THE BUCHAREST ACADEMY OF ECONOMIC STUDIES

ISSUE
4

Database Systems Journal

ISSN: 2069 – 3230 Volume II (June 2011)

Journal edited by Economic Informatics Department

DBJOURNAL BOARD

Contact
Calea Dorobanților, no. 15-17, room 2017, Bucharest, Romania
Web: http://dbjournal.ro/
E-mail: editor@dbjournal.ro

Contents

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.

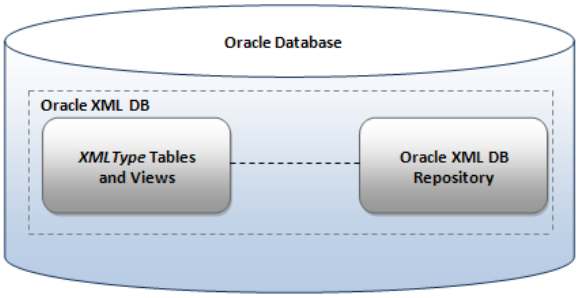

Figure 1 –Main components of Oracle XML DB
(Source: adapted from [13])

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:

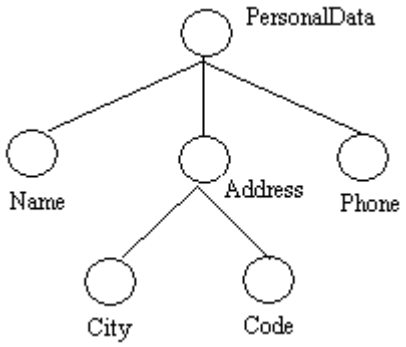

Figure 2 – XML hierarchy for personal data

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>'));
```

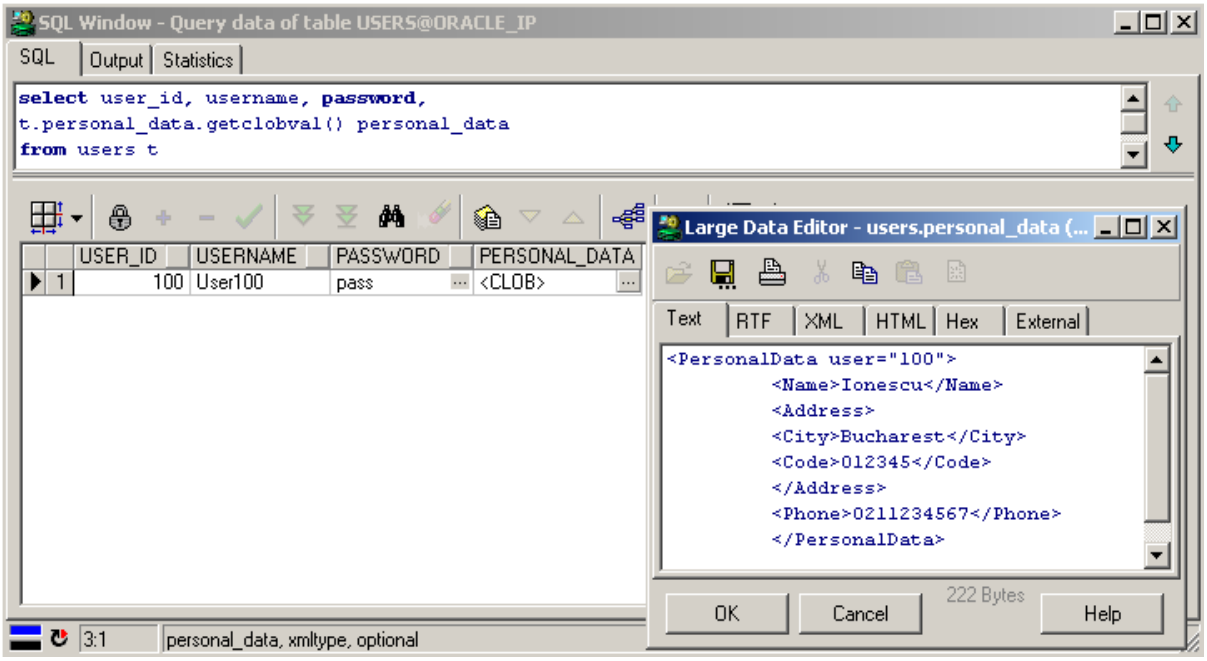

Figure 3 – The modality to visualize the XML data

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.

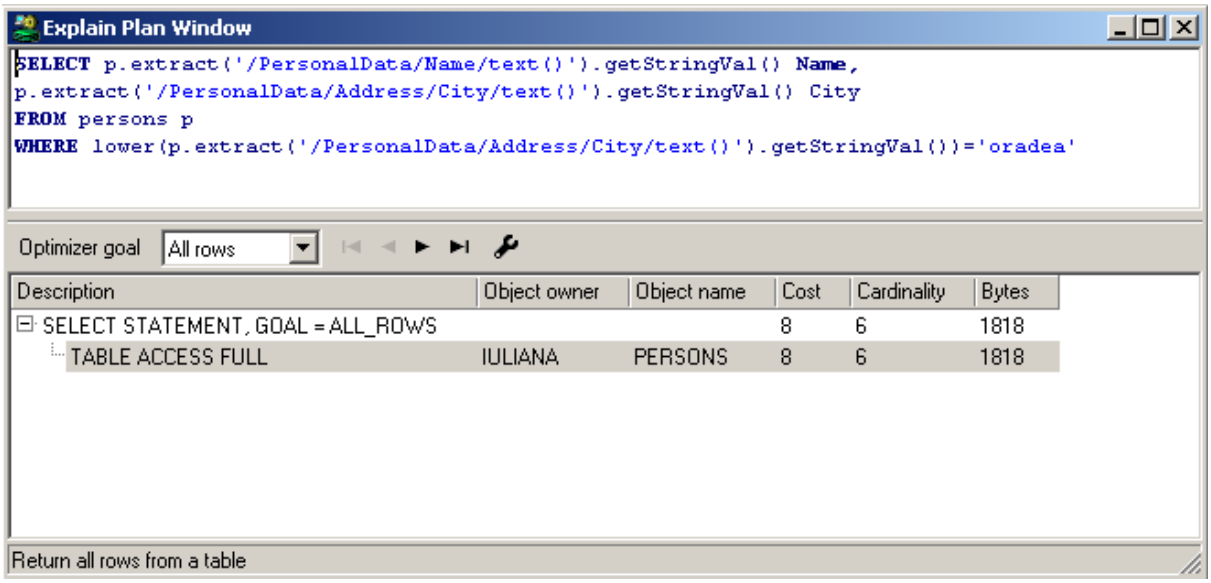

Figure 5 – The execution plan before creating the index

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.

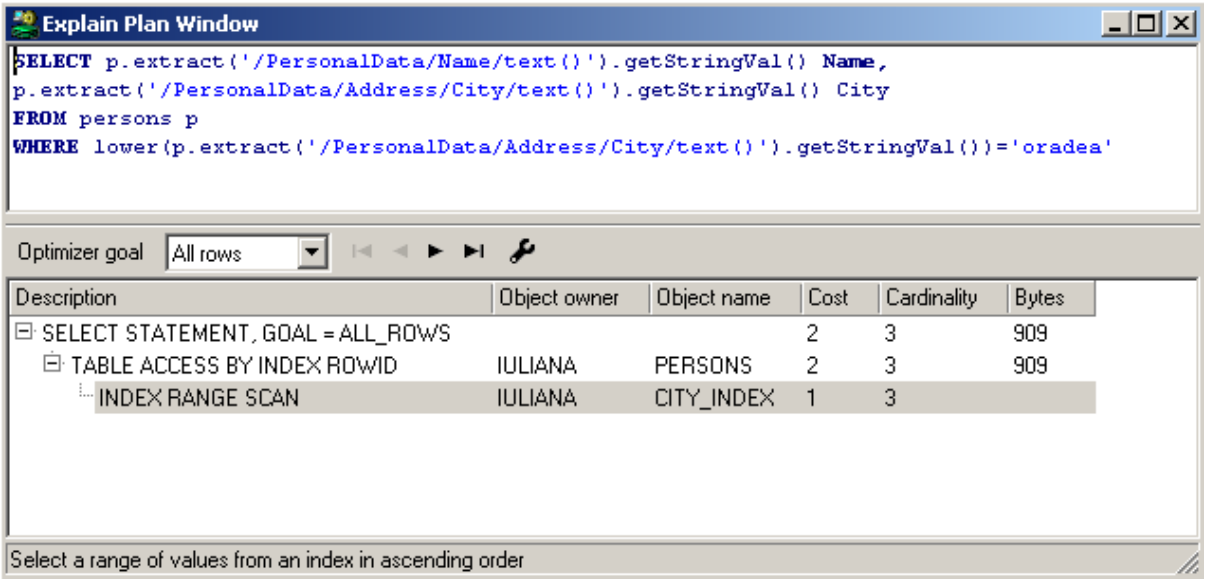

Figure 6 – The execution plan after creating the index

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.

Increasing Database Performance using Indexes

Cecilia CIOLOCA, Mihai GEORGESCU
Economic Informatics Department, Academy of Economic Studies
Bucharest, ROMANIA
cecilia_cioloca@yahoo.com, mihai.georgescu@europe.com

The performance issues are presented. The process of performance tuning is described. The indexing concept is introduced. The two types of defining an index are identified. The paper presents and explains the way a clustered and nonclustered index works. A compared analysis between clustered and nonclustered indexes is created. The benefits of using indexes on databases are identified and explained. A demonstration of using SEEK and SCAN indexes is presented, showing the logic behind these two types of indexes. Cases when an index must be used and how this index is created, are presented along with the performance improvement that is obtained after successfully creating the index.
Keywords: *Clustered index, Nonclustered index, Optimization, Database, Performance*

1 Introduction

Performance is one of the most important metric that describes if a project is a success or a mistake. It is also one of the most common problems the programmers are dealing with. Either if we are taking into consideration a new starting project or an application that is already running on production we should always keep in mind the performance aspects. This means that the design for performance process should start early in the development of an application. The architecture of the system should be design in a manner to meet the performance requirements and to allow performance tuning. The book [1] presents techniques to improve or fine tune a database to achieve maximum performance.

There is no recipe of designing perfect databases, but there are techniques and tips that can improve the quality of the design. Improving the database performance is a cycling activity that should be including in each of development stage.

The performance tuning process includes three steps:

- Time response measurement before tuning;
- Tuning performed;
- Time response measurement after tuning.

The database designer should focus on those techniques that provide the most benefits. Among all the techniques of improving the database performance, indexing and query optimization stand up as they provide visible results. On the other hand, abusing indexes and inappropriate indexes might harm the performance of the system.

The structure of the database used for this demonstration is described in figure 1.

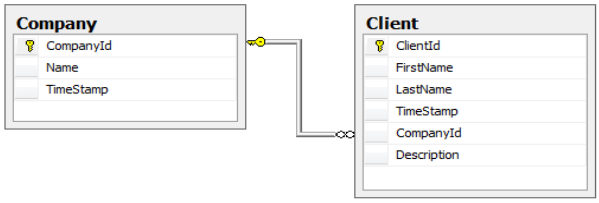

Fig. 1 – Structure of test database

In order to define the index concept we will take a look at how the data is retrieved from database with no indexes.

```
SELECT * FROM Client WHERE
LastName = 'LastName4'
```

In the above query the SQL Server will look in all the rows from the beginning of the table to the end of it searching for those rows that are meeting the condition in the

WHERE clause. If the searched table contains few rows the response of the above query might be very prompt, but in case of tables that contain millions of rows the query might take a while. For this reason creating an index on the table allows SQL Server to get the result without searching the whole data in the table. Indexing is the best way to reduce the logical read and disk input/output as it provides a mechanism of structured search instead of row by row search [8].
Basically an index is a copy of the data in the table, sorted in a certain logical manner. There are two different ways of defining an index:

- *Like a dictionary*: a dictionary is a list of words ordered alphabetically. An index defined like a dictionary is a set of data ordered in a lexicographic manner. For this reason the search on the index will not include all the rows but will position easier based on the ordered data.
- *Like a book index*: this approach of creating an index will not alter the layout of the data in the table, but just like the index of a book will position the data in the table to the corresponding position in the table. An index defined in this manner will contain the data in the indexed column and the corresponding row number of the data.

An example of how to design a spatial database is presented in detail in [3].

2 Clustered indexes

A clustered index is an index that contains the table data ordered in a physical manner. When creating a clustered index on a table, the data rows are reordered on the disk based on the index key sequence so that it meets the indexed ordering. For this reason only one clustered index is allowed to be created on one single table.
For the Client table, when creating a clustered index on *FirstName* column, the data in the table is physically alphabetically sorted based on *FirstName* value. When inserting a new row into the database, it will be inserted in a certain position so that the sorting is still kept. Figure 2 schematically presents the tree structure of the clustered index on *Client* table column *FirstName*.

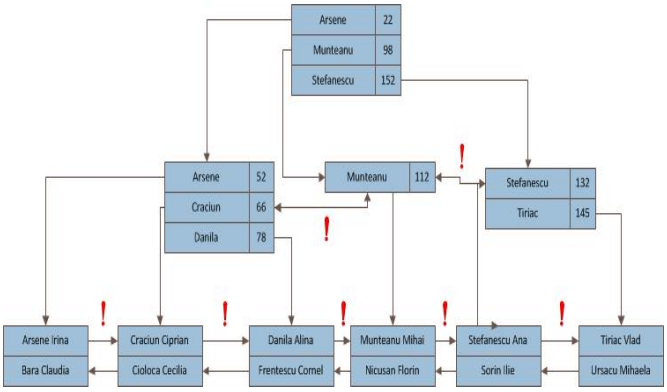

Fig. 2 – The structure of the clustered index

As shown in the figure above the leaf nodes represent the actual data pages while the intermediate nodes of the tree structure are index pages. All the pages in the structure are linked. The top node in the structure is the root index page, while the middle level nodes are intermediate index pages. Each row in an index page refers either another index page or a data page. This reference is a pointer to a page number. The root index page contains a row with the value *Munteanu* which points to the intermediate index page number 98, while the index page 98 contains a row with the value *Danila* which points to the data page 78. The data page 78 contains two rows with the corresponding values.
When searching using the clustered index, the row to row search will be avoided. For the following query

```
SELECT * FROM Client WHERE LastName='Cioloca'
```

SQL Server will first get the root index page from the sysindexes table, and then it will search in the rows of it to find the highest key value not greater then *Cioloca*, and will

find the value *Arsene*. The pointer corresponding to the key value *Arsene* refers to the index page 22. Now, SQL Server will move to index page 22, where it looks again for the highest key value not greater than the searched value. This value is Craciun and it points to the data page 66. SQL Server moves to the data page 66, where it finds the two rows: *Craciun Ciprian* and *Cioloca Cecilia*. It finds the value that meats the search criteria and returns it.

A table that has no clustered index is called heap table. The data in this type of table is not stored in any particular order. Each row in a nonclustered index has a pointer that refers to the location of that particular data row. This pointer is called raw locator, and it can point to a row in the heap or to a clustered index key value, depending if the table has or not a clustered index. If the table has no clustered index then the row located will point to a data row in the heap. In case the table has a clustered index, the row locator will point to a index key value into the clustered index. In the figure 3 the relationship between clustered and nonclustered indexes is presented.

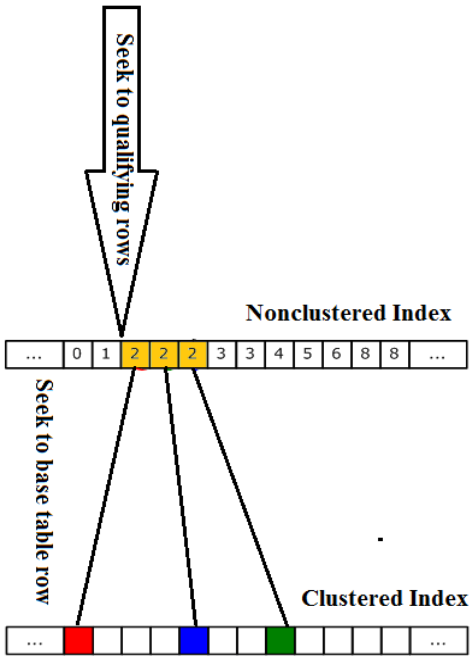

Fig. 3 – Relationship clustered-nonclustered indexes

When using clustered indexes there are few aspects that should be taken into consideration:

- The clustered index should be created prior the nonclustered index. The reason why this is recommended is that the nonclustered index row locators should point to a corresponding key value in the clustered index. If the nonclustered index is created first then the row locators will point into the heap and the relationship between the clustered and nonclustered indexes will be compromised;
- The reordering of the index occurs every time the index is changed. The dependency between the clustered and nonclustered indexes should be kept in mind when changing the clustered index. For this reason the index should be dropped and then recreated so that the nonclustered are rebuilt correctly;
- The clustered index is not recommended to be used when working with columns that are frequently updated as each time the column is updated the clustered and nonclustered indexes must be rebuild;
- The clustered index is efficient when retrieving presorted data and when retrieving range of data.

As recommended in [2], a clustered index should be used when: Consider using a clustered index when the following occur:

• The physical ordering supports the range retrievals of important queries, or equality returns many duplicates;

• The clustered index key is used in the ORDER BY clause or GROUP BY clause of critical queries;

• The clustered index key is used in important joins to relate the tables—that is, it supports the foreign key;

• The clustered index columns are not changed regularly.

3 Nonclustered indexes

Similar to the clustered indexes, nonclustered indexes are balanced tree structures that start from a root index node, and include intermediate index nodes and leaf nodes. The main differences between clustered and nonclustered indexes is that in case of nonclustered indexes the key values are not ordered in any specific way and that the leaf nodes from nonclustered indexes are not data pages but index pages that point to either a index node into a clustered index or to a data row into the heap. Figure 3 describes the tree structure of a nonclustere node. The dotted line suggests that the leaf nodes point to data rows into the heap table.

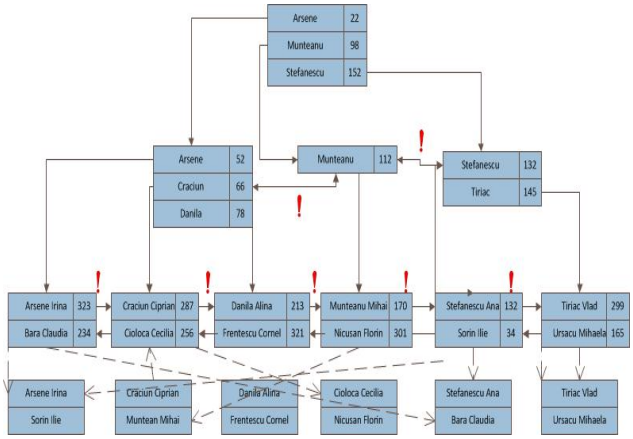

Fig. 3 – Structure of nonclustered index

Due to its properties, a nonclustered index does not physically change the order of the rows into the table. The figure above also shows that each data row is pointed by an index node, which was not the case for clustered index. The nonclustered index is larger than its counterpart, as a supplementary index level is introduced and it has to contain the pointers to the data rows.

Even when the clustered index rows have physically changed the nonclustered row locaters continue to point to the same clustered index nodes. Instead of updating each row locator for each nonclustered index node, SQL Server adds a pointer to the old data page in order to point to the new datapage. This involves a new level of indirection. The leaf index nodes from the nonclustered index still point to the old data page, while these old data pages are pointing to the new ones. This way the user is transferred to the new location of the data pages.

While there can be only one single clustered index for each table, there can be a bunch of nonclustered indexes on the same table. The reason of that is that this type of index does not change the physical order of the data into the table.

When using nonclustered indexes it is useful to keep in mind the following aspects:

- Nonclustered indexes might be find very usefull when retrieving a small set of data from a large amount of data. The nonclustered indexes is also recommended when using wide keys and when the table columns are frequently updated;
- Nonclustered indexes are not the best when retrieving large sets of data, as by using this type of index there still must be read a large amount of data pages. In this case the use of clustered indexes is recommended.

Using noncluster indexes should be used when:

- Once or more rows will be retrieved—that is, the query is highly selective.
- The nonclustered index key is used in the ORDER BY clause or GROUP BY clause of critical queries.
- The nonclustered index key is used in important joins to relate the tables.
- A covered query is required.

4 Index SEEK versus Index SCAN

In order to evaluate the performance of an sql query, one of the most common tool is SQL Server trace which provides information about the "logical reads" counter.

As shown in [7], the logical and physical reads are explained: "*The I/O from an instance of SQL Server is divided into logical and physical I/O. A logical read*

occurs every time the database engine requests a page from the buffer cache. If the page is not currently in the buffer cache, a physical read is then performed to read the page into the buffer cache. If the page is currently in the cache, no physical read is generated; the buffer cache simply uses the page already in memory."

Physical and logical reads are available in SQL Server 2008 by using the following command:

```
SET STATISTICS IO ON
```

After successfully running this command, all future queries will show the operations performed.

First we will create a nonclustered index on column *LastName*:

```
CREATE NONCLUSTERED INDEX
LastNameIndex on Client(LastName)
```

By using the system procedure *sp_helpindex* we can check what indexes are defined on a table:

```
sp_helpindex Client
```

In figure 4, we can see that we have 2 indexes defined on our Client table, PK_Client index is generated automatically when we specified the primary key for the table.

	index_name	index_description	index_keys
1	LastNameIndex	nonclustered located on PRIMARY	LastName
2	PK_Client	clustered, unique, primary key located on PRIMARY	ClientId

Fig. 4 – Current indexes on table Client

To understand how index seek and index scan works, the execution plan is turned on and the following queries are executed:

Query 1:

```
select * from Client where
LastName = 'LastName4'
```

Query 2:

```
select * from Client where
FirstName = 'FirstName4'
```

We observe for the first query that SQL Server requested a page from the buffer cache only 15 times. Also in figure 5, by examining the execution plan, the database engine used an Index Seek to fetch the information that was already sorted using the LastNameIndex that we created earlier.

The second query used an Index Scan which means that the entire Client table was scanned to find the Clients that have *LastName* = '*LastName4*' and requested data from cache 19458 times, a lot more than the first query which used the index.

Another information that the execution plan provides is the query cost. For the first query is almost 0% compared to the second query which scanned the entire table.

Query 1:

```
(4 row(s) affected)
Table 'Client'. Scan count 1, logical reads
15, physical reads 0, read-ahead reads 0, lob
logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
```

Query 2:

```
(4 row(s) affected)
Table 'Client'. Scan count 1, logical reads
19458, physical reads 0, read-ahead reads 0,
lob logical reads 0, lob physical reads 0,
lob read-ahead reads 0.
```

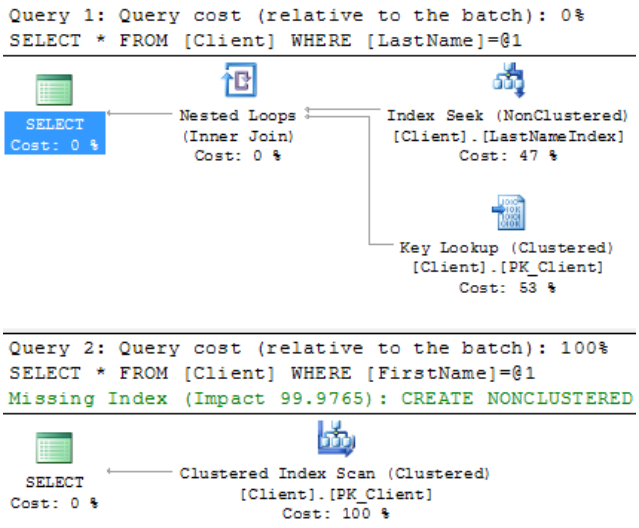

Fig. 5 – Execution plan of Query 1 and Query 2

An index scan either clustered or nonclustered will do a complete scan in a

clustered or nonclustered index. Neither index uses the b-tree structure of the index but only reads the leaf pages in order using each page's reference to the next in the chain.
An index seek is a seek through the b-tree structure of a nonclustered index, from the root down to the leaf. A clustered index seek is looking in the structure of a clustered index from the root down to the leaf.
Index seek is preferred for the highly selective queries, when the query is returning a fewer number of rows, less than 15% of total rows in the table.
If there is an index on a table and if the query is fetching a large amount of data, more than 70% of total rows in a table, then the database engine will scan all the data pages to retrieve the data rows.

5 Clustered indexes compared to nonclustered indexs

Clustered and Non Clustered indexes have the following common characteristics as presented in [5]:
Both clustered and nonclustered indexes are created based on one or more columns. The columns are put in the index in the order you specify them in the CREATE INDEX statement. They are also sorted in this order as well. Indexes are first sorted on the first column in the index, then any duplicates of the first column and sorted by the second column, etc;
Neither clustered or nonclustered indexes will guarantee the sort order of the data when it is being returned to you. If the order of the data matters to you, you should always sort the data with the ORDER BY clause in your select statement.
Both clustered indexes and nonclustered indexes take up additional disk space. The amount of space that they require will depend on the columns in the index, and the number of rows in the table. The clustered index will also grow as you add columns to the table;
Adding indexes (both clustered and nonclusterd) will increase the amount of execution time that the INSERT, UPDATE and DELETE statement take, as the data has to be updated in the table an also in each index;
Columns of the IMAGE data type cannot be indexed using normal indexes;

Clustered Indexes

SQL Server supports only one clustered index for each database table. This index should be the most common column that is in the WHERE clause for queries against this table. In most cases the search is by primary key and this index is automatically created.
The reordering of the data occurs every time the index changes;
Reorders the way records are physically stored;

Nonclustered Indexes

Nonclustered indexes, also called "indexes" are the normal indexes that are created on your tables. SQL Server supports 999 nonclustered indexes per table.
Each nonclustered index can contain upto 1023 included columns. Columns of the TEXT, NTEXT and IMAGE data types cannot be included either.
Nonclustered indexes are not copies of the table but a sorting of the columns that reffer to the data pages in the clustered index. Because of this, the clustered index affects all other indexes.
There are two modes for the nonclustered index: non-unique and unique.The non-unique type specifies that the index allows identical rows to be inserted, compared to the unique type which does not allow any identical rows.
As presented in [6], the nonclustered index does not re-order the actual table data. If the clustered index is missing, then the nonclustered index will point to the actual data in table.
Logical order of the index is not identical with the physical order of the rows stored on disk.

Nonclustered indexes should be used for queries that return small amount of data and to columns used in WHERE clauses that return exact matches. Large result sets are reading more tables pages anyway so they will not benefit from a nonclustered index.

6 Index fragmentation

When the user performs operations like INSERT, UPDATE or DELETE on the database, table fragmentation may occur. Also when changes affect the data that is covered by the index, then index fragmentation occurs and the information in the index is scattered over the database. Due to this, when a query is performed against a heavy fragmented table, the operation will take longer time.

Index fragmentation comes in two different forms: external fragmentation and internal fragmentation. In each of these forms, the pages within the index are used inefficiently. External fragmentation means that the logical order of the pages are wrong and internal fragmentation represents that the amount of data stored within each page is less than the data page.

Each leaf page of an index must be in a logical order otherwise external fragmentation occurs. When an index is created, the index keys are sorted in a logical order on a set of index pages. Each time new data is inserted in the index, there could be the possibility that the new keys are inserted between existing keys. This may lead to the creation of new index pages to accommodate the existing keys that were moved so that new keys can be inserted in correct order.

Let's assume the current index structure presented in figure 6.

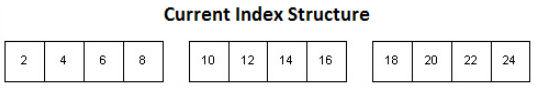

Fig. 6 – Current index structure

When an INSERT statement is executed, new data is added to the index. In our example we will add 5 and automatically a new page will be created and the 7 and 8 will be moved to the new page in order to make room for the 5 on the original page. Due to this, the index will be out of logical order as seen in figure 7.

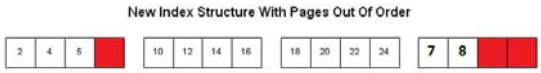

Fig. 7 – New index structure

This type of fragmentation will affect the performance of queries that do not have specific searches or that return unordered result sets, but will affect queries that returned ordered sets. An example of an ordered result set is a query that is returning everything from page 4 to 12. This query has to complete an extra page switch in order to return the 7 and 8 pages. If the fragmentation affects tables with hundreds of pages the amount of extra page switches will be significantly greater.

In order to determine the level of fragmentation the following command can be used:

```
DBCC SHOWCONTIG (TableName)
```

The syntax of this command is:

```
DBCC SHOWCONTIG
[({ table_name
    | table_id
    | view_name
    | view_id }
    [ , index_name | index_id ]
   )
]
[ WITH { ALL_INDEXES
| FAST [ , ALL_INDEXES ]
| TABLERESULTS [ , { ALL_INDEXES } ]
[ , { FAST | ALL_LEVELS } ]
    }
]
```

After running this command on a table or view, the results are presented below:

```
DBCC SHOWCONTIG scanning 'Person' table...
Table: 'Person' (839842950); index ID: 1,
database ID: 5
```

```
TABLE level scan performed.
- Pages Scanned.................: 53600
- Extents Scanned...............: 6712
- Extent Switches...............: 8579
- Avg. Pages per Extent.........: 8.0
- Scan Density [Best Count:Actual
Count].......: 78.09% [6700:8580]
- Logical Scan Fragmentation .....: 5.31%
- Extent Scan Fragmentation ......: 43.19%
- Avg. Bytes Free per Page........: 753.2
- Avg. Page Density (full)........: 90.69%
DBCC execution completed. If DBCC printed
error messages, contact your system
administrator.
```

The command returns the number of pages scanned, the number of extents scanned the number of times the DBCC statement moved from one extent to another while parsing the pages of the table or index, the average number of pages per extent, the scan density.

Pages Scanned: If the number of rows contained in the table or index divided by the approximate row size is significantly greater than the number of pages scanned then there could be internal fragmentation of the index.
Extents Scanned: Take the number of pages scanned and divide that number by 8, rounded to the next highest interval. This figure should match the number of extents scanned returned by DBCC SHOWCONTIG. If the number returned by DBCC SHOWCONTIG is higher, then you have some external fragmentation. The seriousness of the fragmentation depends on just how high the shown value is from the estimated value.
Extent Switches: This number should be equal to (Extents Scanned – 1). Higher numbers indicate external fragmentation.
Extent Scan Fragmentation: Shows any gaps between extents. This percentage should be 0% and higher percentages indicate external fragmentation.

By analyzing the results provided by DBCC SHOWCINTIG on our "Person" table we can see that the number of extent switches is much greater than the number of extents scanned. In this case there is external fragmentation.

The fragmentation can be reduced and the performance improved by using the following methods:

- Dropping and recreating the index;
- Rebuilding the index using the DBCC DBREINDEX statement;
- Defragmenting an index by using the DBCC INDEXDEFRAG statement.

Dropping and rebuilding an index has the advantage of completely rebuilding an index and does reorder the index pages, compacting the pages, and dropping any unneeded pages. This operation should be done on indexes that show high levels of both internal and external fragmentation.

DROP INDEX and CREATE INDEX

Rebuilding the index can reduce fragmentation and it is done by using the following statement:

DBCC DBREINDEX

This operation is similar to dropping and creating the index, except that it will rebuild the index physically allowing the SQL Server to assign new pages to the index and reduce both internal and external fragmentation. This statement also recreates indexes with existing constraints.
Defragmenting an index by using the DBCC INDEXDEFRAG statement reduces the external fragmentation by rearranging the existing leaf pages of an index to the logical order of the index key and internal fragmentation by compacting the rows within index pages then discarding unnecessary pages. The time needed to execute this statement is longer than recreating an index if the amount of fragmentation is high. DBCC INDEXDEFRAG can defragment an index while other processes are accessing the index, eliminating the blocking restrictions.

7 Conclusions

Using indexes on the database improves the performance of the whole system. These indexes are of different type and should be used in an intelligent way in order to achieve the whished behavior. Clustered indexes should be used for queries that return a large amount of data. The clustered index is recommended to be created on the primary key of the table. For all other columns nonclustered indexes should be created.
A noncluster index is better to be used when running queries that return a smaller amount of data. By using this type of indexes large amount of data will have to read in lot of data pages, and the performance of the system will decrease.
Choosing the number and type of indexes can dramatically affect the performance and can mean the difference between data being retrieved in seconds, with few disk I/Os or minutes, even hours, with many disk I/Os. Choosing the optimum number of indexes to support the critical queries is therefore an extremely important task.
Before creating a new index one must make sure that there aren't any unused indexes. Also remove the minimally useful or redundant/duplicate indexes like subsets of other indexes.
Combine existing indexes to create a consolidated index:

Index1:
```
(LastName, FirstName) INCLUDE (phone)
```

Index2:
```
(LastName, FirstName, MiddleName) INCLUDE (PIN)
```

Combined Index:
```
(LastName, FirstName, MiddleName) INCLUDE (phone, PIN)
```

Verify the health of the existing indexes periodically for:

- Splits using sys.dm_db_index_operational_stats
- Fragmentation using sys.dm_db_index_physical_stats
- Contention using sys.dm_db_index_operational_stats

The database performance can be improved by reducing the table or index fragmentation. This is achieved by dropping and recreating the index, rebuild the index by using DBCC DBREINDEX statement or by defragmenting the index using DBCC INDEXDEFRAG statement.

8 Acknowledgment

This work was cofinaced from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POSDRU/107/1.5/S/77213 „Ph.D. for a career in interdisciplinary economic research at the European standards".

References

[1] G. Fritchey, S. Dam, *SQL Server 2008 Query Performance Tuning Distilled*, Apress, 2009, ISBN 978-1-4302-1903-3, 560 pages
[2] K. England, *Microsoft SQL Server 2000 Performance Optimisation and Tuning Handbook*, Digital Press, 2001, ISBN 978-1555582418, 320 pages
[3] A. Mocanu, M. Velicanu, *Building a Spatial Database for Romanian Archaeological Sites*, Database Systems Journal, Vol. II, ISSN: 2069 – 323, p. 3-12.
[4] http://www.sql-server-performance.com/2010/logical-reads/
[5] http://itknowledgeexchange.techtarget.com/sql-server/back-to-basics-clustered-vs-nonclustered-indexes-whats-the-difference/
[6] http://www.devtoolshed.com/content/clustered-index-vs-non-clustered-index-sql-server
[7] http://blogs.msdn.com/b/craigfr/ archiar/2006/06/30/652639.aspx
[8] C. Churcher, *Beginning Database Design. From novice to professional*, Appress, 2007, ISBN 1-59059- 69- 9, 267 pages

Cecilia CIOLOCA has graduated the Faculty of Economic Cybernetics, Statistics and Informatics from the Bucharest Academy of Economic Studies in 2008. She is currently conducting doctoral research in Economic Informatics at the Academy of Economic Studies. She is experienced in software development using open source technologies. She has successfully completed several projects using various programming languages and open source frameworks. She is proficient in using Java Enterprise Edition.

Mihai GEORGESCU has graduated the Faculty of Electronics, Telecommunications and Information Technology – Politehnica University Bucharest in 2005 and also completed a master program at the Faculty of Economic Cybernetics, Statistics and Informatics from the Bucharest Academy of Economic Studies in 2007. He has experience in the development of software products using various programming languages like C/C++, C# both Windows Forms and Web.

A Grid Architecture for Manufacturing Database System

Laurentiu CIOVICĂ, Constantin Daniel AVRAM
Economic Informatics Department, Academy of Economic Studies Bucharest, ROMANIA
laurentiu.ciovica@gmail.com, costin.avram@gmail.com

Before the Enterprise Resource Planning concepts business functions within enterprises were supported by small and isolated applications, most of them developed internally. Yet today ERP platforms are not by themselves the answer to all organizations needs especially in times of differentiated and diversified demands among end customers. ERP platforms were integrated with specialized systems for the management of clients, Customer Relationship Management and vendors, Supplier Relationship Management. They were integrated with Manufacturing Execution Systems for better planning and control of production lines. In order to offer real time, efficient answers to the management level, ERP systems were integrated with Business Intelligence systems. This paper analyses the advantages of grid computing at this level of integration, communication and interoperability between complex specialized informatics systems with a focus on the system architecture and data base systems.
Keywords: *enterprise resource planning, architecture, grid computing, data base systems.*

1 Introduction

The first attempts to computerize the business functions within an enterprise have been materialized in the 60s, in small and isolated applications, developed internally. These applications were specifically designed to provide support for financial-accounting operations and for stock management in order to streamline the production process.

Due to rapid changes inside companies and to new requirements, existing software products continuously needed adaptive and progressive maintenance. Their advantages allowed expansion in other business areas, where they interacted with other applications even though these where not designed to interact and work together. The evolution process continued and the software products required more maintenance, expansion of functionalities and integration. Application packages have emerged being added to existing software collections. On this background, in the late '80s the ERP - Enterprise Resource Planning - systems were born.

The offered solutions were integrated into a single package. The request of such software products was huge and soon the ERP's were everywhere. In [1] and [2] are described the most important advantages of ERP systems. With a single and centralized database system the organization has access to **high quality information** obtained from consistent, accurate and elementary data. Databases' normalization within ERP systems and the usage of a single and central database, **minimize data redundancy**.

The ERP systems provide **small response times**. They can be easily reconfigurable and they allow adaptation to permanent changes in today's volatile economic environment. Process flows, applications, solutions and functional modules are integrated in the EPR system.

Being module oriented the systems allow to add new functionalities and their integration and configuration are possible with acceptable efforts. This gives to ERP systems the advantage of **scalability.** The complexity of ERP applications but also the need of a continuous improvement led to the integration of the maintenance service, offered by suppliers, inside the development process. The result is an **improved maintenance and support system**.

The evolution of ERP products to "ERP-extended" concept, gives them a **collaborative dimension** by integrating applications such as Advanced Planning and Scheduling - APS, Customer Relationship Management - CRM, Supply Chain Management - SCM, Product Lifecycle Management - PLM, Supplier Relationship Management - SRM.

Inside manufacturing organizations the ERP systems are integrated with the production systems, known as Manufacturing Execution Systems. The objective is to better plan and control the production unit.

In today's ERP systems there is a MRP, Material Requirement Planning module that is managing the production cycle, but is not a production dedicated systems.

Manufacturing Resource Planning systems represents the foundation of today's ERP systems. As described in **Fig. 1**, the development of MRP and ERP systems started almost in the same time.

There have also been developed and integrated decision support applications at the top level management known as Business Intelligence, BI. Through the development and integration of web-oriented components, the ERP systems offer the advantage of openness to e-business. This level of integration and communication is possible because of Service Oriented Architectures, SOA, which are focused on the entire enterprise life cycle.

The concept is called Enterprise Service Oriented Architecture or simpler ESA, Enterprise Service Architecture. Enterprise Service Oriented Architectures allow manufacturing organizations to optimize the production lines and to develop new products personalized and adapted to their customers' needs.

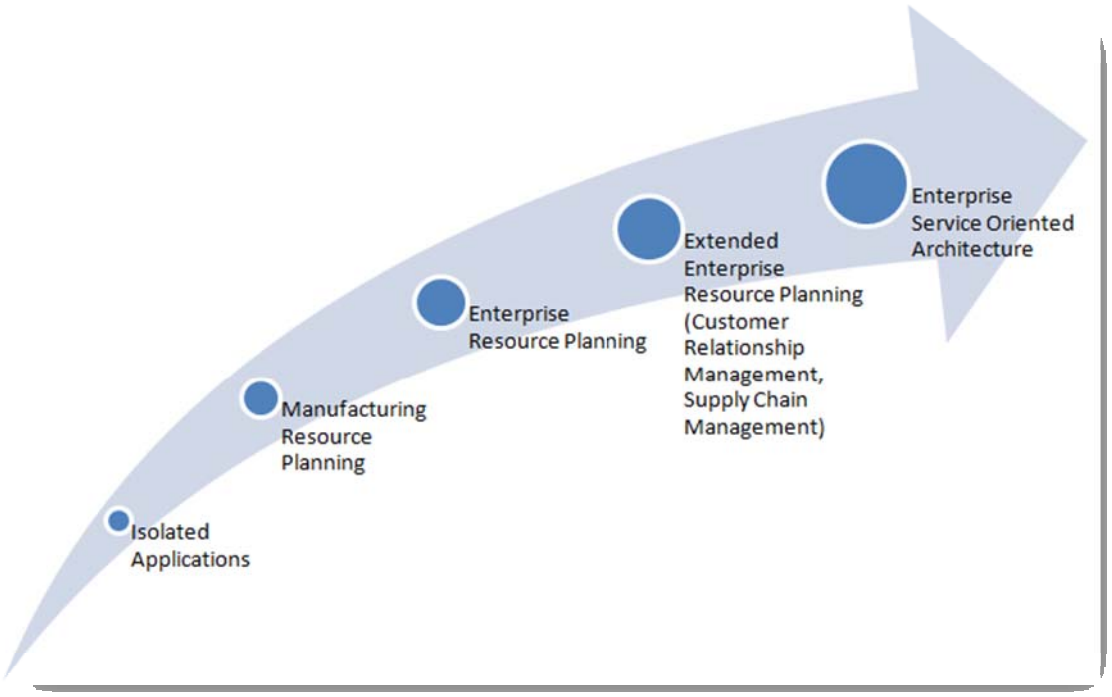

Fig. 1. Enterprise software evolution

According to **Fig. 1,** standard, isolated software applications that were using the power of mainframe computers were integrated in unique software systems for better communication and interoperability, giving life to the ERP systems.

The ERP systems became an imperative requirement especially for the manufacturing enterprises were specialized

systems for production management were being developed - Manufacturing Resource Planning. This phase of software integration in single platforms was supported from the technical point of view by client-server architectures as described in **Fig. 2**. The evolution of software integration didn't stop here as ERP concept was evolved in extended ERP by integrating it with specialized systems for the management of customers and vendors in order to optimize even more the product life cycle management by eliminating delays caused by stock failure or raw material failure.

The final stage of software integration was born thanks to web service technologies and is represented by Service Oriented Architectures and Enterprise Service Oriented Architectures.

Grid computing and cloud computing technologies are already empowering these architectures in ways that will allow production industries to become capable of real-time equipments configuration and transformation in order to execute differentiated and personalized products.

2 Grid computing inside ERP architecture

The ERP systems are transactional applications, client-server and distributed. Servers are often centralized, but the customers are spread throughout the organization. Multinational organizations consolidate their servers into a single point, e.g. the headquarters of the organization, and can access this point from around the world. A general ERP architecture includes three functional entities. The database component is the central repository of transactional data both within and outside the organization. The second component consists in the applications client from where queries on database are initiated and where the processed results from these queries are sent. The third component is an intermediate zone between clients and database and is represented by the ERP internal applications. The ERP systems can be implemented as a two-tier architecture meaning that there is a single level which handles and manages the database and the applications, or a three-tier architecture can be implemented where, in addition to the client level, there are two more levels, one for database and a separate one for applications. Inside two-tier architecture, the database server and applications server are physically installed on the same equipment but remain logically different entities. **Fig. 2** describes three-tire architecture.

The techniques of grid computing are a higher growing concern in the informatics systems world and especially in the case of integrated ERP systems. In [3] we find an ERP architecture based on the model Open Grid Service Architecture, OGSA. The author states that this architecture solves the inefficiency problems of sharing the distributed resources through the benefits of the OGSA model, through the web services, by using the grid computing techniques and through the use of XML and SOAP protocols, in order to integrate core business services of the organization.

The grid computing techniques are found in the components of ERP systems and also at the level of ERP platform itself. The databases used by ERP systems are scalable and allow the storage and retrieval of any type of data in a grid environment. Data are saved in a single virtual group that is shared and made available depending of the competing demands of users, ensuring also the optimization of memory used and avoiding locking the requests between them. Databases are distributed in several physical servers and grouped at logical level into clusters, providing scalability without affecting applications that use the database system. The organization of database servers into clusters enables the possibility to extend the server capacities, by adding new nodes in high demand areas and eliminates the risk of a single point of failure. User applications have permanent access at the fair and efficient level of resources and computing

power through management functions in terms of work load. Not only can the database of ERP systems be distributed on multiple physical servers, but also the application level.

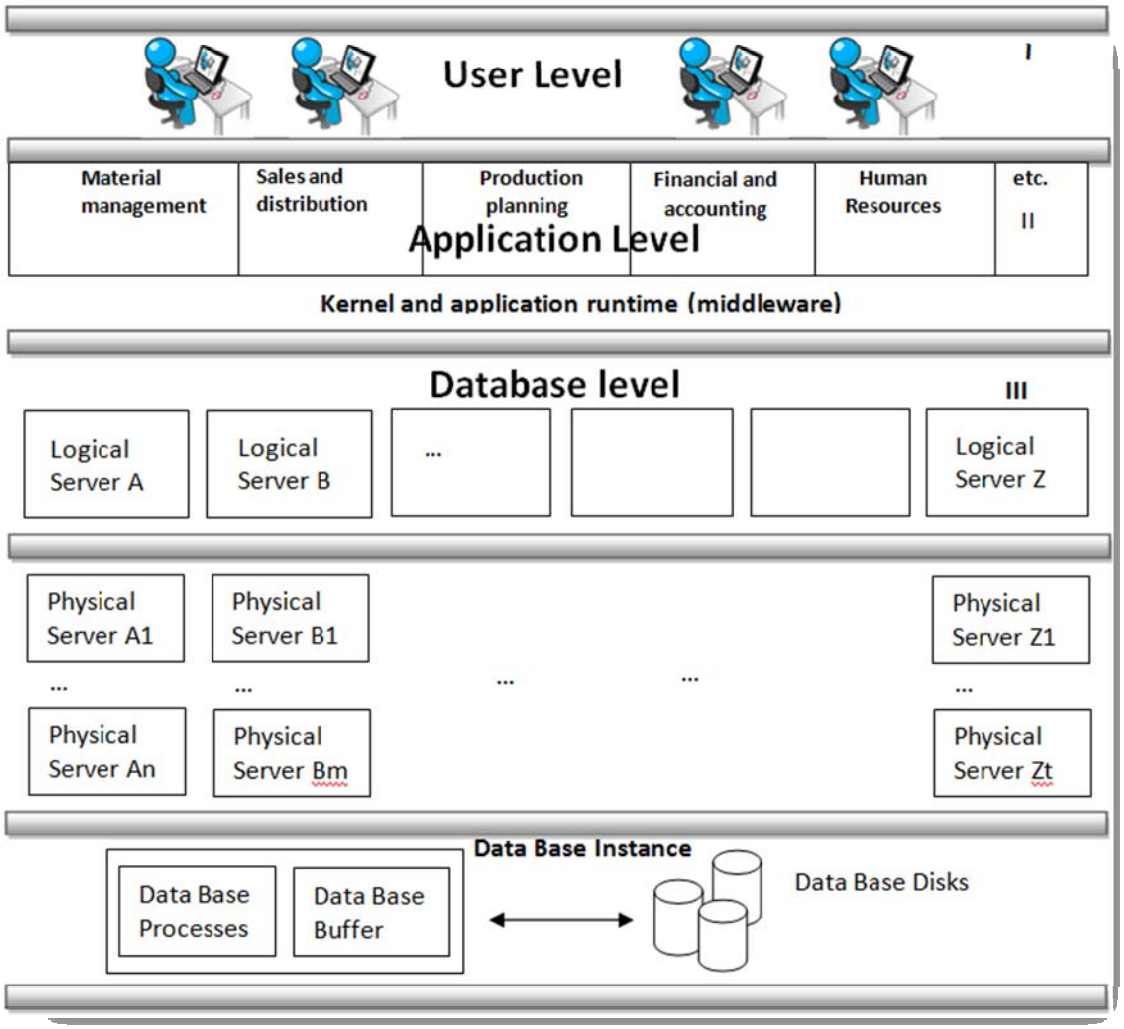

Fig. 2 ERP Architectures

The application server is accessed by the end user as a single point, but planning systems and optimizing of end-user access to ERP system resources, optimally redirects the requests on multiple applications servers. At the programming level, it can be transmitted on which application server will run a particular query. In retrieving data from ERP databases, the authors experienced interfaces that extracted in a work session tens of millions of records in order to transfer them to data warehouses. The interfaces were running during the night, but in many cases, the amount of data was so big that twelve hours were not sufficient to complete the execution, which was unacceptable because during the day the system should have been as free as possible for the key users. The problem was solved by applying grid computing techniques. Assessments and estimations of the volume of data were made and regarding their homogeneity, so that the final result to be divided into partial results.

The advantage of this technique allowed the execution of multiple partial jobs, each one using the resources of a different application server. In **Fig. 2** the existence of an intermediate level between the database

server and the applications server can be observed.
Due to the database distribution and of the applications on multiple servers, an intermediate level is necessary to ensure optimal communication between the two levels. The end user is not involved in this phase. His requests, launched from clients all over the world, are taken over and managed with maximum efficiency by the grid architecture in a non transparent way for the end user.

3 A grid software architecture for manufacturing organizations

The authors think that customers today are more and more oriented on personalized products and they want to participate in an active way during the phase of product design.
In particular, fashion industry shows an increased interest in customers' opinion and the trend is to design and develop personalized fashion products matching perfectly clients' demands.
Personalized demands can be limited to some characteristics such as measure, or can even include personalized product design.
Pomarfin Company, a Finland leading manufacturer for casual footwear is known for its "left foot" concept. The idea is to have 3D scanners available inside their stores in order to obtain the perfect measure for each customer. Clients will choose the model and thanks to the 3D scanners the shoe model they like most will come in perfect size, especially for them.
This paper is extending the "left foot" concept by assuming that very soon 3D scanners will be a technology available for everybody.
Customers will design, from their home, the perfect show model, they will obtain their exactly measure using personal 3D scanners and they will order via internet their products. Software architecture for the development and implementation of such business model is described in **Fig. 3.**
It's a grid architecture because different software components with dedicated data base systems are working together to serve a specific customer request. The grid is controlled by a central element, a Service Proxy containing:

- a message service used for sending messages between the production grid components
- a web service mediation used to convert different messages from different components into a general language; it can be associated with a common protocol or a communication standard
- an enterprise service bus used to receive data from third parties components and systems using different types of protocols

The grid architecture contains a web portal which helps end customers to design their personalized products and place the order inside the manufacturing system.
It contains ERP systems with all its components and applications. The data base system used for the ERP system is using a grid model as described in **Fig. 2**.
The ERP system is integrated with Customer Relationship Management Systems and Supplier Relationship Management Systems. Different data from different data base systems are grouped in data warehouses.
The ERP system in connected to the production systems used at shop floor lever. Between the two systems there is a middleware component, the manufacturing execution system.
As described in [4] the manufacturing execution system is responsible with the management of production. It represents a communication layer between the ERP as transactional system and the systems dedicated to production lines.
The Business Process Management inside our grid architecture enables the organization to define executable processes using visual flow steps.
It allows process adaptation to new market requirements and ensures a competitive

advantage for the enterprise.
This component is essential for our grid architecture as the main purpose of this platform is to offer real time transformation and configuration of the business model due to the products' differentiation.
The Enterprise Decision Management component represents a tool for managing business rules which management of the business model inside the software architecture.

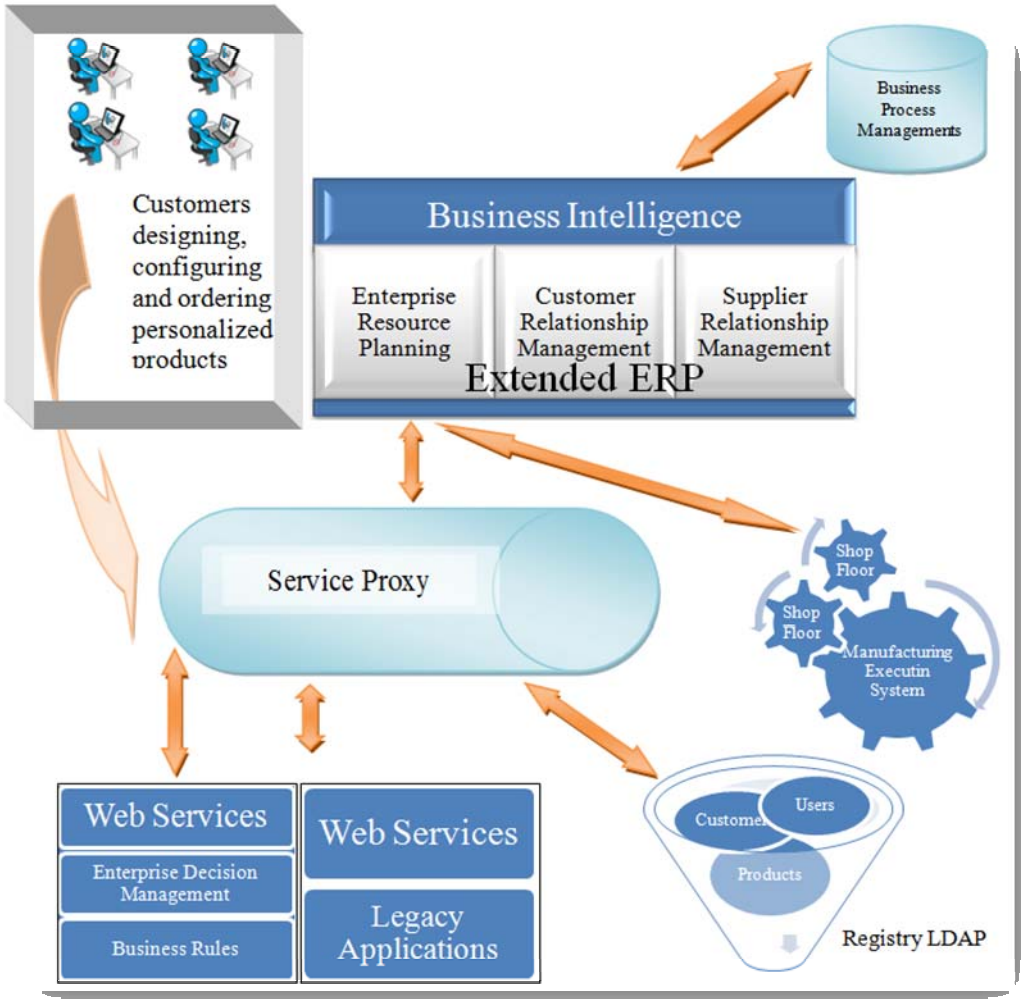

Fig. 3 A grid software architecture for manufacturing organizations

A detailed analysis of the database systems must be undertaken considering the complexity and the number of specialized software applications integrated in a single software architecture, in order to allow more interoperability.

Each component described in **Fig. 4** works with a specific data base model and a database system must be designed for the entire architecture in order to ensure its stability and efficiency.

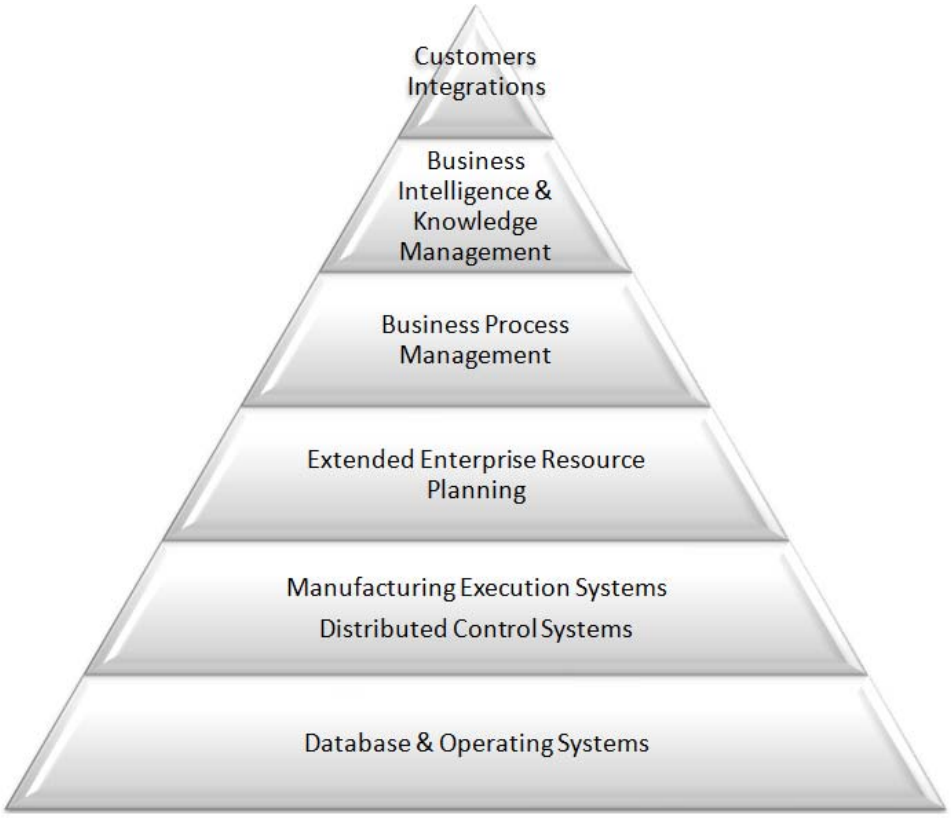

Fig. 4 Components of grid software architecture for manufacturing organizations

4 Data Base Systems Architecture – An overall view

As mentioned earlier in article, the customers will order their desired product through a web application capable in interacting with 3D scanners.

Beside the scanning functionality the application will expose functionalities like: getting all available products along with their models directly from manufacturers, getting all applicable colors, shapes, sizes, etc – basically everything that can be customized and is supported by the manufacturer, the possibility to see the order status in every moment, the possibility to assign notifications channels through which the system will notify the user about any problems that may occur with their order, etc.

The user will interact with a highly usable interface through which in matter of minutes he can choose and customize his desired product and place the order.

From architectural point of view the application itself interacts with third-party applications, services, systems in a transparent manner for the user.

Basically the portal – the web application with which the user interacts – is just a client for the main application server which coordinates the entire order flow and interacts with all necessary third-parties systems.

From database system architectural point of view the system communicates with multiple database servers, as shown in **Fig. 5** – one server per a major functionality domain: e.g customer's database server, product information system database server, orders database server, etc.

In order to handle the huge number of customers the database servers are distributed and a load balancing router is implemented.

In order to avoid data looses and corruption the distributed systems are in sync by using native DBMS replication and specialized data sync applications.

As shown in **Fig. 5** the application server uses the CDB – Customers DB, ODB –

Orders DB and PDB – Product Information System DB as primary data sources.

These sources are shared within the E-ERP itself.

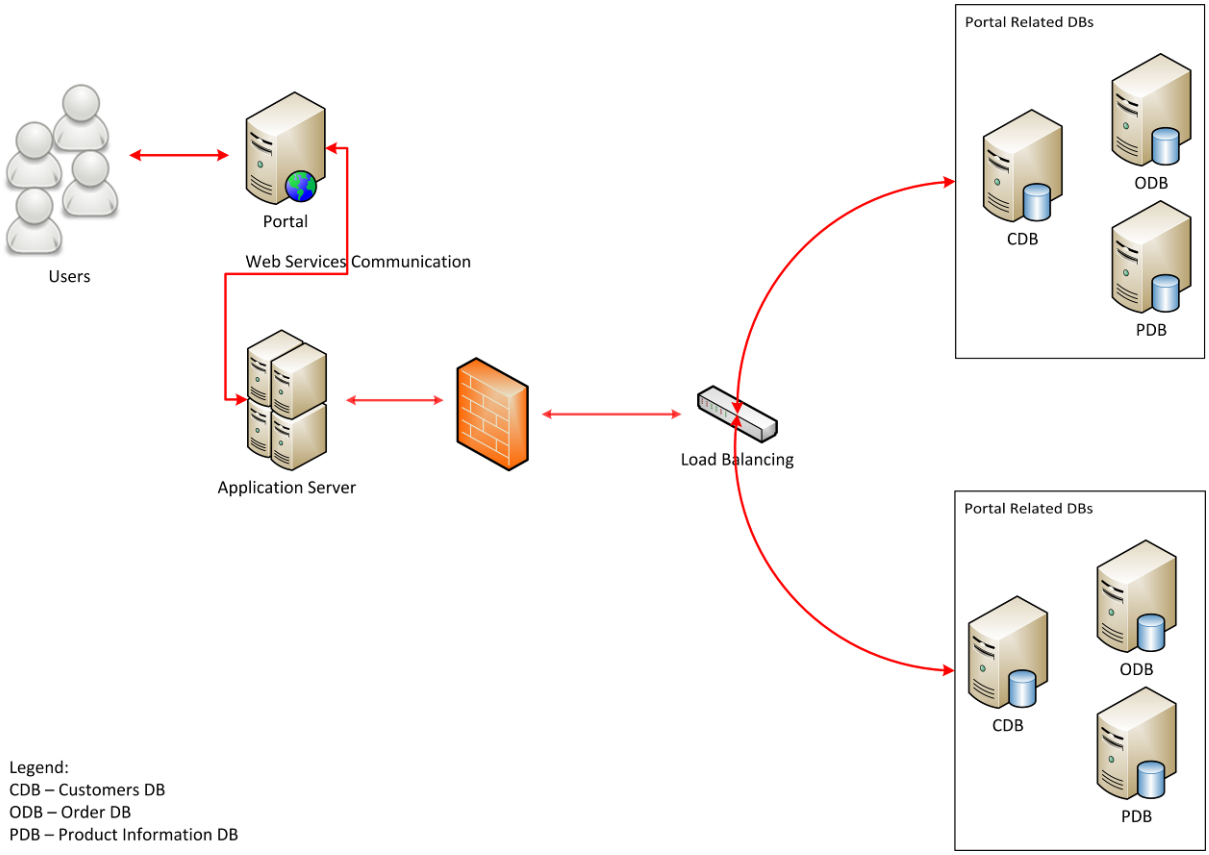

Fig. 5 DB System architecture - overall view

As shown in **Fig. 6** where a database system design is described – a system is composed from one master database along with its mirror database and slave databases.

The idea behind this design is that all the update requests are redirected by a router to the master database and all those read requests to the slave's databases. In order to keep the data in sync inside the system, native replication processes are used. For backup purposes the mirror database is a clone of the master database. The cloning process can be done once or multiple times per day.

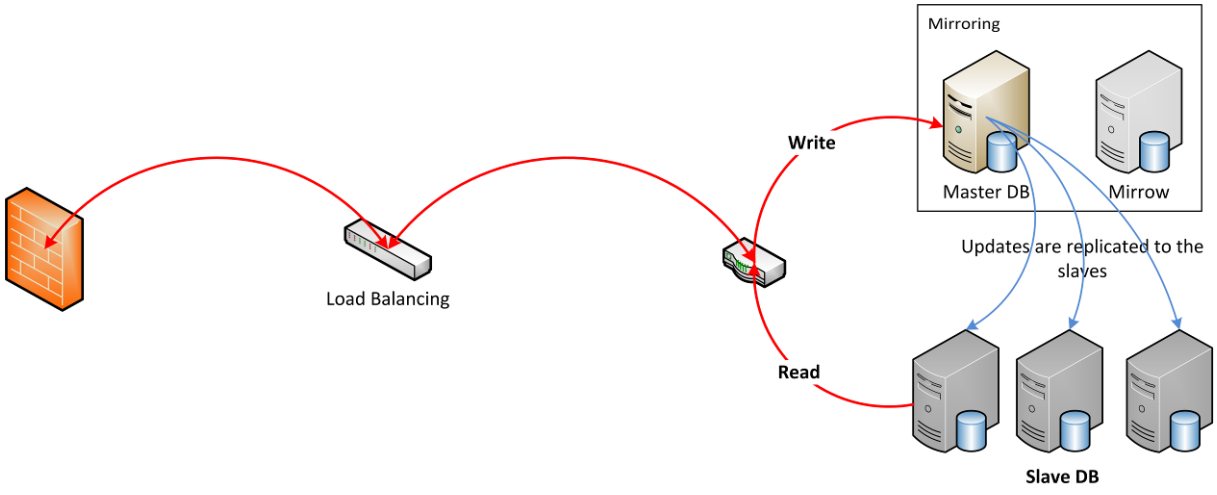

Fig. 6 Database system architecture

For our system beside the customer's databases the most important and vital data source is the one from the Product Information System. This data source contains all those information related to products, their models, sizes and other ways of customization.

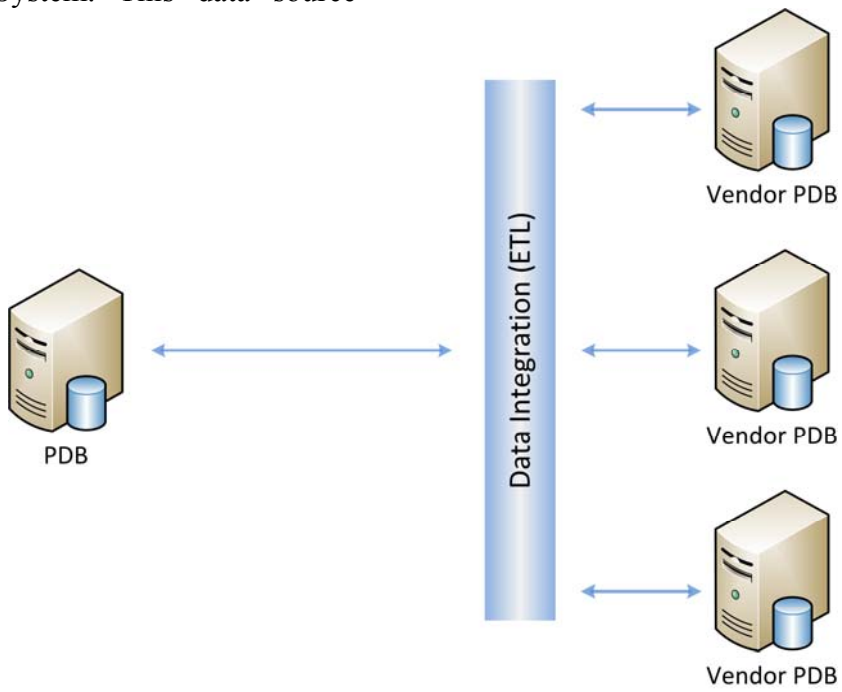

Fig. 7 Product Information System Database Server

As show in **Fig. 7** the data is imported from third-party vendor's database servers. The data importing and synchronization process is done by a specialized application capable in extracting, transforming and storing the necessary and required data.
This application translates the data acquired from vendors in a format understood by our application server and its modules.

5 E-ERP Database System Architecture

As presented earlier in article the proposed ERP - Extended has in its structure specialized modules like: Customer Relationship Management - CRM, Supply Chain Management - SCM, Business Intelligence and Knowledge Management – BI & KM, Manufacturing execution system – MES, and others.
Each module has its own customized database system. The required data is acquired from other data sources inside the entire application system.
We are presenting in **Fig. 8** the main modules and their interactions. Thereby the MES DB contains data acquired by extracting and transforming data from database systems such as: PDB and CDB.
The CRM DB contains data from systems such as ODB and CDB.
The BI & KM module uses data from almost all the existing systems – as show in **Fig. 8** – from systems such as ODB, CDB, PDB, MES DB, CRM DB and so on.
Each module uses specialized application capable in extracting and transforming all the necessary and required data from other internal / external database systems.
The proposed architectural design allows a big database server distribution factor. Thus each database server can be hosted on a single machine, everywhere. Besides this feature, the proposed architectural design eases the use of grid computing design in order to increase performance and speed.

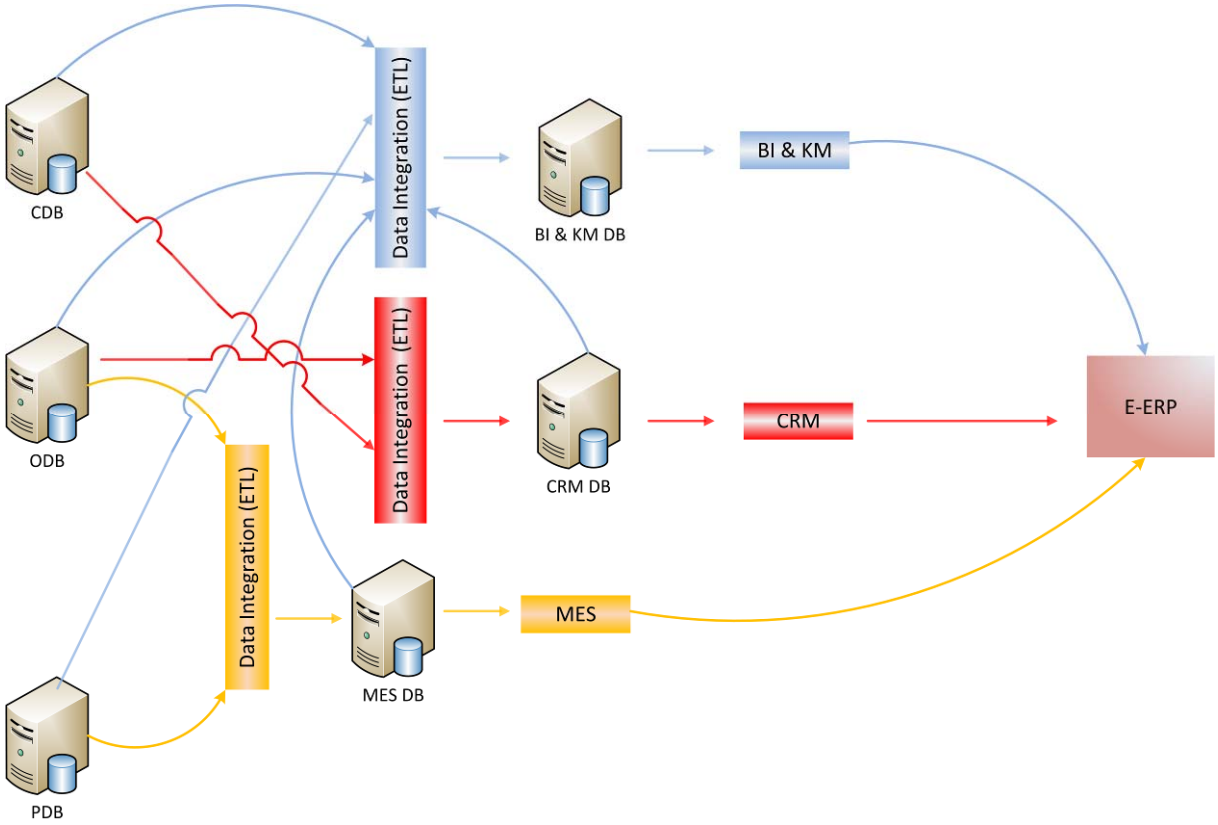

Fig. 8 E-ERP Database System Architecture

6 Conclusions

Grid technologies allow a full integration and information partitioning by providing access to several applications to the IT infrastructure of the organization instead of requiring dedicated servers to be installed for a sole application with a well defined purpose.

By using grid computing the software architecture proposed for the manufacturing organizations, allows dynamic grouping in computer sets, application servers and databases in order to ensure an efficient support to volatile needs in the business environment. ERP platforms become simultaneously more flexible, scalable, with an increased performance level and high availability.

Grid technologies allow resource allocation and efficient use of processing power by optimizing the IT infrastructure.

An organization can overcome the critical moments in an IT infrastructure zone using other resources available at that time. Registering an additional cost is avoided in this way by improving IT performances in an area only to overcome a crisis by splitting processing power to different sources.

This kind of split permits replacing components of the IT infrastructure without affecting functionalities of the system. A better organizational management is obtained by the access to a better image of the IT infrastructure and through and a unique resource focused management process is achieved. ERP systems offer today complete grid computing solutions by allowing access to the system using the internet and sharing application servers and databases.

7 Acknowledgments

This work was co-financed from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013; project number POSDRU/107/1.5/S/77213„ Ph.D. for a career in interdisciplinary economic research at the European standards”

References

[1] I. Lungu, A.R. Bologa, V. Diaconita, A. Bara and I. Botha, Integration of Informatics Systems. Buchares, Romania: ASE Publishing House, 2007, pp. 1-297.

[2] L. Hurbean and D. Fotache, Integrated software solutions for business management-ERP. Bucharest, Romania: Economics Publishing House, 2007, pp. 1-264.

[3] *Development of a Grid ERP Architecture: Integration of Grid Computing and Enterprise Resource Planning Application,* accessed on 29/05/2011, http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=4681186&queryText%3Derp+grid+computing%26openedRefinements%3D*%26searchField%3DSearch+All

[4] A. Dashchenko, *Reconfigurable Manufacturing Systems and Transformable Factories*. New York, SUA: Springer Publishing House, 2010, pp. 1-759

Laurentiu CIOVICĂ has graduated the Faculty of Science, in 2008 gaining a Bachelor of Science degree in Information Technology with a thesis on Translators and Interpreters for Code Generation and Software Optimization. In 2010 he gained a Master of Management degree in the field of Cybernetics, Statistics and Economic Informatics with a thesis on Intelligent Agents. He is currently a PhD student at Academy of Economic Studies in Bucharest. He is the author and co-author of more than 12 scientific articles in the field of software quality and optimization, code generation techniques, collaborative systems, data bases, programming environments and techniques, mobile platforms and economic informatics systems. Besides the scientific activity he is also an active software developer, being the author of few applications. His area of interests includes among others: software quality, optimization techniques and algorithms, code generation techniques, economic informatics systems, intelligent and collaborative systems, mobile platforms.

Constantin-Daniel AVRAM has graduated the Faculty of Economic Cybernetics, Statistics and Informatics in 2005. He holds a master degree in Computerized Project Management. He is the author and co-author of several scientific articles in the field of ERP systems, manufacturing-oriented informatics systems, software recognition, risk management, software and projects quality assurance, project management. He is currently attending a PhD Program in Economic Informatics. He has more than six

years of experience in ERP implementations. He was involved in large and very large international ERP projects.

Grid Database - Management, OGSA and Integration

Florentina Ramona PAVEL (EL BAABOUA)
Academy of Economic Studies ROMANIA, Bucharest
pav_florentina@yahoo.com

The problem description of data models and types of databases has generated and gives rise to extensive controversy generated by their complexity, the many factors involved in the actual process of implementation. Grids encourage and promote the publication, sharing and integration of scientific data, distributed across Virtual Organizations. Scientists and researchers work on huge, complex and growing datasets. The complexity of data management within a grid environment comes from the distribution, heterogeneity and number of data sources.
Early Grid applications focused principally on the storage, replication and movement of file-based data.. Many Grid applications already use databases for managing metadata, but increasingly many are associated with large databases of domain-specific information. In this paper we will talk about the fundamental concepts related to grid-database access, management, OGSA and integration.
Keywords*: grid, data grid, grid computing, database, management, integration.*

1 Introduction

A data grid is a grid computing system that deals with the controlled sharing and management of large amounts of distributed data. Data grids should provide a low level framework for data management activities.
The size and number of data collections has been growing rapidly (petabytes), the costs of computation and data storage decrease and performances increase.
Data grids allow to store, manage and share large data collections, huge amount of files, geographically distributed databases across virtual organizations.
Data management represents the real challenge for the next generation petascale grid environments. In the last few years, there was an increasing interest in fine grained (database related) grid data management activities and services connected with database access, metadata management, data integration, data transformation, data flow. Grid Services for database access and integration play a strategic role and provide added value to a production grid environment since they allow to aggregate data, join datasets stored at different sites, infer new knowledge by analyzing structured and distributed data, manage monitoring and accounting information.[4]
Research and development activities relating to the Grid have generally focused on applications where data is stored in files.
Grid Services for database access and integration play a strategic role and provide added value to a production grid environment since they allow to aggregate data, join datasets stored at different sites, infer new knowledge by analyzing structured and distributed data, manage monitoring and accounting information.

2 Grid Computing and Common Benefits

Grid computing is a term that has been applied to various architectures designed to

deliver the benefits of an IT grid. It is an approach to computing that detaches the software functionality from the specifics of hardware deployment by blending system and storage resources into a continuum of resources that can be allocated to, and deallocated from a particular function or functional locus, in this case, a database.
It enables administrators to assign computing tasks to computing resources, and it assigns data to storage resources in a way that enables such resources to be easily added or removed or tasks and data to be moved as needed.
In the case of database workloads, grid computing contrasts with the classic model that involves dedicated servers associated with dedicated storage in that the servers and storage are fluid. They can be assigned, added, and reassigned as necessary without upsetting the overall topology of the database server environment.

The key benefits of grid computing come in the form of resource flexibility, scalability, and optimization of operations through parallel processing. These benefits are expressed through an architecture that gives users the following capabilities:
• To avoid unnecessary hardware, power, and staffing costs of overprovisioning IT systems, commonly done to avoid capacity upgrades.
• When capacity upgrades are necessary, to scale incrementally by adding (or in some cases, redeploying) system and storage resources without expensive "forklift upgrades" or time-consuming and error-prone upgrade procedures.
• To ensure continuous availability through the provisioning of redundant resources, ensuring automatic failover when necessary.
• To increase transaction throughput through parallelization of tasks.
All these benefits combine to enable better business agility in responding to changes in load or business priorities.

3 Metadata – importance in Grid

The use of metadata in Grid applications tends to be quite simple it is mainly for mapping the logical names for datasets into the physical locations where they can be accessed.
Metadata will be important for many Grid applications, in the following activities:

• Management or scheduling through provision of system and administrative information.
• Data discovery or interpretation through provision of data structure and content information.
• Resource or access method selection, through indexes or summaries.
• Data selection or evaluation, to inform human judgements about the data. [13]

Almost all aspects of metadata can have components that are application specific. [13]
Many applications involve portals, workflows or bespoke code that first examines metadata according to user requirements and then uses these metadata to locate the data, describe which data are accessed, determine what transformations are necessary, to steer analyses and visualizations, and to carry forward information into automatically generated metadata associated with result sets.
As users and developers develop more sophisticated applications, more sophisticated metadata systems and tools will be required.
The use of metadata to locate data has important implications for integrating databases into the Grid because it promotes a two-step access to data . [13]
In step one, a search of metadata catalogues is used to locate the databases containing the data required by the application. Those data are then accessed in the second step. A

consequence of two-step access is that the application writer does not know the specific resource that will be accessed in the second step. The application must be general enough to connect and interface to any of the possible resources returned in step one. [13]
Ideally the two-step approach requires that all resources should provide the same interfaces, but variation in facilities, interfaces and representations is inevitable. [13]
OGSA-DAI services provide metadata about the DBMS, e.g. whether it is an Oracle, DB2 or MySQL, DBMS (Database Management System), system that are being exposed to the Grid. Also metadata are provided about the capabilities of the DBMSs that are being exposed to the Grid through the service interfaces as well as any inherent capabilities of the services themselves. The connection technology that is employed to connect to the databases is also exposed for clients capable of using such information.
The metadata may be provided statically, that is when the service is configured, or dynamically, which may require additional coding. On the whole the static metadata model is extensible so that communities that employ OGSA-DAI to access databases within a Grid context can provide community-specific metadata for the databases they expose to the Grid.

4 The need for databases in Grid environments

Early Grid applications were often closely associated with devices or tools that read and/or generated flat files. Consequently, support for files rather than for the management of structured data had the highest profile in the early Grid toolkits.
However, over time, the file management systems and registries associated with Grid toolkits themselves became complex, and database management systems (DBMSs) were increasingly used to store Grid metadata. Contemporaneously, the requirements of the scientific computing community have become more sophisticated with, for example, biological and astronomical communities generating large quantities of data that increasingly use databases for storage and retrieval. Similarly, engineering, medical research, healthcare and many governmental systems can also take advantage of Grids that access and integrate multiple and distributed collections
of structured data. [13]
Researchers, initially led by Globus and IBM, began in 2001 developing new Grid standards and technology. The result was the Open Grid Services Architecture (OGSA). OGSA presents a picture of the Grid where Grid resources and services are represented by instances of Grid services.
Grid services, are stateful service instances supporting reliable and secure invocation, lifetime management, notification, policy management, credential management, and virtualization. The OGSA-DAI project is developing Grid services that represent data resources, where, by a data resource, we mean any physical or logical entity that is able to source or sink data. These underlying data sources and sinks, together with any associated management infrastructure, are referred to as physical data resources. The term data resource is then used to represent the aspects and capabilities that are exposed to the Grid.
If the OGSA is to support a wide range of communities, then database integration is vital. As the Grid becomes commercially important, database vendors will embed the middleware functionality directly into their products to provide support for OGSA Grid integration.
Similarly, it is vital that those designing standards for Grid middleware take into

account the special requirements to easily integrate databases across a Grid. One of the motivations for OGSA-DAI is to expose and formulate such requirements. Together, these converging developments will reduce the amount of middleware required to integrate databases into the OGSA Grid.
OGSA-DAI has designed, developed and released a collection of services for integrating database access and description with the core capabilities of OGSA, this allowing structured data resources to be seamlessly integrated into OGSA Grid applications.
Relational database vendors support the integration of their products with Web Services from within SQL queries, the creation of Web Services from stored procedures, and the publication of Web Services based on database requests.
The European Data Grid has developed Spitfire, in a Grid settings, a Web Service interface to relational databases for metadata management. Spitfire has developed an infrastructure that allows a client to query a relational database over GSI-enabled HTTP(S). An XML-based protocol is used to represent the query, and its result. Provide a number of facilities for automating the management of data and its referential integrity.
OGSA-DAI currently only provides interfaces to relational and XML database management systems. [3]
OGSA-DAI allows developers to define their own activities and make them available to consumers. This feature has been exploited by a number of research groups and could, for example, be used to expose specialist functionality such as local data mining capability to database consumers. There is clearly also a relationship between OGSA-DAI and other data Grid functionalities. [3]

The prime goals of OGSA-DAI were:

• Provide controlled exposure of physical data resources to the Grid.
• Support access to heterogeneous physical data resources through a common interface style.
• Provide base services that allow higher-level data integration services to be constructed.
• Leverage emerging Grid infrastructure for security, management, accounting etc.
• Standardise data access interfaces through the GGF DAIS WG.
• Provide a reference implementation of the DAIS specification.
OGSA-DAI should be seen as one of a range of components that together support access, sharing, management and coordinated use of data on the Grid. [3]

5 Operations on a Data Resource

Managing the interaction between a data resource and the Grid involves defining the operations that may be performed on a physical data resource and the data requirements for these operations.
• Update operation, data must be delivered to the data source.
• Query operation, data may be transported away, via a delivery mechanism, from the data resource.
OGSA-DAI is not defining any new query languages; the GDS is acting as a conduit through which existing query languages may be directed to the physical data resource.
Figure 1 presents primary mode of operation employed by OGSA-DAI: a Grid service presents some view of a data resource, a query document is submitted to the Grid service, and is evaluated to produce a result document, usually returned to the client. The nature of the query document submitted to the Grid service and the subsequent result document depends on the type of the data resource that the Grid service is configured to represent. For example, a relational database may accept SQL queries. [3]

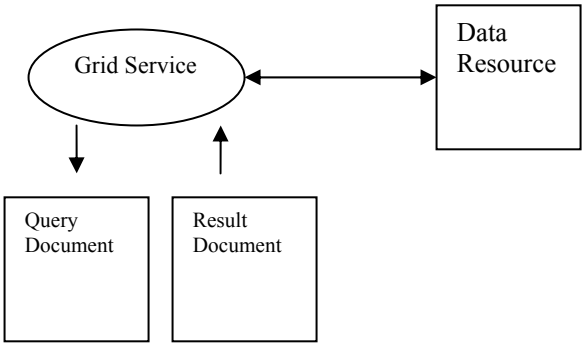

Fig.1. GDS mode of operation

If the data in question is transported somewhere else in the Grid then a GDSF may be used to represent the data at the destination point. Alternatively, the data may be represented in some other non-Grid-enabled storage system in which case it may be referenced using out-of-band techniques.
GDSF (Grid Data Service) is defined to represent the point of presence of a physical data resource on the Grid. It is through the GDSF service instances that a physical data resource's capabilities and meta-data are exposed.

Consider the case where a GDS is used to request data from a physical data resource:

• If the results are anticipated to be small then the client may request that the data is returned synchronously, i.e. in-lined in the response to the original query.
• Out-of-band delivery mechanisms might be used to transfer data resulting from a query. A new GDSF could then be created against the physical data resource to which results have been delivered - see Figure 2. [3]

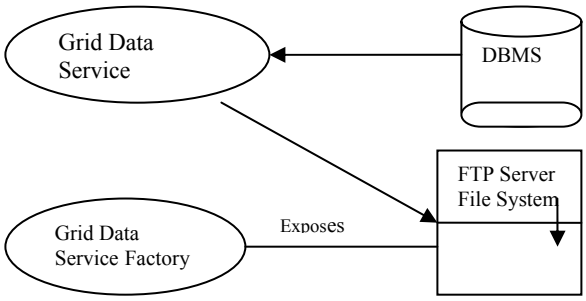

Fig.2. GDS delivery to file system.

• Delivery from one GDS to another may be used as a mechanism for transferring data. The results could then be served by a new GDS.
It is not the intention of OGSA-DAI to build delivery technology or indeed Grid services that represent the data that is being delivered. The interface to delivery and the specification of what is to be delivered across a particular interface is of interest to OGSA-DAI.

6 Grid-DBMS: Key Issues

A grid database management system should provide transparent, secure and efficient management of data sources in a grid environment. Since the beginning of the grid era many efforts were directed towards computational access and storage management. Grid database management was addressed starting from the year 2000 (EDG-Spitfire (Bell et al. 2002), GRelC (Aloisio et al. 2005), and OGSA-DAI (Antonioletti et al. 2005)). In the following we describe some basic elements connected with database management in distributed environments, highlighting how they impact on the application domains and why they are so relevant for end users.
Next subsections will be devoted to the discussion of data representation, data organization, data models, query languages, data access, data integration, access control and data flow. [4]

Data Representation
To be domain-independent, data grids must provide support (in terms of access and management) to every type of data format, structure and representation.
Data can be both structured and unstructured,
characterized by different formats, coding, precision, accuracy and semantics.
Some examples concern bioinformatics (i.e. textual files, relational data sources), astrophysics (i.e. relational DBMS with postGIS extensions), climate scientists (i.e. XML) data banks. [4]

Data Organization, Data Models and Query Languages
Data can be organized following several data models such as relational and hierarchical. Support in terms of relational or XML engines is widely provided by existing systems: Postgres, MySQL, IBM/DB2, as well as XIndice, eXist. Such DBMSs provide full support in terms of database access and management functionalities. Different data models adopt different query languages such as SQL (for the relational one) and XPath and XQuery (for the hierarchical one). Data grids must provide support to all of them. [4]

Data Access
Even if DBMSs provide a lot of functionalities for the management of data sources, they are not fully compatible with existing grid middleware. They can be accessed in grid by using a "grid-DBMS" interface. This grid interface must provide full support to all of the query languages (SQL, XQuery, XPath, etc.) concerning the
target data resources (transparency requirement with regard to the query language). The specific part of the grid-DBMS that makes a data resource accessible in grid (or "grid enabled") is called Grid Database Access Service (Grid DAS).
It must provide secure, transparent, robust and efficient access to heterogeneous and distributed databases exposing standard interfaces to enable interoperability with other grid components and/or services.
Several research projects exploit the service-in-themiddle or front-end approach to provide such kind of functionalities, that is, they focus on the development of a transparent, secure and robust grid interface to existing DBMSs. On the contrary, vendor-specific products (i.e. Oracle 11g) generally exploit an embedded approach providing within the product, software modules to run on a grid environment. [4]

Data Integration
While the Grid DAS is a basic service to expose databases in grid (it provides a first level of virtualization), the Grid Data Integration Service (Grid DIS) is a further necessary building block if we want to provide aggregation capabilities (second level of virtualization).
A Grid DIS can be centralized or distributed and in some cases it is integrated into the related Grid DAS providing what we call a Grid DAIS.
Data integration is strongly challenging since it allows both to integrate data within several application-level domains (bioinformatics, astrophysics, financial, etc.) and system-level distributed environments for monitoring and accounting purposes. [4]

Data Access Control
Data access control is more important to ensure that the confidentiality of the data is preserved/maintained against unauthorized accesses.
The facilities that the Grid provides to control access must be very flexible in terms of the combinations of restrictions, available policies, etc. User-centric and VO-centric (Virtual Organization), data access control allow managing policies

at each level of granularity addressing local site autonomy and user-level policies management (in the first case) and flexibility, scalability and manageability in the VO-level policies management (in the second case).
A combined User-VO data access control allows mixing the benefits related to the two approaches (any combination of insert, update, and delete privileges can be defined with the right level of granularity).
The Grid must provide the ability to control access based on user role (as it usually happens for DBMSs). Role based access control is fundamental for collaborative working, when several individuals may perform the same role at the same time and provides a scalable and manageable way to split users in subclasses with specific and well-known privileges.
Granting and revoking activities must be dynamically performed by administrators and should be easily carried out by using high level interfaces such as data grid portals.
Data access policies should be managed at the Grid- DBMS layer, without entirely relying on the back-end framework. This could enable data access control for trivial data resources such as text files and prevent the access attempts to the back-end systems for unauthorized users. [4]

7 Transparency, Efficiency and Interoperability

Transparency is a common requirement for grid services and fundamental to make virtualization a reality. There are various possible types of transparency in a distributed environment.
• Physical data location: the physical location of a database in the grid must be hidden/virtualized by the grid service.
• Naming: an application must be able to access a data source without knowing its name or location.
These kind of information must be managed by means of mapping, alias, which conceal data that are not relevant to the end-user, such as connection string for the databases, DBMS port, login and password.
• Data replication: replication of data improves performance, availability and fault tolerance. The user must not be aware of the existence/management of multiple physical copies of the same data source, has just to deal with the logical (virtualized) data source name.
• DBMSs heterogeneity: many different RDBMSs exist, such as ORACLE, IBM/DB2, MySQL, SQLite. An increasing number of applications interact with not relational databases such as flat files and XMLbased documents in the bioinformatics and climate change domains. This kind of heterogeneity must be properly handled in order to provide a uniform access interface to different data sources and a grid database access service independent of the back-end systems. [4]

Efficiency
Performance plays a fundamental role in the data grid environment. High throughput, concurrent accesses, fault tolerance, reduced communication overhead, are important goals that must be achieved by exploiting among the others data localization and query parallelism. Efficient data delivery mechanisms can reduce the connection time and the amount of transferred data. [4]

Security is crucial for the management of a database in data grid environment. Data security aims at protecting data against unauthorized accesses by preventing unauthorized users from accessing data and protecting information exchanged in the data grid network. Authentication is strongly required to check user's identity, authorization concerns privileges and read/write permissions. Users must be able to "log on" (authenticate) just once and then

have access to any resource in the Grid that they are authorized to use, without further user intervention.
Most important production/research grids adopt the de-facto standard for security Globus Grid Security Infrastructure (GSI).
It provides full security support concerning data encryption, data integrity, protection against replay attacks and detection of out of sequence packets. GSI is widely used both in gLite and Globus based grid environments.
The Spitfire (European DataGrid Work Package 2, Project Spitfire) has implemented a security architecture based on transport-level SSL security and mapping of Grid credentials to database roles. [4]

Interoperability can be achieved by standard adoption. To achieve interoperability, using the method of defining and adopting common open standards and architectures is a common approach, which relies heavily on standardization and implementation processes. Because comprehensive implementations and roll-outs spanning different Grid communities are difficult, costly and often politically charged, in certain scenarios a coupling of the architectures is a reasonable alternative.
The loss of interoperability in a Grid of middlewares may lead to problems in the Grid's operation. [4]

8 Grid Integration

The full integration of database technologies with Grid middleware is widely recognized.
There are two main dimensions of complexity to the problem: reconciling implementation differences between server products within a single database paradigm (IBM, Oracle, Microsoft, etc.) and the variety of database paradigms (object, relational, XML, etc.). Each DBMS is the result of many hundreds of person-years of effort to provide a wide range of functionality, valuable programming interfaces and tools, and important properties such as security, performance and dependability.
Grid Data Integration service (GDI), this service provides XML schema mapping utilities for semantically connected XML data sources. To this aim the GDI extends the OGSA-DAI by introducing a new activity devoted to the reformulation of an XPath query by using the XMAP reformulation algorithm.
There is a considerable history in database research of semantic data modelling and data integration techniques, both being dimensions of the problem outlined above for Grid data services required in the earth sciences.
Data modelling has evolved from Codd's relational model through the ER model of Chen to semantic and fully object-oriented models incorporating inheritance, aggregation, and behaviour.
The databases literature also contains a considerable history of data integration methods. These have been developed for a rich range of problems including reverse and re-engineering, schema translation, and database integration.
Proven approaches for DBMS integration that might be examined for their applicability in a file-based data Grid include: data warehousing where data is imported enmasse from legacy databases and transformed into a common data model, and wrapper/mediator architectures where heterogeneous local sources are mapped to a global schema and integrated through middleware.
For integration of heterogeneous file-stores in a data Grid, the warehousing and federation models are impractical. Instead, a wrapper/mediator approach is required, with a common data model exposed through semantic data services.
The requirement for data integration on the Grid has led to a significant amount of activity in the GGF DAISWG, with

specifications developed for relational and XML Grid Database Services.
What is really needed in a data Grid is a semantic data integration framework that allows the request on this global geographic dataset.

9 Conclusions

Data services for the Grid have focussed so far primarily on encapsulating data syntax (distributed relational databases, file format and location).
The elements of a generic framework would include:

- A meta-model for constructing semantically-rich domain specific data models independent of storage concerns
- A data storage description language for describing the construction of semantic data object instances.
- A canonical process for serialising semantic data instances in service workflows.

Both implicit and explicit knowledge-bases or ontologies are supported by the general framework.
Data access services may be built on top of a data model constructed according to the framework. These could be exposed through Activity extensions in OGSA-DAI.

10 Acknowledgement

The language used to convey requests to a database service, the database can be SQL for relational database services and XQuery for XML database services.
A single database system may support multiple paradigms.
With Grid Control can be manage a lot of things than just the database. For example, if you run Siebel, PeopleSoft, E-Business Suite, BI-EE, or custom Java application along with Oracle database you can manage them together with Grid Control.
Data grids provide what we need to can manage distributed data, the logical name spaces needed to assemble collections, a common infrastructure base upon which multiple types of data management environments may be implemented.

References

[1] Andrew Borley, Neil Hardman, Alan Knox, Simon Laws , James Magowan, Manfred Oevers, Ed Zaluska, "Grid Data Services – Relational Database Management Systems", Version 1.0 , In: *5th Global Grid Forum*, July (2002), Edinburgh, Scotland. pp. 1-22

[2] Carl W. Olofson, "Grid Computing with Oracle Database 11g" ,Sponsored by: Oracle Corporation, March (2008)

[3] Ali Anjomshoaa, Mario Antonioletti, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil P Chue Hong, Brian Collins, Neil Hardman, George Hicken, Ally Hume, Alan Knox, Mike Jackson, Amrey Krause, Simon Laws, James Magowan, Charaka Palansuriya, Norman W Paton, Dave Pearson, Tom Sugden, Paul Watson and Martin Westhead, "The Design and Implementation of Grid Database Services in OGSA-DAI" - *Proceedings of UK e-Science* All Hands Meeting (2003) 2-4th September, Nottingham, UK pg 795

[4] Sandro Fiore, Salvatore Vadacca, Alessandro Negro and Giovanni Aloisio, "Grid database management: issues, requirements and future directions", University of Salento & SPACI Consortium Euro Mediterranean Centre for Climate Change viale Gallipoli, 49 – 73100 Lecce – Italy, February (2004)

[5] Tuecke, S. ,"Grid Security Infrastructure (GSI) Roadmap", (2001) , Internet Draft

[6] Timo Baur , Rebecca Breu, Tibor Kálmán, Tobias Lindinger, Anne Milbert, Gevorg Poghosyan , Helmut Reiser, Mathilde Romberg, "An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations", J Grid

Computing (2009) 7:319–333, DOI 10.1007/s10723-009-9134-3

[7] AndrewWoolf, Ray Cramer, Marta Gutierrez, Kerstin Kleese van Dam, Siva Kondapalli, Susan Latham, Bryan Lawrence, Roy Lowry, Kevin O'Neill, "Semantic Integration of File-based Data for Grid Services", Conference: Cluster Computing and the Grid - CCGRID , pp. 182-188, (2005) DOI: 10.1109/CCGRID,1558552

[8] John Wiley & Sons, Ltd. Concurrency Computat.: IBM Systems Journal Pract. Exper. (2005); 17:357–376

[9] Amy Krause (EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK), Susan Malaika (IBM Corporation, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, CA 95141, USA), Gavin McCance (Department of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK), James Magowan (IBM United Kingdom Ltd, Hursley Park, Winchester S021 2JN, UK), Norman W. Paton (Department of Computer Science, University of Manchester, Oxford Road, Manchester M134 9PL, UK), Greg Riccardi (Department of Computer Science, Florida State University, Tallahassee, FL 32306-4530, USA. + National e-Science Centre, 15 South College Street, Edinburgh EH8 9AA, UK), "Grid Database Service Specification", GDSS-0.2 4th October (2002)

[10] Moore, R., A. Rajasekar, "Common Consistency Requirements for Data Grids,Digital Libraries, and Persistent Archives", Grid Protocol Architecture Research Group draft, Global Grid Forum, April (2003)

[11] "Open Grid Services Architecture Data Access and Integration", http://www.ogsadai.org.uk

[12] Parent, C. and S. Spaccapietra, "Database Integration: The Key to Data Interoperability", In Advances in Object-Oriented Data Modeling, ed. M.P. Papazoglou et. al., The MIT Press (2000).

[13] Mario Antonioletti, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil P. Chue Hong1, Brian Collins, Neil Hardman, Alastair C. Hume, Alan Knox, Mike Jackson, Amy Krause, Simon Laws, James Magowan, NormanW. Paton, Dave Pearson, Tom Sugden1, PaulWatson and Martin Westhead, "The design and implementation of Grid database services in OGSA-DAI", Concurrency Computat.: Pract. Exper. 2005, 17:357–376, Published online in WileyInterScience(www.interscience.wiley.com). DOI: 10.1002/cpe.93

Florentina Ramona **Pavel (El Baaboua)** graduated from the Computer Science for Business Management, of the Romanian – American University in 2005. At present she is a PhD candidate at the Academy of Economic Studies and PhD assistant at the Romanian – American University of Bucharest. She is co-author of two books and articles in informatic fields.

Considerations Regarding Designing and Administrating SOA Solutions

Vlad DIACONITA
The Bucharest Academy of Economic Studies
diaconita.vlad@ie.ase.ro

Solutions like SOA, Cloud, SaaS, Iaas or PaaS are not only buzzwords, they became a business reality because they are relative cheap and easy to use. SOA and Cloud are tightly linked because most cloud solutions are being defined using SOA making them feasible from the business perspective, because it's hard to move to cloud when you are using a tightly coupled architecture. Big companies such as Oracle, Microsoft, IBM or Amazon offer many commercial solutions providing software as a service, as well as hosted and managed alternatives to classical deployment. For firms that are building private clouds and for service providers that are building public clouds, diverse solutions are offered by the big players for platform as a service and infrastructure as a service.
Keywords: *SOA, web services, modeling, cloud*

Introduction

From an evolution perspective, some authors say ([1], [4]) that the last decade is marked by the development of SOA and cloud computing. The main characteristic of SOA is the ability to be reused in various applications, using service communication by sending information in a loose coupled environment [5]. The idea of exposing resources as web services, making them accessible is older but building the components, tools and infrastructure to accomplish this was the problem. Development of virtualization in organizational environment allowed hooking up applications with various operating systems, enhancing the portability. Like shown in [6], moving towards implementing Web applications that consume a large variety of Web services is the current hype in application space and the mobile application market is searching for solutions to empower mobile devices with Web services integration while minimizing the existing performance issues. By using service-oriented strategies, companies even starting with few resources can run their businesses entirely using cloud. For example, as part of Amazon's AWS Relational Database Service (RDS) someone can rent an Oracle database license if they don't have it on their own. This starts at 16 cents an hour for a small instance going up to $3.96 for a quadruple extra large. The price includes the software, the hardware resources and Amazon's RDS management capabilities [9].

Cloud and SOA

Researchers are trying to bring SOA, web services and cloud services under a common terminology and approach.

Gartner defines cloud computing as a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies. This is a slight revision of Gartner's original definition published in 2008. Gartner has removed *massively scalable* and replaced it with *scalable and elastic* as an indicator that the important characteristic of scale is the ability to scale up and down, not just to massive size. The five attributes of cloud computing are: service-based, scalable and elastic, shared, metered by used, uses internet technology [16].

SOA is providing the architecture, governance and orchestration for services to be delivered using cloud mechanisms, both internally and externally across the Internet.

An author that wrote many books on SOA, Dave Linthicum has shown that cloud computing should be a logical extension of SOA best practices [14].

As explained in Thomas ERL' latest book [13], the rise of cloud computing put SOA back in the spotlight, *even organizations that shunned SOA now have one. It's called the cloud.* He also renamed his online publication, formerly called SOA Magazine, as Service Technology Magazine:

The SOA Magazine has been renamed to the Service Technology Magazine. Articles will of course continue to be focused on service-oriented architecture and service-orientation, but will also address topics related to service technology innovations, such as those fostered by the on-going emergence of cloud computing platforms. I'd like to invite you all to contribute your expertise as we continue to explore how this new generation of architectural models, paradigms, and technologies is changing the way we view and leverage IT [15].

So, we can consider *service technology* as the common identifier.

Defining SOA

There are many definitions of SOA targeted to different audiences (managers, designers, programmers, sales persons etc).

When an enterprise level SOA application is being developed, many people are involved, some of whom are end-user developers. For example, business process experts know about the business context but may not necessarily be professional programmers, and are often responsible for identifying and selecting which services will be used ([2],[3]).

Executives are increasingly frustrated with their inability to quickly access the information needed to make better decisions and to optimize their business [10]. To them, SOA can be showed as a set of services that can be exposed to customers, partners, and between the different departments of the enterprise. These services can be combined and recombined to serve the needs of the business. Applications serve the business because they are composed of services that can be quickly modified or redeployed in new business contexts, allowing the business to quickly respond to changing customer needs, business opportunities, and market conditions.

To an IT designer, SOA is the architectural solution for integrating diverse systems by providing an architectural style that promotes loose coupling and reuse.

To a programmer, SOA is a model where web services and contracts are used for interoperability. The web services, used as part of SOA, facilitate communication using messages, without detailed knowledge of each other's IT systems.

The term *service orientation* is often seen as identical to SOA but some authors [1] consider it broader and represent a way of thinking about services in the context of business and IT.

Like shown in [7] and [8], web services are self contained, modular business applications that have open, Internet-oriented, standards-based interfaces. They allow flexible and dynamic software integration that is often referred to as the *Find-Bind-Execute* paradigm. Using standard Internet technology, Web services facilitate cross-organizational transactions and thus outsourcing of software functionality to external service providers. Thus, service-oriented computing requires an infrastructure that provides a mechanism for coordination between service requesters and providers. Three main technologies are currently used to implement Web services: SOAP, WSDL and UDDI.

Everware-CBDI, a global technology consulting company sees Service Oriented Architecture (SOA) as the principles, patterns and policies that enable application functionality to be provided and requested as services published at a granularity relevant to the Service Consumer, which are abstracted away from the implementation using a single, standards-based form of interface. The evolution of the implementation strategies is offering services in SOA in the CLOUD. SOA should provide reference architecture for service classification, policy implementation and governance, contracts, determining sharing and generalization at many levels.

SOA can also be seen as a collection of services, classified into types, arranged into layers and governed by architectural patterns and policies as shown in figure 1.

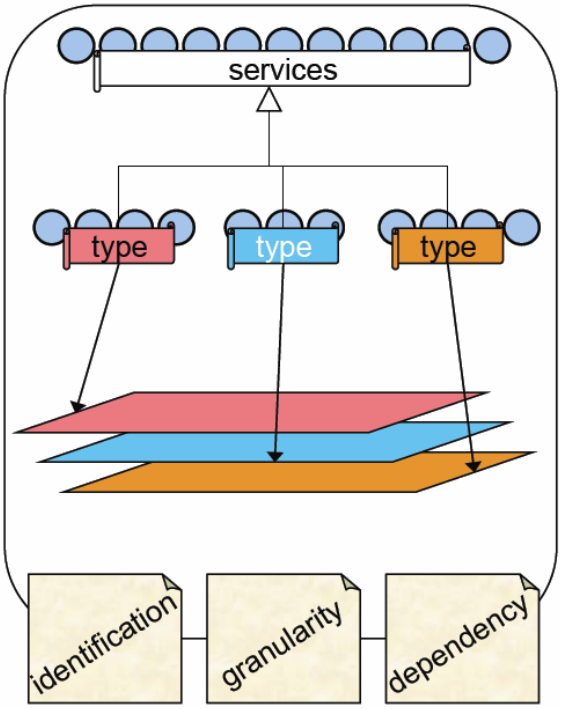

Figure 7. The service arhitecture, source: http://everware-cbdi.com/cbdi-forum

Modeling SOA Infrastructure

A SOA infrastructure can be divided into three categories: consumer, functional, and operational to provide an abstraction that can be used and reused across an organization. Infrastructure architects can use this model as a basis for determining the necessary software products or technologies necessary to provision that building block.

The consumer access component includes the infrastructure needed by the people to access the services, including: browsers, data channels and portals.

Internet browsers are used to expose functionality using a Web interface rendered on a user's browser.

A data channel is where consumers can provide or consume large amounts of data. The movement of bulk data over networks addresses raw unstructured data, structured data, images, and any large data that requires high performance. The infrastructure provides a software-based mechanism designed to move large data files using compression, blocking, and

buffering methods to optimize transfer times. Infrastructure architectures need to determine whether there are business needs for such a bulk data transfer component and provision accordingly.

A Web portal presents information from diverse sources in a unified way. The ESB makes possible the communication between service requestors and service providers. It enables the substitution of service providers or implementations transparent to service requesters. The ESB usually many ways to attach requesters and providers, and it allows intermediary services to be sequenced between them. The ESB can also supply an extensive set of capabilities dependent on business needs and implementation in areas like integration, communications, security, signal processing, QOS and service management.

In SOA, the integrations are pushed outward, toward the applications themselves, leaving the bus to speak a standardized language [11]. Like modeled in figure 2, an ESB is a connectivity infrastructure for integrating applications and services, while EAI focuses on the applications integration. ESB infrastructure does more than integration because it performs routing of messages between services, converts transport protocols between consumers and providers, transforms message formats between requesters and providers, and distributes business events from disparate sources. ESB is the central integration backbone fulfilling the various integration patterns. Enterprise service bus, being the essential and core infrastructure for application integration, ought to be versatile, adaptive, high-performing and assuring, highly available and scalable. In order to accomplish the much-acclaimed process integration, service orchestration capability has to be bestowed with ESBs. The importance of orchestration as a shared component gives ESBs the flexibility and the power towards the success demanded and desired.

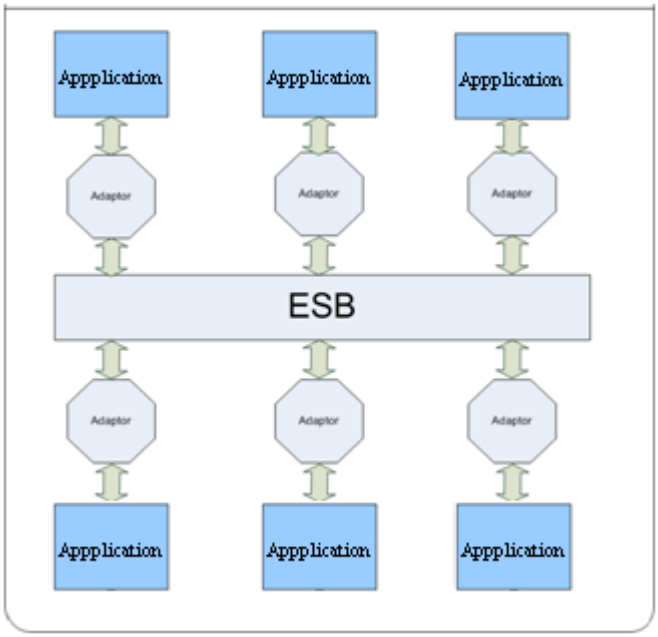

Figure 8. ESB communication

Although EAI solutions can address all of these aspects, integration technologies are usually much more narrowly focused. ESB handles a variety of

interaction patterns, including events. ESB requires management such that the status of a business transaction can be assessed. Although the ESB will not be the only technology to assist in business activity management, it will be a part. ESB product technologies will be federated such that various technologies (e.g., gateways and appliances) can be used to fulfill a single purpose and provide a single interface to applications. Diverse platforms can be supported allowing different ESB technologies to operate as a single logical one. Even if both EAI and ESB can use web services, the latter takes more advantage of the technology and also promotes greater levels of modularity and decoupling of the infrastructure using services. ESBs use registries to assist with locating services, while EAI infrastructures often couple the requester and provider.

From a physical point of view, SOA architecture is very similar to the 3tier web one, the logic is on a server where it is divided into several units. Differences arise from the criteria for sharing the logic, the place where logical units exist and how they interact. In Web architecture there are components that are designed with different levels of functionality and granularity and some emphasis is put on reusing them.

SOA is based on components, modeling takes account of the creation of services that will encapsulate some of the components or all components. The encapsulation of service logic is used to provide interface functions using an open, independent of the technologies used to implement logic. Properly designed, loosely coupled services can easily be combined, aggregated and reused, hence resulting in high scalability SOA solutions. In figure 3 it's presented an endpoint from which 3 different web services can be called, separately or they can be aggregated [17].

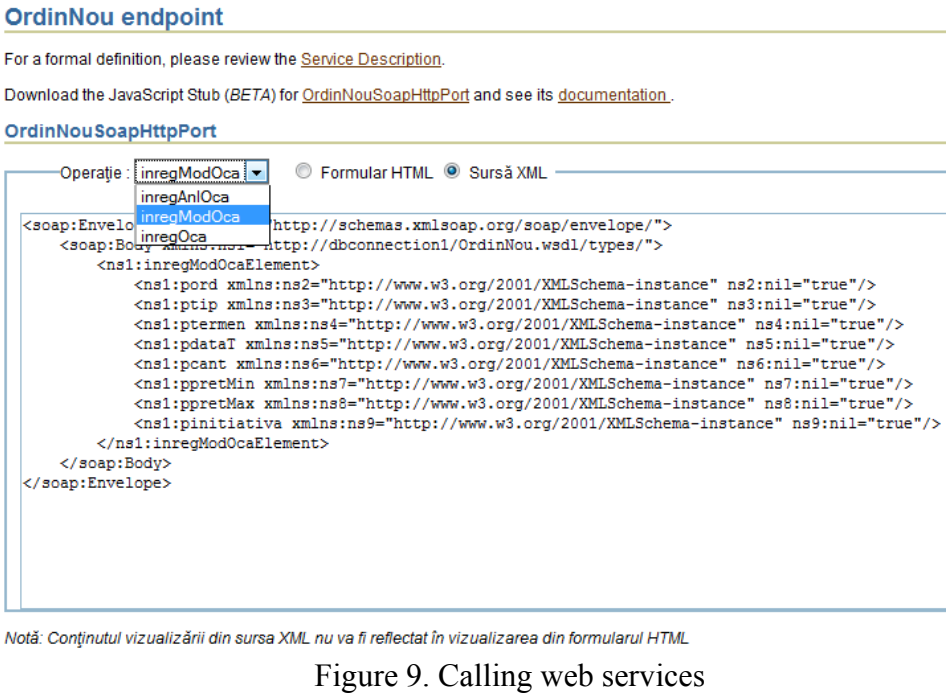

Figure 9. Calling web services

The result of calling a web service, inregAnlOca in this case, is in the form of a XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns0="http://dbconnection1/OrdinNou.wsdl/types/"><env:Body><ns0:inregAnlOcaResponseElement><ns0:result>I-It is confirmed the anulment of the 4328884 order, quantity 500</ns0:result></ns0:inregAnlOcaResponseElement></env:Body></env:Envelope>
```

SOA projects are of different sizes and not all of them require service modeling. SOA projects whose goals include engineering business applications that are built to change service modeling requirements. Some say that new modeling techniques are not required because nothing fundamental is changed. Others say that classic Agile methods or object development methods provide insufficient guidance for SOA projects [1]. Agile methods focus on iterative development, allowing requirements and the solution to grow by collaboration using cross-functional teams including various stakeholders. Object methodologies focus on object modeling and object technologies. In both methods, which represent best practices in system development methods, the focus on service development is absent, mainly because this wasn't popular at the time of these methods development. Organizations developing SOA will need to improve these methods to identify and build reusable, reconfigurable, and flexible services by identification, specification, and realization of these constructs: business processes, services, components, information, rules and policies.

Models help people to deal with complexity by representing complex things at a higher level of abstraction. SOA can help to elevate the level of abstraction by separating the provider from the consumer of the service. The service model identifies the business processes that uses services or identifies the services and the components that recognize the business functionality. Service modeling, as an iterative and business centric process, should focuses on the set of business capabilities and related IT functionality as a set of services, the components that implement them, and the processes that invoke them or put the services together into a composite service or application, should be seen as a whole and address the modeling of activities or flows, services, and their components. Modeling is an iterative and business centric process. A service needs to be modeled from a business and a runtime perspective such that the service fulfills a part of a business process that can be shared and reused.

Like shown in [12], services should be defined and described top-down at enterprise level in Service Oriented Enterprise (SOE). From a functional decomposition of well defined business functionality the business functions can be identified. These business functions can be decomposed in lower level services. As shown in [18], business logic is the defining element for a business being in the process of modeling and automation, and it includes both business rules and workflow (process), which describes the transfer of documents or data from one participant (person or software system) to another. Business Rules refers to the

multitude of policies, procedures or definitions that govern how an organization works together with its interaction with customers or partners. These may be external rules, coming from legal regulations that must be observed by all organizations acting in a certain field, or internal rules which define the organization's business politics and aim to ensure competitive advantages in the market. Starting from the previous observations, it is obvious the important role that business rules play within the development process of a software system. Top-down domain decomposition using process modeling and decomposition, variation-oriented analysis, policy and business rules analysis, and domain specific behavior modeling should be done in parallel with a bottom-up analysis of existing legacy assets that are candidates for componentization (modularization) and service exposure [12].

In SOA, the system operates as a collection of services and each service may interact with various other services to accomplish a certain task. The operation of one service might be a combination of several low level functions. In that case, these low level functions are not considered services.

Most web services are based on document type messages that are designed to be as independent as possible. Using SOAP headers, the actual messages may be accompanied by a wide range of metadata, and general processing instructions.

Processing is distributed; each service has a specific functional border and specific resource requirements. Communication between the supplier and the consumer of services can be synchronous or asynchronous, can use templates and a large part of the business logic is contained in the messages.

Information processing is accomplished at the application server and/or database level. Communication in classical architecture is achieved using protocols such as TCP/IP, DCOM (Distributed Component Object Model) or CORBA (Common Object Request Broker Architecture), protocols which have reached maturity. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification. Nowadays SOA is based on communications using services which imply: serialization, de-serialization and transmission of SOAP messages containing XML documents. Operations that are executed include: conversion of data from relational databases in XML, XML document validation, document transmission and retrieval of information needed by the recipient. Although progress has been made for SOAP classical communications, Remote Procedure Call (RPC) are faster. A SOAP-based communications network facilitates the creation of services that can communicate according to various templates. Even if the synchronous communications is well implemented, asynchronous communications is encouraged to optimize processing and communications. WS-* specifications can be used, especially WS-BPEL.

The technologies used on the Internet have undergone many changes and improvements but remained basically the same: HTTP, HTML or XML. If in traditional web architectures, web services use XML messages are optional, if modern implementations of SOA are almost mandatory.

When the logic of a system is divided and distributed, the implementation of security measures such as authentication and authorization is not as straightforward as it was in client-server architecture. Information travels through multiple servers (bumps) and it is often necessary at least to encrypt it or at least the sensitive information: password, bank account, etc. SOA brings some changes to this model, being based on the WS-Security which puts the logic of security-

related messages. In SOAP messages, the header may store security related information will be accompany the messages. This approach is necessary to maintain autonomy and loose coupling between services.

Conclusion

Although at first glance SOA administration may seem simple, things usually develop to a point where services are highly aggregated and reused and the administration becomes difficult. Then it is necessary to use stronger service agents or private service agents. When we have an application that includes various components, management is not easy, the following have to be monitored: the connections, the instances, the problems with the connections, the resources employed and the tasks related to database administration. The clients connect at first at first to the web server which interacts with the application server, so it is important to carefully administer it to ensure scalability. Most application servers and database management systems provide mature interfaces that can be accessed with a web browser. In SOA solutions, additional problems may arise with regard to communications using SOAP messages. Management errors can be done using the exception mechanism provided by different WS-* extensions. A good strategy to encourage the reuse and aggregation of an internal solution is to create a private agent service. UDDI can be used to standardize the interface of the agent services and so the system services can be easily discovered.

References

[1] Kerrie Holley, Dr. Ali Arsanjani, *100 SOA Questions Asked and Answered,* Prentice Hall, ISBN 978-0-137-08020-5, 2010

[2] S.Y. Jeong, Y. Xie, J. Beaton, B.A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D.K. Busse, *Improving Documentation for eSOA APIs through User Studies*, in Proc. IS-EUD, 2009, pp.86-105.

[3] Thomas Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, 2005, ISBN: 0-13-185858-0

[4] Marinela Mircea, Marian Stoica, Cloud Computing Solutions For Service Oriented Organizations Management, Proceedings of The Tenth International Conference on Informatics in Economy IE 2011

[5] Mircea, M. ,Andreescu, A.I., *Extending SOA to Cloud Computing in Higher Education*, The 15th IBIMA conference on Knowledge Management and Innovation: A Business Competitive Edge Perspective, Cairo, Egypt 6-7 November 2010. Norristown: International Business Information Management Association

[6] Alin COBÂRZAN, *Consuming Web Services on Mobile Platforms,* Informatica Economică vol. 14, no. 3/2010, ISSN 1453-1305

[7] S. Agarwal, S. Lamparter and R. Studer, *Making Web services tradable: A policy-based approach for specifying preferences on Web service properties*, Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 7, No. 1, January 2009, pp. 11-20.

[8] Vlad DIACONIȚA, *Hybrid Solution for Integrated Trading,* Informatica Economică vol. 14, no. 2/2010, ISSN 1453-1305

[9] Maureen O'Gara, *Oracle Goes to Amazon*, .NET Delopers Journal, Mai 2011

[10] Mario Godinez, Eberhard Hechler, Klaus Koenig, Steve Lockwood, Martin Oberhofer, Michael Schroeck *The Art of Enterprise Information Architecture: A Systems-Based Approach for Unlocking Business Insight,* IBM Press, ISBN: 978-0137035717, aprilie 2010

[11] Vlad Diaconita, Ion Lungu, Adela Bara, *Technical solutions for integrated trading on spot, futures and bonds stock markets (extended version),* WSEAS Transactions on Information Science and

Applications Volume 6, Issue 5, 2009, Pages 798-808 , ISSN: 1790-0832, Indexed by Scopus, ACM
[12] http://www.enterprise-architecture.info/EA_Services-Oriented-Enterprise.htm
[13] Stephen Bennett, Thomas Erl, Clive Gee, Robert Laird, Anne Thomas Manes, Robert Schneider, Leo Shuster, Andre Tost, Chris Venable, *SOA Governance: Governing Shared Services On-Premise & in the Cloud,* Prentice Hall/PearsonPTR, Aprilie 2011
[14] David S. Linthicum, *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide*, Addison-Wesley Professional; 1 edition, October 2009
[15] http://www.servicetechmag.com/, ISSUE June 2011, Editorial
[16] Daryl C. Plummer, David Mitchell Smith, Thomas J. Bittman, David W. Cearley, David J. Cappuccio, Donna Scott, Rakesh Kumar, Bruce Robertson, *Five Refining Attributes of Public and Private Cloud Computing*, May 2009
[17] Pethuru Cheliah, *Empowering the Discipline of Cloud Integration – Part II*, Service Technology Magazine Issue LI, June 17, 2011
[18] Alexandra FLOREA, Anca ANDREESCU, Vlad DIACONITA, Adina UTA, *Using SOA for achieving enterprise interoperability,* Proceedings of The Tenth International Conference on Informatics in Economy IE 2011

Vlad DIACONIȚA 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. He has graduated the faculty at which he is now teaching in 2005 and since 2010 holds a PhD in the field of Cybernetics and Statistics. He is the co-author of 2 books in the domain of economic informatics, 3 articles in ISI journals, 4 articles in Scopus journals, 4 articles in ISI proceedings, 6 papers in B+ journals and 8 papers in the proceedings of international conferences. He participated as team member in 3 research projects that have been financed from national research programs. He is a member of the IEEE Computer

Society and the INFOREC professional association. Domains of competence: Database systems, Data warehouses, OLAP and Business Intelligence, Integrated Systems, SOA.

Natural versus Surrogate Keys. Performance and Usability

Dragos-Paul POP
Academy of Economic Studies, Bucharest, ROMANIA
dragos_paul_pop@yahoo.com

Choosing the primary key for a table proves to be one of the most important steps in database design. But what happens when we have to pick between a natural or a surrogate key? Is there any performance issue that we must have in mind? Does the literature have a preferred pick? Is usability a concern? We'll have a look at the advantages and disadvantages of both natural and surrogate keys and the performance and usability issues they address.
Keywords: *primary keys, natural keys, surrogate keys, superkey, candidate key, unique key, performance, usability.*

1 Introduction

Choosing a primary key is really important because it affects the database at the performance and usability levels. The literature speaks of both natural and surrogate keys and gives reasons for choosing one over the other.

Before we get to talk about natural and surrogate keys in a relational transactional table, we must define a few other key concepts used in the relational database model architecture. The concepts to be defined are the superkey, the candidate key, the unique key and the primary key.

2 Key concepts

The superkey is defined as being a set of attributes of a given table that verifies one main condition. The condition in question is that there are no two distinct tuples in that table with an identical value for the superkey set. Also, the attributes comprising the superkey set are said to be functionally dependent. This makes true the following statement: if S is a superkey for the relation R, than the cardinality of the projection of R over S is the same as the cardinality of R. After a table has gone through normalization, we can say that all its attributes form a superkey, because there are no two tuples that are identical for all the values of the set.

From the superkey concept, we can define the candidate key. This is sometimes called a minimal superkey, because a candidate key is, in fact, a minimal set of attributes necessary to uniquely identify a tuple. In other words, a set of attributes is said to be a candidate key if there are no two tuples with the same value for the key and there is no other subset of these attributes that can form a candidate key. This is where the "minimal" property of the candidate key derives from. A table can have multiple candidate keys.

Another concept related to the superkey is the unique key. Again, just like a superkey and a candidate key, the unique key can uniquely identify each row in a table. Although this is not a rule, unique keys tend to only comprise a single column. The difference between candidate keys and unique keys is that, in practice, unique keys do not enforce the NOT NULL constraint. This means they can contain the NULL value and still uniquely identify table rows. Why? Because of the way NULL is treated by the database management systems. NULL is not a value, but the absence of a value, so the unique key concept holds true even for rows with NULL for the unique key. This is because identification of two equal keys is done based on their values, and since NULL is not a value, two keys containing NULL are not considered to be equal. This is by no means to be taken as a rule, because it differs in implementation across database management systems. A better definition of a unique key is that two

tuples cannot have the same value for the unique key if NULL values are not used. So, a unique key only uniquely identifies rows that contain a value other than NULL for the key. As for the candidate key, a table can have multiple unique keys.

The primary key is probably the most important concept in database design. A primary key is, basically, one of the candidate keys in a table. It is a unique key that does not contain (and never will) NULL values can also be made a primary key. For some tables, even a superkey can be a primary key (but that is a little odd). So how and why is a primary key different form all the others? A table can only have one primary key and this key is the preferred way of identifying individual tuples.

3 Natural and surrogate keys

Choosing the primary key has proven to be the difficult part in database design. This is because there are two types of primary keys: natural and surrogate.

The natural key, also called a domain key or an intelligent key, is a candidate key that is logically related to the table. That is, it has business meaning, or business value. It is something that can be found in nature, it makes sense.

A natural key is a single column or set of columns that uniquely identifies a single record in a table, where the key columns are made up of real data. When I say "real data" I mean data that has meaning and occurs naturally in the world of data.

A natural key is a column value that has a relationship with the rest of the column values in a given data record. Here are some examples of natural keys values: Social Security Number, ISBN, and TaxId.

On the other hand, the surrogate key is not derived from real data; it does not have any business meaning or logic. It is a key most often generated by the database or made up using an algorithm.

A surrogate key like a natural key is a column that uniquely identifies a single record in a table. But this is where the similarity stops. They are keys that don't have a natural relationship with the rest of the columns in a table. The surrogate key is just a value that is generated and then stored with the rest of the columns in a record. The key value is typically generated at run time right before the record is inserted into a table. It is sometimes also referred to as a dumb key, because there is no meaning associated with the value. Surrogate keys are commonly a numeric number.

An important distinction between a surrogate and a primary key depends on whether the database is a current database or a temporal database. Since a current database stores only currently valid data, there is a one-to-one correspondence between a surrogate in the modeled world and the primary key of some object in the database. In this case the surrogate may be used as a primary key, resulting in the term surrogate key. In a temporal database, however, there is a many-to-one relationship between primary keys and the surrogate. Since there may be several objects in the database corresponding to a single surrogate, we cannot use the surrogate as a primary key; another attribute is required, in addition to the surrogate, to uniquely identify each object.

Authors have argued that a surrogate should have the following characteristics:

- the value is unique system-wide, hence never reused
- the value is system generated
- the value is not modifiable by the user or application
- the value contains no semantic meaning
- the value is not visible to the user or application
- the value is not composed of several values from different domains

In practice, the surrogate key is frequently a number generated by the database management system. For example, Oracle uses sequences to accomplish this task, while SQL server gives the "identity

column" option. PostgreSQL users have the "serial" option, and MySQL ones use an auto_increment attribute. Having the key independent of all other columns insulates the database relationships from changes in data values or database design (making the database more agile) and guarantees uniqueness.

4 Surrogate Key Implementation Strategies

There are several common options for implementing surrogate keys:

- Key values assigned by the database. Most of the leading database vendors – companies such as Oracle, Microsoft and IBM – implement a surrogate key strategy called incremental keys. The basic idea is that they maintain a counter within the database server, writing the current value to a hidden system table to maintain consistency, which they use to assign a value to newly created table rows. Every time a row is created the counter is incremented and that value is assigned as the key value for that row. The implementation strategies vary from vendor to vendor, sometimes the values assigned are unique across all tables whereas sometimes values are unique only within a single table, but the general concept is the same.
- MAX() + 1. A common strategy is to use an integer column, start the value for the first record at 1, then for a new row set the value to the maximum value in this column plus one using the SQL MAX function. Although this approach is simple it suffers from performance problems with large tables and only guarantees a unique key value within the table.
- Universally unique identifiers (UUIDs). UUIDs are 128-bit values that are created from a hash of the ID of your Ethernet card, or an equivalent software representation, and the current datetime of your computer system. The algorithm for doing this is defined by the Open Software Foundation (www.opengroup.org).
- Globally unique identifiers (GUIDs). GUIDs are a Microsoft standard that extend UUIDs, following the same strategy if an Ethernet card exists and if not then they hash a software ID and the current datetime to produce a value that is guaranteed unique to the machine that creates it.
- High-low strategy. The basic idea is that your key value, often called a persistent object identifier (POID) or simply an object identified (OID), is in two logical parts: A unique HIGH value that you obtain from a defined source and an N-digit LOW value that your application assigns itself. Each time that a HIGH value is obtained the LOW value will be set to zero. For example, if the application that you're running requests a value for HIGH it will be assigned the value 1701. Assuming that N, the number of digits for LOW, is four then all persistent object identifiers that the application assigns to objects will be combination of 17010000,17010001, 17010002, and so on until 17019999. At this point a new value for HIGH is obtained, LOW is reset to zero, and you continue again. If another application requests a value for HIGH immediately after you it will be given the value of 1702, and the OIDs that will be assigned to objects that it creates will be 17020000, 17020001, and so on. As you can see, as long as HIGH is unique then all POID values will be unique.

The fundamental issue is that keys are a significant source of coupling within a

relational schema, and as a result they prove difficult to refactor. The implication is that you want to avoid keys with business meaning because business meaning changes. However, at the same time you need to remember that some data is commonly accessed by unique identifiers, for example customer via their customer number and American employees via their Social Security Number (SSN). In these cases you may want to use the natural key instead of a surrogate key such as a UUID or POID. [3]

5 Tips for Effective Keys

How can you be effective at assigning keys? Consider the following tips, by Scott W. Ambler [3]:

- Avoid "smart" keys. A "smart" key is one that contains one or more subparts which provide meaning. For example the first two digits of an U.S. zip code indicate the state that the zip code is in. The first problem with smart keys is that have business meaning. The second problem is that their use often becomes convoluted over time. For example some large states have several codes, California has zip codes beginning with 90 and 91, making queries based on state codes more complex. Third, they often increase the chance that the strategy will need to be expanded. Considering that zip codes are nine digits in length (the following four digits are used at the discretion of owners of buildings uniquely identified by zip codes) it's far less likely that you'd run out of nine-digit numbers before running out of two digit codes assigned to individual states.
- Consider assigning natural keys for simple "look up" tables. A "look up" table is one that is used to relate codes to detailed information. For example, you might have a look up table listing color codes to the names of colors. For example the code 127 represents "Tulip Yellow". Simple look up tables typically consist of a code column and a description/name column whereas complex look up tables consist of a code column and several informational columns.
- Natural keys don't always work for "look up" tables. Another example of a look up table is one that contains a row for each state, province, or territory in North America. For example there would be a row for California, a US state, and for Ontario, a Canadian province. The primary goal of this table is to provide an official list of these geographical entities, a list that is reasonably static over time (the last change to it would have been in the late 1990s when the Northwest Territories, a territory of Canada, was split into Nunavut and Northwest Territories). A valid natural key for this table would be the state code, a unique two character code – e.g. CA for California and ON for Ontario. Unfortunately this approach doesn't work because Canadian government decided to keep the same state code, NW, for the two territories.
- Your applications must still support "natural key searches". If you choose to take a surrogate key approach to your database design you mustn't forget that your applications must still support searches on the domain columns that still uniquely identify rows. For example, your Customer table may have a Customer_POID column used as a surrogate key as well as a Customer_Number column and a Social_Security_Number column. You would likely need to support searches

based on both the customer number and the social security number. Searching is discussed in detail in Best Practices for Retrieving Objects from a Relational Database.

- Don't naturalize surrogate keys. As soon as you display the value of a surrogate key to your end users, or worse yet allow them to work with the value (perhaps to search), you have effectively given the key business meaning. This in effect naturalizes the key and thereby negates some of the advantages of surrogate keys. [3]

6 Advantages and disadvantages

Of course, there are a lot of advantages and disadvantages of using natural or surrogate keys. Authors are divided between the two strategies. Below there is a listing of pros and cons of using both natural and surrogate keys, as Gregory A. Larsen list them:

6.1 Surrogate Key Pros and Cons

A definite design and programming aspect of working with databases is built on the concept that all keys will be supported by the use surrogate keys. To understand these programming aspects better, review these pros and cons of using surrogate keys. [4]

Pros:

- The primary key has no business intelligence built into it. Meaning you cannot derive any meaning, or relationship between the surrogate key and the rest of the data columns in a row.
- If your business rules change, which would require you to update your natural key this can be done easily without causing a cascade effect across all foreign key relationships. By using a surrogate key instead of a natural key the surrogate key is used in all foreign key relationships. Surrogate keys will not be updated over time.
- Surrogate keys are typically integers, which only require 4 bytes to store, so the primary key index structure will be smaller in size than their natural key counter parts. Having a small index structure means better performance for JOIN operations.
- It's easy to create a naming system for surrogate keys, so that remembering the primary key of a table can be made a lot easier. [4]

Cons:

- If foreign key tables use surrogate keys then you will be required to have a join to retrieve the real foreign key value. Whereas if the foreign key table used a natural key then the natural key would be already be included in your table and no join would be required. Of course this I only true if you only needed the natural key column returned in your query
- Surrogate keys are typically not useful when searching for data since they have no meaning.
- Surrogate keys have no knowledge level value. The most important function of the PK is as an interaction element between the real-world and the database. It is thorough the primary key that we usually query the database. The primary key is of fundamental importance if we are to "usefully" relate the concepts of the database to the real world. [4]

6.2 Natural Key Pros and Cons

Having natural keys as indexes on your tables mean you will have different programming considerations when building your applications. You will find that pros and cons for natural keys to be just the opposite as the pros and cons for surrogate keys. [4]

Pros:

- They already exist in the schema. There is no need for additional columns that would load the tables.
- Will require less joins when you only need to return the key value of a foreign key table. This is because the natural key will already be imbedded in your table.
- Easier to search because natural keys have meaning and will be stored in your table. Without the natural key in your table, a search for records based on a natural key would require a join to the foreign key table to get the natural key. [4]

Cons:

- Requires much more work to change a natural key, especially when foreign relationship have been built off the natural key.
- Your primary key index will be larger because natural keys are typically larger in size then surrogate keys.
- Since natural keys are typically larger in size then surrogate keys and are strings instead of integers joins between two tables on a natural key will take more time.
- Kind of hard to remember the name of the key for every table in the database [4]

7 Performance issue

The next scenario is built to test the performance of natural and surrogate keys. We will see when and if one is better than the other.

The test business logic is simple and it is about the commercial activity of a company that sells goods. The test entities are described as follows:

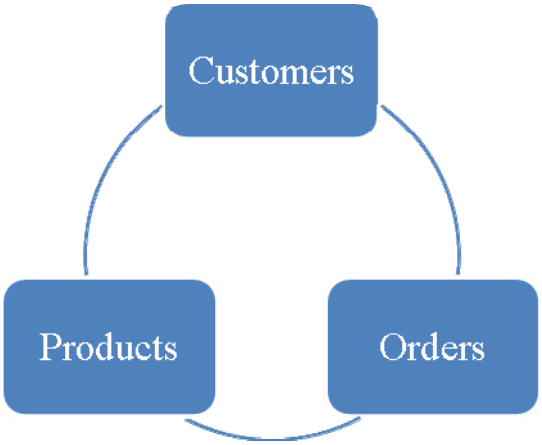

Fig 10. Database logical entities

After undergoing normalization, we get the following database structure:

- Customers
  - This table will hold all the information related to the customers, such as first and last names, email address, telephone number, address and so on
- Products
  - Here we will have details about the products that are being sold: name, price, stock etc.
- Categories
  - This table stores information about different categories of products
- Orders
  - This is the main table that holds information about customer orders, such as order date, serial number, total value
- OrderDetalis
  - The last table is used to store information about individual lines in a customer order: product, quantity, price at buying time

There are two test cases: one in which we will chose a natural key for the primary key of every table and one in which a surrogate key will be used. The tables will be loaded with data and will be tested to see the response times of simple selects and joins. The test database management systems is Oracle 10g Express Edition. The test computer is equipped with an Intel Core i5 750 processor, 4 GB of RAM and two 500

GB hard-disks at 7200 rpm connected in RAID level 0.
The test scenario uses the following table descriptions:

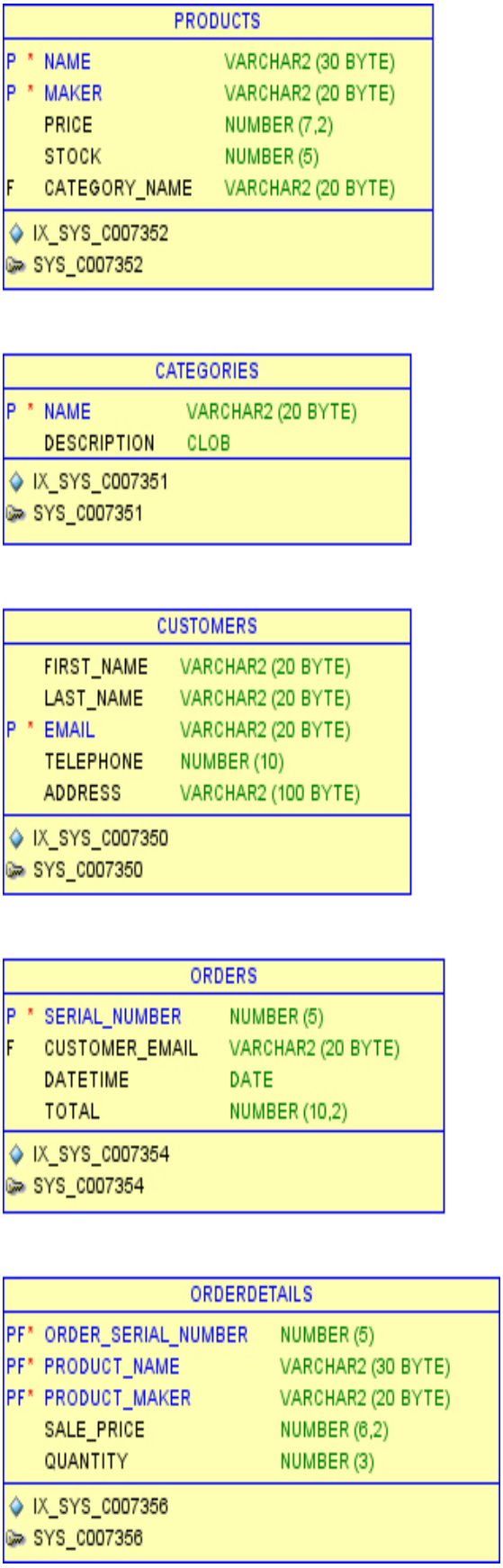

Fig. 11. Natural keys database

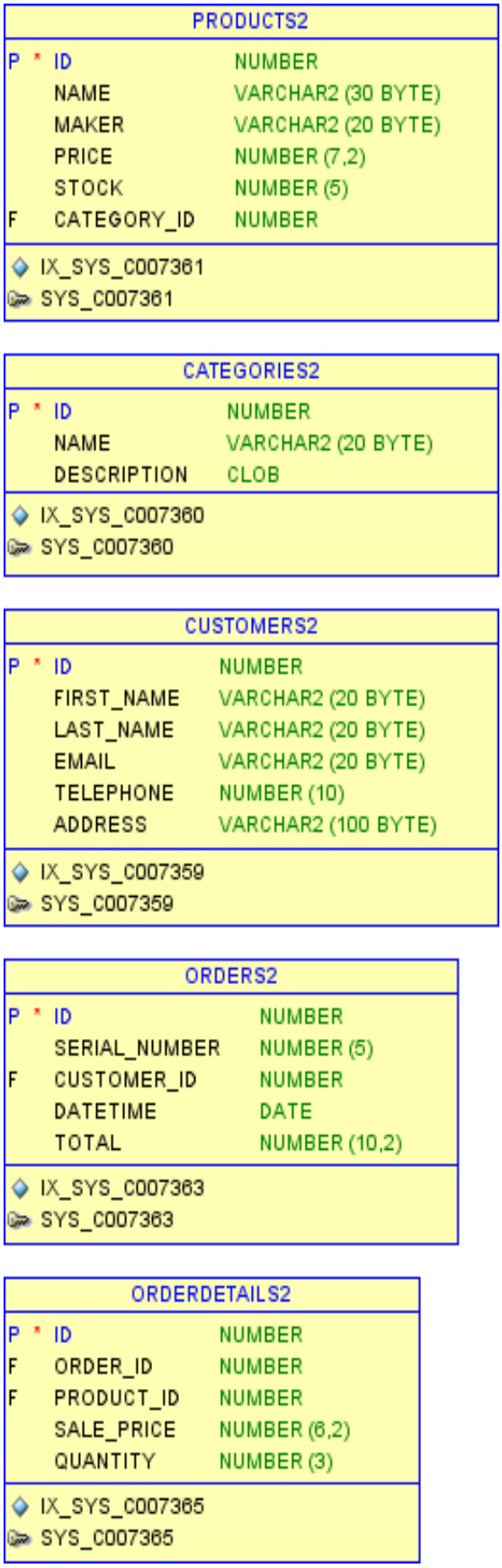

Fig. 12. Surrogate keys database

Table 1. Query results

SQL Query	Execution time
`select * from customers2, products2, categories2, orders2, orderdetails2 where customers2.id = orders2.customer_id and products2.category_id = categories2.id and orders2.id = orderdetails2.order_id and orderdetails2.product_id = products2.id`	18 ms
`select * from customers, products, categories, orders, orderdetails where customers.email = orders.customer_email and products.category_name = categories.name and orders.serial_number = orderdetails.order_serial_number and orderdetails.product_maker = products.maker and orderdetails.product_name = products.name`	20 ms
`select * from orderdetails2`	15 ms
`select * from orders2`	15 ms
`select * from categories2`	15 ms
`select * from products2`	18 ms
`select * from customers2`	14 ms
`select * from orderdetails`	17 ms
`select * from orders`	18 ms
`select * from categories`	28 ms
`select * from products`	18 ms
`select * from customers`	15 ms

8 Conclusions

As we can see from the results above, choosing surrogate keys as primary keys does not always mean adding columns to tables. Also, query times are improved, because primary indexes are smaller. This is due to the fact that surrogate keys use an integer data type, while the natural keys they replaced used a variable length character data type.

In the end, although surrogate keys tend to be better for performance, people still use natural keys just because they feel better. Generally, database designers are inclined to use surrogate keys, because making things abstract is their main issue, while application developers go with natural keys, because they have more business logic.

9

Acknowledgements

This work was cofinaced from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POSDRU/107/1.5/S/77213 „Ph.D. for a career in interdisciplinary economic research at the European standards".

References

[1] Breck Carter, "Intelligent Versus Surrogate Keys". Internet: http://www.bcarter.com/intsurr1.htm, October 6, 1997 [Mar 18, 2011]

[2] Michelle A. Poolet, "SQL by Design: How to Choose a Primary Key". Internet: http://www.sqlmag.com/article/systems-administrator/sql-by-design-how-to-choose-a-primary-key, April 01, 1999 [March 18, 2011]

[3] Scott W. Ambler, "Choosing a Primary Key: Natural or Surrogate?". Internet: http://www.agiledata.org/essays/keys.html, 2005 [Mar 18, 2011]

[4] Gregory A. Larsen, "SQL Server: Natural Key Verses Surrogate Key". Internet: http://www.databasejournal.com/features/mssql/article.php/3922066/SQL-Ser

[5] ver-Natural-Key-Verses-Surrogate-Key.htm, January 31, 2011 [Mar 18, 2011]

[6] Michelle A. Poolet, "Surrogate Key vs. Natural Key". Internet: http://www.sqlmag.com/article/data-modeling/surrogate-key-vs-natural-key/2, January 24, 2002 [Mar 18, 2011]

Dragos-Paul POP graduated from the Faculty of Computer Science for Business Management at the Romanian-American University of Bucharest in 2007 (Bachelor's degree) and in 2009 (Master's degree), specialising in Economic IT. He is curently a Ph.D. candidate at the Academy of Economic Studie in Bucharest. He works as an assistant teacher at the Romanian-American University in Bucahrest, teaching computer architecture, operating systems, advanced web programming and databases. His main domains of interest are web technologies, database technologies, programming languages, networking, hardware and operating system