Current Support of XML by the "Big Three"

Irena Mlynkova

Martin Necasky

Abstract

XML technologies have undoubtedly become a standard for data representation and manipulation. Thus it is inevitable to propose and implement efficient techniques for managing and processing XML data. A natural alternative is to exploit tools and functions offered by relational database systems. Even though the native XML databases are undoubtedly more efficient, relational databases are still more popular among XML users due to their long history, maturity and reliability.

In this paper we provide an overview of XML-processing functions that are currently supported by the so-called "Big Three", i.e. *Oracle 11g*, *IBM DB2 9*, and *Microsoft SQL Server 2008*. We firstly show what are the key aspects a user may require from an XML-enabled database. Then, we provide an overview of their support in the respective systems. And, finally, we compare and contrast the findings so that advantages and disadvantages of the particular systems are apparent.

Table of Contents

Introduction

Without any doubt the eXtensible Markup Language (XML) [xml] is currently one of the most popular formats for data representation. The wide popularity naturally invoked an enormous endeavor to propose faster and more efficient methods and tools for managing and processing XML data. Soon it was possible to distinguish several different directions based on various storage strategies. The four most popular ones are methods which store XML data in a file system, methods which store and process XML data using an (object-)relational database management system ((O)RDBMSs), methods which exploit a pure object-oriented approach, and native methods that use special indices, numbering schemas, and/or structures suitable particularly for tree structure of XML data.

Naturally, each of these approaches has both keen advocates and objectors who emphasize its particular advantages or disadvantages. The situation is not good especially for file system-based and pure object-oriented methods. The former ones suffer from inability of querying without any additional preprocessing of the data, whereas the latter approach fails especially in finding a corresponding efficient and comprehensive tool. Expectably, the highest-performance techniques are the native ones, since they are proposed particularly for XML processing and do not need to artificially adapt existing structures to a new purpose. Nevertheless, the most practically used ones are undoubtedly methods which exploit features of (O)RDBMSs. We speak about so-called *XML-enabled databases*. The reason for their popularity is that (O)RDBMSs are still regarded as universal and powerful data processing tools which can guarantee a reasonable level of reliability and efficiency.

The key aim of this paper is to provide an analysis of XML support that is offered by the three leading database vendors and their systems, i.e. *Oracle 11g*, *IBM DB2 9*, and *Microsoft SQL Server 2008*, sometimes denoted as the "Big Three". We firstly show what are the key aspects a user may require from an XML-enabled database management system. Then, we provide an overview of their support in the respective systems including examples that depict their functionality. And, finally, we compare the key findings so that advantages and disadvantages of the particular systems are more apparent.

The paper is structured as follows: Section 2 introduces the problems and issues related to XML processing in general. Section 3 describes how these issues are faced in the Oracle 11g database, Section 4 does the same for IBM DB2 9 database and Section 5 for Microsoft SQL Server 2008. Section 6 overviews the key findings and provides the general comparison of the three systems. And, finally, Section 7 provides conclusions.

General Requirements for XML Processing

In general the basic idea of XML processing based on an (O)RDBMS is relatively simple. The XML data are firstly stored into relations - we speak about so-called *XML-to-relational mapping* or *shredding XML data into tables*. Then, each XML query posed over the data stored in the database is *translated* to a set of SQL queries (which is usually a singleton). And, finally, the resulting set of tuples is transformed to an XML document. We speak about *reconstruction* of XML fragments.

Consequently, the primary concern of the database-based XML techniques is the choice of the way XML data are stored into relations. On the basis of exploitation or omitting information from XML schema we can distinguish so-called *generic* [generic],[generic02] and *schema-driven* [schemadriven], [schemadriven02] methods. From the point of view of the input data we can distinguish so-called *fixed* methods [generic],[generic02],[schemadriven],[schemadriven02] which store the data purely on the basis of their model and *adaptive* methods [flex_semi],[flex_lego],[aam],[hill], where also sample XML documents and XML queries are taken into account to find more efficient storage strategy. And there are also techniques based on user involvement which can be divided to *user-defined* [comerc] and *user-driven* [XCacheDB],[shrex],[usermap], where in the former case a user is expected to define both the relational schema and the required mapping, whereas in the latter case a user specifies just local mapping changes of a default storage strategy.

Approaching the aim from another point of view, the SQL standard has been extended by a new part *SQL/XML* [sqlxml] which introduces new XML data type and operations for both XML and relational data manipulation within SQL queries. It involves functions such as, e.g. `XMLELEMENT` for

creating elements from relational data, `XMLATTRIBUTES` for creating attributes, `XMLDOCUMENT` or `XMLFOREST` for creating more complex structures, `XMLNAMESPACES`, `XMLCOMMENT` or `XMLPI` for creating more advanced parts of XML data, `XMLQUERY`, `XMLTABLE` or `XMLEXISTS` for querying over XML data using XPath [xpath10],[xpath20] or XQuery [xquery], etc.

As we have mentioned in the introduction, the native XML databases differ from the XML-enabled ones in the fact that they do not adapt an existing technology to XML, but exploit techniques suitable for XML tree structure. Most of them use a kind of *numbering schema*, i.e. an index that captures the XML structure. Examples of such schemas are *Dietz's encoding* [dietz], *interval encoding* [xiss], *prefix encoding* [dyntree], *ORDPATHS* [ordpath] or *APEX* [apex]. And, naturally, such indices can be also exploited in relational databases to optimize query processing.

Last but not least, not only storing, querying and indexing are the key operations with XML data. A natural requirement is also the ability to check data validity or to apply XSL [xslt] transformations. However, such features do not need to be incorporated within the database, but they can be ensured using a kind of middleware. On the other hand, one of the key problems of each application is that it is usually dynamic, i.e. the data change. From a short-time period it means that we need to be able to update the data (in XML technologies it is currently covered by the *XQuery Update Facility* [xquery_update_facility]). On the other hand, from a long-time period also the problem of *XML data evolution* occurs [xevolution], i.e. the situation when the modifications of the data are more significant, they usually violate validity and new storage techniques need to be established.

In the following sections we show how these issues are faced in Oracle, IBM DB2 and Microsoft SQL Server.

Oracle 11g

The Oracle corporation[1]calls its ORDBMS Oracle as well. Recently, *Oracle 11g Release 1 (11.1)* [oracle1],[oracle2] has been released.

Storing XML Data

Oracle supports two types of storage strategies - XML data type and a user-defined XML-to-relational mapping. The XML data type is called `XMLType` and its basic usage is very similar to classical atomic SQL data types such as `INTEGER` or `VARCHAR` as depicted by the following example.

Example 1. Oracle: Basic usage of `XMLType`

```
CREATE TABLE person (id NUMBER, desc XMLTYPE);

INSERT INTO person (id, desc) VALUES (1, XMLTYPE(
  '<record>
     <name>Irena Mlynkova</name>
     <date>13/1/2003</date>
     <email>irena.mlynkova@mff.cuni.cz</email>
   </record>'));

INSERT INTO person VALUES (2,
   XMLTYPE(bfilename('mydirectory', 'person.xml'),
           nls_charset_id('AL32UTF8')));

SELECT p.desc.getCLOBVal() FROM person p;
```

The `XMLType` can be stored in three possible ways. If we choose the *structured* storage, the XML data are stored as a set of objects in a set of respective relations. Apparently, this strategy is suitable for data-centric, highly structured XML documents. In case of *binary* storage the data are stored in

[1] `http://www.oracle.com`

a binary format optimized for XML. This approach is suitable for semi-structured XML data which cannot be fully shredded into tables. And, finally, in case of the *non-structured* storage the XML data are stored in the form of `CLOB`. This approach ensures the highest level of *round-tripping*, however at the cost of low efficiency of more complex operations than retrieval of a whole document.

The storage strategy is specified in the `CREATE TABLE` command using `STORE AS` clause. With regard to the three strategies we have three options - nothing, `CLOB` or `BINARY XML`. In addition, we can add characteristics determining the (un)necessity of data validity, e.g. `XMLSCHEMA` *schema_name*, `ALLOW ANYSCHEMA` or `ALLOW NONSCHEMA`, and the root element using `ELEMENT` *element_name* clause.

Example 2. Oracle: Determining storage strategies

```
CREATE TABLE person (id NUMBER, desc XMLTYPE)
XMLTYPE desc STORE AS CLOB
   XMLSCHEMA "http://www.example.com/personschema.xsd" ELEMENT "record";
```

If we do not specify anything, the default storage strategy is structured. In this case Oracle supports two options - each XML collection, i.e. an element with `maxOccurs > 1` can be stored either into a `VARRAY` or into `LOB`. By default all XML collections are stored into `LOB`s, however the user-required modifications can be specified within the `CREATE TABLE` command using clauses `VARRAY` and `LOB`. In addition, in both the cases we can also specify respective storage characteristics.

Example 3. Oracle: User-defined mapping in `CREATE TABLE`

```
CREATE TABLE person (id NUMBER, desc XMLTYPE)
XMLTYPE desc
   XMLSCHEMA "http://www.example.com/personschema.xsd" ELEMENT "record"
VARRAY desc."XMLDATA"."email"
   STORE AS TABLE tableOfEmails (
      (PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
LOB (desc."XMLDATA"."date")
   STORE AS (TABLESPACE USERS ENABLE STORAGE IN ROW
             STORAGE(INITIAL 4K NEXT 32K));
```

Apparently, this approach can be used only in case of simple XML data, since in more complex cases it is quite user-unfriendly. Hence, Oracle supports also another option - annotating the XML schema of the XML data. The XML schema must be first associated with two namespaces `http://xmlns.oracle.com/xdb` and `http://xmlns.oracle.com/2004/CSX`. Then, the elements, attributes or complex types can be annotated using attributes such as `SQLName`, i.e. the name of respective SQL attribute for an element or an attribute stored into a single column, `SQLType`, i.e. name of object type for complex types or SQL type for simple types, `storeVarrayAsTable`, i.e. requirement for storing all collections into `VARRAY` type, etc.

Indexing XML Data

Similarly to indexing of relational data, also indexing of XML data is denoted for the purpose of higher efficiency of respective operations. The indexing approaches in Oracle naturally depend on the selected storage strategy. In case of structural storage, we can exploit classical relational indices. In case of binary or non-structured storage we can exploit XML indices similar to numbering schemas used in native XML databases - so-called `XMLIndex`.

Example 4. Oracle: Indexing an `XMLType`

```
CREATE INDEX myXMLindex ON person.desc INDEXTYPE IS XDB.XMLIndex;
```

The `XMLIndex` consists of three parts - *path index* that indexes all paths of the XML tree, *order index* that indexes relations parent-child, ancestor-descendant and sibling and *value index* that indexes all values. The position of each node is preserved using a variant of the ORDPATHS numbering schema.

Querying XML Data

For the purpose of XML querying Oracle supports two options - XQuery and SQL/XML. Evaluation of the queries can be optimized by the indices as described before.

Example 5. Oracle: SQL/XML and XQuery querying

```
SELECT id, EXTRACTVALUE(desc, '/record/name') "person_name"
FROM person
WHERE id > 10;

SELECT id, XMLQUERY('for $i in /record
                     where $i/name != "Irena Mlynkova"
                     order by $i/name
                     return $i/name'
                     PASSING BY VALUE desc RETURNING CONTENT) XMLData
FROM person;
```

Other Operations

Apart from storing and querying XML data, Oracle supports also other XML-related operations, in particular validity checking and XSL transformations. Validity can be checked in two ways - either within the `CHECK` clause of `CREATE TABLE` command or using a built-in function of `XMLType`.

Example 6. Oracle: Validity checking

```
CREATE TABLE person (id NUMBER, desc XMLTYPE)
CHECK (XMLIsValid(desc) = 1))
XMLTYPE desc
   XMLSCHEMA "http://www.example.com/personschema.xsd" ELEMENT "record";

SELECT p.desc.isSchemaValid(
   'http://www.example.com/personschema.xsd','record')
FROM person p;
```

XSL transformations can be applied using the `XMLTRANSFORM` function on any `XMLType` column. It returns the result as `XMLType` as well.

Example 7. Oracle: XSL transformations

```
CREATE TABLE tableXSL (xsl XMLTYPE);

SELECT XMLTRANSFORM(p.desc, x.xsl).GetClobVal()
FROM person p, tableXSL x
WHERE p.id = 2;
```

Updating XML Data

If we apply the classical `UPDATE` operation on an `XMLType` column of a relation, it causes replacement of the whole XML document stored in it. Naturally, this kind of update operation is required only in very special cases. On the other hand, neither XQuery nor SQL/XML involves update operations, whereas the XQuery Update Facility is not supported in Oracle so far. Nevertheless, Oracle supports own set of SQL functions that enable to replace, insert and delete various XML nodes of XML data. Examples of these functions are `updateXML`, `insertXMLbefore`, `insertXMLafter`, `appendChildXML` or `deleteXML`.

Example 8. Oracle: Update operations

```
UPDATE person
SET desc = APPENDCHILDXML(desc, 'record/name', XMLType('<dg>PhD</dg>'))
WHERE id = 1;
```

XML Schema Evolution

Oracle is able to cope with the problem of XML schema evolution, i.e. a situation when XML schema of the stored data is modified which can cause violation of validity or changes in the storage strategy.

Oracle supports two kinds of schema evolution. In *copy-based* schema evolution all instance documents that conform to an old schema are copied to a temporary location in the database, the old schema is deleted, the evolved schema is registered, and the instance documents are inserted into their new locations from the temporary area. Procedure `DBMS_XMLSCHEMA.copyEvolve` is defined for this purpose and its main parameters involve the old and the evolved schema and an XSL script that re-validates the old XML data if necessary.

On the other hand, the *in-place* evolution does not require copying, deleting, and inserting existing data and thus it is much faster. However, it can be applied only when the *backward compatibility* is ensured, i.e. if no changes to the storage strategy are required and the evolution does not invalidate existing documents. `DBMS_XMLSCHEMA.inPlaceEvolve` is the procedure defined for this purpose.

IBM DB2 Version 9

DB2 version 9 [ibm1],[ibm2] is the latest release of database system provided by the IBM corporation[2].

Storing XML Data

DB2 supports similar storage strategies as Oracle, i.e. XML data type called `XML` and shredding into tables. `XML` is stored in a native structure optimized for XML which is similar to binary storage in Oracle. An XML document inserted into an `XML` column is stored separately from the base table and the column itself contains only a document ID. The shredding strategy exploits a user-defined strategy, where the required mapping is specified using XML schema annotations from namespace `http://www.ibm.com/xmlns/prod/db2/xdb1`. They involve elements/attributes such as `rowSet` for specifying target table name, `column` for specifying column name, `condition` that determines if decomposition inserts a row into a table, `defaultSQLSchema` that specifies the default SQL schema, `expression` that specifies a customized expression whose result is inserted into the respective table, etc.

Indexing XML Data

DB2 supports three XML indices. The *XML region index* stores the locations of each XML document stored in the database. An XML document is stored in one or more regions and the XML region index provides a logical mapping of these regions to retrieve document data. DB2 creates an XML region index for each table with `XML` column automatically. The *XML column path* index is also created automatically for each `XML` column and provides mappings of unique XML paths to their IDs. The last XML index called *XML index* allows enhancing the query performance by indexing XPath expressions. It is not created automatically but must be specified explicitly for each particular XPath expression. A sample XML index is depicted in the following example.

Example 9. DB2: XML index

```
CREATE INDEX person_name_idx ON person(desc) GENERATE KEY USING
XMLPATTERN '/record/name' AS SQL VARCHAR;
```

[2] http://www.ibm.com

Querying XML Data

Similarly to Oracle, DB2 supports XQuery for querying XML data and SQL/XML for exporting relational data to XML and embedding XQuery to SQL queries. In addition, it supports a special non-standard feature - embedding SQL queries in XQuery.

Example 10. DB2: Embedding SQL in XQuery

```
XQUERY
for $person in db2-fn:sqlquery('SELECT desc
                                FROM person WHERE id > 1000')/record
return $person/name;
```

Other Operations

Naturally, DB2 allows validating XML documents against XML schemas. In particular, `XMLVALIDATE` function returns a copy of the input XML value augmented with information obtained from XML schema validation, including default values and type annotations.

Example 11. DB2: XML schema validity checking

```
INSERT INTO person (id, desc)
VALUES (3, XMLVALIDATE('<record>...</record>'
     ACCORDING TO XMLSCHEMA URI 'http://www.example.com/personschema.xsd'
     LOCATION '/mydirectory/person.xml'));
```

On the other hand, XSLT transformations are supported by a built-in engine in DB2 version 9.5 and higher. For transformations, DB2 introduces the `XSLTRANSFORM` SQL function similar to the Oracle one.

Updating XML Data

DB2 allows updating whole XML documents in `XML` columns using standard `UPDATE` SQL command. Moreover, since version 9.5, DB2 supports the XQuery Update Facility as well. It allows changing values of specific XML nodes, replacing nodes as well as inserting, deleting and renaming particular nodes.

Example 12. DB2: XQuery Update Facility

```
UPDATE person
SET desc = xmlquery('
            copy $new := $desc
            modify do replace value of $new/record/email
            with "mlynkova@ksi.mff.cuni.cz"
            return $new')
WHERE id = 1;
```

XML Data Evolution

Lat but not least, DB2 also considers evolution of XML schemas. However, only the case when a new version of an XML schema is backwardly compatible with an old version is considered. Then, a stored procedure `XSR_UPDATE` replaces an existing schema with a new one.

Microsoft SQL Server

The last but not least database vendor is naturally the Microsoft corporation[3]. It produces an object-relational database system whose latest version is called *SQL Server 2008* [micro1],[micro2],[micro3].

Storing XML Data

XML data can be stored in SQL Server in `LOB` data type, native XML data type called `XML` or shredded into tables. The `LOB` option is directed for applications that require the exact copy of the stored XML data including insignificant white-spaces, order of attributes, etc. In case of XML data type the XML data are stored in a native XML repository which preserves all the XML-relevant items. And in the last case a classical user-defined storage strategy is supported, where an annotated XSD determines the particular schema.

Both the `LOB` and `XML` types are used in the same way as in the previous cases, i.e. as classical atomic SQL data types. The set of mapping annotations is defined in namespace `urn:schemas-microsoft-com:mapping-schema` and involves attributes such as `relation` for specifying the relation to store the XML node into, `field` for specification of a particular column, `key-fields` for specifying the columns that uniquely identify the relation, `relationship` for specifying key-foreign key relationship between relations, etc.

Indexing XML Data

An XML index can be created on XML data type columns. The so-called *primary XML index* indexes all tags, values and paths over the XML instances in the column (exploiting the ORDPATHS schema). Having a primary index, we can create any of the allowed *secondary indices* - `PATH`, `PROPERTY` or `VALUE`. The `PATH` index builds a B+ tree on (path, value) pair of each XML node in document order over all XML instances. The `PROPERTY` index creates a B+ tree clustered on the (PK, path, value) tuple within each XML instance, where PK is the primary key of the base table. And, finally, the `VALUE` index creates a B+ tree on (value, path) pair of each node in document order across all XML instances.

Example 13. SQL Server: Indices

```
CREATE TABLE person (id INT PRIMARY KEY, desc XML)

CREATE PRIMARY XML INDEX idx_desc on person (desc)

CREATE XML INDEX idx_desc_path on person (desc)
   USING XML INDEX idx_desc FOR PATH

CREATE XML INDEX idx_desc_property on person (desc)
   USING XML INDEX idx_desc FOR PROPERTY
```

In addition, SQL Server enables to create a full-text index (that can be used also for other SQL data types) over the XML data type column. Then, the SQL function `CONTAINS` enables to check whether the XML instance contains the given string anywhere in the document.

Example 14. SQL Server: Full-text indices

```
CREATE FULLTEXT CATALOG ft AS DEFAULT

CREATE FULLTEXT INDEX ON person (desc) KEY INDEX PK_docs_023D5A04

SELECT * FROM person
WHERE CONTAINS (desc, 'mff.cuni.cz')
```

[3] `http://www.microsoft.com`

Querying XML Data

Firstly, the XML data type can be queried using XQuery. For this purpose SQL Server supports various built-in functions having the XQuery query as a parameter, such as `exist` checking existence of nodes, `value` returning an SQL value and `query` returning XML data type result. In case of shredded XML data, function `nodes` returns one row for each node that matches the query.

To ease usage of both SQL and XML data in queries, SQL Server exploits the idea of data binding, i.e. mapping SQL values to XML values. For this purpose it provides two functions - `sql:variable` to use the value of an SQL variable and `sql:column` to use values from a relation column in XML query.

Example 15. SQL Server: Data binding in SQL queries

```
DECLARE @date varchar(20)
SET     @date = '13/1/2003'
SELECT  desc
FROM    person
WHERE   desc.exist ('/reord[date = sql:variable("@date")]') = 1
```

Contrary to the previous two cases the SQL Server's *SQLXML* has nothing in common with the SQL/XML standard. The idea is similar - to bridge the gap between SQL and XML data - however the syntax and usage is different. The `OPENXML` construct provides an SQL view of XML data using the user specified mapping, that can be specified either explicitly (as in the following example) or using a parameter that denotes a general mapping strategy.

Example 16. SQL Server: `OPENXML`

```
SELECT *
FROM OPENXML (@docHandle, '/record')
WITH (PersonName  varchar(10)   'name',
      PersonID    int           '@id',
      PersonEmail varchar(10)   'email')
```

Conversely, the `FOR XML` construct enables to create an XML view of SQL relations. The way the data are mapped can be influenced using four modes. The `RAW` mode generates a single `<row>` element per each row in the rowset that is returned by the `SELECT` statement; its columns are mapped either to attributes or subelements depending on other parameters. The `AUTO` mode generates nesting in the resulting XML using heuristics based on the way the `SELECT` statement is specified. The `EXPLICIT` mode allows more control over the shape of the XML view using a set of special directives. And, finally, the `PATH` mode together with the nested `FOR XML` query capability provides the flexibility of the `EXPLICIT` mode in a simpler manner - using paths and nested queries.

Example 17. SQL Server: `FOR XML` query and result

```
SELECT ProductModelID as "@id", Name
FROM Production.ProductModel
WHERE ProductModelID=122 or ProductModelID=119
FOR XML PATH ('ProductModelData')

<ProductModelData id="122">
  <Name>All-Purpose Bike Stand</Name>
</ProductModelData>
<ProductModelData id="119">
  <Name>Bike Wash</Name>
</ProductModelData>
```

Other Operations

Similarly to the previous cases, validity checking is supported in SQL Server. For this purpose it introduces a so-called `SCHEMA COLLECTION` which enables to store one or more XML schemas. An `XML` column or variable can be then associated with such collection and, hence, validity checking is ensured.

Example 18. SQL Server: `SCHEMA COLLECTION` and its usage

```
CREATE XML SCHEMA COLLECTION MyColl AS '
   <schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.com/personschema.xsd">
     <!- ... ->
   </schema>'

CREATE TABLE person (id INT PRIMARY KEY, desc XML (MyColl))
```

On the other hand, the XSL processing is not a direct part of SQL Server, e.g., in the form of a built-in function. However, it can be easily extended with such functionality using a stored procedure implemented using an external tool.

Updating XML Data

The XML data type supports a built-in function `modify`, that enables to update the respective XML data. As a parameter it gets the required operation. In particular, subtrees can be inserted before or after a specified node, or as the leftmost or rightmost child. Attribute, element, and text node insertions are all supported as well. Deletion of subtrees is supported and scalar values can be replaced with new scalar values.

Example 19. SQL Server: Updating XML data

```
UPDATE person SET desc.modify('
  insert <dg>PhD</dg>
  after (/record/name[.="Irena Mlynkova"])')
```

XML Data Evolution

SQL Server also deals with evolution of XML schemas, even though only in a very special way and only for `XML` columns. In fact, it is easily ensured using `SCHEMA COLLECTION`s and their ability to be extended with new schemas.

Example 20. SQL Server: Adding a schema to `SCHEMA COLLECTION`

```
ALTER XML SCHEMA COLLECTION MyColl ADD '
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.example.com/personschema2.xsd">
     <!- ... ->
  </schema>'
```

Consequently, the respective `XML` column or variable can contain data valid against both old and new XML schemas in the collection.

Overview and Comparison

For better lucidity the following table provides and overview of the XML-related functions that are (not) supported in the three systems and their key characteristics.

Table 1. Overview and comparison of key XML features

Feature	Oracle	DB2	SQL Server
XML data type	`XMLType`	`XML`	`XML`
	Structured, binary, non-structured	Binary, structured	`LOB`, native, structured
Mapping	User-defined	User-defined	User-defined
	Names, data types and storage strategies (`VARRAY` vs. `LOB`)	Relations, columns, conditions, expressions	Relations, columns, keys, relationships
Indexing	`XMLIndex`	Region index, column path index, XML index	Primary, secondary (`PATH`, `PROPERTY`, `VALUE`), full-text index
	ORDPATHS, path index, axes index	Indexing particular XPath expressions	ORDPATHS
Querying	XQuery, SQL/XML	XQuery, SQL/XML, SQL embedded to XQuery	XQuery, SQLXML (`OPENXML`, `FOR XML`)
Other operations	Validity checking, XSL transformations	Validity checking, XSL transformations	Validity checking, XSL transformations only via an external tool
Updating	Own functions for inserting, replacing, deleting nodes	XML Update Facility	Own function with parameter for inserting, replacing, deleting nodes
Evolution	With/without backward compatibility	Backward compatibility must be ensured	Only for `XML` columns using `SCHEMA COLLECTION`s

As we can see, in general, all the three vendors follow the same pattern and try to support as much XML functionality as possible. The most advanced and, at the same time, standard-conforming support has Oracle, whereas the SQL Server traditionally ignores the proposed standards the most.

Under a closer investigation we can see that there are some significant differences in respective areas of XML support. Firstly, while all three systems support a kind of XML data type as well as shredding into relations, the storage strategies do not follow any kind of common standards. In addition, in case of user-defined mapping, all the systems require quite a highly skilled user, i.e. user-driven strategies are not supported.

As for the query capabilities, all the three systems support several kinds of indices that enable to speed up XML query evaluation. Naturally, they are highly related to the selected storage strategy. All the systems support the XQuery language and its embedding in SQL to enable working with both XML and SQL data at the same time. To further increase this ability, both Oracle and DB2 support the SQL/XML standard, while SQL Server provides own set of functions called SQLXML. Surprisingly, DB2 also supports a new feature - embedding SQL queries into XQuery.

Considering other operations with XML data, all systems naturally support validity checking and XSL transformations. On the other hand, considering 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 only DB2 already supports the XQuery Update Facility, i.e. a standard approach.

Last but not least, the important aspect of XML data evolution is being considered by all the systems, but only Oracle enables to deal with significant structural changes using XSL transformations. However, in this case the user responsibility for data correction is full, there are no automatic options.

Conclusion

The aim of this paper was to provide an overview of XML support in the currently most popular (object-)relational database management systems - Oracle 11g, IBM DB2 9, and Microsoft SQL Server 2008 - the so-called "Big Three". Firstly, we discussed the main topics related to XML processing. Then, we described how these issues are faced in particular systems. And, finally, we provided their mutual comparison. Our aim was to show which of the solutions already proposed in the scientific world are being exploited in the industry and to what extent. This text should serve as a good source of information for developers who search for a system supporting particular functions, as well as for researchers looking for an up-to-date topic with practical exploitation.

Acknowledgement

This work was supported in part by the Czech Science Foundation (GA#R), grant number 201/09/P364.

Bibliography

[xml] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau: Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C, 2008.

[generic] D. Florescu and D. Kossmann: Storing and Querying XML Data Using an RDMBS. IEEE Data Eng. Bull., 22(3):27–34, 1999.

[generic02] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. ACM Trans. Inter. Tech., 1(1):110–141, 2001.

[schemadriven] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In VLDB'99, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[schemadriven02] K. Runapongsa and J. M. Patel. Storing and Querying XML Data in Object- Relational DBMSs. In EDBT'02, pages 266–285, London, UK, 2002. Springer.

[flex_semi] M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In Selected papers from the 3rd Int. Workshop WebDB'00 on The World Wide Web and Databases, pages 151–170, London, UK, 2001. Springer.

[flex_lego] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In ICDE'02, pages 64–75, Washington, DC, USA, 2002. IEEE.

[aam] W. Xiao-ling, L. Jin-feng, and D. Yi-sheng. An Adaptable and Adjustable Mapping from XML Data to Tables in RDB. In VLDB'02 Workshop EEXTT and CAiSE'02 Workshop DTWeb, pages 117–130, London, UK, 2003. Springer.

[hill] S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In DASFAA'03, pages 337–344, Kyoto, Japan, 2003. IEEE.

[comerc] S. Amer-Yahia. Storage Techniques and Mapping Schemas for XML. Technical Report TD-5P4L7B, AT&T Labs-Research, 2003.

[XCacheDB] A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. The VLDB Journal, 14(1):30–49, 2005.

[shrex] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to- Relational Mapping Problem. In WIDM'04, pages 31–38, New York, NY, USA, 2004. ACM.

[usermap] I. Mlynkova. A Journey towards More Efficient Processing of XML Data in (O)RDBMS. In CIT'07, pages 23–28, Los Alamitos, CA, USA, 2007. IEEE.

[sqlxml] ISO/IEC 9075-14:2003. Part 14: XML-Related Specifications (SQL/XML). Int. Organization for Standardization, 2006.

[xpath10] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C, November 1999. http:// www.w3.org/TR/xpath.

[xpath20] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simeon. XML Path Language (XPath) 2.0. W3C, January 2007. http://www. w3.org/TR/xpath20/.

[xquery] S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. W3C, January 2007. http://www.w3. org/TR/xquery/.

[dietz] P. F. Dietz. Maintaining Order in a Linked List. In STOC'82, pages 122–127, New York, NY, USA, 1982. ACM.

[ordpath] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In SIGMOD'04, pages 903–908, New York, NY, USA, 2004. ACM.

[apex] C.-W. Chung, J.-K. Min, and K. Shim. APEX: an Adaptive Path Index for XML Data. In SIGMOD'02, pages 121–132, New York, NY, USA, 2002. ACM.

[xiss] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In VLDB'01, pages 361–370, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers, Inc.

[dyntree] E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In PODS'02, pages 271–281, New York, NY, USA, 2002. ACM.

[xevolution] M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In EDBT'06, LNCS, pages 1143–1146. Springer, 2006.

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

[xquery_update_facility] D. Chamberlin, D. Florescu, J. Melton, J. Robie, and J. Simon. XQuery Update Facility 1.0. W3C, srpen 2007. http://www.w3.org/TR/xquery-update-10/.

[oracle1] Oracle Database 11g. Oracle Corporation. http://www.oracle.com/technology/ products/database/ oracle11g/.

[oracle2] Oracle XML DB Developer's Guide 11g Release 1 (11.1). Oracle Corporation. http:// download.oracle.com/docs/cd/B28359_01/appdev.111/b28369/ toc.htm.

[ibm1] DB2 Product Family. IBM. http://www-01.ibm.com/software/data/db2/.

[ibm2] IBM DB2 for Linux, UNIX, and Windows: Managing XML Data - Best Practices. IBM. http:// download.boulder.ibm.com/ibmdl/pub/software/dw/dm/ db2/bestpractices/DB2BP_XML_0508I.pdf.

[micro1] Microsoft SQL Server 2008. Microsoft Corporation. http://www.microsoft.com/ sqlserver/2008/.

[micro2] XML Best Practices for Microsoft SQL Server 2005. Microsoft Corporation. http: //msdn.microsoft.com/ en-us/library/ms345115.aspx.

[micro3] White Paper: What's New for XML in SQL Server 2008. Microsoft Corporation. http:// www.microsoft.com/sqlserver/2008/en/us/ wp-sql-2008-whats-new-xml.aspx.