Managing XML Data to optimize Performance into Object-Relational Databases

Iuliana BOTHA
Academy of Economic Studies, Bucharest, Romania
iuliana.botha@ie.ase.ro

This paper propose some possibilities for manage XML data in order to optimize performance into object-relational databases. It is detailed the possibility of storing XML data into such databases, using for exemplification an Oracle database and there are tested some optimizing techniques of the queries over XMLType tables, like indexing and partitioning tables.
Keywords: *object-relational database, XML data, optimizing technique, index, partitioned table.*

1 Introduction

In the last decades, the world economy was characterized by the transition from industrial to information society, which is governed by a new set of rules that use digital technologies for accessing, processing, storing and transferring the information.

In all fields of activity are required accurate and timely obtained information. This information is obtained from primary data collected and organized into databases following extensive processes performed using complex software products. Modern information systems are currently structured in different types and are practically identified with the complex and changing economic activity.

Currently, organizations are required to store and process increasing quantities of data, requiring recourse to modern information technology, databases, data warehouses, Internet and intelligent systems.

Thus, in recent years are rapidly developed some new ways to store and manipulate multimedia and spatial data. Since relational databases (RDB) have limitations in the case of special data (like multimedia, spatial, XML), the most effective way proves to be the use of object-relational databases (ORDB) [1].

2. Brief considerations about XML technology

eXtensible Markup Language (XML) is a platform-independent format for representing data and was designed as a standard for information exchange over the Internet. XML enables easy exchange of information, which allows interoperability between applications due to data encapsulation with metadata.

The studies [5], [6] and [7] present two approaches for storing XML data: through native XML databases or using mapping techniques for translate XML data into a relational or object-relational database. Also, they propose mapping algorithms and rules from XML Schema to object-relational database schema.

Current paper will expose the possibility of storing XML data into object-relational databases, using for exemplification an Oracle database. The main advantage of using object-relational databases is that we can get the benefits of both relational and object-oriented technologies. However, this translates into lower performances due to XML data mapping to the relational data, which can produce a database schema with many relations.

Storing data as XML also provides certain facilities. First, XML is self-describing, and applications can consume XML data without knowing their schema or structure. XML data are always arranged hierarchically as a tree. XML tree structure has a parent node, known as an XML document. If a set of XML nodes have no parent node, it is an XML fragment. Second, the ordering is maintained in the XML document. Thirdly, the scheme declaration provides validation into the document. XML is a language used to define a structure for a valid XML document or fragment. XML Schema allows the declaration of optional sections or types inside generic scheme that supports any XML fragment. This means that XML data can be used for representing semi-structured or unstructured data. Fourth, XML allows searching. Due to the hierarchical structure, multiple algorithms can be applied to search within the tree structure. Fifthly, XML data are extensible. XML data can be manipulated by inserting, modifying and deleting nodes. This means you can create new XML instances of existing XML structures.

3. Brief considerations about object-relational databases

The object-relational databases are a hybrid type of databases, which use the best facilities of its predecessors (relational and object-oriented databases) [3]. In other words, they can be considered an object-oriented extension of the relational databases. The internal logic of storing and retrieving data is the same like in the relational case. The main difference consists in new data types, some of them user defined (like object classes), and in the ability of manipulate them. Multimedia, spatial and XML data are important resources, which need to be manipulated, in order to use them in specific applications.

However, one can observe that, in a table of such database, the main part of the columns have nothing in particular, being just standard columns. The exceptions come from these columns that contain complex data: large objects (LOB), object types, spatial data, XML data [1].

The new standard for object-relational design is SQL:1999 and provides support for user defined abstract data types, which can be used in the same way as the standard data types. This allows for the encapsulation of an object within another object. Also, SQL:1999 added support for XML platform for data representation using text files.

As stated in [2], using this hybrid type of database has its main reason for that:

- In many cases, the existing applications are already based on a relational data model. This calls for coexistence with the relational model as long as we do not want to redesign the applications based on a common object model to be included in a single OODB;
- Performance and scalability are important properties of an application, and in this respect, OODBMS have not yet shown advantages over RDBMS.

The main issue, in the case of object-relational databases, is how to store objects using tables and how to transform complex requirements of applications into properties stored in databases, all in a simple and clear way, that keep the structure of object-oriented application, reduce programming effort and maintain a reasonable level of performance [10].

As specified in [1], the object-relational database management system (ORDBMS) offer is very generous and covers a wide scale of cost and performance, going from the DBMS that can be used for free (unlicensed or with public license, such as PostgreSQL) to the commercial ones such as Oracle 10g, DB2 UDB 8, and SQL Server 2005. All these DBMS types extend their relational model with abstract data types and object-oriented properties.

4. Managing XML Data in Oracle ORDBMS

Oracle is a relational database management system (RDBMS), but since version 10g is included into the category of relational DBMS extended with facilities for defining and processing the types of objects - ORDBMS. Thus, the system can distinguish between types (classes) of objects and objects (instances of objects types) [8].

Oracle specific procedural language, PL/SQL, supports object-oriented programming features and objects types (equivalent with objects classes). An object type encapsulates a data structure with functions and procedures for handling data. The variables that form the object type are called attributes. The functions and procedures that manipulate the attributes are called methods. The definition of objects types and the methods are stored in the database. Instances of these types of objects can be stored in tables and used as variables in PL/SQL programs [9].

A new Oracle database functionality consists into XML data management, through Oracle XML DB component. It provides high-performance storage and retrieval of XML data.

The main components of Oracle XML DB are shown in Figure 1 and the most important features are highlighted in [13] and summarized below:

- Supports XML Schema data models;
- Provides methods for navigating and querying XML data;
- Allows DML statements over the XML data;
- Allows standard methods for accessing and updating XML, including W3C XPath recommendation and the ISO-ANSI SQL/XML standard;
- The transfer of XML data in and out of Oracle Database can be made using FTP, HTTP or WebDAV;
- Enables the management of the XML hierarchy;
- Includes a XML repository that allows XML content to be organized and managed;
- Provides a storage-independent, content-independent and programming-language-independent infrastructure for storing and managing XML data;
- Supports standard APIs used for programmatic access and manipulation of XML content using Java, C, and PL/SQL;
- Allows specific memory management and optimizations;
- Allows Oracle Database main features, such as reliability, availability, scalability, and security for XML content.

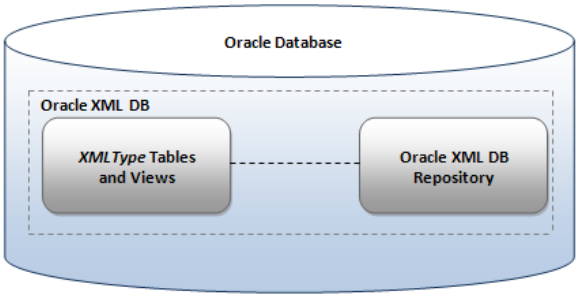

Figure 1 –Main components of Oracle XML DB
(Source: adapted from [13])

In today's organizations, the data is managed differently depending on their structured or unstructured format. Thus, unstructured data is stored into tables, while structured data is stored into LOB data files (Large Objects).

Oracle database allows XML data to be stored and managed whether they are structured, unstructured or semi-structured data. Using Oracle it can be performed XML operations on object-relational data, but also SQL operations on XML documents.

As shown in figure above, when we use Oracle XML facilities we discuss about *XMLType* data type and XML repository.

XMLType is an Oracle server data type, similar to the native data types like DATA, NUMBER or VARCHAR2. *XMLType* allows the database to understand that a column or table contains XML and also provides methods that allow standard operations such as XML Schema validation and XSL transformations.

According to [12], the modalities to store *XMLType* data are the following:

- Structured storage, in tables or views, when we discuss about structured data;
- Large objects (LOB) storage, when we discuss about unstructured or semi-structured data and we need to store XML document as a whole.

Table 1 listed below indicates the main features of each type of storage:

Table 1 – The main features of each *XMLType* storage modality
(Source: adapted from [12])

CHARACTERISTIC	STRUCTURED STORAGE	LOB STORAGE
Database schema flexibility	Limited flexibility for schema changes	Good flexibility for schema changes
Data integrity and accuracy	Limited data integrity. Maintains DOM fidelity.	Maintains the original XML byte for byte - important in some applications
Performance	Good performance for the DML statements	Medium performance for the DML statements
SQL features	Good accessibility to existing SQL features, such as constraints, indexes, and so on	Medium accessibility to SQL features
Space needed	Consume less space when used with Oracle XML DB	Can consume considerable space

We can use *XMLType* as the data type of columns in database tables or views, as shown in the following example:

```
CREATE TABLE users
(
user_id NUMBER(3),
username VARCHAR2(15),
password VARCHAR2(20),
personal_data XMLTYPE
);
```

The structure for the *XMLType* data can be visualized in the tree-structure represented in the Figure 2:

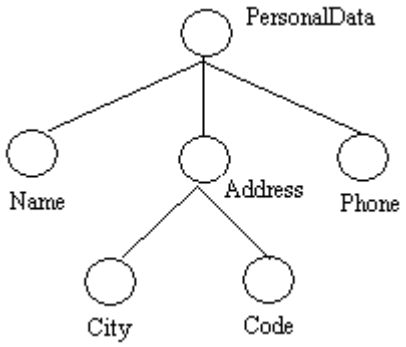

Figure 2 – XML hierarchy for personal data

If we choose to store XML data in an *XMLType* column as a CLOB column, we have the possibility to specify LOB storage characteristics for that column, as shown in the following example:

```
CREATE TABLE users2
(
user_id NUMBER(3),
username VARCHAR2(15),
password VARCHAR2(20),
personal_data XMLTYPE
)
XMLType COLUMN personal_data
STORE AS CLOB
(
TABLESPACE lob_example
STORAGE
(
INITIAL 4096
NEXT 4096
)
CHUNK 4096
NOCACHE
LOGGING
);
```

In order to create an *XMLType* instance will be used the *XMLType()* constructor applied to a VARCHAR2 string or to a CLOB (Character Large Object) data. The stored data can be seen as in Figure 3.

```
INSERT INTO users VALUES
(100, 'User100', 'pass',
XMLType('<PersonalData user="100">
          <Name>Ionescu</Name>
          <Address>
          <City>Bucharest</City>
          <Code>012345</Code>
          </Address>
          <Phone>0211234567</Phone>
          </PersonalData>'));
```

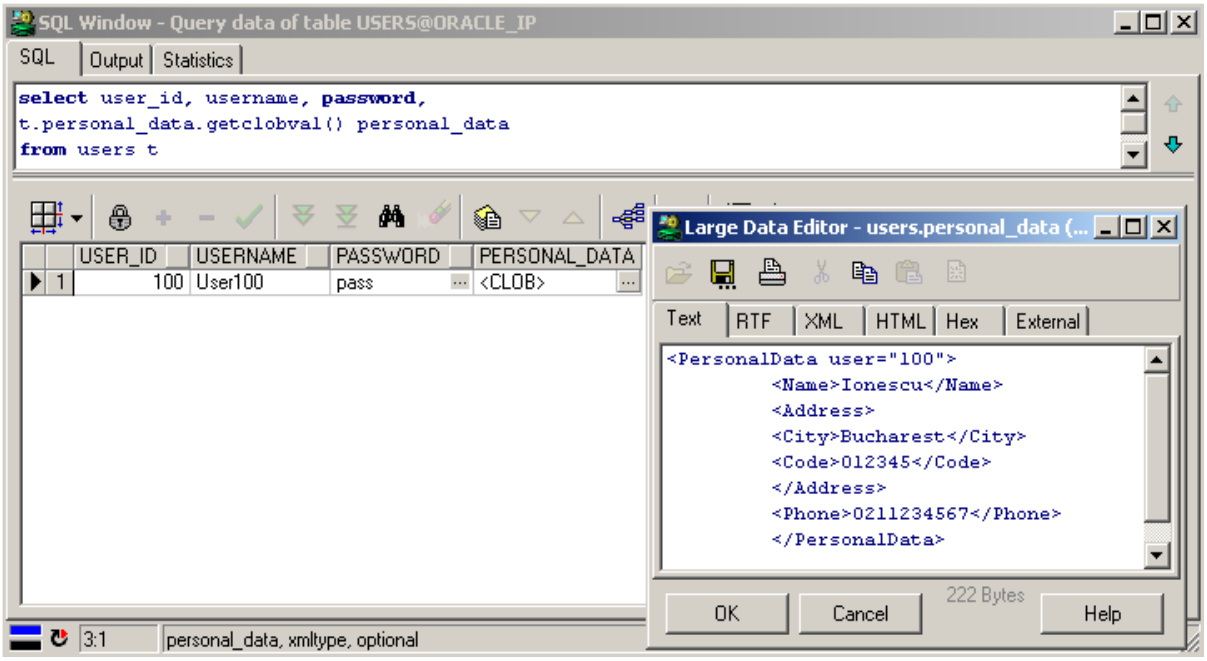

Figure 3 – The modality to visualize the XML data

Another way for using this data type allows us to create tables of *XMLType*. Thus, the below example creates the person table of *XMLType*. In this case, the default type of storage is CLOB based.

```
CREATE TABLE persons of XMLType;
```

XML type offers great search and query facilities. The developers have the ability to use different methods that allow manipulation of XML data, like: *extract(), createXML(), existsNode(), getCLOBVal(), getStringVal() or getNumberVal()*.

In order to query a table which has a *XMLType* column, simple or complex, will be used the method *extract()* of the object type. The result of the method will be a VARCHAR2 value.

```
SELECT u.username,
u.personal_data.extract('/Personal
```

```
Data/Name/text()').getStringVal()
Name,
u.personal_data.extract('/Personal
Data/Address/City/text()').getStri
ngVal() City
FROM users u;
```

The above query retrieves values of the nodes from XML structure presented in Figure 2, by using the path to these nodes.

USERNAME	NAME	CITY
User100	Ionescu	Bucharest

Figure 4 – Result of the SELECT statement

The others DML statements (update and delete) are no different from updating or deleting rows containing any other standard data type. Obviously, specific *XMLType* methods can be used in order to identify rows to update or delete, like in the following example:

```
DELETE FROM users u
WHERE
upper(u.personal_data.extract('//C
ity/text()').getStringVal()) =
'BUCHAREST';
```

Other modalities to manipulate the *XMLType* data use PL/SQL or Java programs. In addition, for loading the XML documents into the repository can be used the PL/SQL standard package *DBMS_XDB*, which stores under a given path the document.

5. Optimizing database performance by managing XML Data

Database performance can be optimized through a severe management of XML data and appropriate optimizing techniques, like indexing and partitioning tables.

When a query is executed over a table with *XMLType* columns, the query optimizer takes into consideration many factors related to the objects referenced and the conditions specified in the query, in order to identify the most efficient technique. The query optimizer estimates the cost of the execution plan, which is an estimated value that depends on resources used to execute the statement (in terms of I/O, CPU and memory) [4].

Oracle uses indexes to avoid the need for full-table scans which are required when the query optimizer cannot find an efficient way to service the SQL statement.

An index is used to find data quickly, regardless of the amount of data. The structures used by Oracle to create and maintain indexes are B-tree and bitmap indexes.

The oldest and most popular type of indexing is a classic B-tree index. A B-tree consists of a root node that contains one page of data, 0 or more additional pages containing intermediate levels, and a leaf level. Leaf level contains entries that correspond to ordered data that are indexed.

Oracle bitmap indexes are very different from standard b-tree indexes. This type of index creates a two-dimensional array with one column for every row in the table being indexed. Each column represents a distinct value within the bitmapped index. The array created represents each value within the index multiplied by the number of rows in the table.

An interesting and important feature in Oracle indexing is represented by function-based indexes. Thus, are created indexes on expressions, internal functions, and user-defined functions in PL/SQL or Java. A function-based index ensures matching any condition in a query and replaces the unnecessary full-table scans with super-fast index range scans.

In order to identify how database performance can be optimized, we will execute some queries on *XMLType* tables stored into repository from Oracle database.

First, performing a query against the USER_XML_TABLES data dictionary view will mark the *XMLType* tables from the repository:

```
SELECT table_name, storage_type,
xmlschema FROM user_xml_tables;
```

The result obtained in this case indicates the Persons table as *XMLType* table, stored with the object-relational storage option, as we can see in table properties.

We will now execute the query below to see if its execution plan is optimal:

```
SELECT
p.extract('/PersonalData/Name/text
()').getStringVal() Name,
p.extract('/PersonalData/Address/C
ity/text()').getStringVal() City
FROM persons p
WHERE
lower(p.extract('/PersonalData/Add
ress/City/text()').getStringVal())
='brasov';
```

As we can observe in the Figure 5, the execution plan for the query performed involves an inefficient TABLE ACCESS FULL operation, with a cost of execution estimated at 8.

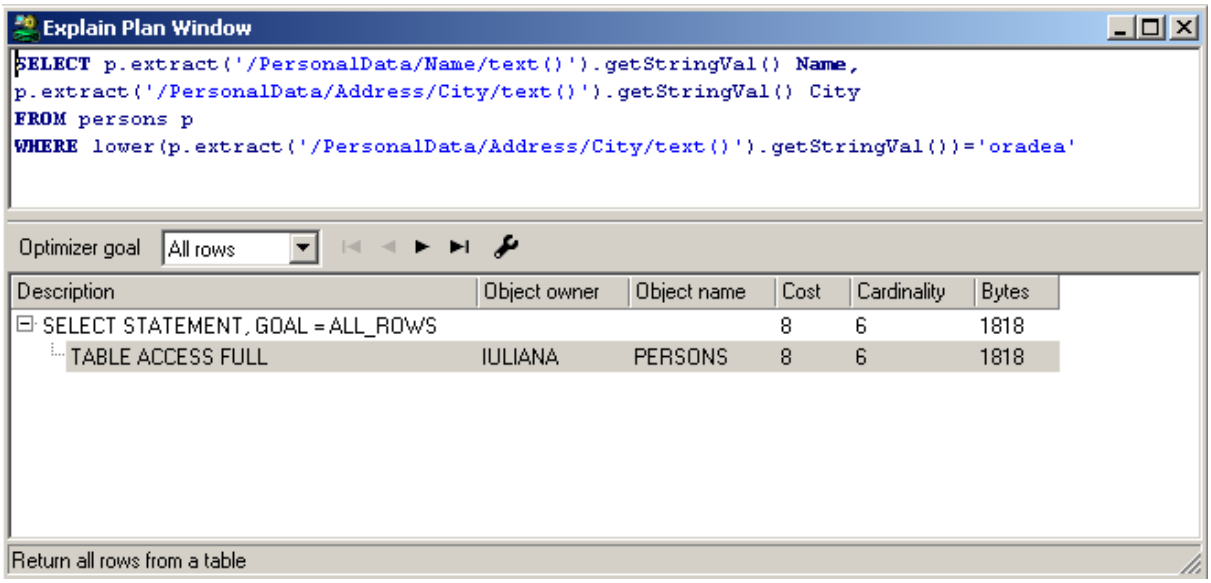

Figure 5 – The execution plan before creating the index

To increase performance of query execution, a function-based index will be created:

```
CREATE INDEX city_index ON persons
p(lower(p.extract('/PersonalData/A
ddress/City/text()').getStringVal(
)));
```

To examine the created indexes on a table, can be run the query shown below:

```
SELECT index_name, index_type,
table_name
FROM user_indexes
WHERE table_name='PERSONS';
```

The statistics for the execution plan are not refreshed automatically, but at a specific time or when this is an explicit requirement. In this case, we collect information about the tables in the current scheme using a function included in the standard package DMBS_STATS:

```
BEGIN
DBMS_STATS.GATHER_TABLE_STATS(user
, 'persons');
END;
/
```

After running the PL/SQL block above, we will check the execution plan again, by running the SELECT statement tested before creating the index. The result

indicates that the query execution plan has improved (the cost for executing the query is now estimated at 2), as shown in the Figure 6. Also, TABLE ACCESS FULL operation has been replaced by more efficient TABLE ACCESS BY INDEX ROWID and INDEX RANGE SCAN operations.

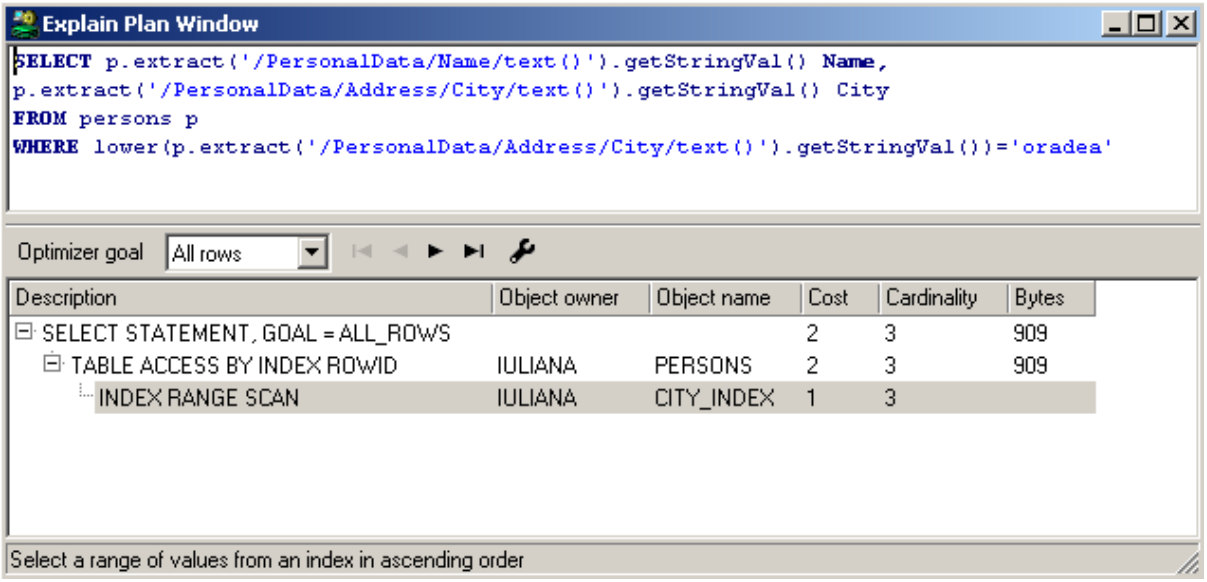

Figure 6 – The execution plan after creating the index

Moving forward in order to identify optimizing techniques, we will study the effects of partitioning against the queries built on *XMLType* tables.

The main objective of the partitioning technique is to radically decrease the amount of disk activity and to limit the amount of data to be retrieved.

Tables are divided into partitions using a partitioning key. This is a set of columns that will determine by their conditions in which partition a given row will be stored.

Partitioning for object-relational storage was introduced in Oracle Database 11g to help simplify XML data life-cycle management and performance [11].

We will create an *XMLType* table with partitioned object-relational storage using a XML Schema for identification of the XML hierarchy elements. Then, the table will be populated with data selected from the Persons table.

```
CREATE TABLE person_part OF
XMLTYPE
XMLSCHEMA
"http://localhost:8080/orabpel/xml
lib/XMLSchema_persons.xsd"
ELEMENT "personal_data"
PARTITION BY LIST
(personal_data.address)
(PARTITION a VALUES ('Bucharest'),
PARTITION b VALUES ('Iasi'),
PARTITION c VALUES ('Oradea'),
PARTITION d VALUES ('Brasov')
);
```

The XML Schema which is pointed in the CREATE TABLE statement is presented below:

```
<?xml version="1.0" encoding="ISO-
8859-1" ?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/X
MLSchema">
<xs:element name="personal_data">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="name"
type="xs:string"/>
    <xs:element name="address">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="city"
type="xs:string"/>
          <xs:element name="code"
```

```
type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="phone"
type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

For better performance improvement of the queries we can chose to create indexes on the partitioned *XMLType* table. The execution plan resulted will be more efficient for large data sets.

7. Conclusion

The paper presents the object-relational database main features and the possibilities for integration with XML technology. We have presented and tested two optimizing techniques used by Oracle database for the queries that are built on *XMLType* tables.

8. Acknowledgment

This paper presents some results of the research project PN II, TE Program, Code 332: "Informatics Solutions for decision making support in the uncertain and unpredictable environments in order to integrate them within a Grid network", financed within the framework of People research program.

References

[1] Iuliana Botha, AndaVelicanu, Adela Bâra, "Integrating Spatial Data with Object Relational-Databases", *Journal of Database Systems*, no.1/2011, pp. 33-42, ISSN: 2069–3230

[2] Gheorghe Sabau, "Comparison of RDBMS, OODBMS and ORDBMS", *The Proceedings of the 8th International Conference on Informatics in Economy,* Bucharest, 2007, pp. 792-796, ISBN 978-973-594-921-1

[3] Michael Stonebraker, Dorothy Moore, "Object-Relational DBMS - The Next Great Wave", Morgan-Kaufmann, 1996, ISBN: 155-860-397-2

[4] Adela Bâra, Ion Lungu, Manole Velicanu, Vlad Diaconița, Iuliana Botha, „Extracting data from virtual data warehouses – a practical approach of how to improve query performance", *The Proceedings of the 7th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, 2008, pp. 509-514, ISBN: 978-960-6766-41-1, ISSN: 1790-5109

[5] Laila Alami Kasri, Noureddine Chenfour, "Model of Storage XML Database based on the Relational-Object Model", *International Journal of Engineering Science and Technology*, Vol. 2(11), 2010, ISSN 0975–5462

[6] Irena Mlynkova, Jaroslav Pokorny, "From XML Schema to Object-Relational Database – an XML Schema-driven mapping Algorithm", *Proceedings of the 3rd IADIS International Conference WWW/Internet*, Madrid, Spain, 2004, pp 115 - 122, ISBN 972-99353-0-0

[7] Irena Mlynkova, Jaroslav Pokorny, "XML in the World of (Object-) Relational Database Systems", *Information Systems Development: Advances in Theory, Practice, and Education*, Vilnius, Lithuania, 2004, pp. 63 - 76, ISBN 978-0-387-25026-7

[8] Manole Velicanu, *Dicționar explicativ al sistemelor de baze de date*, Economica Publishing House, Bucharest, 2005, ISBN 709-114-0

[9] Manole Velicanu, Ion Lungu, Iuliana Botha, Adela Bâra, Anda Velicanu, Emanuil Rednic, *Advanced Database Systems*, AES Publishing House, Bucharest, 2009, ISBN: 978-606-505-217-8

[10] *Oracle Database, Application Developer's Guide: Object-Relational*

Features, Oracle tutorial, December 2003

[11] *Using Oracle XML DB to Optimize Performance and Manage Structured XML Data*, Oracle tutorial, http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/11g/r2/prod/appdev/xmldb/xmldb_structured/optimizeandmanageXMLdata_v3.htm

[12] http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96620/xdb04cre.htm

[13] *Oracle XML DB Developer's Guide*, Oracle tutorial, http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm

Iuliana BOTHA is an Assistant Lecturer at the Economic Informatics Department at the Faculty of Cybernetics, Statistics and Economic Informatics from the Academy of Economic Studies of Bucharest. She has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2006 and the Databases for Business Support master program organized by the Academy of Economic Studies of Bucharest in 2008. Currently, she is a PhD student in the field of Economic Informatics at the Academy of Economic Studies. She is co-author of 4 books, 8 published articles (2 articles ISI indexed and the other 6 included in international databases), 16 scientific papers published in conferences proceedings (among which 4 paper ISI indexed). She participated as team member in 4 research projects that have been financed from national research programs. From 2007, she is the scientific secretary of the master program *Databases for Business Support* and she is also a member of INFOREC professional association. Her scientific fields of interest include: Databases, Database Management Systems, Design of Economic Information Systems, Business Intelligence.