Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation

Girts Karnitis and Guntis Arnicans
Faculty of Computing
University of Latvia
Riga, Latvia
{girts.karnitis, guntis.arnicans}@lu.lv

***Abstract* — Relational databases remain the leading data storage technology. Nevertheless, many companies want to reduce operating expenses, to make scalable applications that use cloud computing technologies. Use of NoSQL database is one of the possible solutions, and it is forecasted that the NoSQL market will be growing at a CAGR of approximately 50 percent over the next five years. The paper offers a solution for quick data migration from a relational database into a document-oriented database. We have created semi-automatically two logical levels over physical data. Users can refine generated logical data model and configure data migration template for each needed document. Data migration features are implemented into relational database browser DigiBrowser. Real patients' database was migrated to Clasterpoint database. The offered approach provides means to obtain at least proof-of-concept for new document-oriented database solution in a couple of days.**

Keywords - database migration; relational database; NoSQL; document-oriented database; meta model; logical data representation

I. INTRODUCTION

Relational databases (RDM) have been the leading data storage technology for years. Nevertheless, changes in the demands for the data processing have caused emergence of new data storage, retrieval and processing mechanisms. One of such mechanisms is a NoSQL (Not only SQL) database that was introduced to label non-relational and distributed data stores. To reduce operating expenses, managers of many large companies are looking forward to the cloud computing solutions. These solutions help to store data efficiently, to compute the massive amount of data, to provide high scalability, to ensure high performance and availability at low costs. Relation databases are not appropriate for cloud computing, but NoSQL databases are.

The market of NoSQL databases is still small but steady growing. More and more companies are trying to exploit NoSQL databases to improve business. The article [1] describes one such example – company CARFAX that offers online vehicle and valuation website. CARFAX has 13.6 billion records associated with 700 million vehicles, and it was difficult to run a website that uses a relational database. The company is now running a website that is based on NoSQL database MongoDB (108 servers, 10.6 TBs of data, 1.5 billion vehicle documents, the new system works five times faster than legacy one).

Various estimations exist about the future of NoSQL databases market, but all predict a significant increase of share. Company Research and Markets estimated the NoSQL market will be growing at a CAGR of 53.09 percent over the period 2013-2018 [2]. CAGR is an indicator that shows the year-over-year growth rate of an investment over a specified period. The company Allied Market Research forecast a CAGR of 40.8% during 2014 – 2020 and the global NoSQL market is expected to reach $4.2 billion by 2020 [3]. The most significant segments of applications are data storage, metadata storage, cache memory, distributed data depository, e-commerce, mobile apps, web applications, data analytics, social networking.

There are five types of NoSQL databases at present: *column*, *document*, *key-value*, *graph*, *multi-model*. The most important types for some next years are key-value stores and document databases. We concentrate our attention on document databases and call them *document-oriented databases* (several notations exit – *document database*, *document-oriented database*, *document store*). There is a range of document-oriented databases, for instance, *MongoDB*, *Couchbase server*, *CouchDB*, *MarkLogic*, *Cloudant*, *Cassandra*, and *Clusterpoint database*.

Days and sometimes weeks must be spent to convert data from RDB to NoSQL data for the first time. Nowadays many companies want to test whether NoSQL based solution is suited to their needs and allocate some resources to explore technologies and create NoSQL based proof-of-concept. Many resources must be allocated to create data conversion, and no resources are left to create and test functionality.

The problem of migrating data from relational databases to document-oriented databases is addressed in this paper. The proposed technology or ideas can help companies to migrate their legacy systems based on RDB to the document-oriented database. We approbated our ideas by adding new features into the relational database browser *DigiBrowser* (http://digibrowser.com). At least companies can use the DigiBrowser for fast creating of proof-of-concept before starting the migration project.

The remainder of the paper is organized as follows. Section 2 reveals data migration problem from legacy relation database into a document-oriented database or other NoSQL database. Related works are given. Section 3

978-1-4673-7016-5/15 $31.00 © 2015 IEEE
DOI 10.1109/CICSyN.2015.30

explains our approach for data denormalization and creating templates that define logical objects. Section 4 shows how data is transformed according to created definitions. Results of approbation are in Section 5. Finally, we give conclusions.

II. Data Migration Problem and Solutions

Data accumulated by information systems is one of the important assets for most of the companies. Pushed by customer demands and pulled by changes in technologies, companies from time to time migrate from one information system to another. Hence, data from the legacy system has to be migrated to the new system. Despite the significance of this process, knowledge about migration process is limited [4]. F. Matthes and C. Schulz in [4] fixed a state of the art and provided a literature review of the data migration problem.

F. Matthes and C. Schulz define the term *data migration* as follows: "Tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on a conceptual and/or physical level".

Data migration has two important steps: first, restructuring of source data according to requirements of the target system, and second, transferring data from the source to the target database. The academic literature offers several approaches/methods for dealing with these steps: *schema conversion*, *meta-modeling approach*, *ETL* (Extract, Transform, Load), *program conversion*, *model-driven migration*, *automated data migration*.

Theoretical research for RDB to XML/JSON transformation was done when the XML format appeared, but as stated in [5] conversion standard does not exist yet. Two types of transformation approaches exist.

The first type of approaches creates XML document without redundancy and preserving all constraints. All data from RDB to XML is converted. In this approach at first XML schema is generated from RDB, and then data is translated. Main emphasis of authors of [6] is to preserve all RDB constraints and avoid redundancies. Different models are used to enrich semantic of the relational schema of an existing database in the first step; data transformation using the model created is in the second step. Authors of [7] use the so-called Canonical Data Model (CDM). Authors of [8] use extended entity relationship (EER) schema. Approaches mentioned above generate very flat XML structure. Some research is done to create nesting XML schemas, for example [9].

The second type of approaches is a manual creation of conversion mapping from the existing relational database to the existing XML schema. Tools exist to map the relational database to XML schema, for example, Altova XMLSpy.

Our approach is close to the meta-modelling approach [10]. Data conversion from RDB to NoSQL consists of two steps. The first step is to get the physical data structure from the existing relational database and to raise the semantic level of achieved data structure (creating of the so-called *meta-model*). Conversion scope is also defined. The second step is to convert data from the relational database to NoSQL in compliance with the created data structures (meta-model) and conversion scope.

We derived our solution from the experience of building browsers for relational databases. Paper [11] contains a description of basic ideas and implementation of the browser.

The browser connects to database, reads its schema, and creates data browsing model based on obtained metadata. After that, the user can start data browsing. The user perceives data as a graph. The tool provides the possibility to traverse from one node (data item) to another and look at related data at any moment. In order to get details of data or to move to the other place, the user clicks on links similarly to browsing web pages. The browser can be used by IT specialists as well as by users without specific IT knowledge. It is allowed to modify and customize browsing model. Thus, after customization the tool is more like a simple information system that allows to traverse throughout database, see additional calculated information, and even change data [12].

III. Data Structure Denornmalization

We use as an example a relational database schema showed in Fig. 1.

A. Physical level of data

Let us consider that a university application stores data in a relational database with the given data schema. *Logical objects* (*business concepts* [4]) used in our example are *Course*, *Teacher*, *Student*, *Lecture*, *Examination*, *Grade*, *Attendance*. Data are with complex structure and tightly connected. For instance, a Grade not only contains information about the level of the grade, but also specifies what student has received the grade of a particular course and what teacher has granted the grade. The object Grade is a complex object at the logical level. On the other hand, the Grade is a table in the relational database that contains only essential data of the object Grade. Developer defined *views* might exist in the relational database. The view often is some logical object that is formed from several physical tables and can be used as part of the target document.

A part of relational database metadata used for data conversion to NoSQL is shown in Fig. 2. A relation database contains *Table-like-structures*: *Tables* and *Views*. The Table-like-structure has a *Name*. Tables and Views do not differ from the data translation perspective. The Table-like-structure contains *Attributes* (columns). Each Attribute has a *Name* and *Data type*. A Table has *constraints*. *Primary Keys* (*PK*), *Unique Keys* (UK) and *Foreign Keys* (*FK*) are essential for data translation because these constraints define the relation between tables and records.

Data in a database is usually normalized; therefore, one logical level object is stored in a bunch of related database tables. A relational model is a low-level physical model. It shows how data is stored but does not show the natural relationship of data or full view of complex data objects.

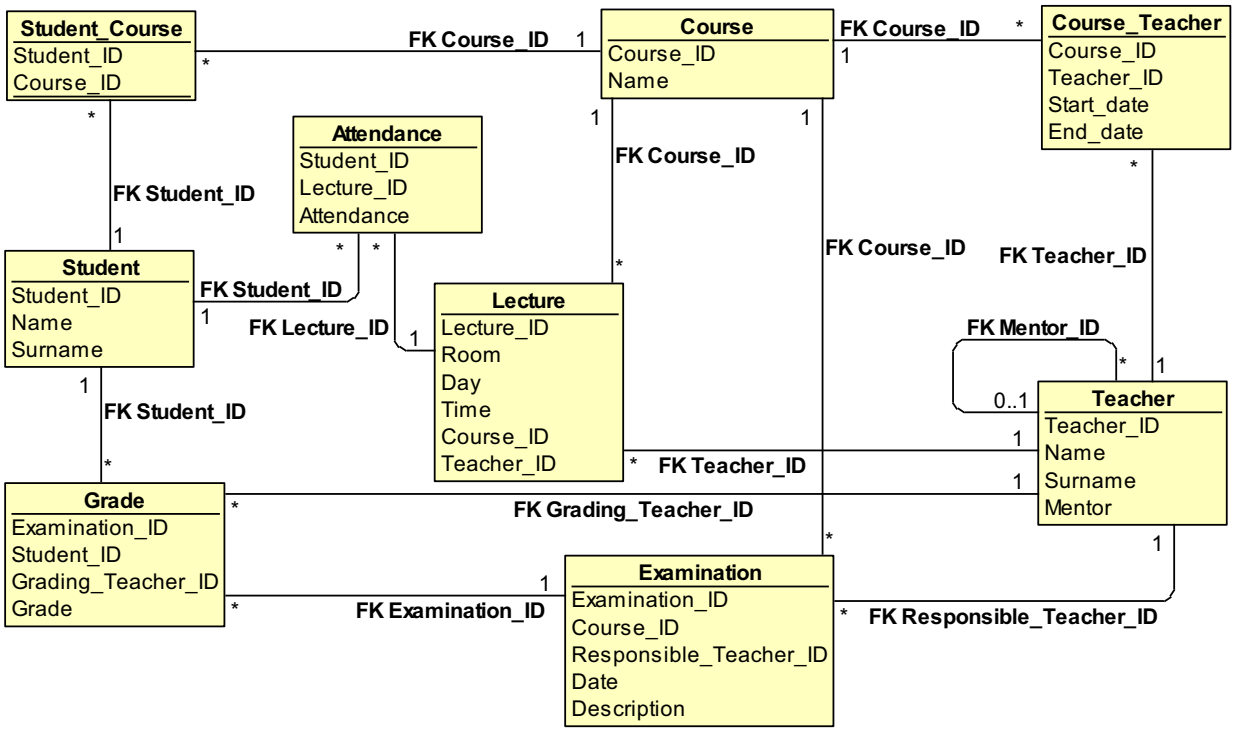

Figure 1. Physical data model of the example database

N:N type relations cannot be naturally defined in the relational model. Instead, a specific table is created for each relation. These tables are normal database tables with FK to the related data tables. Sometimes additional information is stored in such table. For instance, a teacher instructs courses for known time periods. Tables Teacher, Course, Course_Teacher contains all teaching facts (*Teacher* teaches *Course* from *Start_date* till *End_date*). It is hard to tell whether table Course_Teacher contains natural objects or only relations between Teacher and Course. Thus, we cannot automatically determine a semantic of distinct tables or group of tables.

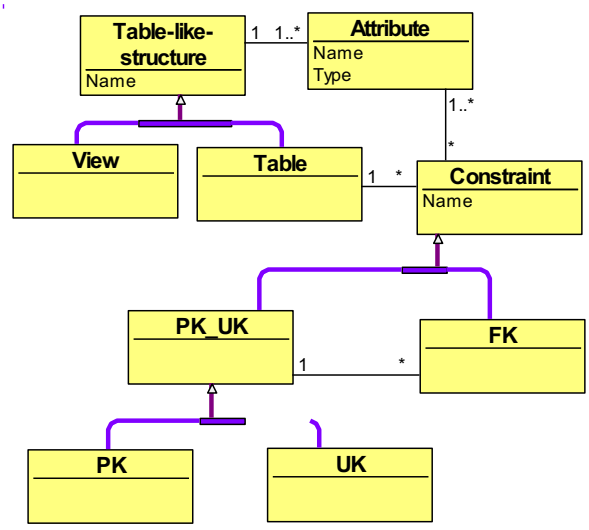

Figure 2. Part of relational database metadata

B. *First Logical Level of Data*

The logical level shows how we perceive data in logical units or entities, but not how data is physically stored in a database. ER models and modeling languages are used to show logical data model. The main differences between logical and physical data models are the following: in logical data model the names of tables and attributes are created in natural language; there are no specific codifier tables, but codified values are included in the main data objects; N:N relations are allowed; there are no surrogate keys, each entity has a primary key that can be combined from more than one attribute and can have attributes with long data type, for example, text.

We supplement the physical model with additional information. That allows us to show data at the logical level and convert data into logical documents. The metamodel containing information about physical and logical levels is shown in Fig. 3. Let us consider that this model describes a *first logical level of data.*

The *Table-like-structure* is the main object. That is analogous to the Table-like-structure in physical model. The Table-like-structure contains two types of attributes – *Physical attributes*, which come from a database, and *Virtual attributes,* which are calculated from other attributes in the moment of data processing. The objects in the metamodel

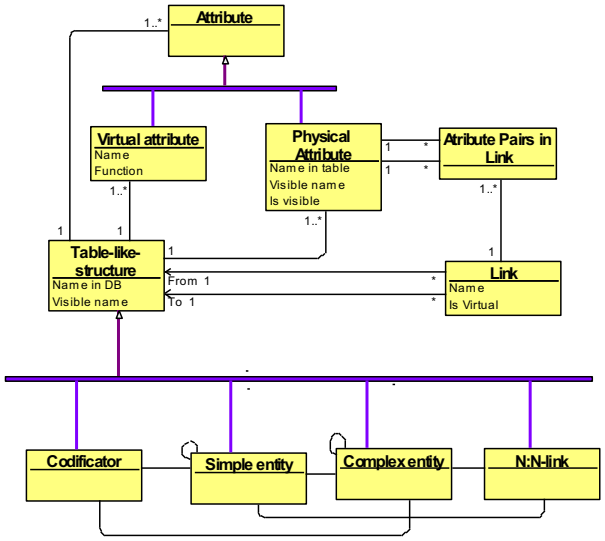

Figure 3. Part of logical model metadata

have their physical names in the relational database (*Name in table*), as well as logical name to be used during data processing (*Visible name*). By default, all logical names are the same as corresponding physical names until user change them.

Relations (*Link*) link together the Table-like-structure objects. Relations contain attributes that define which attribute names must be equal in the related Table-like-structures (like in FK-PK relationships). Physical relations are defined in a database (Is_Virtual has value *false*); virtual relations are additional relations defined only in the logical model (Is_Virtual has value *true*). Both types of relations are used for navigation. The relationship is used only for one way navigation in each particular moment. Therefore, each relationship can be perceived as two one-way relationships. That allows us to define relations with any cardinality (1:1, 1:N, N:N).

The Table-like-structure class has four subclasses: *Codifier*, *Simple entity*, *Complex entity*, *N:N-link*. Codifier contains code, as well as coded data (there are no tables with type Codifier in the example model; a sample is a table with colors that contains fields *id* and *color*). A Simple entity contains information about simple objects that are stored in one table (for instance, Course and Student). A Complex entity contains information about a complex object that is stored logically in more than one table (for instance, Grade, Lecture, Teacher, Attendance, and Examination). N:N-link table is used to code N:N relationships in a relational database (for instance, Course_Teacher, Student_Course).

The semi-automatic algorithm to determine Table-like-structure type exists [13]. We use this algorithm to build logical objects from tables in the relational database.

C. *Second Logical Level of Data*

The next step is obtaining a *second logical level of data* for each target document. The Table-like-structure (TLS) is taken as a basis for the document. Let us call this TLS as *tls_in_focus*. First, we automatically create tree from Table-like-structures that serves as a *template draft* for document generation. Second, we refine the created tree by adding additional Table-like-structures and removing excessive Table-like-structures. At last, we choose what attributes are excluded from the document (flag *is_visible* is set to *false* for the chosen attributes).

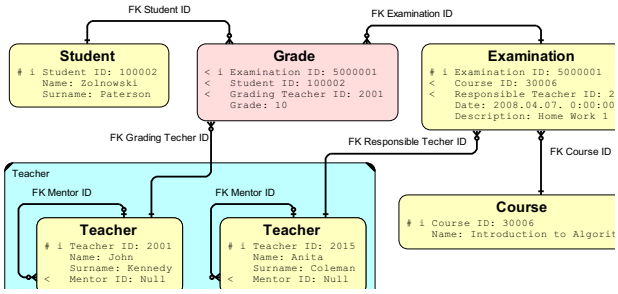

Figure 4. Example how business concept *Grade* might be formed

Template draft can be generated by various algorithms. A simple approach is demonstrated in Fig. 4. Let us choose the TLS Grade as a tls_in_focus. We build a graph recursively by adding as nodes TLSs that are on the opposite side of links with cardinality 1 on the opposite side. First, Grade adds Student, Teacher, and Examination. Second, Examination adds Course and another instance of Teacher.

The described approach usually collects essential data of the logical object. However, the target document may require more data. For instance, a document based on Student can require data about attended lectures and earned grades. Algorithm in Fig. 5 describes another approach. The idea behind algorithm *dfs_selecting* is traverse model graph by principle to add TLS as tree nodes while a path from the root to any leave contains unique TLS. This algorithm is a combination of depth-first search in graph and backtracking algorithm. Unfortunately, this approach usually adds too many TLS that are weakly related to the tls_in_focus.

At time of writing this paper, we use the algorithm given in Fig. 6. The idea is to exploit each link between TLSs not more than one time, and TLS is visited (added) based on the width-first graph search algorithm. Any link is allowed to be used only in one direction.

We search for another tree building algorithms,

```
dfs_selecting(parent_node, used_tls)
  used_tls.add(parent_node.tls)
  for each constraint of parent_node.tls
    related_tls = get_related_tls(parent_node.tls, constraint)
    child_node = create_tree_node(related_tls)
    parent_node.add_child(child_node)
    dfs_ selecting (child_node, used_tls)
  used_tls.remove(parent_node.tls)

// Sample of the function call
used_tls = create_list()
root_node = create_tree_node(tls_in_focus)
dfs_selecting(root_node, used_tls)
```

Figure 5. Algorithm *dfs_selecting* for a draft of document template

```
bfs_selecting(root_node, used_links)
  queue = create_queue()
  queue.enqueue(root_node)

  while not queue.is_empty
    parent_node = queue.degueue()
    for each link of parent_node.tls
      related_tls = get_related_tls(parent_node.tls, link)
      if not used_links.find(link)
        used_links.add(link)
        child_node = create_tree_node(related_tls)
        parent_node.add_child(child_node)
        queue.enqueue(child_node)

// Sample of the function call
used_links = create_list()
root_node = create_tree_node(tls_in_focus)
bfs_selecting(root_node, used_links)
```

Figure 6. Algorithm *bfs_selecting* for a draft of document template

particularly, by using TLS types Codifier, Simple entity, Complex entity, N:N-link table.

The document template refining is performed manually by experts. Templates can be created for any desirable tls_in_focus. Many templates may be defined for each tls_in_focus, for instance, with few related TLSs for concentrated documents and with many related TLSs for very detailed documents.

IV. DATA TRANSFORMATION

Data transformation and document generation are based on the created template. Queries to the relational database are executed. All queries are generated according to template and metamodel transformations.

Final document structure and syntax depend on the target document-oriented database. Various algorithms can be created for desirable document format. As an example, we offer a simple algorithm that exploits TLS types (Fig. 7). Essential data are TLS names, attribute names and attribute values (field values in a relational database). This information is printed by command *print*. To show sublevel, we print '{' for sublevel start and '}' for sublevel end.

Pseudo-command "*for each* link *from* parent_node *to* child_node *where* child_node *is child of* parent_node" allows to find such links between two TLS that correspond to the document template, and to iterate through these links.

The given algorithm does not exploit link names (it is possible to give a name to any link in the logical model). Using link names can make the document more readable. For instance, document Student can include Teacher data in several roles: a person who reads a lecture, a person who is responsible for an examination, and a person who grades the examination.

V. RESULTS

We have implemented the ideas described in the paper into tool DigiBrowser. Initially, DigiBrowser was designed only for relation database browsing, and data migration to

```
dfs_printing(record, parent_node)
  if not parent_node.type = "N:N-link"
    print parent_node.tls.visible_name
  if child_node.type = "Simple entity" or
                                child_node.type = "Complex entity"
    print "{"

    for each field of record
      if field.is_constraint
        for each link from parent_node to child_node
                          where child_node is child of parent_node
          if field.is_visible
            print field.name, field.value
          related_records = get_related_records(record, link)
          for each related_record of related_records
            dfs_printing(related_record, child_node)
      else
        if field.is_visible
          print field.name, field.value

  if child_node.type = "Simple entity" or
                                child_node.type = "Complex entity"
    print "}"

// Sample of the function call
dfs_printing(record_in_focus, root_node)
```

Figure 7. Sample algorithm for document creation

NoSQL database is an additional feature. Actually, instead of data browsing on the screen we write data to the file.

DigiBrowser can connect to any relational database stored in *Oracle Database*, *Microsoft SQL Server*, *MySQL*, *PostgreSQL*, and a browsing and traversing can be performed immediately. It is possible to connect to *Virtuoso Universal Server* (RDF data browsing) and *XML file*. For more details see DigiBrowser homepage (http://digibrowser.com).

Gints Ernsons, CEO of Clasterpoint Ltd (www.clusterpoint.com) says: "We used Digibrowser to denormalize a complex hospital IT system from legacy SQL data containing 100 tables, 500 columns and 1 million patients data into Clusterpoint XML database model creating a single e-health record per patient in less than two days. DigiBrowser works for us like a charm!". The same data migration was performed for MongoDB. Another data formats supported by DigiBrowser are XML and JSON. These formats allow migrating to a new system that supports XML or JSON documents.

We have measured patients' data conversation to XML and JSON formats. The desktop computer used for measurement has the following characteristics: Intel i7-4600 2.1Ghz, RAM 8GB, SSD disk 256GB SAMSUNG MZ7TE256HMHP, MS Windows 8.1 64bit, MS SQL Server 2008 with patients' database. The total number of patients was 1.5 million. Total database size was 4.2GB. Document extraction templates took data from 10.5 million rows. All conversation lasted 100 minutes (approximately 15 thousand documents per minute). Of course, document generation

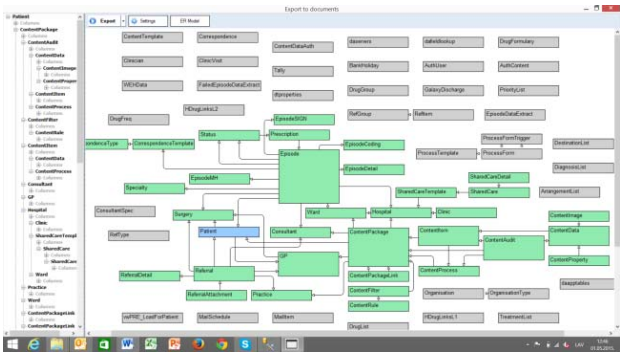

Figure 8. Source data model for patient's document

speed depends on database structure, size and document template.

A screenshot of medical data migration is shown in Fig. 8. The left part contains document template tree. The right part contains corresponding logical model. The blue rectangle is a tls_in_focus, and green rectangles are other TLSs that belong to the document.

VI. CONCLUSIONS

The offered data migration from a relational database into the document-oriented database can be used in practice. We included data conversion features into relational database browsing tool DigiBrowser. Developers of document-oriented database Clusterpoint approbated DigiBrowser and converted legacy patients' database to 1.5 million patients' documents in a couple of days. IT specialists were not familiar with the given database and were forced to explore the database by DigiBrowser before creating the logical data model and defining of converting templates.

If the relational database is not huge, it is possible make migration of the relational database on a daily basis. NoSQL database can provide additional services for the legacy system such as better information searching and presentation.

Further studies have to be done to improve given methods. Various new algorithms might be created for recognizing table types, defining document structure, choosing fields for migration, and forming target documents.

ACKNOWLEDGMENT

The research leading to these results has received funding from the research project "Information and Communication Technology Competence Center" of EU Structural funds, contract nr. L-KC-11-0003 signed between *ICT Competence Centre* and *Investment and Development Agency of Latvia*, Research No. 1.8 "Technology for denormalisation of relational data bases"

REFERENCES

[1] B. Butler, "The NoSQL database market is a small but increasingly important segment of the database market, and it's giving SQL database vendors and users a scare", Network World, Oct 27, 1014, http://www.networkworld.com/article/2839141/cloud-computing/nosql-takes-the-database-market-by-storm.html

[2] Research and Market, "Global NoSQL Market 2014-2018", Report ID 2776270, Mach 2014, http://www.researchandmarkets.com/reports/2776270/global_nosql_market_20142018

[3] Allied Market Research, "Global NoSQL Market (Type, Application, Verticals and Geography) Size, Global Trends, Company Profiles, Demand, Insights, Analysis, Research, Report, Opportunities, Segmentation and Forecast, 2013 - 2020", Report, April 2015, http://www.alliedmarketresearch.com/NoSQL-market

[4] F. Matthes and C. Schulz, "Towards an integrated data migration process model", Software Engineering for Business Information Systems (sebis), 2011.

[5] B. Mohamed and A. Elalami, "The migration of data from a relational database (RDB) to an object relational (ORDB) database", Journal of Theoretical and Applied Information Technology, 20th December 2013. Vol. 58 No.2 ISSN: 1992-8645 ,E-ISSN: 1817-3195.

[6] P. Paolo and E. Quintarelli, "An algorithm for generating XML Schemas from ER Schemas." In Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems, SEBD, 2005, pp. 192-199.

[7] M. Abdelsalam, A. Ali, and N. Rossiter, "An integrated approach to relational database migration", In the Proceeding of International Conference on Information and Communication Technologies-2008 (IC-ICT'08), pages 01-06, Bannu, Pakistan (2008).

[8] F., Joseph, et al. "Translating relational schema with constraints into XML schema." International Journal of Software Engineering and Knowledge Engineering 16.02, 2006, pp. 201-243.

[9] D. Lee, M. Mani, F. Chiu, and W. Chu, „NeT & CoT: translating relational schemas to XML schemas using semantic constraints, in: Proceedings of the 11th CIKM, 2002, pp. 282-291.

[10] M. Jeusfeld, and U. Johnen, "An executable meta model for re-engineering database schemas", (P.Loucopoulos, Ed.) International Journal on Cooperative Information Systems, 4(2-3), 1995, pp. 473-483, Springer Berlin Heidelberg.

[11] G. Arnicans, G. Karnitis, "Prototype for Traversing and Browsing Related Data in a Relation Database," in: Barzdins (ed.), Scientific Papers University of Latvia, Vol 756, Computer Science and Information Technologies, University of Latvia, 2010, 59-74.

[12] G. Arnicans, G. Karnitis, "DigiBrowser: Tool for Relational Data Browsing and Traversing,", 2012, unpublished, http://digibrowser.com/download.php?id=DIGIBrowser_ToolForRelationaDataBrowsingAndTraversing&page=documentation

[13] G. Arnicans, "Application generation for the simple database browser based on the ER diagram," in Janis Barzdins, editor, Databases and Information Systems, Proceedings of the Third International Baltic Workshop, Volume 1, pages 198-209. Riga, 1998.