Language Integrated Query as a Canonical Data Model for Virtual Data Integration

Vladimir Klyuchikov

Lomonosov Moscow State University, Moscow, Russia
kluchvlad@gmail.com

Abstract. Nowadays data using by organizations in different business areas are very heterogeneous. This raises the issue of data integration. Two main classes of data models used for data representation can be distinguished: relational (SQL) and non-relational (NoSQL) data models. Data models of the classes differ a lot, for instance, relational models are applied for structured data, and NoSQL models mostly applied for semi-structured data. For the issues of data integration it is required to find a model that can unify relational and NoSQL models. A candidate for such unifying data model is Language Integrated Query – LINQ. The aim of this work is to validate that LINQ can successfully serve as unifying data model in data integration systems intended to integrate both relational and NoSQL data sources.

Keywords: Virtual Data Integration, Canonical Data Model, Language Integrated Query

1 Introduction

Currently, there is an exponential increase of the volume of experimentally obtained data in science and industry. The data can be obtained from various sources. For example, researchers can get scientific data from sensors during the experiments. Credit banks capture transactions each borrower and generate a credit history, that can be used in advance. The list of users and personal information, feedbacks and stories in social networks are also a data that continuously increase volume. The number of organizations that get the emerged data in different areas is also large. Data that is stored in various sources like web log files, web pages, documents, etc. possess different levels of structurization: structured, semi-structured, and unstructured data. Data in sources can be presented in various data models. Two large classes of data models can be distinguished: relational (mostly SQL) and NoSQL data models. NoSQL is a group of models with flexible schemas that can be further classified in four main categories: key-value, column-oriented, document and graph data models. These models are intended to represent semi-structured data or even schema-less data. NoSQL models are implemented in respective database management systems. To deal with heterogeneity of data the respective data integration methods and tools are required.

Frequently, the data sources are scattered and getting them directly from sources and processing is quite problematic. That data can be localized in *data lakes* [1] that are

repositories for large quantities and varieties of structured or unstructured data. In data lakes the data are stored as-is: no initial structuring or transforming data are presumed. Different kinds of analytics – from dashboards and visualizations to big data processing, real-time analytics, and machine learning – are provided over data lakes to guide better decisions. Data lakes can be created for specific cases, such as analytics, machine learning, real-time data movements or on-premises data movements.

Even if the problem of localization of the initial data is solved, the problem of data integration from different sources remains urgent. Data integration requires the following conceptual specifications: *a global schema* and *mappings binding the global schema and source schemas*. The global schema is the integrated schema serving as a unified representation of schemas of participating data sources. Data integration methods are implemented within data integration systems (DIS).

Two kinds of data integration can be distinguished: materialized and virtual integration. *Materialized data integration* usually proceeds within *data warehouses* [13]. Each source can possess a schema that differs from the warehouse (global) schema. The data are reshaped into global schema using Extract-Transformation-Load (ETL) processes (that implement conceptual mappings) and stored (materialized) into the warehouse database.

Virtual DISs developed for concrete subject areas are called *subject mediators* (mediators, in short). To have access to "fresh" information, a virtual integration system is preferred to a warehouse since it avoids having to propagate updates of the data source to the warehouse. The process of answering user queries in virtual DIS is performed as follows [3]:

- a user poses a query in terms of global schema;
- the query is rewritten into a set of subqueries, each subquery is formulated in terms of some source schema;
- each subquery is passed to a specific *wrapper* of the relevant data source to be executed there;
- answers returned from the wrappers are collected, combined and returned to the user.

Three main techniques for definition of conceptual mappings between global and local schemas for virtual integration are known: Global-as-View (GAV) [14], Local-as-View (LAV) [14] and Global-Local-as-View (GLAV) [4, 8].

The canonical model plays a role of a unifying model, in which the source data models can be represented without loss of information [25].

Various kinds of data models with different semantics are used as canonical models: relational models [9] and their extensions [17], object models [3], XML [15], hypergraph models [23], Web Ontology Language (OWL) [18], RDF [24] etc. However, nowadays one of the properties that the canonical model should possess is the ability to unify the SQL and NoSQL source data models. One of the promising models that hopefully has this property is the Language Integrated Query (LINQ) that is a part of C# language standard ([6], *12.17 Query expressions*). Meijer et al. in [16] shows the duality of SQL and NoSQL models using an area of mathematics called category theory. LINQ is used by Meijer as the representation language to illustrate the duality. LINQ does not

require strict data typing – if in relational models the data is strictly typed, in non-relational values are dynamically typed. Also, LINQ does not require preliminary data normalization, which is required in SQL and which, in this case, is not used in NoSQL models. Therefore, based on research in [16], it can be assumed that LINQ can be successfully applied as a canonical model in virtual integration of both SQL and NoSQL data sources. This work aims to confirm this assumption.

To achieve the goal, the following problems have to be solved: (i) selection of concrete data models to be unified in LINQ; (ii) implementation of DIS prototype based on the Global-As-View approach and LINQ as the canonical data model, and (iii) evaluation of DIS via use case in some subject area. In this paper the related work is shortly overviewed and the current progress of DIS prototype implementation is reported and illustrated by a use case.

The remainder of this paper is structured as follows. In Section 2 related work is discussed. The phases of the DIS prototype development and description of its components are shown in Section 3. The initial steps on evaluation of the DIS prototype by a use case are described in Section 4. Finally, Section 5 draws conclusions and points out future work.

2 Related Works

During recent years quite a number of different systems have been developed to implement virtual data integration. Various canonical models and conceptual specification approaches were applied.

In 2000 the Agora [15] system based on Local-As-View approach was developed. In Agora the XML is used as a canonical model. The system is intended to integrate relational or XML sources. The queries are posed using XQuery language.

In [2] the Automed system is presented. This DIS is based on a hypergraph data model and implements Both-As-View approach. In addition to the models accepted by Agora as source data models, Automed can also accept flat files as a data sources for processing and accessing it using the AutoMed Intermediate Query Language (AIQL).

A virtual data integration approach is also used in [1], where the authors considers RDF-based Data Integration Framework, and in [22], where the authors show how the Ontology Web Language (OWL) can be used as a canonical data model.

In [3] a virtual data integration system is presented and used for problem solving in the field of astronomy. As a canonical model the SYNTHESIS [11] language that is a combined object and frame data model. The SYNTHESIS canonical data model was intended to unify wide spectrum of data models like XML, relational, RDF etc. For instance, in [21] the main principles of conceptual mapping of array data model (ADM) into the SYNTHESIS language are analyzed and illustrated and in [19] the mapping of the RDF language into the SYNTHESIS is considered.

In [14] the MetaMed system used for integration of medical data is proposed. The extracted metadata are stored using the RDF and are structured by OWL ontologies. For posing queries the SPARQL language is used. The authors integrate clinical docu-ments and laboratory results that presented in DASTA format and imaging examination as DICOM files.

In [17] the possibility of using of SQL++ language as a canonical model for the relational and JSON data sources is analyzed. The authors discuss the use of unifying language in FORWARD virtual database query processor. SQL, NoSQL, SQL-on-Hadoop and NewSQL databases are integrated.

In fact, none of known existing DISs support the integration of the whole range of the NoSQL resources. However, with Mejer's research in [16] we can suggest that LINQ can serve as a unifying data model for a wide range of NoSQL models. LINQ is a query expression language within the C# language, and there is an advantage of LINQ over the SQL model: each class in LINQ may either have scalar properties, or contain arbitrary values, including other rows (or nested collections), that are typical for NoSQL models. Meijer et al. demonstrate the principles of interpretation of LINQ queries using category theory. They correlate data structures of SQL and NoSQL models using a notion of *duality* from category theory to conclude that NoSQL is a *dual* to SQL, and NoSQL can be called *coSQL*.

3 LINQ as a Canonical Data Model: Validation Steps

Validation that LINQ can be used as a canonical data model consists of three steps:

1. selection of source data models to be unified in LINQ;
2. implementation of a DIS prototype based on the GAV approach with LINQ as the canonical data model;
3. evaluation of DIS prototype using a concrete use case.

For validation LINQ as a canonical data model there were chosen five heterogeneous data models: SQLite as a relational model and four NoSQL data models. It is worth to admit that from a set of various NoSQL model classes the four main classes with different storage structure and data manipulation facilities were chosen. These classes are key-value, column-oriented, document and graph data models:

- *Key-Value Model.* The simplest model in representation is a key-value model. The representation consists of composition of a *key* and a *value*. A key-value NoSQL implemented systems allow either simple data types (e.g., numerals and strings) or the use of lists and sets of values of simple types. The databases based on this model do not support complex queries to be performed on the data stored in database, but only the search keys. The relationships in terms of reference keys and provides no referential integrity constraint are not supported in the key-value model [7].
- *Column-oriented Model.* Since the column model is organized in terms of columns and rows, it can be called as a modification of relational model. The modification is that the tables in column database can contain not only scalar values, but also nested tables. In other words, the rows do not store a tuple, but a set of attributes of the same type, while the set of attributes of a column contains the information from a given instance. Such feature allows queries to be performed more efficiently, although when recovering a complete instance, it may become more costly [7].
- *Document Model.* The databases based on the document model store the grouped data entities in document as an object that are composed by keys and values. Mainly,

the documents are usually serialized in JSON syntax. The keys are generated randomly by the database or manually at persistence time. The document database allows complex queries involving different collections of documents. In the model is necessary that document has a database reference to another database, but it does not guarantee referential constraint [7].

- *Graph Model.* In the graph model the data items are connected by relationships by means of a graph structure. The graph model consists of (i) nodes, that correspond to data instances; (ii) edges, that refer to maintained relationships among node instances; and (ii) properties that relate to data instances. Edges contain linked input and output nodes. That feature guarantee referential integrity by ensuring that input node always makes reference to the output node. The access keys to the nodes are automatically set by the system. However, it is possible to establish unique constraints for other node properties [7].

The following DBMSs are chosen to be integrated in this work: SQLite[1] database based on a relational model, Accumulo[2] as a key-value store, HBase[3] as a column store, MongoDB[4] database based on document model and Neo4j[5] graph database.

Mentioned databases are planned to be integrated within a DIS prototype. The prototype should contain three main components: query rewriting component, DBMS wrappers supporting selected data models, and data merging component. The whole architecture that is planned to be realized is shown in Fig. 1.

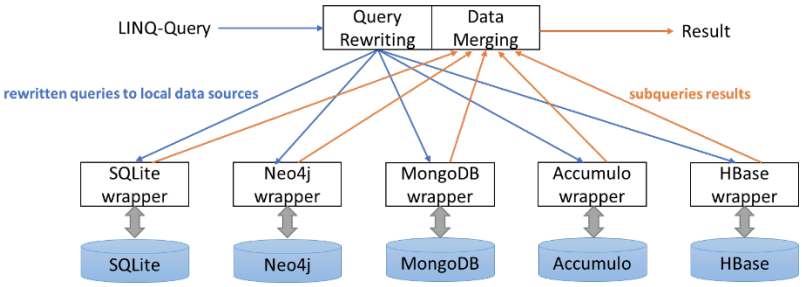

Fig. 1. The architecture to validate LINQ as a canonical model

Query rewriting is the first phase of DIS prototype operation. A query posed by a user should be rewritten into five subqueries [10]. After that rewritten subqueries are transformed into the source query languages. The wrappers for relational data models are integrated in some development environments, such as Microsoft Visual Studio or LinqPad[6], the default algorithm is LINQ-to-SQL[7] query transformation. The wrappers

1 https://www.sqlite.org/index.html
2 https://accumulo.apache.org/
3 https://hbase.apache.org/
4 https://www.mongodb.com/
5 https://neo4j.com/
6 https://www.linqpad.net/
7 https://docs.microsoft.com/en-gb/dotnet/framework/data/adonet/sql/linq/

for chosen NoSQL models should be additionally developed. The development of these wrappers is considered as a future work. Transformed subqueries are passed to source DBMSs and results are directly extracted from the source databases. Since subqueries return results separately, the obtained data should be merged together in data merging component and then presented to the user.

4 Application of LINQ as a Canonical Data Model: a Use Case

Application and evaluation of LINQ as a canonical data model is illustrated in this section with a use case and consist of the following steps:

- a use case subject domain selection;
- definition of the global schema and a set of analytical queries that should illustrate the main LINQ constructs;
- selection of data sources to be deployed into source DBMSs;
- definition of conceptual mappings (based on the Global-as-View approach), that link the local sources' schemas and the global schema;
- illustration of query rewriting and data merging processes.

Each step is described further in Sections 4.1–4.5.

Subject Domain Selection

As the domain for the use case the urban statistics were chosen including transports, demography, environment, immigration, etc. The examples of the subject domain entities and their attributes for transports and demography are illustrated in Table 1.

Various regional agencies or ministries can use the regional urban statistics to analyze the indicators and metrics. The indicators can be used for creating the development programs to improve the life quality in the region.

Global Schema and Analytical Query Example

After selection of the subject domain, the global schema for the domain should be defined. The global schema is used for aggregating all data obtained from heterogeneous data sources.

The global schema should be defined using the canonical model. Since the LINQ is a constituent of the C# standard, the entities (classes) and attributes should be defined using C#. An example for the definition of the classes *births* and *deaths* is shown in Table 2.

Table 1. Entities and attributes of the domain

Entities	Attributes
Demography	

births	year, district_code, district_name, neighborhood_code, neighborhood_name, gender, number
deaths	year, district_code, district_name, neighborhood_code, neighborhood_name, age, number
population	year, district_code, district_name, neighborhood_code, neighborhood_name, gender, age, number
unemployment	year, month, gender, demand_occupation, number
immigration_by_nationality	year, district_code, nationality, number
most_frequent_names	order, name, gender, decade, frequency
most_frequent_baby_names	order, name, gender, year, frequency
Transports	
accidents	id, district_name, neighborhood_name, street, weekday, month, day, hour, part_of_the_day, mild_injuries, serious_injuries, victims, vehicles_involved, longitude, latitude
bus_stops	code, transport, longitude, latitude, bus_stop, district_name, neighborhood_name
transports	code, transport, longitude, latitude, station, district_name, neighborhood_name

Table 2. The class declaration in the global schema

```
class births {
  int year;
  string district_name;
  string neighborhood_name;
  string gender;
  int number;
  string city;
}
```

```
class deaths {
  int year;
  string district_name;
  string neighborhood_name;
  string age;
  int number;
  string city;
}
```

In order to validate that LINQ can be used as a canonical data model, it is necessary to define a certain set of queries that covers the capabilities of the LINQ language. As an example, consider a LINQ query that contains filter (*where* clause), *join* operation, aggregation functions *sum*, sub-queries and mathematical operation in *select* clause:

```
var demogr =    from b in births
        join d in deaths on b.year equals d.year
        where b.year<=2017
        select new {y = b.year, brt = b.sum(n=>n.number),
                    dth = d.sum(n=>n.number),
                    diff = b.sum(n=>n.number) - d.sum(n=>n.number),
                    city = b.city};
```

This query should return the annual number of births and deaths from two separated classes (both classes should be joined) and the difference between births and deaths (rate of natural increase). The example uses *births* and *deaths* classes as data sources.

This paper considers only a fragment of a global schema that should be used in analytical queries. A definition of the whole global schema is a future work.

Data Sources

The raw data were extracted from the open data banks or previously developed databases. Data on events and statistics in Barcelona[8] are stored in SQLite relational database. In Neo4j (graph DBMS) a database with the events and statistics in Madrid[9] was created, the data on Bilbao[10] is going to be created in MongoDB (document DBMS). Data and statistics on Malaga[11] are going to be presented in HBase (column store), data on Seville[12] are going to be stored in Accumulo (key-value store).

As the query from Section 4.2 requires *births* and *deaths* entities, the parts of diagrams from relational and graph databases are illustrated in Fig. 2. The postfix *_rel* means that the entity is from relational database. Postfix *_gr* denotes entities from graph database.

On the left side of the Fig. 2 a part of the relational schema is presented: *births_rel* and *deaths_rel* entities. Each entity has a list of attributes that have explicit conducted data types. On the right side a visualization of the graph database is presented. The blue nodes are marked with *births_gr* label and orange nodes are marked with *deaths_gr* label. Since Neo4j does not show all labels of the nodes, there were shown only one label for each node. Other stored data of each label are hidden. In blue nodes the *gender* labels are shown, and in orange nodes the age category of deaths is pointed (the *age* label). Blue and orange nodes are linked by *brt_dth* relationships if *year*, *district_name* and *neighborhood_name* attribute values of *births_gr* and *deaths_gr* entities are equal.

[8] https://opendata-ajuntament.barcelona.cat/en
[9] https://datos.madrid.es/portal/site/egob/
[10] https://www.bilbao.eus/opendata/es/formatos
[11] https://datosabiertos.malaga.eu/dataset
[12] http://sevilla-idesevilla.opendata.arcgis.com/datasets

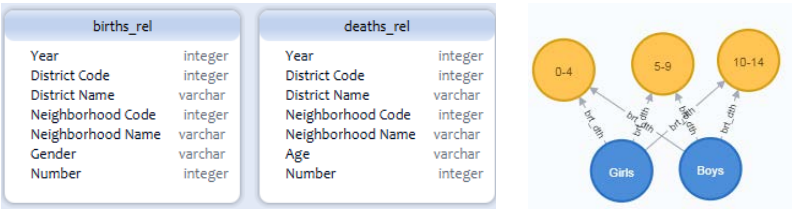

Fig. 2. The diagrams in relational (left) and graph (right) models for the example

The local schema definitions for document, column and key-value models are planned for the future work.

Conceptual Mappings

To define the conceptual mappings, the correspondences of entities and attributes between local and global schemas should be established first. The examples of correspondences between local (relational) and global schema elements are illustrated in Fig. 3.

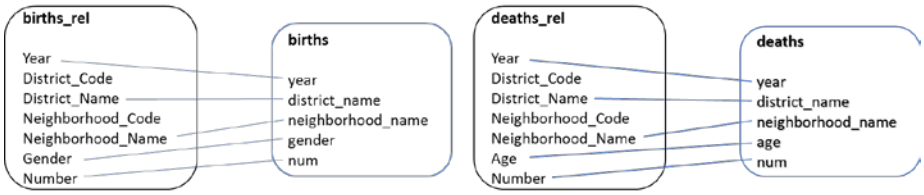

Fig. 3. Correspondences of the relational database (black) and global schema (blue)

According to GAV approach [14], the global schema entities should be represented as views over the local schema using the canonical data model. The correspondences can be produced either manually by an expert or automatically based on various approaches: machine learning approach [5], linguistic processing [26] etc.

For example, the conceptual mappings between *births* and *deaths* entities in global schema and *births_rel* and *deaths_rel* entities in local schema (SQL model) can be defined in the following way:

```
var births =  from b in births_rel
                select new {year = b.Year, district_name = b.District_Name,
                  neighborhood_name = b.Neighborhood_Name,
                  gender = b.Gender, num = b.Number, city = "Barcelona"};

var deaths =  from d in deaths_rel
                select new {year = d.Year, district_name = d.District_Name,
                  neighborhood_name = d.Neighborhood_Name,
                  age = d.Age, num = d.Number, city = "Barcelona"};
```

City is not the attribute of any local source and its value is generated within views. The value of *city* attribute is a constant that identifies the data source – the city to which

the extracted data refer. As it is described earlier, the database based on relational model stores data on Barcelona.

Conceptual mappings for the graph database look almost the same, the only difference is that *births_gr* and *deaths_gr* entities are used instead of *births_rel* and *deaths_rel*.

Conceptual mappings can be constructed manually by an expert or in semi-automated way. This paper discusses a simple example in which conflict situations do not arise during construction of conceptual mappings. In general, various types of conflicts such as data type mismatch or structural conflicts requiring combination of several attributes of a local schemas into an attribute of the global schema can occur. Dealing with conceptual mappings in case of conflicts is a future work.

Query Rewriting, Transformation and Data Merging

Query Rewriting. First, the query from Section 4.2 should be rewritten. The query rewriting is processed using the views from Section 4.4 on the basis the GAV approach [14]. For instance, the rewritten query to the relational database is formed by replacing *births* and *deaths* by the bodies of views from Section 4.2 as nested queries:

```
var demogr =    from b in (from b in births_rel
                select new {year = b.Year, district_name = b.District_Name,
                    neighborhood_name = b.Neighborhood_Name,
                    gender = b.Gender, num = b.Number, city = "Barcelona"})
        join d in (from d in deaths_rel
                select new {year = d.Year, district_name = d.District_Name,
                    neighborhood_name = d.Neighborhood_Name,
                    age = d.Age, num = d.Number, city = "Barcelona"})
        on b.year equals d.year
        where b.year<=2017
        select new {y = b.year, brt = b.sum(n=>n.num),
                dth = d.sum(n=>n.num),
                diff = b.sum(n=>n.num)- d.sum(n=>n.num),
                city = b.city};
```

The rewritten query for the graph database looks almost the same. The only difference is that in the rewritten query for the graph database that the value of the *city* attribute is *"Madrid"* instead of *"Barcelona"*.

Query Transformation. To extract the data from local sources rewritten queries should be transformed into the source query languages. For instance, the rewritten LINQ query should be transformed in relational data model wrapper as follows:

SELECT b.year as y, sum(b.num) as brt, sum(d.num) as dth,
sum(b.num) - sum(d.num) as diff, b.city
FROM
(SELECT b.Year as year, b.District_Name as district_name, b.Neighborhood_Name as neighborhood_name, b.Gender as gender, b.Number as num, "Barcelona" as city FROM births_rel b) b
INNER JOIN
(SELECT d.Year as year, d.District_Name as district_name, d.Neighborhood_Name as neighborhood_name, d.Age as age, d.Number as num, "Barcelona" as city
FROM deaths_rel d) d ON b.year = d.year
WHERE b.year <=2017
GROUP BY b.year, b.city

Considering that the attribute value *city = "Madrid"* in the graph data source, the rewritten LINQ query is transformed into Cypher graph query language (supported by Neo4j) as follows:

MATCH (b:births_gr)
WITH [{year: b.Year, district_name: b.District_Name, neighborhood_name: b.Neighborhood_Name, gender: b.Gender, num: b.Number, city: "Madrid"}] as b
UNWIND b as b
MATCH (d:deaths_gr)
WITH [{year: d.Year, district_name: d.District_Name, neighborhood_name: d.Neighborhood_Name, age: d.Age, num: d.Number, city: "Madrid"}] as d
UNWIND d as d
MATCH (b)--(d)
RETURN b.year as y, sum(b.num) as brt, sum(d.num) as dth, sum(b.num)-sum(d.num) as diff, b.city

Data Merging. Finally, the results extracted from sources should be merged in data merging component. For instance, the query to the relational database returns the result that is shown on Fig. 4(a), the query to the graph database returns the result that is shown on Fig. 4 (b). The result presented to a user in this case is just a merge of 4(a) and 4(b) results.

a)

year	births	deaths	difference	city
2015	13510	15478	-1968	Barcelona
2016	13630	15183	-1553	Barcelona
2017	13526	15564	-2038	Barcelona

b)

year	births	deaths	difference	city
2015	68892	38007	30885	Madrid
2016	73824	38096	35728	Madrid
2017	69613	38450	30866	Madrid

Fig. 4. The query results from relational (a) and graph (b) databases

5 Conclusions and Future Work

This paper presents an approach for validation that Language Integrated Query (LINQ) is able to serve as a canonical data model for virtual data integration in the world of SQL and NoSQL databases. The architecture of DIS to validate LINQ as a canonical

model is presented. A use case intended to illustrate key steps of data integration is provided. The subject domain of the use case is defined, conceptual mappings between source schemas and the global schema are illustrated as well as query rewriting and transformation for the relational and Neo4j data models.

Future work includes development of wrappers for chosen NoSQL models as well as query rewriting and data merging components of the prototype DIS. Complete use case description including global and local schemas and their conceptual mappings accompanied by a set of comprehensive analytical queries and their results is also a future work.

Acknowledgements

This work is supervised by Sergey Stupnikov, lead research scientist at the Federal Research Center "Computer Science and Control" of Russian Academy of Sciences.

References

1. Amini, A., Saboohi, H., and Nematbakhsh, N.: An RDF-based data integration framework. In: National Electrical Engineering Conference (NEEC) 2008, Najafabad (2008).
2. Boyd, M., Kittivoravitkul, S., Lazanitis, C., McBrien, P., and Rizopoulos, N.: AutoMed: a BAV data integration system for heterogeneous data sources. In: Persson A., Stirna J. (eds) Advanced Information Systems Engineering. CAiSE 2004. Lecture Notes in Computer Science **3084**. Springer, Berlin, Heidelberg (2004).
3. Briukhov, D.O., Vovchenko, A. E., Zakharov, V.N., Zhelenkova, O.P., Kalinichenko, L.A., Martynov, D.O., Skvortsov, N.A., and Stupnikov, S.A.: The middleware architecture of the subject mediators for problem solving over a set of integrated heterogeneous distributed information resources in the hybrid grid-infrastucture of virtual observatories. Informatics and Applications **2** (1), 2–34 (2008).
4. Briukhov, D., Kalinichenko, L., and Martynov, D.: Source registration and query rewriting applying LAV/GLAV techniques in a typed subject mediator. In: Proceedings of the 9th Russian Conference on Digital Libraries, RCDL'2007, 253–262. Pereslavl, Russia (2007).
5. Bulygin, L.: Combining lexical and semantic similarity measures with machine learning approach for ontology and schema matching problem. In: Proceedings of the XX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2018), 245–249, Moscow (2018).
6. European Computer Machinery Association. Standard ECMA-334: C# Language Specification, 5th edition, December 2017. https://www.ecma-international.org/ publications/files/ECMA-ST/ECMA-334.pdf
7. Freitas, M., Souza, D., and Salgado, A.: conceptual mappings to convert relational into NoSQL Databases. In: Hammoudi, S., Maciaszek, L.A., Missikoff, M.M., Camp, O., Cordeiro, J. (eds.) 18th International Conference on Enterprise Information Systems (ICEIS 2016), **1**, 174–181. SCITEPRESS, Rome (2017).
8. Friedman, M., Levy, A., and Millstein, T.D.: Navigational plans for data integration. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), 67–73. AAAI Press/The MIT Press (1999).
9. Haas, L.M., Lin, E.T., and Roth, M.A.: Data integration through database federation. IBM Systems Journal **41** (4), 578–596 (2002).

10. Hai, R., Quix, and C., Zhou, C.: Query rewriting for heterogeneous data lakes. In: Benczúr A., Thalheim B., Horváth T. (eds) Advances in Databases and Information Systems. DBIS 2018. Lecture Notes in Computer Science **11019,** 35–49. Springer, Cham (2018).
11. Kalinichenko, L.A., Stupnikov, S.A., and Martynov, D.O.: SYNTHESIS: A language for canonical information modeling and mediator definition for problem solving in heterogeneous information resource environments. IPI RAN, Moscow (2007).
12. Khine, P. and Wang, Z.: Data lake: a new ideology in big data era. In: Guchi, K., Chen, T. (eds.) 2017 4th Annual International Conference on Wireless Communication and Sensor Network (WCSN 2017) **17**, Wuhan (2017).
13. Kimball, R. and Ross, M.: The Data Warehouse Toolkit. 3nd edn. John Wiley & Sons, Inc., Indianapolis, IN (2013).
14. Lenzerini, M.: Data integration: a theoretical perspective. In: Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 233–246. Madison, Wisconsin, USA (2002).
15. Manolescu, I., Florescu, D., Kossmann, D., Xhumari, F., and Olteanu, D.: Agora: living with XML and relational. In: El Abbadi, A., Brodie, M., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K.-Y. (eds.): Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000), 623–626. Cairo (2000).
16. Meijer, E., Bierman, G.: A co-Relational Model of Data for Large Shared Data Banks. Microsoft Research. ACMqueue **3** (9), 1–19 (2011).
17. Ong, K. W., Papakonstantinou, Y., Vernoux, R.: The SQL++ Unifying Semi-structured Query Language, and an Expressiveness Benchmark of SQL-on-Hadoop, NoSQL and NewSQL Databases. CoRR, abs/1405.3631 (2014).
18. Rodríguez-Muro, M., Kontchakov, R., and Zakharyaschev, M.: Ontology-Based Data Access: Ontop of Databases. In: Alani H. et al. (eds): The Semantic Web – ISWC 2013. ISWC 2013. Lecture Notes in Computer Science **8218**, 558–573. Springer, Berlin, Heidelberg (2013).
19. Skvortsov, N.A.: Mapping of RDF Data Model into the Canonical Model of Subject Mediators. In: Smirnov, V., Stupnikov, S. Proceedings of the 15th All-Russian Scientific Conference “Digital Libraries: Advanced Methods and Technologies, Digital Collections” (RCDL 2013), 95–101 (2013).
20. Stupnikov, S. and Kalinichenko, L.: Extensible Unifying Data Model Design for Data Integration in FAIR Data Infrastructures. In: Manolopoulos Y., Stupnikov S. (eds): Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2018). Communications in Computer and Information Science **1003**, 17–36. Springer, Cham (2019).
21. Stupnikov, S.: Unification of an array data model for the integration of heterogeneous information resources. In: Znamenskij, S., Kogalovsky, M. Proceedings of the 14th All-Russian Scientific Conference "Digital libraries: Advanced Methods and Technologies, Digital Collections" (RCDL 2012), 42–52. Pereslavl-Zalessky (2012).
22. Tan, P., Madnick, S., and Tan, K.: Context mediation in the Semantic Web: handling OWL Ontology and Data Disparity Through Context Interchange. In: Bussler, C., Tannen, V., Fundulaki, I. 2nd international workshop on semantic web and databases 2004, LNCS **3372**, 140–154. Springer, Berlin, Heidelberg (2005).
23. Theodoratos, D.: Semantic integration and querying of heterogeneous data sources using a hypergraph data model. In: 19th British National Conference on Databases (BNCOD), 166–182. Springer, Heidelberg (2002).
24. Vcelak, P., Kratochvil, M., Kleckova, J., and Rohan, V.: MetaMed – Medical meta data extraction and manipulation tool used in the semantically interoperable research information

system. In: Chen, Q., Huan, J., Xu, Y., Zhang, T., Wang, L. (eds.) 5th International Conference on Biomedical Engineering and Informatics (BMEI 2012), pp. 1270–1274. IEEE, Chongqing, China (2012).
25. Zakharov, V.N., Kalinichenko, L.A., Sokolov, I.A., and Stupnikov, S.A.: Development of canonical information models for integrated information systems. Informatics and Its Applications **1** (2), 15–38 (2007).
26. Zhang, Y., Wang, X., Lai, S., He, S., Liu, K., Zhao, J., and Lv, X.: Ontology matching with word embeddings. In: Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data, 34–45 (2014).