microsoft SQL Server, and IBM DB, more specifically the goal of this chapter is to provide an insight into the features with the help of which XML data are being processed in these databases.

2.1. Overall Introduction to Oracle XML DB

In 1979, Oracle Corporation introduced Oracle Version 2 as the first commercially available SQL-based database. After 22 years in 2001, Oracle9/Database was released to the world market. It included Oracle XML database that was designed to introduce the ability to store and query XML data [22].

Oracle XML DB is a set of Oracle Database technologies related to high-performance XML storage and retrieval [2]. It provides the storing and manipulating over XML data by including the tools for interaction between SQL and XML data models.

Oracle XML DB includes the following features to process XML data:

- Support for the World Wide Web Consortium (W3C) XML and standard methods to perform XML data. The XML data models are comprised into Oracle database
- Interoperation between XML and SQL data. Using SQL to perform with XML data and vice versa
- Light-weighted repository of database content including XML using file/folder/URL representation with wild-known standard ways on working with XML data
- New storage-independent, content-independent, programming language-independent infrastructure to perform XML data (for example: managing XML documents hierarchies)
- XML-specific memory management and optimizations

To build the XML-enabled application that will run in Oracle Database the set of components, tools and utilities are designed, called Oracle XML Developer's Kit (XDK) which can be used combined with Oracle XML DB.

2.1.1. Oracle XML DB Software Architecture

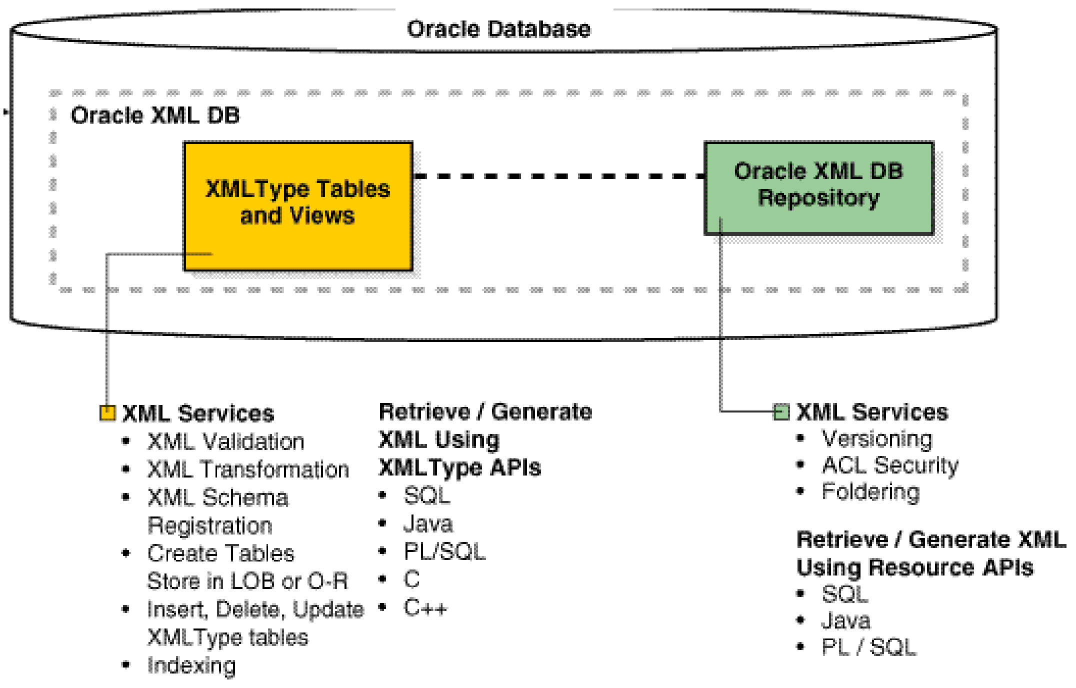


Figure 2.1: Oracle XML DB architecture

Figure 2.1 represents Oracle XML DB architecture. The main two features of the database architecture represented in the picture are storage of XML tables and views and Oracle XML DB repository. XMLType Storage includes the main services related to XML data manipulation such as basic functions upon data (create table, insert, delete, update), indexing, validation, transformation etc. The data can be retrieved from XMLType Storage using Structured Query Language (SQL), Java, Procedural Language/Structured Query Language (PL/SQL), C, C++. XML DB Repository includes versioning, ACL security, and foldering as its services. The data from repository can be retrieved using SQL, Java, PL/SQL.

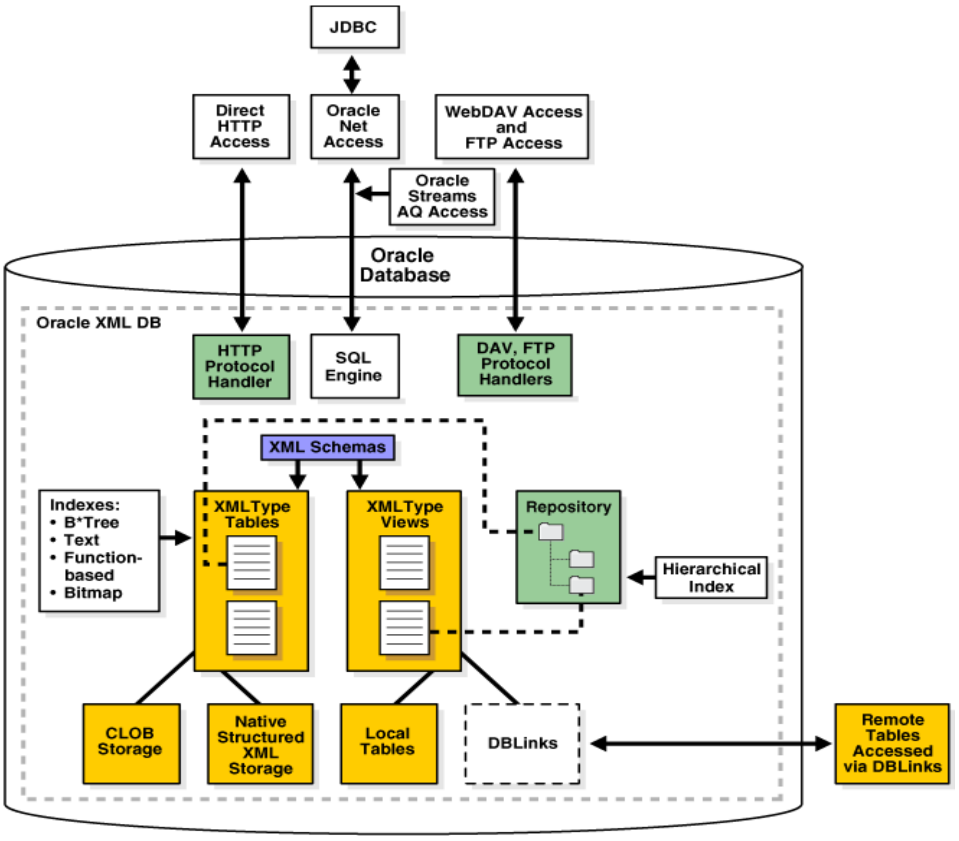


Figure 2.2: Oracle XML Type storage architecture

When any XML schema is registered with Oracle XML DB a set of default system tables are created in XMLType Storage. These tables contain XML instance documents associated with schema. XML schema consists of XMLType tables and views (see Figure 2.2).

There are two possibilities to store XMLType tables which are as a Character Large Object (CLOB) or natively structured (a set of objects). XMLType views can be stored in either local or remote tables which can be accessed using DBLinks. The indexing of XMLType tables and views can be performed using several data structures like B*Tree, Oracle Text, function-based indexes, and bitmap indexes.

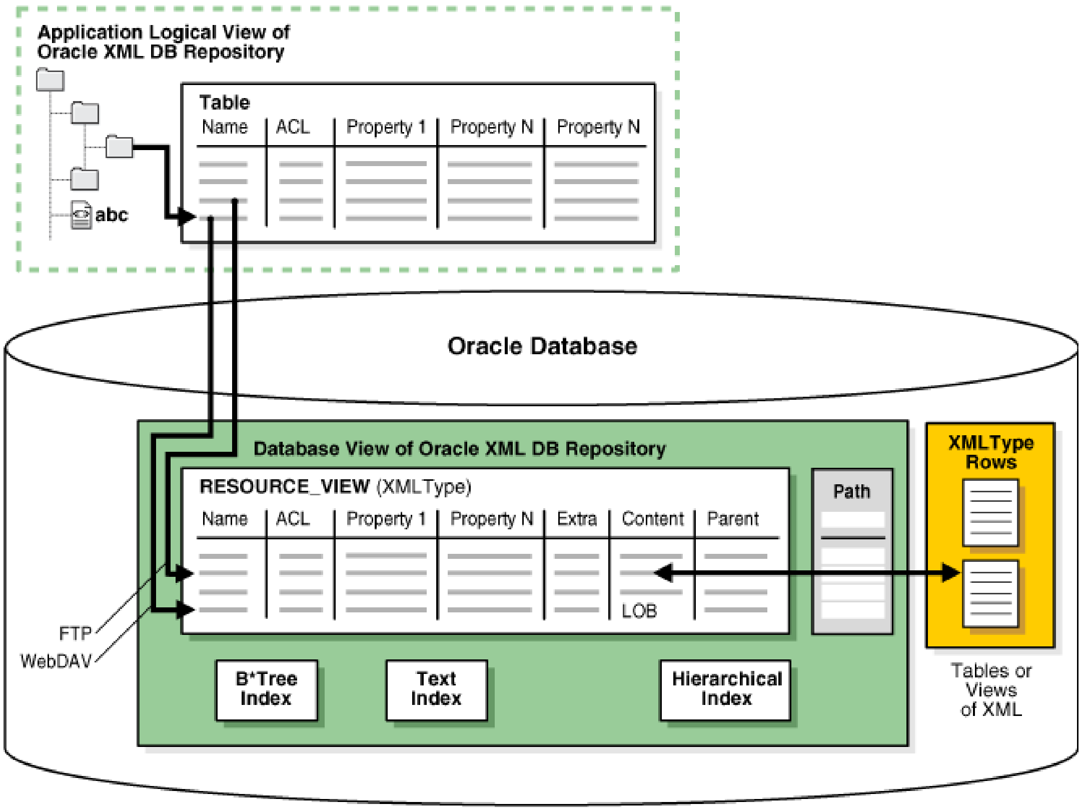


Figure 2.3: Oracle XML DB repository architecture

XMLType documents associated with the schema can be viewed and accessed in XML DB repository. Repository contains resources which are either folders or files. Each of the resources is identified by a path and a name (are contained in the system view PATH_VIEW), has content (can be XML but not necessarily), system-defined metadata (properties), user-defined metadata (which are not the part of the content but are associated with it), access control list (to define access and operation rights restrictions). Repository can be used to access any data in Oracle XML DB(see Figure 2.3).

End users along with different types of software like Microsoft Internet Explorer can directly access Oracle XML DB repository. No middleware is required.

Data in Oracle XML DB repository can be accessed with HTTP(S), Web-based Distributed Authoring and Versioning (WebDAV), File Transfer Protocol (FTP), or SQL using HTTP protocol handler, WebDAV and FTP protocol server, Oracle Net Services including Java Database Connectivity (JDBC) respectively [2].

All the above described protocols are supported with the same architecture that is used to support Oracle Data Provider for .NET (ODP.NET) in a shared server configuration. The Oracle Database shared server process receives the protocol request from the Listener, serves it and sends the response back to the client.

Oracle XML DB supports all major XML, SQL, Java, Internet standards.

2.1.2. When to Use Oracle XML DB

Oracle XML DB is designed for applications where large amounts of data proceeded by the application are represented in XML. It provides high performance manipulation with XML data as storage, processing and retrieval. In addition it provides the interoperation between XML and relational data (for example generating XML documents from existing relational data).

The types of applications which Oracle XML DB is suitable for are internet, content-managing applications, Web services, messaging and the integration of Business-to-Business (B2B) and Application-to-Application (A2A).

A typical Oracle XML DB application has following issues to be solved:

- Large numbers of XML documents must be generated
- Large XML documents must be processed
- Fast searching both within one document and through the collections of documents
- High level of security
- Data processing must be contained in XML documents and traditional relational tables
- Support open standards such as SQL, XML, XPath, and XSLT
- Use of standard internet protocols (HTTP(S)/WebDAV, FTP, JDBC) to access information
- Possibility of full querying from SQL
- Validation of XML documents

To comply with above described requirements a set of XML technologies are integrated into Oracle XML DB.

2.1.3. XML Technologies in Oracle XML DB

For any database to proceed and manipulate with XML data the main demand is to be able to persist XML documents. It also includes all the standard database features to cover not only relational data but also XML data. In addition there must be effective tools for XML documents to be indexed, queried, updated, and searched.

XMLType Datatype

To let the database recognize XML data a native server datatype XMLType is introduced in Oracle XML DB. XMLType provides a set of methods that allow common operations such as validation (method schemaValidate();) or transformation (method transform();) to be performed on XML content. XMLType can be integrated as a column in a relational table; it also can be used with PL/SQL procedures and functions.

XMLType tables and columns can correspond with XML schema. This will insure that all XML data that are stored in the table are validated against the XML schema. The information contained in XML schema also provides more effective way to update and process XML. It also allows storing the content of XML document as a set of SQL objects. All the relational and object-relation data can be wrapped into XMLType views if necessary.

XML Schema Support

Support for the W3C XML Schema Recommendation is a key feature in Oracle XML DB. XML Schema specifies the structure, content and certain semantics of a set of XML documents [4].

With XML Schema it is possible to specify a set of options for tables and columns of XMLType that are based on the schema maintaining DOM precision at the same time. All SQL methods and functions that allow an XML schema to check whether instance documents conform to their specifications are included into Oracle XML DB.

XML/SQL Duality

The XML/SQL duality means basically the interoperation between XML and SQL data. In other words the tools to manipulate with SQL data can be used over XML documents and vice versa. It brings flexibility to use the most appropriate and effective methods for solving different types of problems.

The essential difference between SQL and XML data is that SQL data are structured and viewed at through a table/raw metaphor whereas XML data are either semi structured or unstructured and are viewed at through hierarchy/node metaphor. In Oracle XML DB this traditional boundary is erased which allows the same data to be exposed both as rows in a table and nodes in an XML document. This brings the great advantage of data being interchangeable between different types of applications.

Oracle XML DB introduces the SQL functions defined in the SQL/XML standard which are either designed to query and access XML content as part of normal SQL operations or to generate XML from the result of a SQL SELECT statement.

XPath Expressions

For searching collections of XML documents or extracting a subset of the nodes within one XML document XPath expressions can be used along with the SQL/XML functions and their corresponding XMLType methods.

There are two ways which XPath expressions can be evaluated by Oracle XML DB. These are dependent on the used method of XML data storage:

- <u>Structured-storage XML data:</u> XPath expression inside SQL/XML function is translated into an equivalent SQL query. This process is called *XPath Rewrite* and it occurs mostly when performing queries and UPDATE operations. This allows optimizing the processing of XPath notation as a purely relational SQL statement.

- Unstructured-storage XML data: XPath is evaluated by building a DOM tree and resolving XPath programmatically. If the expression implies updating a DOM tree the whole XML document has to be rewritten [6].

With the above described features Oracle XML DB provides a high performance when storing and retrieving complex, large, or many XML documents.

XML type data generate higher overhead than other representations. For this reason, Oracle XML DB introduces a number of further features that are designed specifically to improve the performance of XML processing [2].

2.2. Overall Introduction to IBM DB2 pureXML

Since the 1970s IBM Research invented a complete family of data servers started on mainframe platforms such as Virtual Machine (VM) and Multiple Virtual Storage (MVS). In 1983, DB2 for MVS Version 1 appeared. "DB2" abbreviation was used to indicate a shift to the new relational databases. In 1996, IBM announced DB2 Universal Database (UDB) Version 5 for distributed platforms. On a growing demand of XML documents storage and usage the DB2 9's pureXML technology was released in July 2006 [21].

PureXML [7] is a new set of technologies in DB2 9 that includes functions and features that allow XML data to be stored natively in a database tables. It provides us with more efficient management over large number of XML documents and integration of XML data with traditional data.

The pureXML technology includes following features:

- PureXML data type and storage techniques to manipulate efficiently with XML data taking into account a hierarchical structure of XML documents
- Indexing technology to accelerate the searching within one XML document or a collection of XML documents
- New query language support (for XQuery and SQL/XML) that corresponds with all industry standards and includes query optimization techniques
- Support for XML Schema validation and management
- Integration with all popular application programming interfaces (API) and development environments
- Integration of XML into existing relational models
- The applying of all enterprise standards like reliability, scalability, performance, security to XML data

DB2 9 includes a set of technologies to improve and simplify the development of both XML centric and hybrid applications. It supports the following programming languages and application interfaces:

- Languages: C/C++, Java, C#, Visual Basic, Cobol, PHP
- Interfaces: JDBC, .NET, Embedded SQL, SQLJ

Along with languages and interfaces DB2 9 includes a capability to query data using XQuery and SQL/XML.

DB2 9 provides us with the possibility to store and access both relational and XML data in the same database. The collection of XML documents are stored in tables that contain one or more columns based on XML data type. XML document is stored by being parsed into a set of fragments that helps to preserve its hierarchical structure.

DB2 9 allows us to design a database that maintains the consistency and integrity of different data types with the possibility of creating a hybrid database.

2.2.1. IBM DB2 pureXML Architecture

Figure 2.4 represents architecture of the hybrid database system:

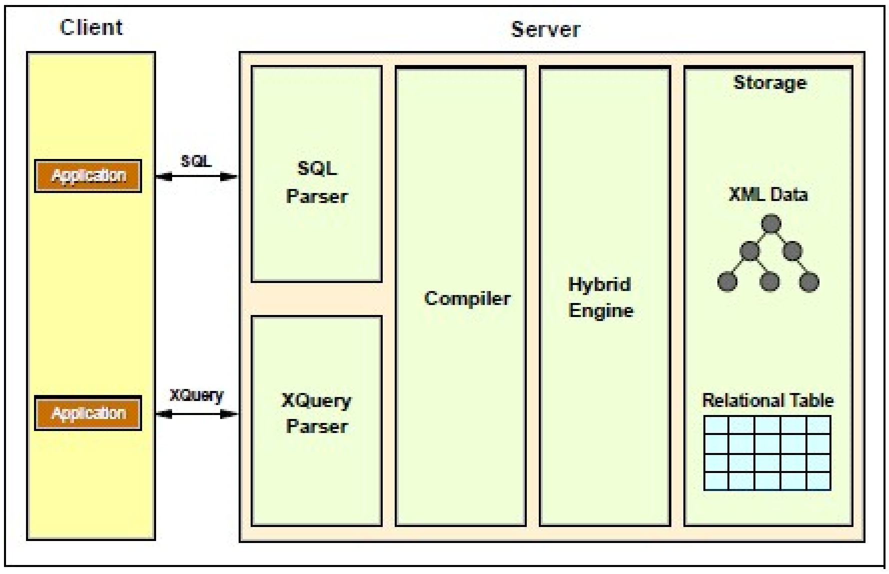


Figure 2.4: IBM DB2 hybrid database architecture

On the server side there is one hybrid engine that manipulates with both relational and XML data, which are stored in tabular and tree structure respectively. To process the two querying languages from the applications on the client side there are two different parsers, SQL parser and XQuery parser. To handle both languages a single compiler and optimizer is used in DB9. The hybrid database can process the combinations of SQL and XQuery from the applications to access both relational and XML data.

In DB2 hybrid database XML is treated as a native data type. DB2 pureXML provides us with possibility of keeping XML documents in its hierarchical structure. For these purposes relational and XML data are stored separately.

In order to insert and import XML documents into an XML column, it is essential that they are separated from the other contents of the table and has its own individual table space. The mechanism of this type of storage is similar to the long data types (LONG VARCHAR or LOB data). Because XML data objects are stored separately from parent table objects XML Data Specifier (XDS) is introduced in the DB2. For each row of XML type column the XDS includes all the information needed to access the XML data stored in the disk.

During inserting and importing XML documents are validated with XML Schemas that are supported in DB2. XML Schemas used for validation are registered in a special XML Schema Repository (XRS) [7]. The reason for the demand of XML Schemas being stored in different repository is its main difference from a relational database schema. XML Schema is a language to describe the structure of the XML documents whereas relational database schema is a collection of database objects used to logically group these and as a name qualifier.

Figure 2.5 represents table, XDS and XML data relationship.

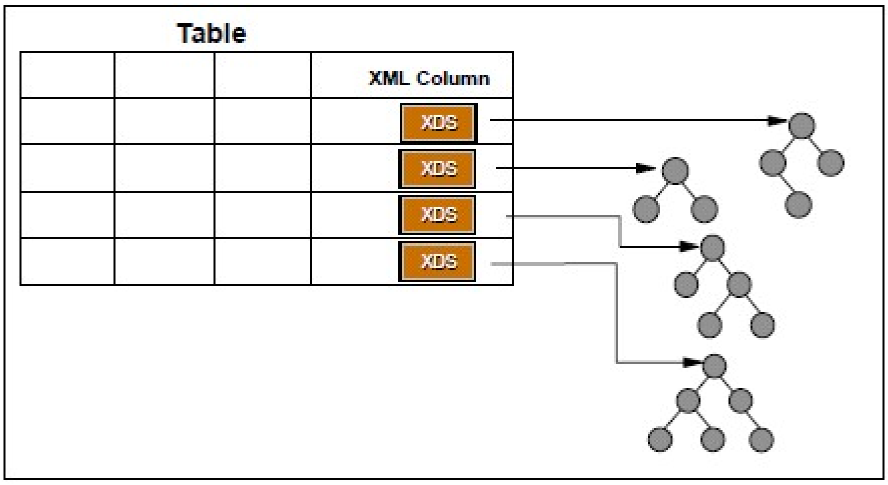


Figure 2.5: IBM table, XDS, XML data relationship

An XML document can also be inserted into a LONG/LOB VARCHAR type column which allows both inserting and selecting the whole document to be faster than the ones using XML native data type. The reason for this is that while inserted as a LONG/LOB VARCHAR XML document is neither parsed nor serialized. But because there is no parsing XML document is not checked for well-formedness which leads to slow searching and extracting.

XML documents can be decomposed in order to map into the relational table and contrariwise decomposed XML documents in relational tables can be composed and published to an XML document. However, during decomposing XML document loses most of its structure (for example the order of the XML document is lost) which leads to published XML document being different from the original one. To maintain the order of the XML document it is essential to store it in either CLOB/LONG VARCHAR or XML type column.

2.2.2. When to Use IBM DB2 pureXML

Building new applications it can be found that the XML data model has several advantages over relational data models, dealing with:

- Semi-structured or unstructured data (biological data, healthcare records etc.)
- Hierarchical, nested or complex data (catalogs, manuals, books etc.)
- Data with dynamically changing schemas (forms, new versions etc.)
- Data including null and multiple values (addresses, phone numbers etc.)

The technologies to store and manage XML data are introduced in DB2 which provides us with possibility of using XML data model with pureXML.

2.2.3. XML Technologies in IBM DB2

Due to the main demand of the hybrid database to proceed the collections of XML documents keeping its natural hierarchical structure it is necessary for the database to include a set of features and mechanisms that are optimized for working with XML type data. It is essential for these purposes so that the hybrid database contains the tools to manage, validate, query, index, and search XML data.

XML Data Type

XML data type is a native data type introduced and supported in DB2 to manipulate with XML data. In order to proceed and manipulate with XML data in hybrid database, all XML documents must be well-formed before being inserted into a database otherwise the statement will fail with error code. All the rules for well-formedness are specified in the W3C standard for XML [8].

After being inserted into a database all XML documents are stored in a parsed tree structure. This internal representation is not compatible with the string data type; however it can be transformed to a string type using the function XMLSERIALIZE. That brings up a question what data should be stored in XML. Generally, the data that is better described with hierarchical form, and has most attributes empty or unknown with a highly complex structure should be represented in database with XML data type.

XML Schema

In hybrid database XML Schema can be used to validate XML instance documents, for these purposes schemas can be registered in XSR that is provided by DB2. XSR is located in the database catalog. To access XSR object information the system view SYSCAT.XSROBJECTS can be queried. XSR objects can also be managed in the DB2 Control Center which also supports adding and dropping XML schemas. In hybrid database a single XML column can contain XML documents with different schemas. The documents can be validated both before inserting and during it.

XML Indexes

Indexes are created to fasten finding and accessing data, and also to improve query performance. DB2 supports creating XML indexes to perform standard functions over XML data. Along with speeding up the queries XML indexes have disadvantages of slowing down such operations as insert, update, delete and also requiring extra space to store the index data.

Unlike relational indexes which can be composed of more than one table columns and provide access to the beginning of the document, every XML index uses a particular XML pattern expression to index the nodes in each of XML documents stored within a single column.

XQuery and SQL/XML

XQuery and XPath are known languages to manipulate with XML data. However, in hybrid databases there is a demand to access both XML and relational data simultaneously. For these purposes, DB2 introduces two possibilities which are embedding SQL in XQuery and embedding XQuery in SQL statements.

To embed SQL statements into XQuery a function db2-fn:sqlquery can be used. SQL full select statement is processed as an argument that must return XML data. Embedded SQL statement is not limited to be applied only to the table containing XML column we are retrieving.

To embed XQuery expressions in SQL the functions XMLQUERY, XMLTABLE, XMLEXISTS can be used. These functions are the part of SQL/XML language that defines a set of functions to query, validate, and transform XML data. SQL/XML can also be used when joining XML data with relational data, and for grouping and aggregating XML data.

Generally, in DB there are four possibilities to query XML data such as: plain SQL to only work with full XML documents (insert, update, delete); SQL/XML to join relational and XML data and use predicates on both of the types, the possibility to extract and work with fragments of XML data; XQuery to query XML data only (extract, transform, join); and XQuery with embedded SQL to use predicates based on non-XML data, call external functions on the XML columns and perform full-text search.

XML Full-Text Search

There are two possibilities in DB2 to perform a full-text search within one XML document or a collection of XML documents. The XQuery support allows the users to search data with XML documents. However, XQuery function provides us only with the possibility of performing a search based on a simple substring match. Because in the XQuery Data Model text is a node, XQuery cannot search efficiently through XML documents that contain large amounts of text.

The solution to the above described issue is a new feature designed in DB2. DB2 Net Search Extender (NSE) allows searching within large unstructured collections of XML documents efficiently.

XML Data Security

There are two ways of controlling the access to the data contained in XML type column such as access control at either row-level/column-level or at XML-nodes level.

For the first type of access control (which users are allowed to access which rows and columns) DB2 feature label-based access control (LBAC) is designed.

To control user access on XML elements within one XML document the VIEW and XMLTABLES function can be used.

2.3. Overall Introduction to Microsoft SQL Server XML DB

Microsoft SQL Server 1.0 was invented in 1989 by Microsoft, Sybase. The support of XML data was first introduced in Microsoft SQL Server 2005 along with Integration and Analysis Services.

Microsoft SQL Server and SQLXML Web Releases introduce a set of features to provide the capabilities of effective XML data management specifically processing of XML data and mapping between relational and XML data. In addition, it includes the tools to query XML data, define XML schemas, index XML type columns which meet all the requirements while developing applications for semi-structured and unstructured data management.

There are two main approaches to manipulate with XML data:

- XML-centric approach: to query and update collections of XML documents using XML views and schemas
- SQL-centric approach: to map the results of querying relational data to XML (FOR XML) and generate relational views from XML (OpenXML) using Transact-SQL

The above described features bring new possibilities to data storage and application development. While developing application it is essential to choose the most suitable format to store data. XML format is appropriate while working with data with unknown or dynamically varying structure; data that may be recursive or hierarchical; data that are queried in parts based on its structure. To support these requirements a set of tools is introduced in Microsoft SQL Server, such as:

- All standard administrative functionality while manipulating with XML data (replication, backup etc.)
- Querying, searching and modifying sections of XML data within one XML document without replacing the whole document

- The possibility of interoperation between XML and relational data within one application
- The requirement of data well-formedness and validation according to XML schemas
- Effective processing of XML data using XML indexes
- Controlled access to XML content

In case the application main demands are storing and retrieving data the alternative of using XML native data type is to store data in LOB (nVARCHAR, nBINARY) column. If the data used in application are highly structured with known schema it is better to use relational data model.

2.3.1. XML Storage and Access in Microsoft SQL Server

There are three main options to store XML data in Microsoft SQL Server. These are the following:

- Storage in native XML data type

  The internal representation of XML data in this case corresponds with the content of XML documents preserving its hierarchical structure, elements and attributes values. For untyped XML documents (that are not validated according to XML schema) the text content stored in database may not be the copy of the original content. Contrariwise, in case XML document is typed all the information about its content is embedded to the internal representation.

- Mapping between XML and relational storage using XML views

  XML document is divided into one or more relational tables using Annotated XML Schema Definition (AXSD) at the relational level preserving the hierarchical structure of the document. However, the order of the elements within the document is ignored. The AXSD in this case can not be recursive.

- Large object storage using nVARCHAR or nBINARY type

  The content of the document is the exact copy of the original one in the internal representation.

In order to comply with the requirements of modern applications the combination of three above described approaches can be used. As an example, the non-recursive parts of XML data can be stored in non-XML columns whereas the recursive parts will be stored in XML data type columns.

Different capabilities of storing XML data varies according to several factors such as: the nature and the extent of the queries, optimization demands (indexing XML data), data modification requirements (transforming and updating of XML data), the support of XML schema. Different choices of XML data storage have different performance characteristics.

In order to use a combination of relational and XML data type columns it is appropriate to use hybrid data model. This type of data modeling may improve the performance in manipulating with data. However, it also requires greater control in prospect of managing the data storage.

XML data type column can be used in combination with the relational typed columns at the same table or in the different table using foreign key. When accessing and retrieving XML data from the table the first option is better to be used when XML index is not needed or the primary matches the clustering key, otherwise the second option should be used.

2.3.2. When to Use Microsoft SQL Server

Storing XML data in a relational database provides benefits in the areas of data management and query processing [9]. In Microsoft SQL Server all standard capabilities and tools while manipulating with relational data are extended to query and modify XML data. It also solves the issue of XML data to interoperate with relational data within one database to meet the requirements of modern applications. New features and tools bring the benefits in conditions of increasing XML usage nowadays.

2.3.3. XML Technologies in Microsoft SQL Server

This section provides an overview of the technologies and tools introduced in Microsoft SQL Server to support XML data in database. It also includes the mechanisms for mapping data back and forth between the relational and XML data models both from client and server sides.

XML Data Type

Microsoft SQL Server version 2005 and above includes a native data type for XML. All XML type parameters and data are structured as in a relational table in MS SQL Server XML DB. For that reason, it is possible to perform both-way conversion meaning to shred XML into multiple rows or retrieve SQL type values from the XML instance using nodes() and value() functions respectively.

Well-formedness of XML documents is essential in order to effectively process XML data. XML data type column can be typed according to one or more XML schemas or left untyped. However, to avoid performance penalty untyped XML documents are checked to be well-formed even thought it is not associated with any XML schema.

It is also possible to convert XML data to string type and reversed using functions CAST and CONVERT.

XML Schema Processing

XML documents, columns, variables and parameters can be optionally typed according to a collection of XML schemas. Every XML instance includes the namespace for XML schemas it is typed to. While inserting, deleting or modifying every XML instance is validated according to XML schema it conforms to.

The validation of XML documents using XML schema assures the well-formedness of XML data which brings the benefits of storage and query optimization. Typed XML instances contain typed values in XML indexes. This provides more effective high performance processing of XML data.

FOR XML and OpenXML features

FOR XML functionality has been expanded in Microsoft SQL Server to provide the efficient way of manipulating with XML data. FOR XML clause is used to generate XML data instances by aggregating the relational row set and can be embedded in standard SELECT statement. It supports three modes: RAW to generate single elements for each row returned, AUTO generates a simple hierarchy based on the lineage information and the order of the data, EXPLICIT to map a specific row set formulated by a single SQL query into XML shape. All three modes are designed to provide the possibility of generating large XML documents efficiently.

OpenXML provides a table structured view over XML documents. It is a part of Transact-SQL and can be embedded in standard SELECT statement. This keyword access to XML data as though it is a relational row set. In order to query XML data using OpenXML the XML document originally is parsed to a tree representation of nodes in XML document. This allows applying SQL-statements to XML content by viewing on it as a relational table. It also can be used to insert XML documents into a database.

XQuery

To world with XML data as a native build-in type in Microsoft SQL Server XQuery can be used through a set of SQL methods. The input parameter for each of these methods must be in nVARCHAR type. For both XML and SQL portions of the query one query engine is used to process it. For this reason when it comes to XML instance it is essential to apply XML indexes in order to optimize the processing of the whole query.

XML Indexing

In Microsoft SQL Server there are for types of indexes to fasten the query processing: one primary index and three secondary. For indexing relational data B^+ trees are used.

A primary XML index creates a B+ tree on all tags, values and paths of XML documents contained in the column. It provides efficient evaluation of queries on XML data preserving document order and document structure.

Secondary XML indexes are designed to speed up different classes of queries. There are three types of secondary indexes: PATH index for path-based queries, PROPERTY index for property bag scenarios, and VALUE index for value-based queries [10].

Client Access to XML Data Type

XML data in the server can be accessed by clients using Open Database Connectivity (ODBC) and XML data are delivered as a Unicode string by Object Linking and Embedding Database (OLE DB).

2.4. Conclusion

For better understanding of similarities and differences in XML support Table 2.1 provides an overview of XML related features that are (not) supported in the three systems and their key characteristics.

Feature	Oracle	DB2	SQL Server
XML type	**XMLType**	**XML**	**XML**
	structured, binary, non-structured	*binary, structured*	*native, structured*
Mapping	**User-defined**	**User-defined**	**User-defined**
	names, data types and storage strategies	*relations, columns, conditions, expressions*	*primary, secondary, full-text index*
Indexing	**XMLIndex**	**Region index, column path index, XML Index**	**Primary, secondary, full-text index**
Querying	**XQuery, SQL/XML**	**XQuery, SQL/XML, SQL embedded into XQuery**	**XQuery, SQL/XML, SQLXML (OPENXML, FOR XML)**
Transformations	**XSL**	**XSL**	**XSL via external tools**
Updates	**Internal functions**	**XML Update facility**	**Internal function with parameter**

Table 2.1: Comparison of key XML features

As we can see, in general, all three systems follow the same pattern when trying to support as much XML functionality as possible, using already existing tools along with creating its own solutions.

Firstly, while all three systems support a kind of XML data type as well as shredding XML data into relations, the storage strategies are specific in each system and not dependent on any kind of common standards. In case of mapping support, all three systems leave the definition on a user completely and do not support embedded user-driven strategies.

As for the query capabilities all the systems provide the functionality to use to use pure XQuery language (in case of referencing to an XML document stored in database directory) and its embedding in SQL (in case of storing XML in relational tables). Both Oracle and DB2 support the SQL/XML standard unlike SQL Server, which represents this functionality with its own set of functions called SQLXML. In addition, DB2 supports a new feature – embedding SQL queries into XQuery.

The transformation operations are represented in all the systems by a natural support of XSL transformations, as for the update operations, each of them has its own approach. Oracle provides a set of update functions, SQL Server provides a single function with multiple parameters and DB2 supports the XQuery Update Facility, i.e. a standard approach.

Considering all the above described similarities and differences, it is seen that Oracle follows the most standard-conforming way of supporting XML technologies, SQL Server tends on creating its own functionality, whereas DB2 combines both approaches according to each XML technology in particular [23].

3. Application Goals

The application created as a part of this thesis is called XMLTest. From now on when referring to this application, we use the name XMLTest. XMLTest is designed as a tool that provides a possibility of running tests on several technologies for XML data storing and querying in three database systems: Oracle Database 11g Release 2, Microsoft SQL Server 2008 R2, IBM DB2 9 Express C. These objectives can be further divided into more detailed points:

- Using different methods of testing XML technologies in a database
- Storing XML data in a database
- Querying XML data in a database
- Representation of the experiment results

3.1. Using Different Methods of Testing XML Technologies in Database

In order to run experiments using XMLTest it is essential to provide it with appropriate data for testing which include XML documents to be tested on, and a set of querying statements to execute. There exist two main approaches for collecting necessary base: creating a unique new experiment or using the data from existing benchmarks.

3.1.1. XML Benchmarks

An XML Benchmark *"is a specification of a set of meaningful and relevant tasks, intended to assess the functionality and/or performance of an XML processing tool or system"*[24]. In other words, XML Benchmark is a tool that estimates the performance of XML technologies or systems by executing a set of tasks upon generated XML data.

Along with performing a unique new experiment, XMLTest allows us to compare database systems performances using one of the most common modern XML Benchmark – XMark [24]. XMark models an internet auctioning application. The workload consists of a large database, in a single XML document and 20 queries that cover the following broad kinds of operations: simple node selections; document queries for which order information are relevant; navigational queries; and computing aggregate functions. The database document is generated with XMark data generator which uses several real data values to produce realistic data. In XMLTest four database documents are pre-generated depending on the chosen size: small (100KB), normal (1MB), big (10MB), and huge (100MB).

3.2. Storing XML Data in Database

As it was described in Chapter 3 there is a wide range of methods to store XML data in databases. There are two main points to consider when choosing one storage method over another: first, whether to store XML data into relational structure or import the document into database directory as an independent object; second, if choosing relational structure whether to store the document as a whole or shred it into several tables and columns.

When working with database systems that are XML-enabled in order to closely examine the ways they handle interaction of XML and relational data, it is more applicable to store XML documents into relational tables. It is also important to examine the ways of querying the whole XML document table using XML technologies rather than using relational tools in case of XML shredding. For these reasons, in XMLTest application all XML documents are stored as a whole into relational tables. It provides us with two ways to do so: store XML document as a CLOB type (in other words, as a text), or maintain its structure by storing it as an XML type.

3.3. Querying XML Data in Database

XMLTest allows us to execute statements upon testing data with two most common technologies: XQuery language and SQL/XML tool.

Using SQL/XML is the most efficient way to apply XML querying languages directly upon an XML document stored in a relational table. More specifically, it allows us to embed an XQuery statement into SQL statement to approach XML data stored in any relational structure.

In case of choosing XQuery language, the querying document is represented with pure XQuery statements which then are then be embedded dynamically into SELECT statements within the program in order to be applied on relational tables.

3.4. Representation of the Experiment Results

Three main sections of experiment results that must be represented in order to perform a complete comparison analysis of the database systems are the following: querying statements execution results, performance statistics, errors during processing.

To fully represent the statements execution results, XMLTest results tab (see Section 4.3) output includes each statement after converting it into SQL/XML (if needed) and a corresponding database results set ('Not succeeded' line in case execution has failed). In case the conversion was not fully proper XMLTest provides the possibility of correcting particular statements.

The performance statistics section (see Chapter 4.3) in XMLTest shows four comparison graphs: the number of successful query executions, total time elapsed for processing, average time elapsed for one statement processing, and total performance coefficient which is composed based on above described values.

The error tab output includes the names of the errors while processing for each database. The error tab allows us to estimate the reason statement execution failed. Typically the most probable errors are ones of syntax mistakes (then it can be corrected), or the case of one or more parts of XQuery languages are not supported by the database system.

4. User Documentation

The following sections describe how to run an experiment using XMLTest.

4.1. Configuration

In order to successfully run the experiments using XMLTest it is necessary to configure database systems so that they meet all the requirements for connection establishing.

4.1.1. Oracle

The following list represents the requirements for Oracle database:

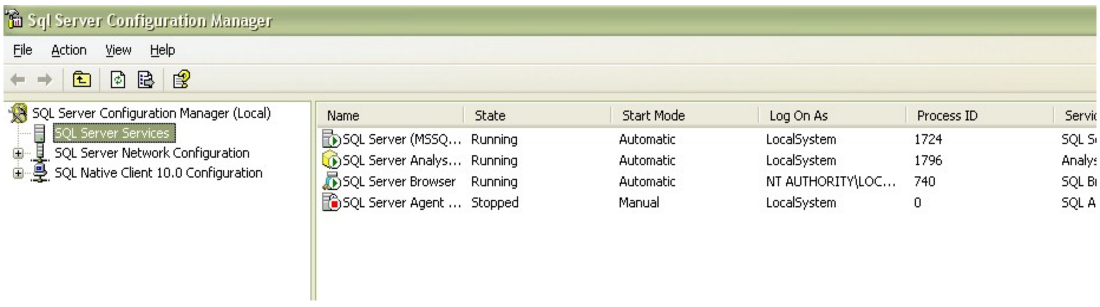
- Installed software: Oracle Database 11g Release 2 [35]
- Database installed and created on localhost, port 1521
- Database name: orcldb, username: SYSTEM, password: Pa$$word1
- Administrator authorization

4.1.2. SQL Server

The following list represents the requirements for SQL Server database:

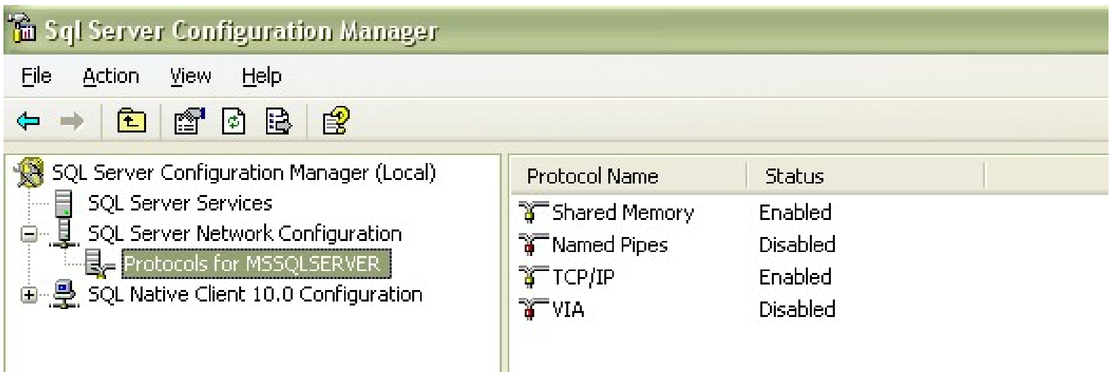
- Installed software: Microsoft SQL Server 2008 R2 [36]
- Database installed on localhost, port 1433
- Database name: tempdb, username: sa, password: Pa$$word1
- SQL Server Browser running (see Figure 4.1); start mode: automatic, built-in account: Local Service
- TCP/IP protocol: enabled (see Figure 4.2); TCP Port: 1433 (has to be specified)

To complete all the following requirements visit SQL Server Configuration Manager:


Sql Server Configuration Manager
File
Action
View
Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration
SQL Native Client 10.0 Configuration

Name	State	Start Mode	Log On As	Process ID	Servi
SQL Server (MSSQ...	Running	Automatic	LocalSystem	1724	SQL S
SQL Server Analys...	Running	Automatic	LocalSystem	1796	Analy
SQL Server Browser	Running	Automatic	NT AUTHORITY\LOC...	740	SQL B
SQL Server Agent ...	Stopped	Manual	LocalSystem	0	SQL A

Figure 4.1: SQL Server Configuration Manager Server Browser running


Sql Server Configuration Manager

File Action View Help

SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration
Protocols for MSSQLSERVER
SQL Native Client 10.0 Configuration

Protocol Name	Status
Shared Memory	Enabled
Named Pipes	Disabled
TCP/IP	Enabled
VIA	Disabled

Figure 4.2: SQL Server Configuration Manager TCP/IP enabled

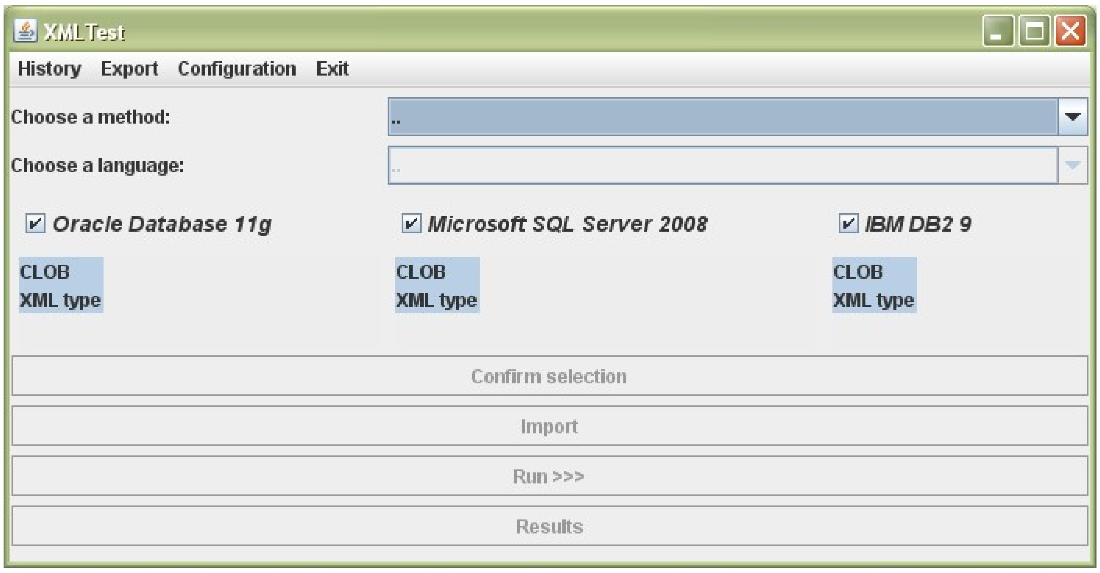
4.1.3. IBM DB2

To complete all the following requirements visit Configure Database Logging Wizard in Control Center Advanced view:

- Installed software: IBM DB2 9 Express C [37]
- Database installed and created on localhost
- Database name: tempdb (no username or password)
- Archive Logging: on
- Number of Primary Log Files: 256, Secondary Log Files: -1, Size of Each Log File: 512

4.2. Data Import

Figure 4.3 shows XMLTest main window at the first use.


XMLTest

History Export Configuration Exit

Choose a method: ..

Choose a language: ..

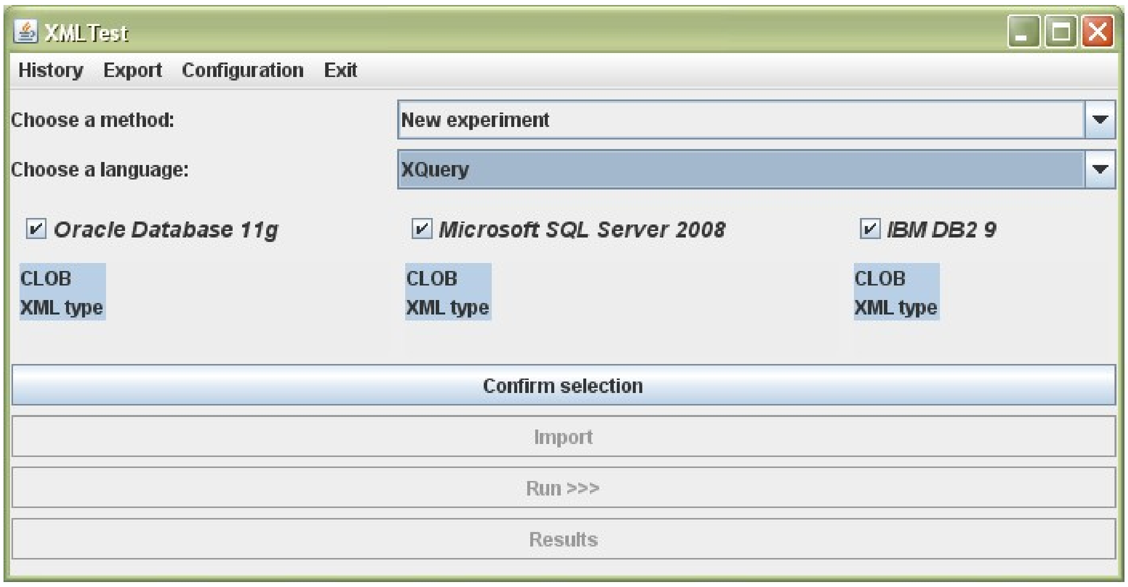
☑ Oracle Database 11g — CLOB, XML type

☑ Microsoft SQL Server 2008 — CLOB, XML type

☑ IBM DB2 9 — CLOB, XML type

Figure 4.3: XMLTest main window before at the first use

"Choosing a method" combo box allows us to choose the type of experiment to run: a new experiment upon user data (see Figure 4.4), or XMark benchmark (see Figure 4.5).



Choose a method: New experiment

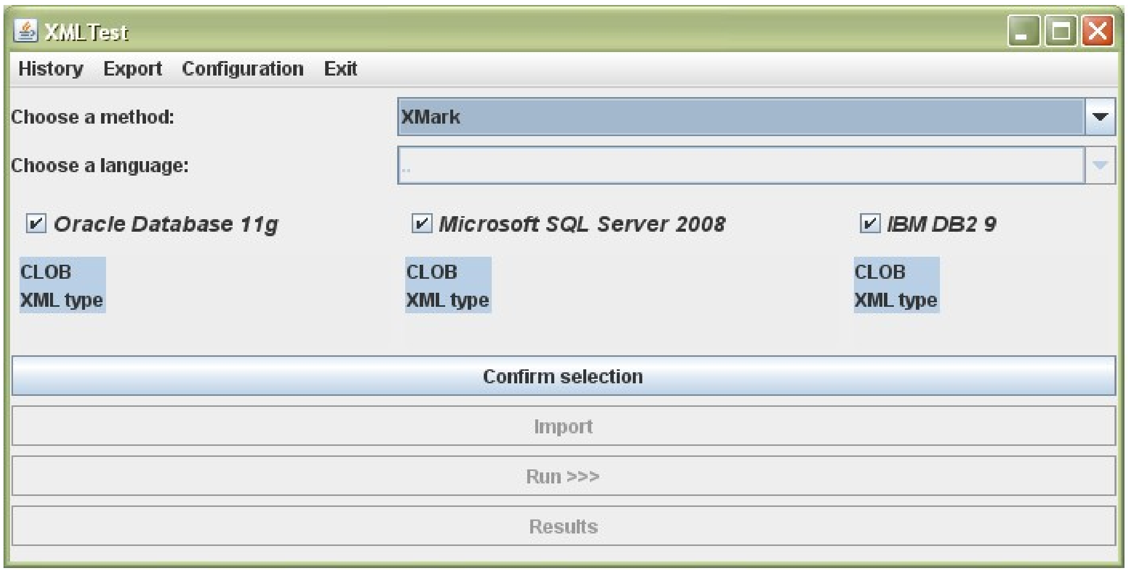
Choose a language: XQuery

☑ Oracle Database 11g — CLOB, XML type

☑ Microsoft SQL Server 2008 — CLOB, XML type

☑ IBM DB2 9 — CLOB, XML type

Figure 4.4: XMLTest choosing a new experiment



Choose a method: XMark

Choose a language:

☑ Oracle Database 11g — CLOB, XML type

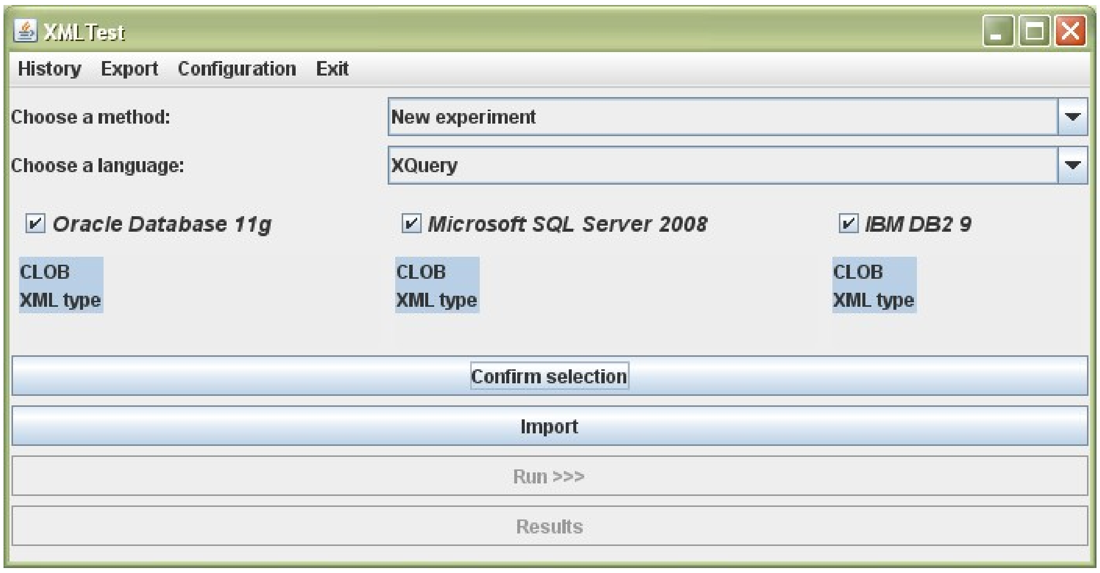
☑ Microsoft SQL Server 2008 — CLOB, XML type

☑ IBM DB2 9 — CLOB, XML type

Figure 4.5: XMLTest choosing XMark benchmark

When choosing a new experiment a language of following querying should be chosen in "choose a language" combo box: pure XQuery, or SQL/XML. When choosing XMark, the querying language is pure XQuery by default, for that reason "choose a language" combo box is disabled.

As the next step, by selecting check boxes and rows in storing method lists we choose database systems to compare and XML data storing method in each of them. To choose more than one storing method Ctrl button should be held while selecting rows from the list. After all the parameters are selected by pressing "confirm" button we save all the options. It is necessary to confirm parameters every time any of it is changed otherwise changes will have no influence on the experiment.



Choose a method: New experiment

Choose a language: XQuery

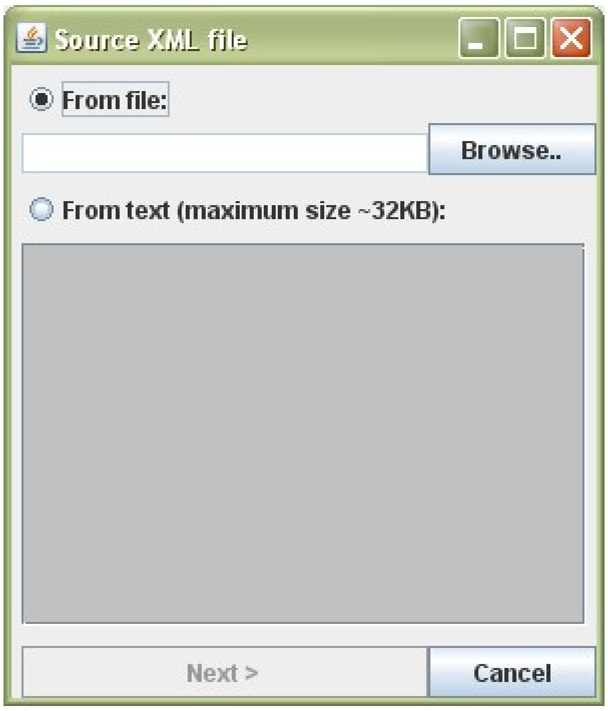
[✓] Oracle Database 11g — CLOB, XML type

[✓] Microsoft SQL Server 2008 — CLOB, XML type

[✓] IBM DB2 9 — CLOB, XML type

Figure 4.6: XMLTest is ready to import data

When all the options for the experiment are final and confirmed the XMLTest application is ready to import data, which is indicated by enabling the "Import" button (see Figure 4.6).

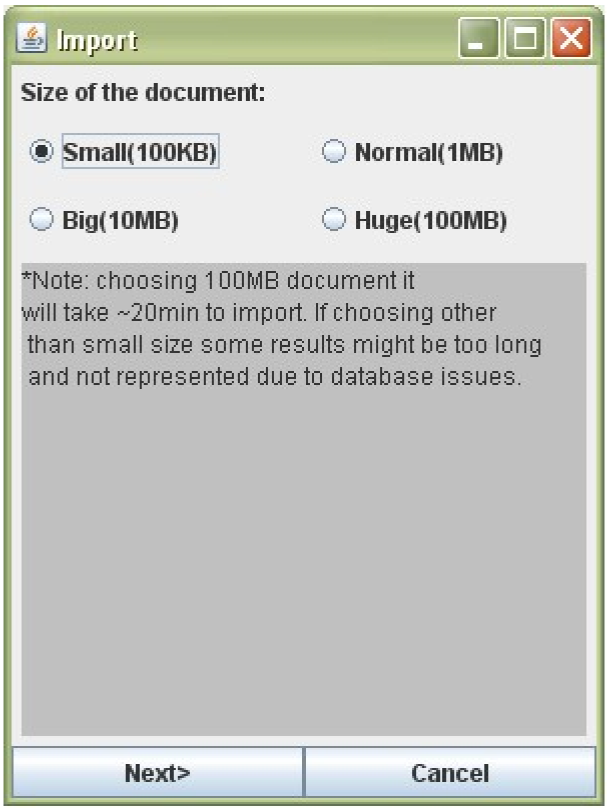


● From file:

○ From text (maximum size ~32KB):

Figure 4.7: Import data when choosing XQuery language

Running a new experiment when XQuery or SQL/XML language is chosen there are two options to load an XML document which will be tested: by selecting a file from computer file system, or by copying the contents of XML document into a text area (see Figure 4.7). In both cases the source XML document must be well-formed and have .xml extension, otherwise it will be not possible to proceed with the next step.



Small(100KB) [✓]
Normal(1MB)
Big(10MB)
Huge(100MB)

*Note: choosing 100MB document it
will take ~20min to import. If choosing other
than small size some results might be too long
and not represented due to database issues.

Figure 4.8: Import data when choosing XMark

In case of running XMark benchmark experiment the XML database document of specified size is already generated and ready for the import (see Figure 4.8).

The second part of the base data for a new experiment is querying document that includes statements to be executed upon source XML document. Each query statement must have an empty line after it, and two empty lines after the last one in the document. Incorrectly formatted querying document might cause errors in processing results.

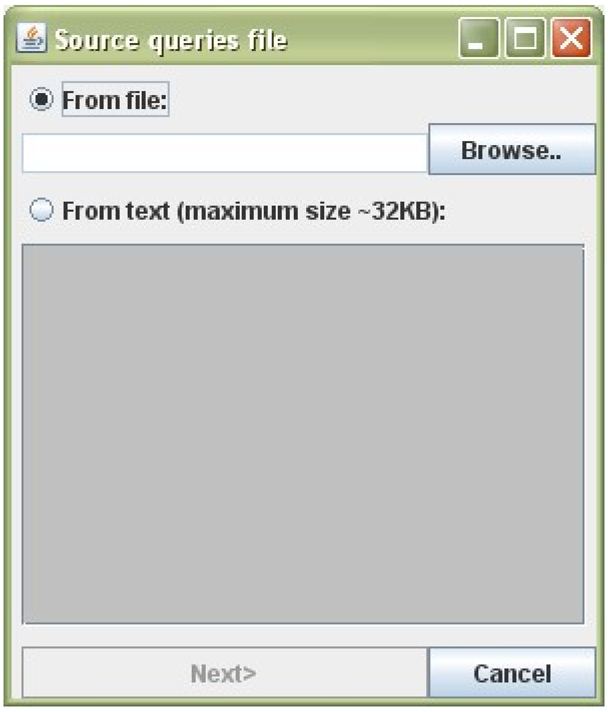
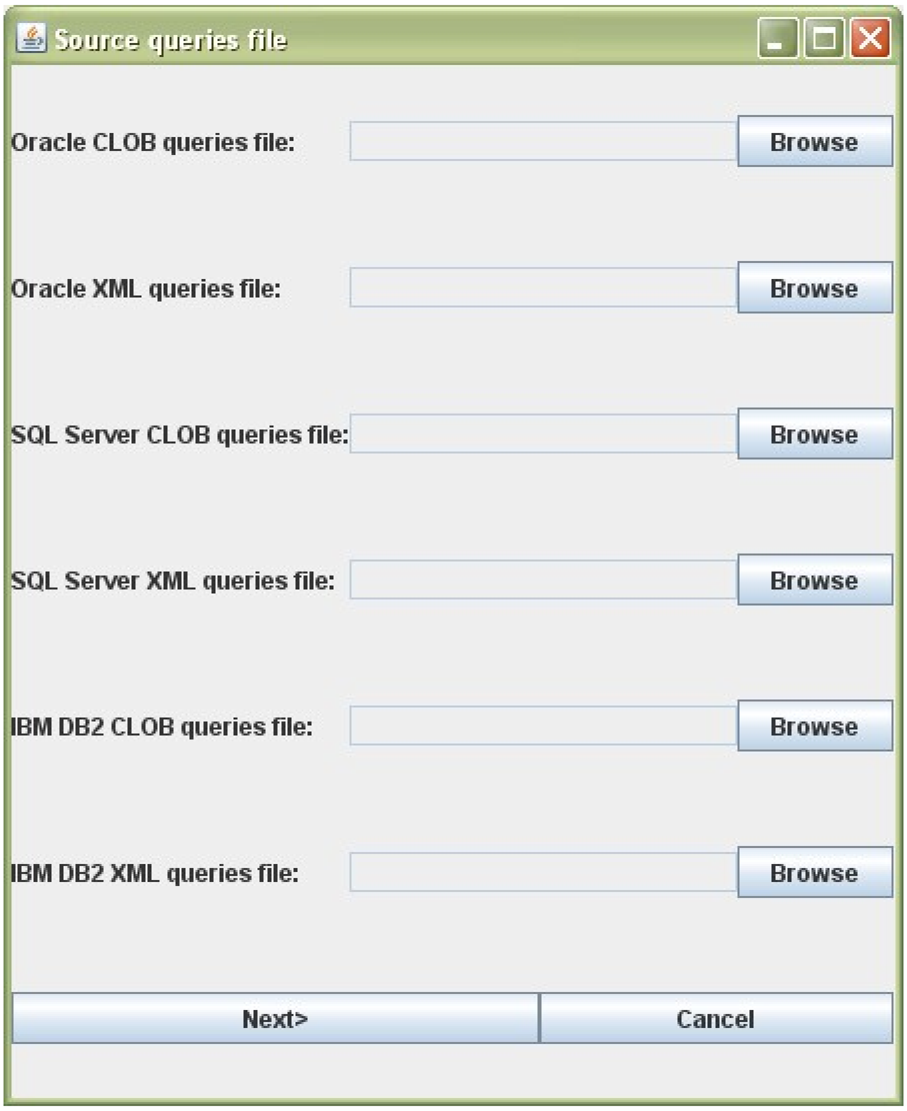


Figure 4.9: Import queries when choosing XQuery

When pure XQuery language is chosen only one document with queries is needed. It then will be automatically converted into SQL/XML statements with further correction options. It can be loaded by choosing a file from computer file system, or by copying queries directly into a text area (see Figure 4.9).



Oracle CLOB queries file:

Oracle XML queries file:

SQL Server CLOB queries file:

SQL Server XML queries file:

IBM DB2 CLOB queries file:

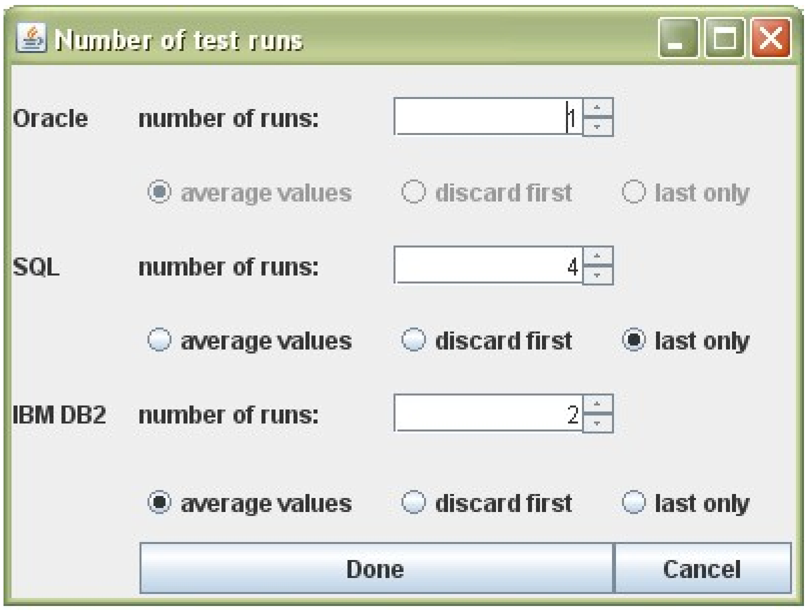
IBM DB2 XML queries file:

Figure 4.10: Import queries when choosing SQL/XML

When SQL/XML language is chosen to process XML data the querying document will not be further converted but executed directly when running the experiment. For that reason an individual querying document is needed for each database system and storing method in particular (see Figure 4.10). The querying document in that case must include valid SQL/XML statements referring to specific column ('xml_document') in a corresponding relational tables ('test_clob' for CLOB storing method, 'test_xml' for XML storing method) with an empty line after each statement, and two empty lines after the last one in the document. The example of valid SQL/XML statement for Oracle CLOB:

"SELECT XMLQuery ('xquery statement embedded' PASSING XMLType (xml_document) RETURNING CONTENT) FROM test_clob".

After all documents necessary for the experiment are selected XMLTest opens "Number of runs" windows (see Figure 4.11)


Number of test runs

Oracle number of runs: 1

[✓ average values] discard first last only

SQL number of runs: 4

average values discard first [✓ last only]

IBM DB2 number of runs: 2

[✓ average values] discard first last only

Done Cancel

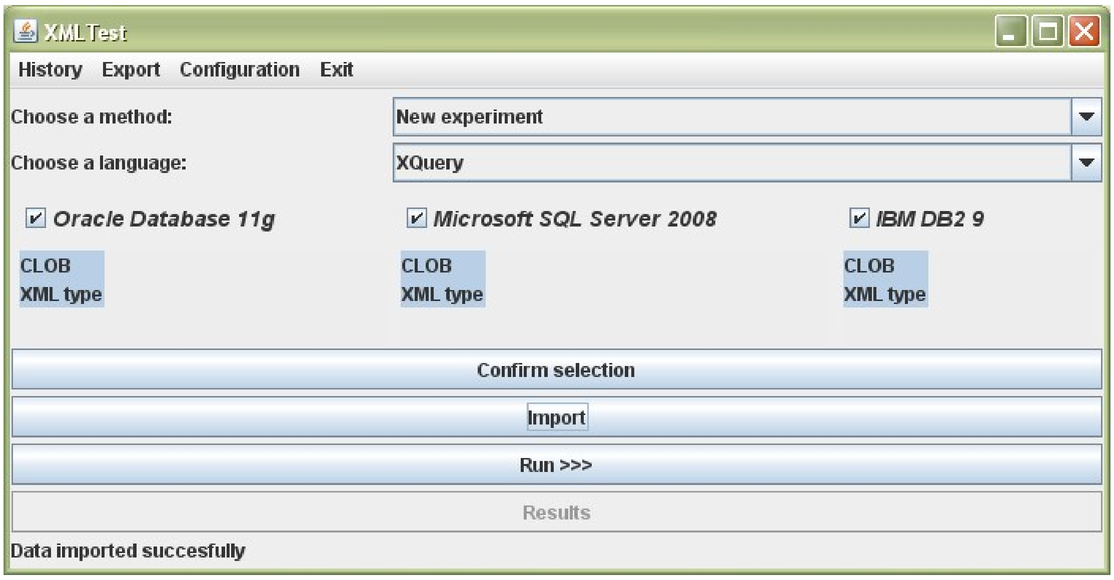
Figure 4.11: XMLTest selecting the number of queries runs

"Number of runs" window allows us to specify further parameters of queries processing and results calculation. For each previously chosen database we can choose how many times each query in querying document will be executed.

In case each query is executed more than once, XMLTest receives a set of time measurements for each execution. Because all these time measurements are based on the execution of the same query XMLTest should produce one result value to represent the speed result. The calculation of this value can be performed in three following ways:

1) Average values: the result time of one query execution is the average time all executions performed on this query.
2) Discard first: the time of the first execution (can be very slow) is ignored while calculating the average time of all executions.
3) Last only: the result time of one query execution is the time elapsed on the last execution.

When the data are imported into chosen databases, XMLTest is ready to run the experiment which is indicated by enabling the "Run" button (see Figure 4.12).



Choose a method: New experiment

Choose a language: XQuery

✓ Oracle Database 11g — CLOB, XML type

✓ Microsoft SQL Server 2008 — CLOB, XML type

✓ IBM DB2 9 — CLOB, XML type

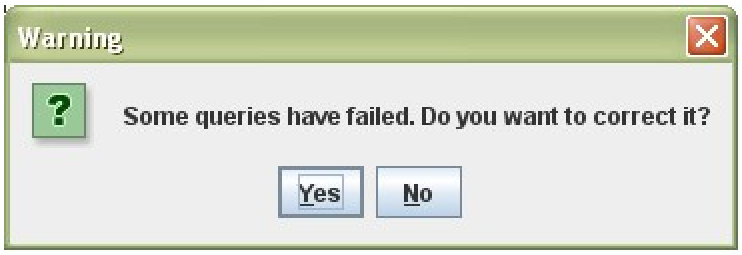
Data imported succesfully

Figure 4.12: XMLTest program is ready to run the experiment

"Run" button starts executing loaded querying document upon XML source document in each chosen database system. After computing the results, "Result" window is opened.

4.3. Results Representation

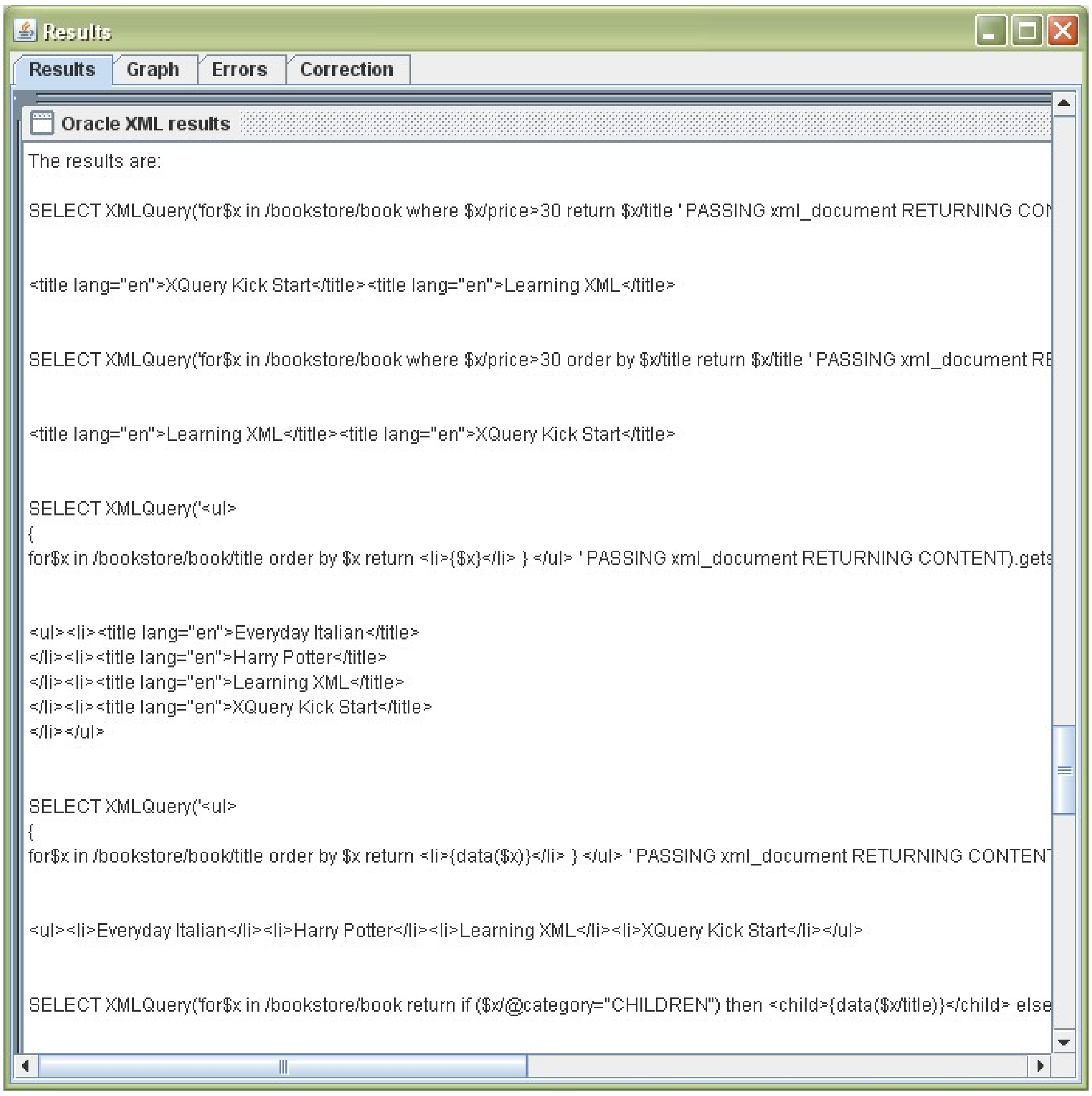
XMLTest results window includes four tabs: queries and their execution results, graphs to illustrate system performance while processing, errors if any, and the correction tab.



Some queries have failed. Do you want to correct it?

Figure 4.13: XMLTest correction dialog

If any of the queries execution failed when opening results window a dialog for immediate correction appears (see Figure 4.13). The "yes" answer leads to a correction tab, whereas "no" answer leads to a results tab.



```
The results are:

SELECT XMLQuery('for$x in /bookstore/book where $x/price>30 return $x/title ' PASSING xml_document RETURNING CON


<title lang="en">XQuery Kick Start</title><title lang="en">Learning XML</title>


SELECT XMLQuery('for$x in /bookstore/book where $x/price>30 order by $x/title return $x/title ' PASSING xml_document RE


<title lang="en">Learning XML</title><title lang="en">XQuery Kick Start</title>


SELECT XMLQuery('<ul>
{
for$x in /bookstore/book/title order by $x return <li>{$x}</li> } </ul> ' PASSING xml_document RETURNING CONTENT).gets


<ul><li><title lang="en">Everyday Italian</title>
</li><li><title lang="en">Harry Potter</title>
</li><li><title lang="en">Learning XML</title>
</li><li><title lang="en">XQuery Kick Start</title>
</li></ul>


SELECT XMLQuery('<ul>
{
for$x in /bookstore/book/title order by $x return <li>{data($x)}</li> } </ul> ' PASSING xml_document RETURNING CONTEN


<ul><li>Everyday Italian</li><li>Harry Potter</li><li>Learning XML</li><li>XQuery Kick Start</li></ul>


SELECT XMLQuery('for$x in /bookstore/book return if ($x/@category="CHILDREN") then <child>{data($x/title)}</child> else
```

Figure 4.14: XMLTest results tab

The results tab represents single queries that have been executed and corresponding results (see Figure 4.14). For each query if executed successfully the result of database system execution is represented, otherwise a string "Not succeeded" is shown.

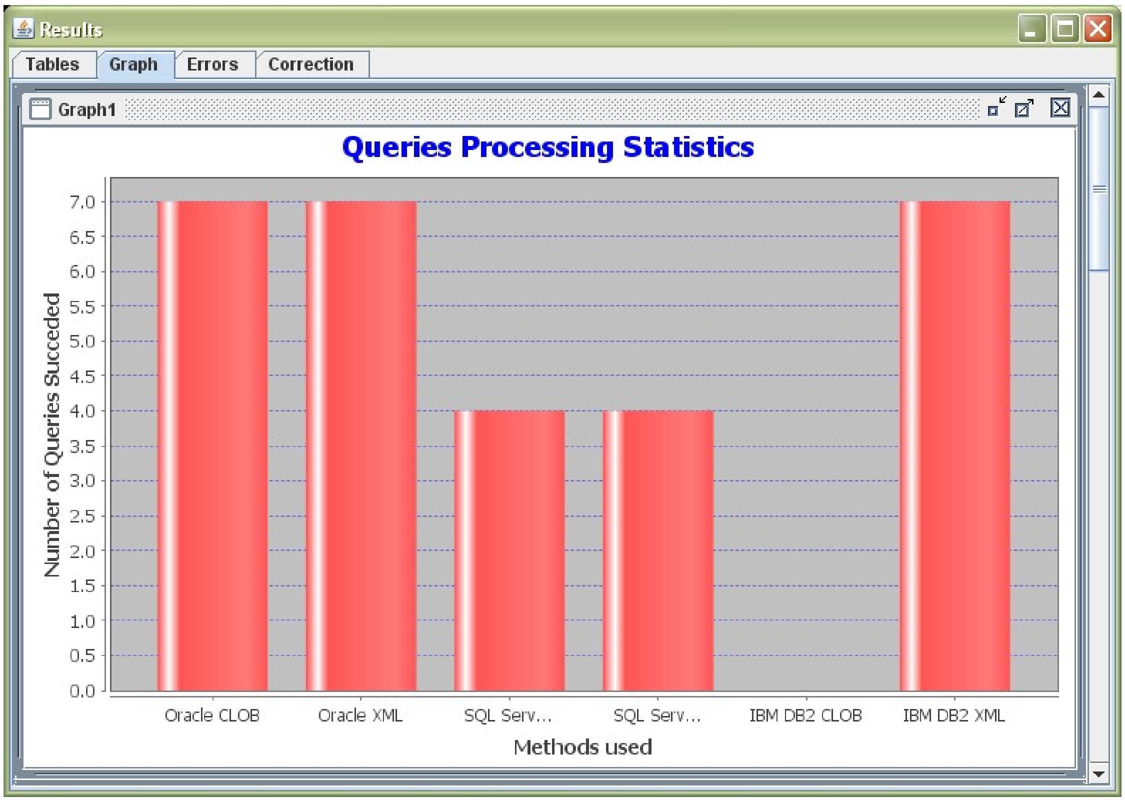
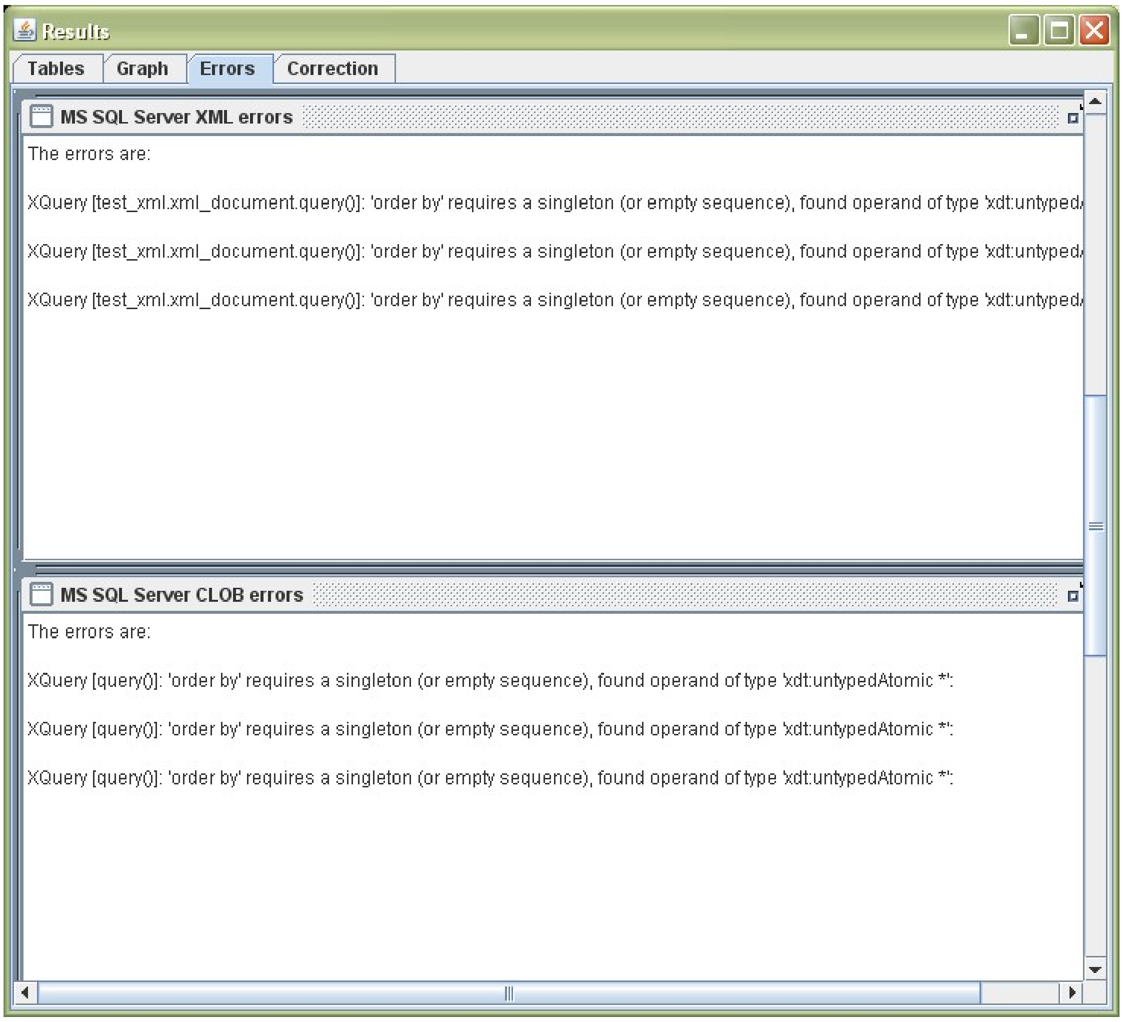


Figure 4.15: XMLTest graph tab

Graph tab represents four graphs. Each graph includes the information about systems chosen and storing methods for each database in particular (see Figure 4.15). The first graph represents the numbers of succeeded queries while processing. The second graph represents total time elapsed on the whole querying document processing. The time is computed by the composition of time elapsed while executing every single query. The third graph represents total performance of the systems which is calculated based on the number of succeeded queries execution and average time on executing a single query. The fourth graph represents the average time elapsed while processing one query.



```
The errors are:

XQuery [test_xml.xml_document.query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untyped

XQuery [test_xml.xml_document.query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untyped

XQuery [test_xml.xml_document.query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untyped
```

```
The errors are:

XQuery [query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untypedAtomic *':

XQuery [query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untypedAtomic *':

XQuery [query()]: 'order by' requires a singleton (or empty sequence), found operand of type 'xdt:untypedAtomic *':
```

Figure 4.16: XMLTest error tab

Error tab represents all the errors while processing queries for each database and storing method (see Figure 4.16). These may include syntax errors in query composition, failures in supporting of specific XQuery sections by each database system, etc.

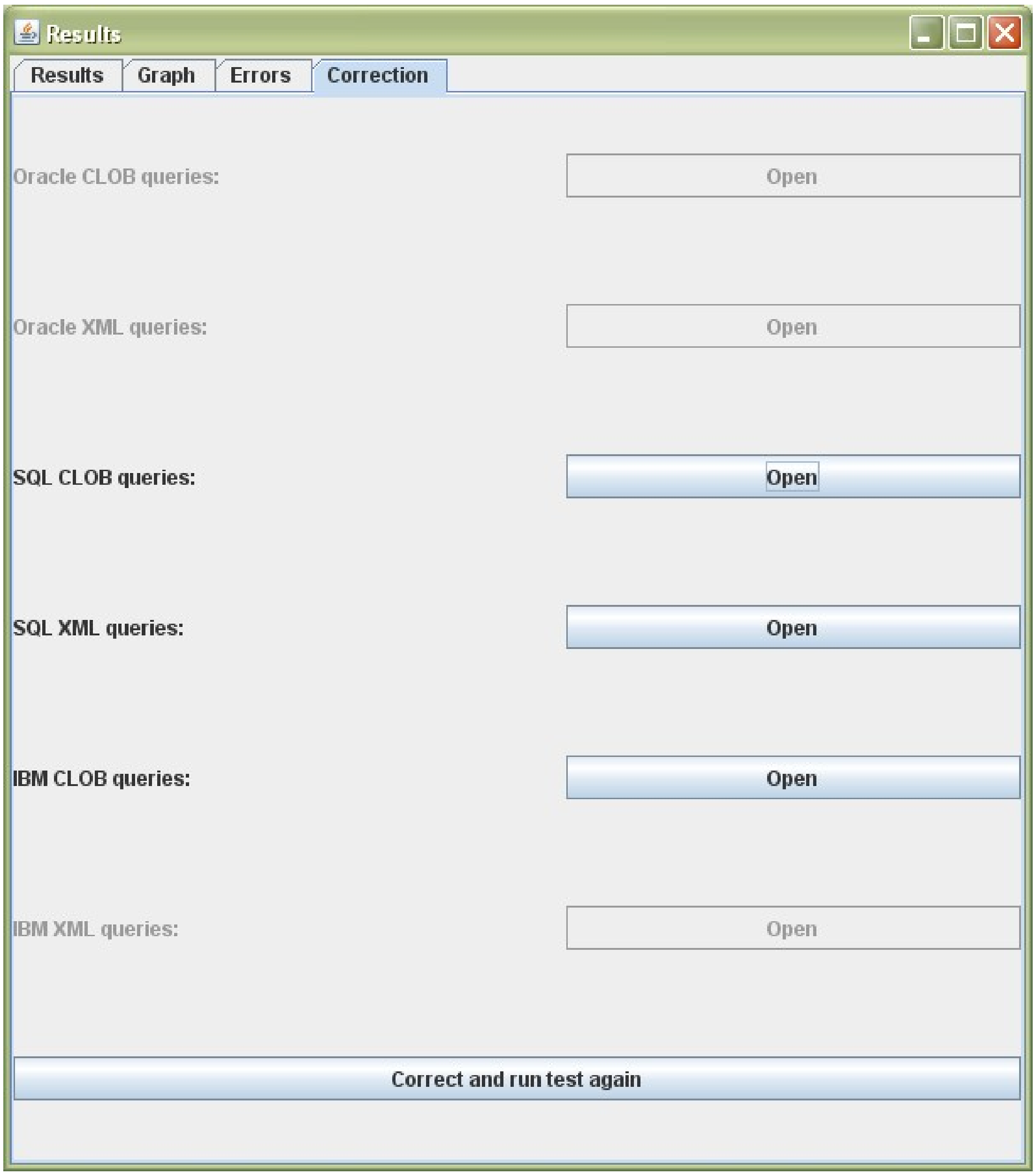
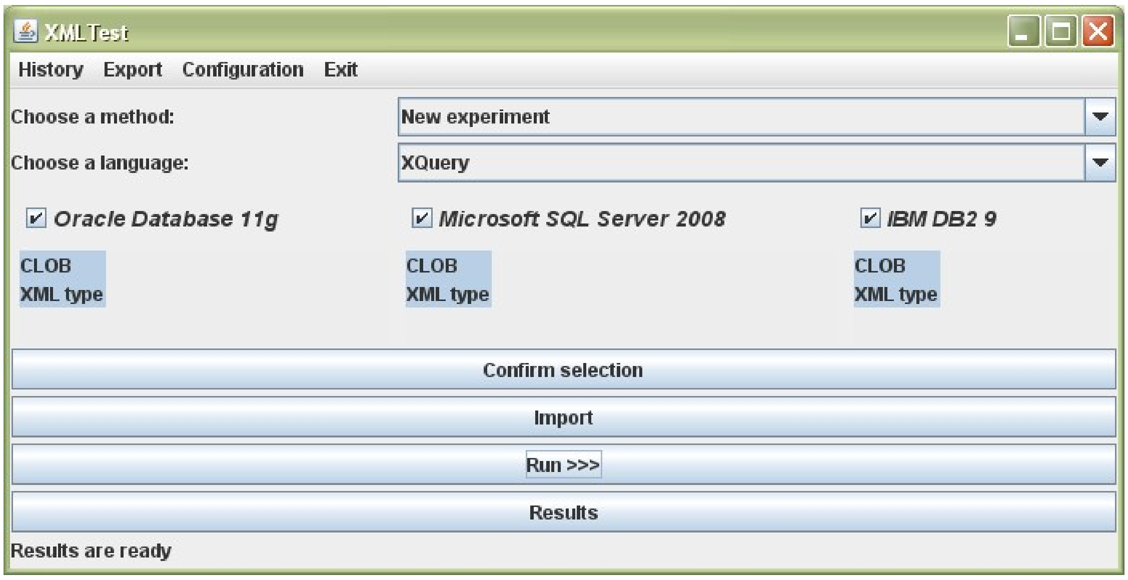


Figure 4.17: XMLTest correction tab

Correction tab represents querying documents for all database systems and storing methods the way it has been executed (after conversion if any). If one or more query has failed during execution, the "open" button for a corresponding querying document is enabled so that it is possible to open the document and correct the mistakes (see Figure 4.17). After correction by pressing "Correct and run test again" button the experiment is run with saved querying documents and a new "result" window is opened.



Choose a method: New experiment

Choose a language: XQuery

☑ Oracle Database 11g ☑ Microsoft SQL Server 2008 ☑ IBM DB2 9

CLOB
XML type

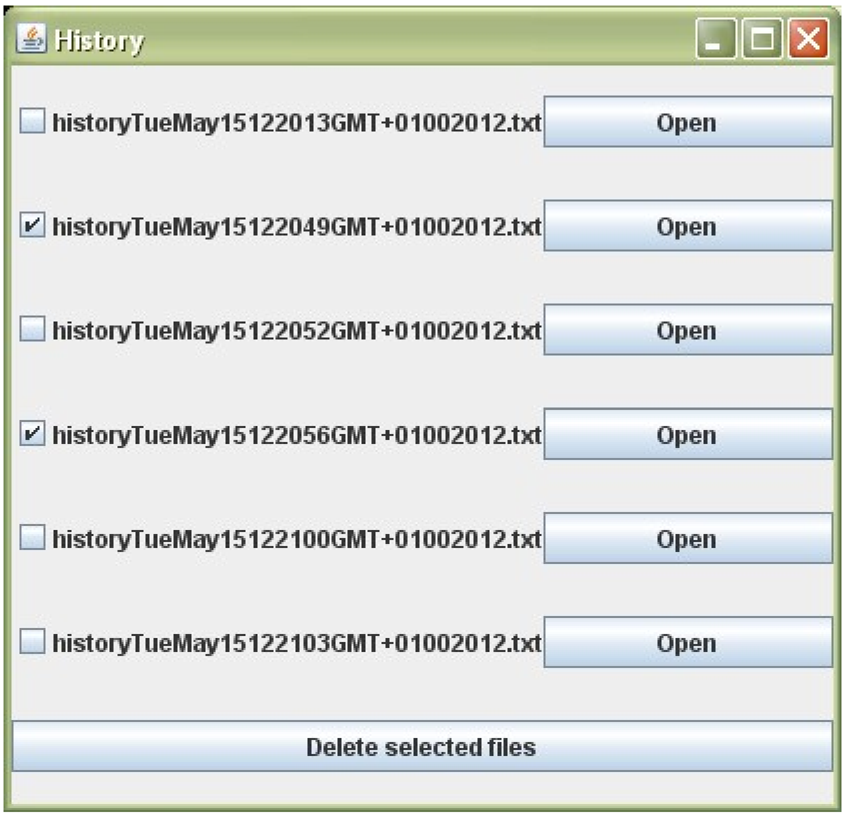
Results are ready

Figure 4.18: XMLTest main window with prepared results

After closing the current "result" window, the "results" button in XMLTest main window is enabled (see Figure 4.18). The button opens the window with current results of the previously run experiment.

4.4. History

XMLTest allows us to save the history of previous experiments (see Figure 4.19).



☐ historyTueMay15122013GMT+01002012.txt

☑ historyTueMay15122049GMT+01002012.txt

☐ historyTueMay15122052GMT+01002012.txt

☑ historyTueMay15122056GMT+01002012.txt

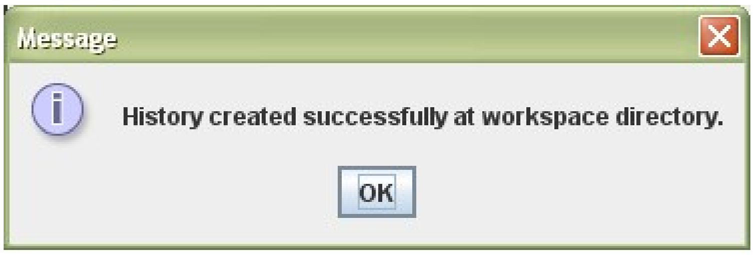
☐ historyTueMay15122100GMT+01002012.txt

☐ historyTueMay15122103GMT+01002012.txt

Figure 4.19: XMLTest history window

All existing history files are stored in /history directory of the project directory. By pressing "show history" menu button the window with all history files is opened. History file includes the failed queries and corresponding errors thrown for each database system and storing method. It also includes the information about times elapsed from the graph tab of experiment results window.

XMLTest allows us to delete chosen history files by pressing "Delete selected files" button (see Figure 4.19).


Message

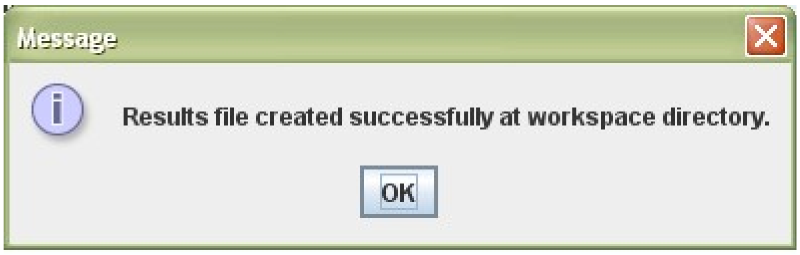
History created successfully at workspace directory.

Figure 4.20: XMLTest history created dialog

When a history file is created and stored successfully, the corresponding message appears (see Figure 4.20).

4.5. Export

XMLTest allows us to export current results into text files with specified path and name.


Message

Results file created successfully at workspace directory.

Figure 4.21: XMLTest results creation dialog

Results file is saved with a .txt extension, it includes single queries and a result of its processing for each database system and storing method from the results tab of experiment results window. If results file is created and saved successfully the corresponding message box appears (see Figure 4.21).

5. Implementation Description

The XMLTest application is written in Java language using Eclipse SDK version 3.7.1 as a development platform. Graphical User Interface (GUI) is written using Swing – the primary Java GUI widget toolkit [25][26].

The XMLTest application architecture is represented with four basic modules, which are responsible for particular sections of the program. In other words, the application is divided into separate parts; each of them is given a certain type of tasks based on application goals. Modules include a set of functions that solves these tasks. Such structure of the source code provides the possibility of extending it to include further database systems or storing methods. The chart in Figure 5.1 shows the relations between different modules of the application.

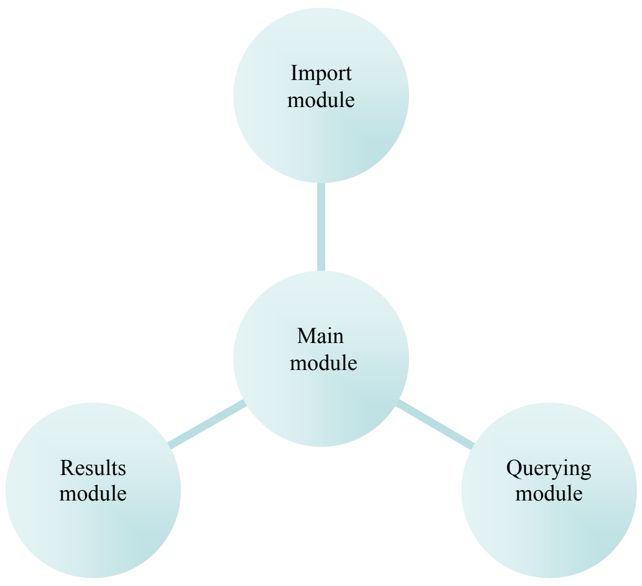


Figure 5.1: The structure of XMLTest application

As can be seen in Figure 5.1, the main module handles the import, querying and results modules operation by performing both side interactions with each of them. The three modules do not communicate with each other, which helps to maintain the consistency of the application and prevents it from internal collisions.

In order to extend the application for further database systems or XML technologies support new functions and procedures should be added into on or more corresponding modules. For example, to implement XSL transformations handling the querying module should be extended of a new function in Test class and the corresponding section should be added into the main module to support XSLT language selection.

5.1. Project Directories

Project directory includes four created directories which are intended to store additional documents that are necessary for the application to run properly.

The /lib directory contains all external libraries that are used in XMLTest application. The meaning and usage of the libraries will be described further in this chapter.

The /data directory contains all the documents needed to run XMark benchmark experiment. It includes XML database documents of sizes 100KB, 1MB, 10MB, and 100MB which were generated using XMark generator called Xmlgen with corresponding parameters. Along with tested documents, the directory contains querying document with twenty XQuery statements to execute.

All history files are stored in /history directory. Querying documents that are converted within the application for further execution are stored in /Queries directory.

5.2. Database Connectivity

In order to perform any tasks on database system the application should be able to establish the connection with the specified databases. To achieve this XMLTest uses JDBC which is a Java-based data access technology that defines how a client may access a database and provides methods for querying and updating data in it [27]. Since XMLTest is designed to work with Oracle, SQL Server and IBM DB2, JDBC is the most applicable technology as it is oriented towards relational databases.

The connection to a database system through a Java program can be established by performing three following steps: registering the corresponding JDBC driver, specifying a database URL, and creating a connection object using DriverManager class which will attempt to load the JDBC driver.

5.2.1. Oracle Database Connection

Oracle JDBC driver is defined with two libraries: ojdbc6.jar and orai18n.jar. Listing 27 illustrates a code that connects the application to Oracle Database 11g Release 2.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
String url1 =  "jdbc:oracle:thin:hr/hr@localhost:1521/orcldb";
conn = DriverManager.getConnection(url1, "SYSTEM", "Pa$$word1");
```

Listing 27: Connection to Oracle Database code

First, the class for JDBC Oracle driver is registered, then the "url1" variable defines URL for created database called "orcldb" on localhost (in some cases it is necessary to specify IP address 127.0.0.1 instead of localhost, to determine this see system "hosts" file), port 1521 (on which the default listener is installed for Oracle database). Last, DriverManager class creates the connection object with declared URL based on specified username and password for the corresponding database.

5.2.2. SQL Server Database Connection

SQL Server JDBC driver is defined with two libraries: jdb2jcc4.jar and sqlj.zip. Listing 28 illustrates a code that connects the application to Microsoft SQL Server 2008 R2.

```
Class.forName("net.sourceforge.jtds.jdbc.Driver");
String url2 = "jdbc:jtds:sqlserver://localhost:1433/tempdb";
conn = DriverManager.getConnection(url2, "sa", "Pa$$word1");
```

Listing 28: Connection to SQL Server database code

First, the class for JDBC SQL Server driver is registered, and then the "url2" variable defines URL for created database. XMLTest uses system database called "tempdb" which is created automatically when installing the database software. SQL TCP/IP listener is disabled by default, so in order to connect to the database from external application TCP/IP protocol should be enabled with specified port 1433 (by first configuring an automatic running of SQL Server Browser service). All above described requirements can be configured using SQL Server Configuration Manager. Last, DriverManager class creates the connection object with declared URL based on specified username and password for the corresponding database (using SQL authorization mode).

5.2.3. IBM DB2 Database Connection

IBM DB2 JDBC driver is defined with one library: jtds-1.2.5.jar. Listing 29 illustrates a code that connects the application to IBM DB2 9 Express C.

```
Class.forName("com.ibm.db2.jcc.DB2Driver");
String url3 = "jdbc:db2:testdb";
conn = DriverManager.getConnection(url3, "", "");
```

Listing 29: Connection to IBM DB2 database code

First, the class for JDBC DB2 driver is registered, and then the "url3" variable defines URL for the created database. With DB2 the database is created on localhost by default with a name specified by user. XMLTest application uses the database called "testdb". Last, DriverManager class created the connection object with declared URL based on specified username and password. In case of connecting to DB2 no username or password is used.

5.3. Main Module

The most significant goal of the main module is the coordination of three other modules operation. The module procedures include tools to call the functions, refer to the objects and receive the information from other modules.

When writing a multi-threaded application using Swing, there are two constraints to keep in mind:

- Time-consuming tasks should not be run on the Event Dispatch Thread. Otherwise the application becomes unresponsive
- Swing components should be accessed on the Event Dispatch Thread only [28]

To maintain the above described constraints all high-performance operations of the application (e.g. importing, querying, results representing) are conducted in separate threads to avoid the collisions between the operations and graphical interface. To handle an inter-thread communication XMLTest main module implements Java utility class SwingWorker which is "an abstract class to perform lengthy GUI-interacting tasks in a dedicated thread" [28].

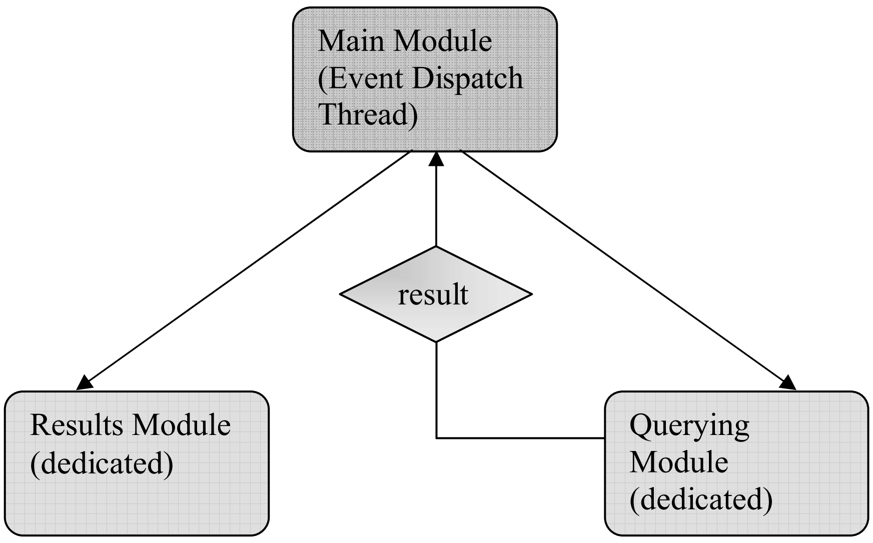

Main Module (Event Dispatch Thread)
result
Results Module (dedicated)
Querying Module (dedicated)

Figure 5.2: Threads interaction scheme

Figure 5.2 illustrates the process of querying XML data and further results representation in XMLTest. The main module runs on the Event Dispatch Thread, to start the experiment on previously imported data a new separate dedicated thread is created. In XMLTest source code this thread is defined by Experiment class (which extends SwingWorker), which starts querying module function to execute a querying script. The results of the execution are forwarded to the main module. It is important to understand that the results of querying module are never stored in the memory as a whole (storing the results in some cases can significantly slow down the application), but each part received by the main module is simultaneously passed to results module in order to be represented. For that reason to avoid the conflict between graphical results representation and querying script execution, the results module is started by the main module in a separate thread which is defined by Draw class (which extends SwingWorker), which in its turn starts results module function to draw results, graph, error, and correction tabs. After the Experiment thread has completed queries execution and all results are represented by results module, the "results" window is opened in XMLTest.

All time-consuming tasks in XMLTest are handled with a similar structure as shown in Figure 5.2.

Along with coordinating an inter-thread communication, the main module solves a set of further tasks such as: conversion of querying documents, creating history files, exporting results files.

5.3.1. XQuery Conversion

As it was described in the previous chapters all pure XQuery querying documents are attempted to be converted by XMLTest automatically into SQL/XML queries to be applicable upon relational tables.

To construct a valid SQL/XML statement, first, the input XQuery statement is modified and prepared to be executed. Then it is wrapped into corresponding SQL clauses to produce correct SQL/XML statement. The algorithm for XQuery modifying is described in Section 5.3.1.1.

The following are corresponding SQL "wraps" for XQuery statements:

1. Oracle Database/CLOB storage type:

   SELECT **XMLQuery**('*embedded statement*' PASSING **XMLType**(xml_document) RETURNING CONTENT) FROM test_clob;

   XMLQuery() function informs a database that the embedded statement is XQuery. PASSING clause refers the execution of the statement to an xml_document column where XML document is stored. The column is of CLOB type. For that reason it is converted into XML using XMLType() function.

2. Oracle Database/XML storage type:

   SELECT **XMLQuery**('*embedded statement*' PASSING xml_document RETURNING CONTENT) FROM test_xml;

3. SQL Server/CLOB storage type:

   SELECT (**CONVERT**(XML, [xml_document])).**query**('*embedded statement*') FROM test_clob;

As in the Oracle Database xml_document CLOB column has to be first converted into XML, using function CONVERT(). Then, function query() executes the embedded statement upon the xml_document column.

4. SQL Server/XML storage type:

   SELECT [xml_document].**query**(*'embedded statement'*) FROM test_xml;

5. IBM DB2/CLOB column:

   IBM DB2 does not support the dynamic conversion of CLOB column to an XML type. For that reason, XQuery statement can not be applied upon CLOB column directly.

6. IBM DB2/XML storage type:

   SELECT **XMLQuery**(*'embedded statement'* PASSING test_xml.xml_document AS "d") FROM test_xml;

   The IBM DB2 "wrap" is similar to the one in the Oracle Database. The main difference is contained in the PASSING clause where we specify the name of xml_document that will be used in XQuery statement's paths as a variable referring to a path "root" (e.g. $d/bookstore/book).

5.3.1.1. XQuery Conversion Algorithm

Since all testing XML documents are stored in corresponding relational tables (which are specified by SQL "wrap", see Section 5.3.1) an XQuery statement can include no references to an external XML document. The references to an XML document stored in file system can be implemented by function doc().

There are two ways of using doc() function in an XQuery statement:

- Direct using. Doc() reference is used directly as a "root" element of every XQuery path. The path is preceded by a keyword 'in' in a 'for' clause of the XQuery statement (see Listing 30).

```
for $x in doc("books.xml")/bookstore/book return $x/title
for $x in /bookstore/book return $x/title
```

Listing 30: Direct doc() conversion

- Variable declaration. The value of doc() function can be assigned to a variable in 'let' clause, which then will be used as a "root" element of every XQuery path. In this case it is preceded with ':=', and the whole declaration clause should be eliminated (see Listing 31).

```
let $auction := doc("auction.xml") return for $b in $auction/site/people rturn $b/name
for $b in /site/people return $b/name
```

Listing 31: Variable doc() conversion

XMLTest handles both above described cases, using the algorithm illustrated in Figure 5.3.

```
clauses := document.split("for");
foreach (clause: clauses) do
  if (clause.contains("doc"))
        set index := clause.index("doc")
        if(clause[index-1] == ":=") begin
             set variable_name to clause[index-2]
             delete clause[index-3] from clause
             delete clause[index-2] from clause
             delete clause[index-1] from clause
             delete clause[index+1] from clause
             end
  add clause to result
  done

  delete variable_name from result
  delete "doc(*.xml)" from result

  return result
```

Figure 5.3: XMLTest conversion algorithm in pseudocode

The querying document is first parsed into 'for' clauses: if doc() function occurs in the document it will be contained fully in one of the clauses. Based on the preceding character it is decided whether doc() is used directly (in) or by assigning to the variable (:=). In the first case the doc() function is deleted from the statement and then the conversion is completed (see Listing 30). In the second case the name of the variable is saved for further usage, then let clause is deleted from the statement and all the variable name occurrences are eliminated (see Listing 31).

5.4. Import Module

When querying XML data stored in a database to compare the systems performance it is first needed to be imported into relational tables of particular databases. XMLTest import module includes a set of functions which perform XML data inserting into specified databases using the chosen storing methods.

In case the XML testing document is small (<32KB) all of the three database systems (Oracle, SQL Server, IBM DB2) follow a similar pattern of importing it into tables by accepting the data from a plain text. Contrariwise, when dealing with large

XML documents each database system provides us with its own set of tools to perform the import process.

5.4.1. Oracle Database Import

Before an XML document can be inserted into Oracle Database the corresponding relational tables should be created for each storing method chosen. XMLTest performs the tables' creation by executing the following SQL statements:

```
sqlText = "CREATE TABLE test_clob(id NUMBER(15), xml_document CLOB)";
stmt.executeUpdate(sqlText);
```

Listing 32: XMLTest create table statement

```
sqlText = "CREATE TABLE test_xml(id NUMBER(15), xml_document XMLType)";
stmt.executeUpdate(sqlText);
```

Listing 33: XMLTest create table statement

In Listing 32 SQL statement create a table called "test_clob" with a column "xml_document" of Oracle CLOB data type.

In Listing 33 SQL statement creates a table called "test_xml" with a column "xml_document" of Oracle XMLType data type.

To insert a large XML document that does not fit to memory as a whole into Oracle database table XMLTest uses Oracle CLOB data type and a character stream. The following sample code demonstrates inserting of XML file into Oracle database:

```
tempClob.open(CLOB.MODE_READWRITE);
Writer tempClobWriter = tempClob.setCharacterStream(0L);
```

Listing 34: XMLTest set character stream sample code

```
pstmt.setClob(1, xmlDocument);

pstmt.executeUpdate();
```

Listing 35: XMLTest set clob sample code

Listing 34 shows how to assign a character stream to CLOB variable in order to then write XML data from a file directly into CLOB using that stream.

Listing 35 shows how to set CLOB object to previously defined prepared statement which insert the CLOB into an Oracle table. More information on prepared statements usage can be found in Oracle documentation [29].

5.4.2. SQL Server Import

Before an XML document can be loaded into SQL Server, the corresponding relational tables should be created for each storing method chosen. XMLTest performs the tables' creation by using the following SQL statements:

```
sqlText = "CREATE TABLE test_clob(xml_document VARCHAR(MAX))";
stmt.execute(sqlText);
```

Listing 36: XMLTest create table statement

```
sqlText = "CREATE TABLE test_xml (xml_document XML )";

stmt.execute(sqlText);
```

Listing 37: XMLTest create table statement

Listing 36 shows how to create a table with a CLOB column in SQL Server database. SQL Server has no native CLOB data type; the equivalent is a VARCHAR type with the size of MAXIMUM value.

Listing 37 shows how to create a table with an XML column in SQL Server database.

To insert a large XML file that does not fit into memory XMLTest application implements an SQL Server BULK load:

```
sqlText = "INSERT INTO test_clob(xml_document) SELECT * FROM " +
        "OPENROWSET(BULK N'" + f.getCanonicalPath() +"', SINGLE_BLOB) rs";

stmt.execute(sqlText);
```

Listing 38: XMLTest bulk load

Listing 38 shows how to load a large XML file directly into "xml_document" column of "test_clob" table. The OPENROWSET function accepts the path to XML document stored in a local file system; the N parameter ensures encoding matching. More information on using bulk loading can be found in SQL Server documentation [30].

5.4.3. IBM DB2 Import

Before an XML document can be loaded into IBM DB2, the corresponding relational tables should be created for each storing method chosen. XMLTest performs the tables' creation by using the following SQL statements:

```
sqlText = "CREATE TABLE test_clob(xml_document CLOB(2g))";
stmt.executeUpdate(sqlText);
```

Listing 39: XMLTest create table statement

```
sqlText = "CREATE TABLE test_xml(xml_document XML)";
stmt.executeUpdate(sqlText);
```

Listing 40: XMLTest create table statement

Listing 39 shows how to create table with a CLOB column in IBM DB2. Note that for large documents to insert it is necessary to specify the maximum size of the CLOB column.

Listing 40 shows how to create a table with an XML column in IBM DB2.

To insert a large XML document that does not fit into memory to a column of IBM DB2 table XMLTest creates a binary stream to read the contents of XML file and then attaches a stream to previously created prepared statement:

```
InputStream inBin = new FileInputStream(f);
pstmt.setBinaryStream(1, inBin, f.length());
pstmt.executeUpdate();
```

Listing 41: XMLTest binary stream insert

In Listing 41 an input binary stream first created taking a parameter "f" which stands for the XML document stored in a local file system. Then, the binary stream is passed to a prepared INSERT statement, with an additional parameter of file length.

5.5. Querying Module

In application source code the XMLTest querying module is defined by "Test" class which accepts one connection object parameter that stands for the connection established to a particular database and implements a test function with a querying file parameter to execute the statements in it.

The correctly formatted querying file passed to a test function is first parsed into single statements based on an empty line after each of them. It then forwards a statement text to a previously created statement object according to the accepted connection. The statement is executed; the results are stored in the Java ResultSet object [31].

The results of statements execution are stored in a two-dimensional String array. For each statement, the result array includes the query text, results of a database query execution, error name in case the execution has failed, and time elapsed on the execution. The elapsed time is calculated in seconds by subtraction of the start time right before the execution started from the end time right after it is completed.

5.6. Results Module

XMLTest results module is designed to represent the results of the experiment in a separate application window. It includes a set of functions to solve the following tasks: represent statement and corresponding results, draw graphs, represent error thrown during the execution, and represent all querying documents used for further correction if needed.

5.6.1. Queries Representation

To represent the statements and corresponding results XMLTest results module defines "drawResults" function that accepts three parameters: a two-dimensional String array (returned result of querying module), the name of the database system, and the name of the storing method used.

Based on the parameters "drawResults" function creates Java JInternalFrame object [32] with a pane to write results into for each database and storing method used. Each statement text is added to a new line; the corresponding result is placed at the next line respectively.

5.6.2. Graphs Representation

To draw the charts XMLTest results module defines "drawGraphs" function which creates four statistics graphs based on the given values. All necessary information to draw graphs are collected and calculated in XMLTest main module and then passed to results module into four arrays. The arrays data types are int or double [33] depending on the graph type.

To draw the chart "drawGraph" function implements external JFreeChart library [34] which consists of four packages: jfreechart-1.0.14-demo.jar, iText-2.1.5.jar, jcommon-1.0.17.jar, and jfreechart-1.0.14.jar.

5.6.3. Errors Representation

The errors in XMLTest are represented with "drawErrors" function. The function is implemented similarly to "drawResults" function described in Section 5.6.1.

5.6.4. Corrections Representation

XMLTest defines "drawCorrections" function to represent all querying documents used during experiment run for each database system and storing method.

When one or more queries have failed during execution it is possible to correct the corresponding querying document immediately after the results are calculated and shown. Every querying document is opened in a simple view window which allows us to edit the document and save the changes.

6. Experiments Presentation

Each of the three databases analyzed previously in this thesis has its own advantages along with the areas that are not completely solved or supported. To illustrate and analyze the real aspects of Oracle, SQL Server and DB2 performance over XML data, this chapter describes the experiments performed on these database systems using XMLTest.

6.1. Experiments Description

The experiments test all three database systems included in XMLTest application: Oracle Database 11g Release 2, Microsoft SQL Server 2008 R2, IBM DB2 9 Express C.

All the experiments use valid testing data of the XMark benchmark experiment. The size of the XML database document is 100KB. The querying document includes twenty XQuery statements which will be converted to appropriate SQL/XML queries during the experiment run.

Experiment results include queries execution performances, total and average times, and error analyzing. These aspects will be closely described and examined in the following sections. From now on to make the experiment results description more comprehensible the names Oracle CLOB, Oracle XML, SQL CLOB, SQL XML, IBM CLOB, IBM XML will be used for Oracle Database CLOB and XML types of storage, SQL Server CLOB and XML types of storage, and IBM DB2 CLOB and XML types of storage respectively.

6.2. Experiment Results Analysis

There are two main requirements that are put on a database system while developing XML-based application: the range of XQuery syntax and function support in a database, and the speed of its execution. The estimation of the most preferable proportion for these parameters is based individually on particular application requirements. The following sections estimate whether the above described parameters are inversely related to each other. It also shows whether it is possible to affect (eventually change) this relation.

6.2.1. Queries Execution Performances

The first aspect of results analysis is the databases success while performing query execution. As all further described experiments will be run upon the same testing XML document and querying document, the changes in various further parameters will not influence these statistics.

The graph in Figure 6.1 represents the number of queries executed successfully in each database.

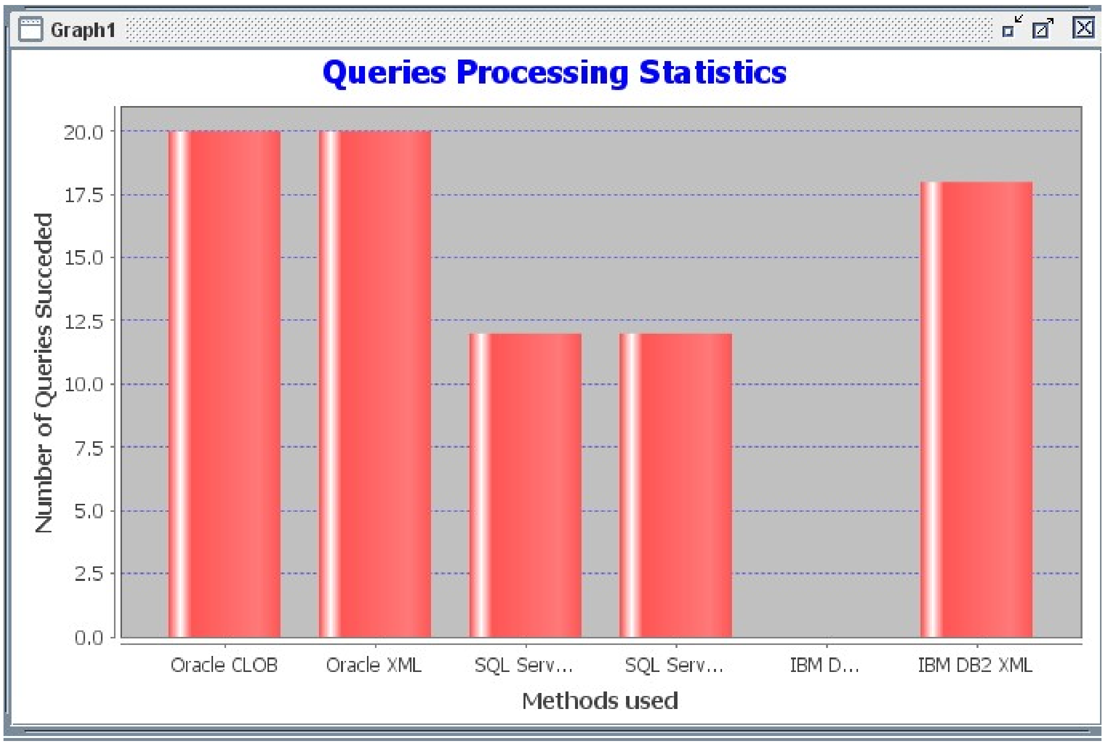


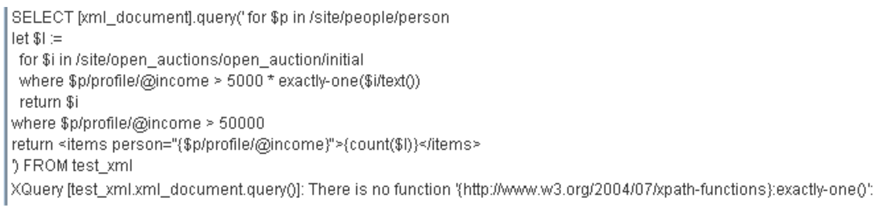
Figure 6.1: XMLTest Queries Processing Statistics graph

As we can see in Figure 6.1 the results are the following:

a) Oracle CLOB and XML: 20 (100%)
b) SQL CLOB and XML: 12 (60%)
c) IBM CLOB: 0, IBM XML: 18 (90%)

The first significant difference in query executions is that unlike Oracle and SQL Server, IBM database produced different results for CLOB and XML storage types. It is due to the fact that there is no dynamic conversion from CLOB to XML supported in the IBM database (see Section 5.3.1). For that reason IBM CLOB will be excluded from further experiments statistics.

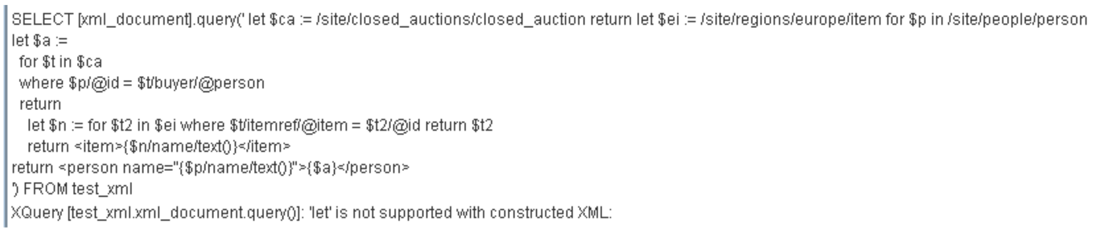
SQL results include eight failed queries, which mean that they contain some XQuery tools that are not supported by the SQL Server.



```
SELECT [xml_document].query(' for $p in /site/people/person
let $l :=
  for $i in /site/open_auctions/open_auction/initial
  where $p/profile/@income > 5000 * exactly-one($i/text())
  return $i
where $p/profile/@income > 50000
return <items person="{$p/profile/@income}">{count($l)}</items>
') FROM test_xml
XQuery [test_xml.xml_document.query()]: There is no function '{http://www.w3.org/2004/07/xpath-functions}:exactly-one()':
```

Figure 6.2: SQL failed function query and corresponding error

Figure 6.2 illustrates a failed query in SQL XML due to the non-support of function exactly-one(). Similarly SQL failed to execute function zero-or-one(). In this experiment these errors caused the failure of three further queries.

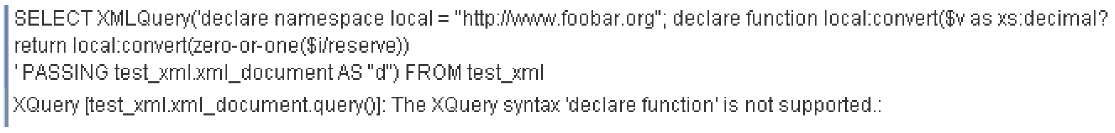


```
SELECT [xml_document].query(' let $ca := /site/closed_auctions/closed_auction return let $ei := /site/regions/europe/item for $p in /site/people/person
let $a :=
 for $t in $ca
 where $p/@id = $t/buyer/@person
 return
  let $n := for $t2 in $ei where $t/itemref/@item = $t2/@id return $t2
  return <item>{$n/name/text()}</item>
return <person name="{$p/name/text()}">{$a}</person>
') FROM test_xml
XQuery [test_xml.xml_document.query()]: 'let' is not supported with constructed XML:
```

Figure 6.3: SQL failed structure query and corresponding error

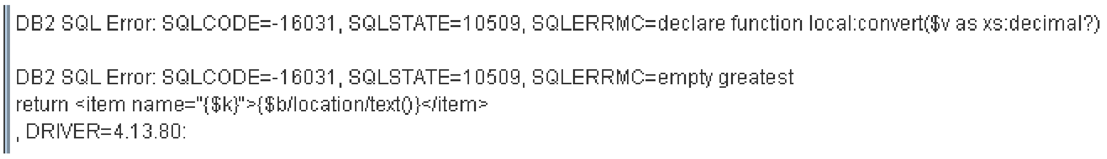
Another cause of queries execution failures in SQL is specific XQuery statement structure sections that are not supported by the database, e.g. assigning XQuery path to a variable (see Figure 6.3).

The most common error while processing a statement is some specific XQuery syntax which is not supported by a database. Both SQL and IBM do not support 'declare function' and 'empty greatest' syntax (see Figures 6.4, 6.5).



```
SELECT XMLQuery('declare namespace local = "http://www.foobar.org"; declare function local:convert($v as xs:decimal?
return local:convert(zero-or-one($i/reserve))
' PASSING test_xml.xml_document AS "d") FROM test_xml
XQuery [test_xml.xml_document.query()]: The XQuery syntax 'declare function' is not supported.:
```

Figure 6.4: SQL failed syntax query and corresponding error



```
DB2 SQL Error: SQLCODE=-16031, SQLSTATE=10509, SQLERRMC=declare function local:convert($v as xs:decimal?)

DB2 SQL Error: SQLCODE=-16031, SQLSTATE=10509, SQLERRMC=empty greatest
return <item name="{$k}">{$b/location/text()}</item>
, DRIVER=4.13.80:
```

Figure 6.5: IBM failed syntax errors

In conclusion, Oracle appeared to be the most successful in the aspect of query processing. Whereas IBM XML successfully executed most of the queries, IBM CLOB does not support XQuery processing at all. As for SQL, both CLOB and XML have managed to execute only 60% of the queries, due to the lack of XQuery function, syntax, and structure support.

6.2.2. Time Statistics

Another aspect that influences the total database performance is the time required for execution of the whole query over a specific document, and the average time spent on one query processing. This parameter usually depends on the way the experiment is run: how many times each query is executed, and how several time measurements for each query are handled.

First, to receive "rough" results (not optimized or influenced), the number of runs for each query and database is specified to one. Figures 6.6 and 6.7 illustrate the total times elapsed on executing the whole querying document, and the average times elapsed on executing one query respectively.

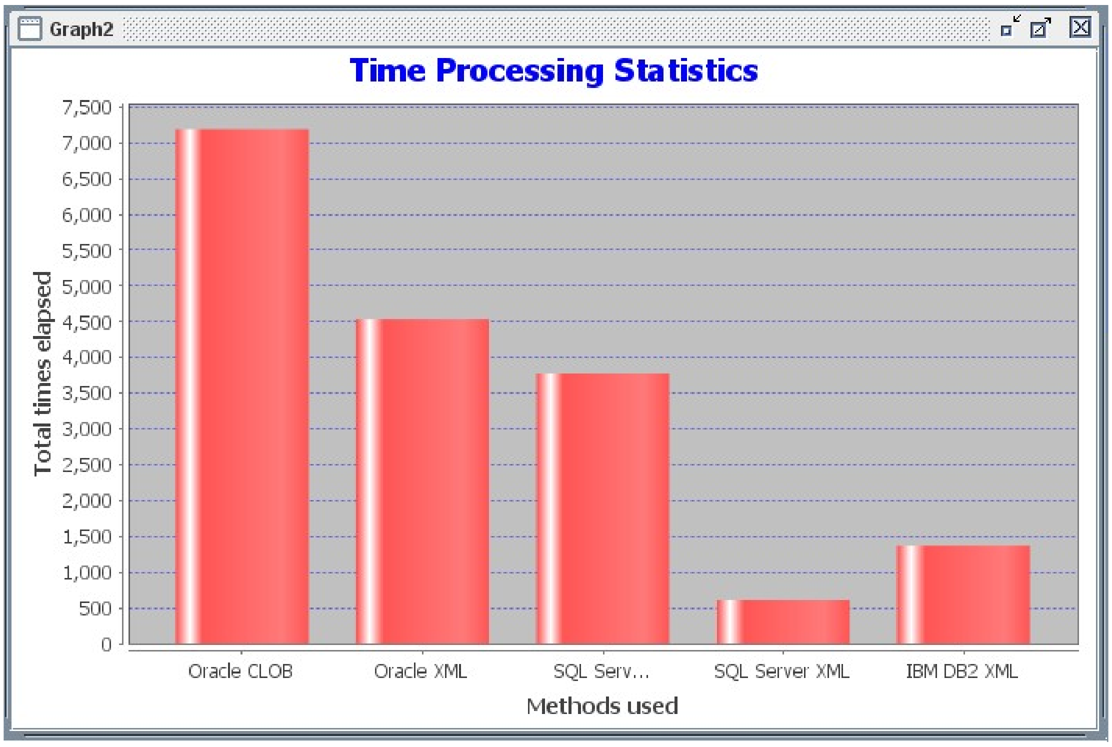


Figure 6.6: XMLTest total time statistics graph for one experiment run

Clearly, the total time elapsed on the processing of the whole querying document depends on the number of queries that have been executed successfully. The important information that is contained in Figure 6.6 is that regardless the database system, the times spent on execution with CLOB storage are always greater than the ones with XML storage. That leads us to a natural conclusion, that native XML data type is more preferable when working with XML data. For that reason CLOB storage will be excluded from further experiments.

To estimate the relation between the number of successfully executed queries and "real" speed of the database the graph in Figure 6.7 demonstrates the average times elapsed on processing one query.

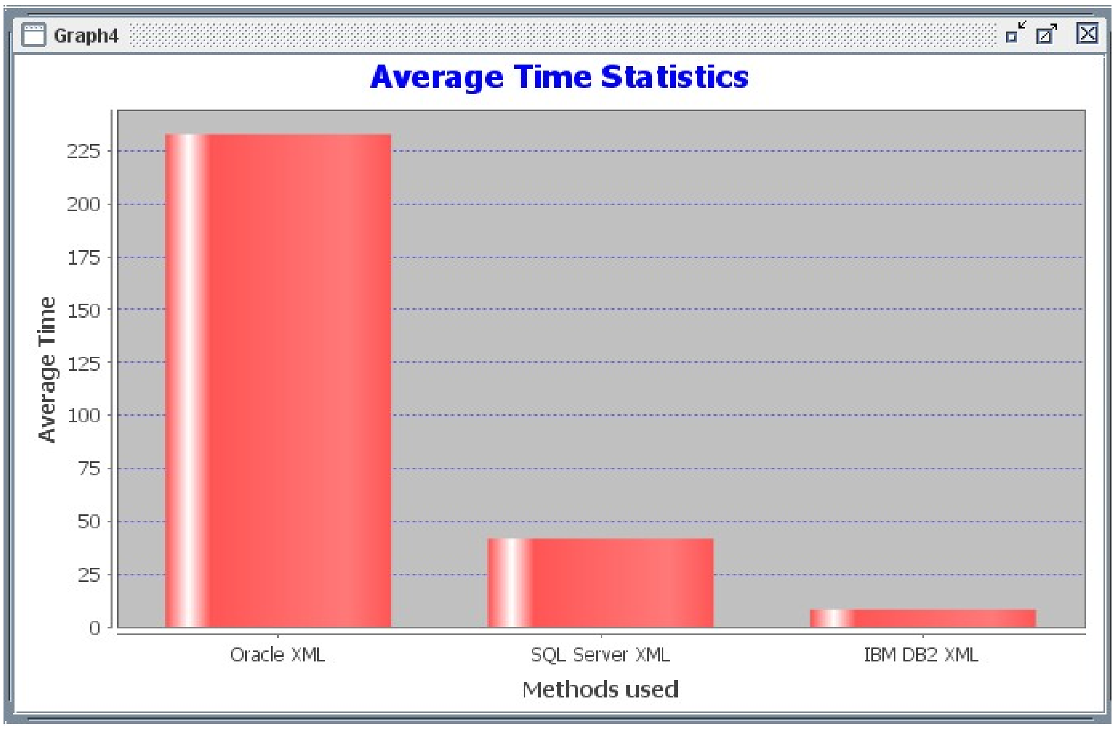


Figure 6.7: XMLTest average time statistics run for one experiment run

The information contained in Figure 6.7 is better to be represented with the results of query execution to demonstrate the relation between execution success and speed. Let each value be represented with a number from one to three, where one is the best success in query execution (the best average time), three is the worst success in query execution (the worst average time). Then the results are following:

a) Oracle XML: execution success – 1, speed – 3;
b) SQL XML: execution success – 3, speed – 2;
c) IBM XML: execution success – 2, speed – 1;

As can be seen from the results, IBM database has the most optimal ratio. These ratios also partly confirm the idea that the success in performing query execution is usually somehow inversely related to processing speed.

In order to estimate whether it is possible to improve the previous time performances results of the database the next three experiments are run with the different set of parameters: average values (see Figure 6.8), discard first (see Figure 6.9), and last only (see Figure 6.10). In all experiments the number of runs is equal to three.

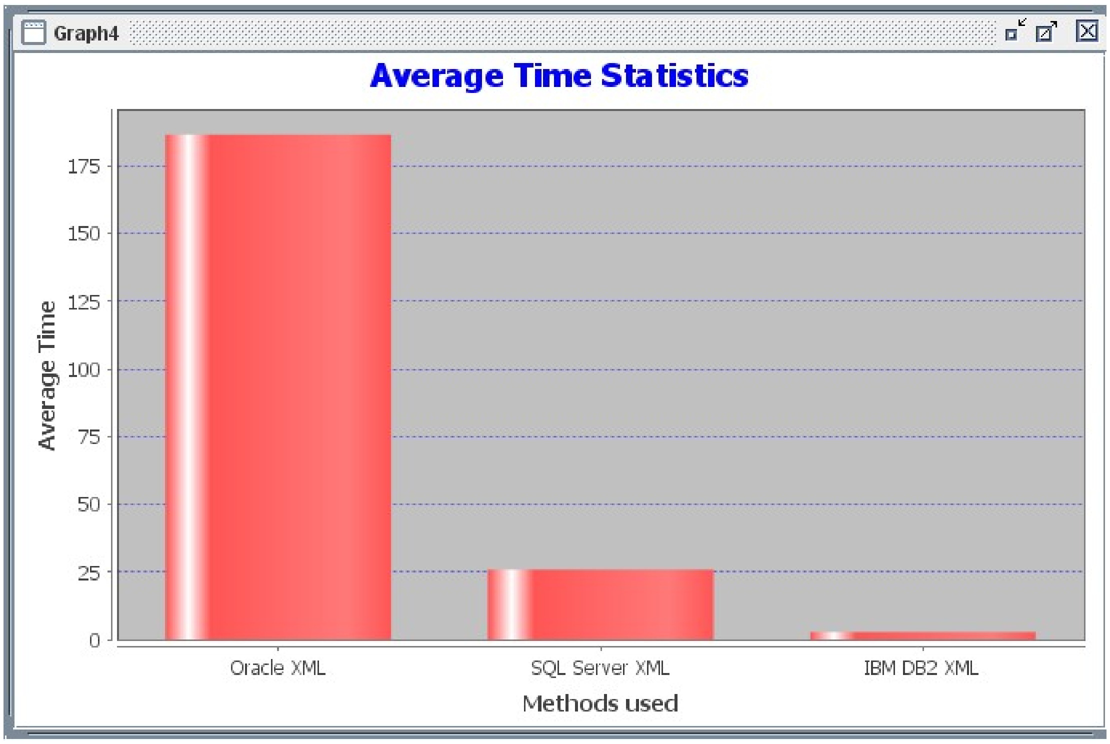


Figure 6.8: XMLTest average times graph for 3 experiment runs

As can be seen in Figure 6.8, the increasing of query execution number significantly reduces the average time elapsed on one query execution, though it takes more time until all the results are calculated and shown.

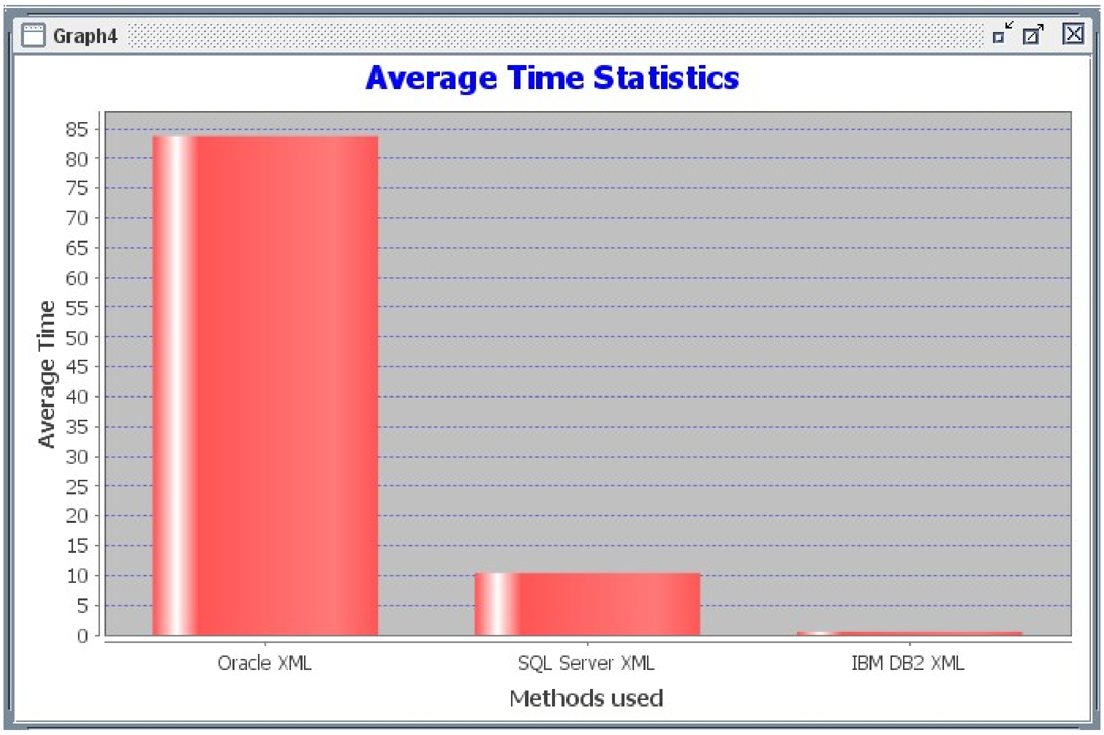


Figure 6.9: XMLTest average times excluding the first execution

The graph in Figure 6.9 shows that the first execution of the query usually takes longer than the rest of executions. For that reason, by discarding the time of the first execution the total average times on execution are decreased.

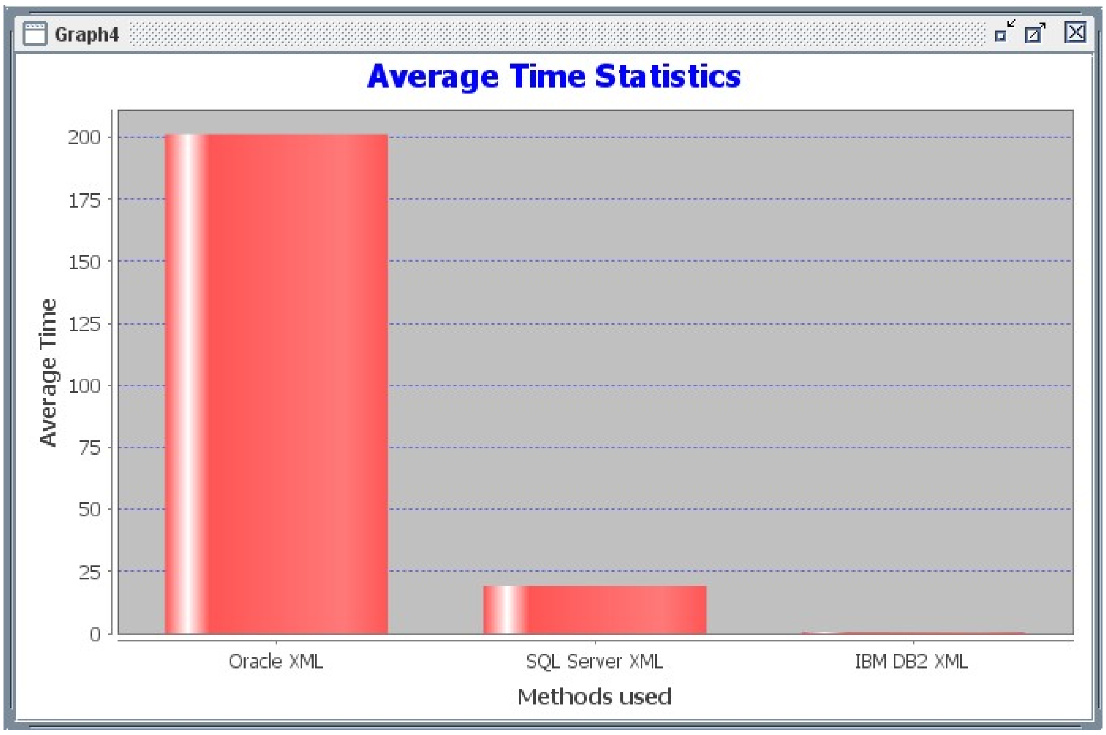


Figure 6.10: XMLTest average times last only execution

In comparison with the previous results (see Figure 6.9) the graph in Figure 6.10 shows that in Oracle XML and SQL XML the last execution of the query is not necessarily the fastest one, whereas in IBM XML it did not show any significant difference.

Based on all previous results it is possible to say that even though the choosing of the most suitable parameters can significantly improve the performance of each database in particular, it does not change the performances relation in general.

7. Summarizing Comparison Analysis

Table 7.1 illustrates summarizing comparison analysis of the three database systems. It includes all the issues that occurred during the work with the databases.

Aspects	Storage type	Oracle DB	SQL Server	IBM DB2
Documentation		Full, includes both theory and practical examples	Poor, fragmented, lack of theory	Focused on practical usage
Software		SQL Developer	SQL Management Studio	Control Center
Connectivity issues		Database created only manually	TCP/IP pre-configuration	-
Data storage issues		Encoding mismatch	-	Manual expanding of the columns
XQuery support	CLOB	✓	✓	x
	XML type	✓	✓	✓
XML handling strong side		Queries execution	Speed	Both
Best average		Discard first	Discard first	Last only
Results representation issues		Results type confusion when working with large data	-	Necessity of enabling log archiving
Errors handling		General info + technical cause	General info	Technical cause

Table 7.1: Summarizing comparison analysis

When creating an XML-based application that designed to work with databases several aspects should be taken into account. The most important thing is the correct decision on what is the main requirement of the application when working with XML data. If the application is focused on querying XML data and obtaining the most possibly full and accurate results it is important to choose the database which strong side is query execution, like Oracle DB. When the application can afford to sacrifice the results accuracy in order to process queries as fast as possible SQL Server should be chosen. If there are no specific requirements for the application in this area, IBM DB2 provides the most optimal relation of these two parameters.

Another important thing to decide is whether the application can afford to imply a big number of prerequisites and manual interferences. In case of choosing Oracle DB, it will be necessary to manually create a new database, specify XML documents encoding when it is needed and correctly deal with the results type conversion when working with large data. SQL Server requires pre-configuration for the connectivity establishment; however, it is the only thing that has to be done manually. When working with large data IBM DB2 is not the best choice as in this case it requires specification of columns length, and working with the sizes of log files.

The last aspect of working with XML data is database failures handling. Oracle DB provides all technical information about the cause of the failure along with a comprehensive explanation. IBM DB2 provides mostly technical information about the cause of the failure, whereas SQL Server includes not very formal, but easy to understand explanation.

Conclusion

The aim of this thesis was to perform a comparison analysis of XML technologies support in Oracle DB, SQL Server and IBM DB2 using created application called XMLTest. XMLTest allows us to run experiments upon XML data stored in each database system. Based on the results obtained from a set of experiments it is possible to define strong and weak sides of each database in particular, estimate the relations between database performances over different criteria, and perform a general comparison of the database systems when working with XML data.

XMLTest is able to store XML documents in a database using different storage methods, and query stored documents using main XML technologies: XQuery, SQL/XML. To run experiments on XQuery language it allows us to use an existing benchmark called XMark. The results of queries processing are represented in a user-friendly way along with creating a set of statistics graphs to compare databases performances based on different aspects. It also allows us to examine the failures of each database by analyzing the errors thrown by a database during processing. The results can be exported to a text file and compared with the results of previous experiments.

However, XMLTest cannot cover all aspects of XML technologies support at a professional level. It is designed to provide a general overview of how modern databases deal with XML data and how this can affect the process of choosing one database over another when creating XML-based applications.

Of course, there are many ways how XMLTest could be improved. The following list shows some ideas which could make XMLTest a more robust application:

- add a support for XSLT and further benchmarks
- add a possibility of shredding XML documents based on XML Schema definition
- allow for a dynamic checking of XML document on well-formedness
- improve the algorithm of XQuery conversion
- correct mistakes in XQuery scripts dynamically based on the experiments results
- optimize application performance when dealing with large data
- provide more effective way to compare the results of queries executions along with comparing it with previous results
- allow users to create their own statistics
- allow users to customize the appearance of application

Bibliography

[1] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C, November 2008, http://www.w3.org/TR/xml

[2] Oracle XML DB Developer's Guide 11g Release 1, http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/toc.htm

[3] Kosek, J., Janota, V.: XML Prague 2009 Conference Proceedings, Prague: MATFYZPRESS, 2009. 276 p., ISBN 978-80-7378-061-6

[4] Sperberg-McQueen, C.M., Thompson, H.: XML Schema, W3C, April 2000, http://www.w3.org/XML/Schema

[5] Oracle XML DB Developer's Guide 11g Release 1: XMLType Operations, http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb04cre.htm#g1050045

[6] Oracle XML DB Developer's Guide 11g Release 1: Rewriting of XQuery and XPath Expressions, http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb01int.htm#BABGCEC F

[7] Hendricks, F., Goutev, D.: DB2 9 pureXML Guide, Redbooks, January 2007, http://www.rebooks.ibm.com/abstracts/sg247315.html

[8] Extensible Markup Language (XML), W3C, January 2012, http://www.w3.org/XML/

[9] What's New for XML in SQL Server 2008?, White Paper, August 2008, http://download.microsoft.com/download/a/c/d/.../WhatsNewSQL2008XML.doc

[10] Pal, S., Zolotov, V.: Performance Optimizations for the XML Data Type in SQL Server 2005, Microsoft Corporation, December 2005, http://msdn.microsoft.com/en-us/library/ms345118(v-sql.90).aspx

[11] Document Object Model (DOM), W3C, January 2005, http://www.w3.org/DOM/

[12] XML DOM Tutorial, W3Schools, http://w3schools.com/dom/default.asp

[13] XML Schema Tutorial, W3Schools, http://w3schools.com/schema/default.asp

[14] XQuery 1.0: An XML Query Language (Second Edition), W3C, December 2010, http://www.w3.org/TR/xquery

[15] XQuery Tutorial, W3Schools, http://w3schools.com/xquery/

[16] XQuery 1.0 and XPath 2.0 Functions and Operators, W3C, February 2005, http://www.w3.org/2005/02/xpath-functions/

[17] XQuery Update Facility 1.0, W3C, http://www.w3.org/TR/xquery-update-10

[18] XSL Transformations (XSLT) Version 1.0, W3C, November 1999, http://www.w3.org/TR/xslt

[19] XSLT Tutorial, W3Schools, http://www.w3schools.com/xsl/

[20] Staken, K.: Introduction to Native XML Databases, October 31, 2001, http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[21] Chong, R., Dang, M.: Introduction to DB2, IBM Press, July 3, 2008, http://www.ibmpressbooks.com/articles/article.asp?p=1163083

[22] Oracle Database Concepts 11g Release 2 (11.2), http://docs.oracle.com/cd/E11882_01/server.112/e16508/intro.htm

[23] Mlýnková, I., Nečaský, M.: Current Support of XML by the "Big Three", http://www.ksi.mff.cuni.cz/~necasky/publications/paper_2009_xmlprague.pdf

[24] Barbosa, D., Manolescu, I., Xu Yu, J.: XML Benchmarks, http://www-rocq.inria.fr/~manolesc/PAPERS/Encyclopedia-XMLBenchmarks.pdf

[25] Swing (Java), Wikipedia, http://en.wikipedia.org/wiki/Swing_(Java)

[26] Graphical User Interface, Wikipedia, http://en.wikipedia.org/Graphical_user_interface

[27] Oracle: JDBC Overview, http://www.oracle.com/technetwork/java/overview-141217.html

[28] Java Documentation: Class SwingWorker<T,V>, http://docs.oracle.com/javase/6/docs/api/javax/swing/SwingWorker.html

[29] The Java Tutorials: Using Prepared Statements, http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

[30] MSDN library: OPENROWSET (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms190312.aspx

[31] Java Documentation: Interface ResultSet, http://docs.oracle.com/javase/1.4.2/docs/api/java/sql/ResultSet.html

[32] The Java Tutorials: Hot to Use Internal Frames, http://docs.oracle.com/javase/tutorial/uiswing/components/internalframe.html

List of Tables

Appendix A

This thesis contains an attached CD with a batch file to run XMLTest, source code of XMLTest and an electronic version of this document.

The disc contains the following directories:

- Program – contains the complete Java source code and a batch file to run XMLTest
- Thesis – contains this text in PDF format