On an Approach to Data Integration: Concept, Formal Foundations and Data Model

© Manuk G. Manukyan

Yerevan State University,
Yerevan, Armenia

mgm@ysu.am

Abstract. In the frame of an extensible canonical data model a formalization of data integration concept is proposed. We provide virtual and materialized integration of data as well as the possibility to support data cubes with hierarchical dimensions. The considered approach of formalization of data integration concept is based on the so-called content dictionaries. Namely, by means of these dictionaries we are formally defining basic concepts of database theory, metadata about these concepts, and the data integration concept. A computationally complete language is used to extract data from several sources, to create the materialized view, and to effectively organize queries on the multidimensional data.

In memory of Garush Manukyan, my father.

This work was supported by the RA MES State Committee of Science, in the frames of the research project N 15T-18350.

Keywords: data integration, mediator, data warehouse, data cube, canonical data model, OPENMath, grid file, XML.

1 Introduction

The emergence of a new paradigm in science and various applications of information technology (IT) are related to issues of big data handling [21]. The concept of big data is relatively new and involves the growing role of data in all areas of human activity beginning with research and ending with innovative developments in business. Such data is difficult to process and analyze using conventional database technologies. In this connection, the creation of new IT is expected in which data becomes dominant for new approaches to conceptualization, organization, and implementation of systems to solve problems that were previously considered extremely hard or, in some cases, impossible to solve. Unprecedented scale of development in the big data area and the U.S. and European programs related to big data underscore the importance of this trend in IT.

In the above discussed context the problems of data integration are very actual. Within of our approach to data integration an extensible canonical model has been developed [16]. We have published a number of papers that are devoted to the investigation of data virtual and materialized data integration problems, for instance [15, 17]. Our approach to data integration is based on the works of the SYNTHESIS group (IPI RAS) [2, 9–12, 22–25], who are pioneers in the area of justifiable data models mapping for heterogeneous databases integration. To support materialized integration of data during creation of a data warehouse a new dynamic index structure for multidimensional data was proposed [6] which is based on the grid files [18] concept. We consider the concept of grid files as one of the adequate formalisms for effective management of big data. Efficient algorithms for storage and access of that directory are proposed in order to minimize memory usage and lookup operations complexities. Estimations of complexities for these algorithms are presented. In fact, the concept of grid files allows to effectively organize queries on multidimensional data [5] and can be used for efficient data cubes storage in data warehouses [13,19]. A prototype to support the considered dynamic indexation scheme has been created and its performance was compared with one of the most demanded NoSQL databases [17].

In this paper a formalization of the data integration concept is proposed using a mechanism of the content dictionaries (similarly ontologies) of the OPENMath [4]. Subjects of the formalization are the basic concepts of database theory, metadata about these concepts and the data integration concept. The result of the formalization are a set of content dictionaries, constructed as XML DTDs on the base of OPENMath and are used to model the databases concepts. With this approach, schema of an integrated database is an instance of content dictionary of the data integration concept. Within the considered approach is provided virtual and materialized integration of data as well as the possibility to support data cubes with hierarchical dimensions. Using OPENMath as the kernel of the canonical data model allows us to use a rich apparatus of computational mathematics for data analysis and management.

The paper is organized as follows: Concept and formal foundations of the considered approach to data integration are presented briefly in Section 2. Canonical data model and issues to support the data integration

Proceedings of the XIX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2017), Moscow, Russia, October 10–13, 2017

concept are considered in Section 3. The conclusion is provided in Section 4.

2 Brief Discussion on Data Integration Approach

The basis of our concept to data integration is based on the idea of integrating arbitrary data models. Based on this assumption our concept of data integration assumes:

- applying extensible canonical model;
- constructing justifiable data models mapping for heterogeneous databases integration;
- using content dictionaries.

Choosing the extensible canonical model as integration model allows integrating arbitrary data sources. As we allow integration of arbitrary data sources a necessity to check mapping correctness between data models arises. It is reached by formalization of data model concepts by means of AMN machines [1] and using B-technology to prove correctness of these mappings.

The content dictionaries are central to our concept of data integration and semantical information of different types can be defined based on these dictionaries. The concept of content dictionaries allows us to extend the canonical model by means of introducing new concepts in these dictionaries easily. In other words, canonical model extension *only* is reduced to adding new concepts and metadata about these concepts in content dictionaries. Our concept to data integration is oriented as virtual and materialized integration of data as well as to support data cubes with hierarchical dimensions. It is important that in all cases we use the same data model. The considered data model is an advanced XML data model which is a more flexible data model than relational or object-oriented data models. Among XML data models, a distinctive feature of our model is that we use a computationally complete language for data definition. An important feature of our concept is the support of data warehouses on the base of a new dynamic indexing scheme for multidimensional data. A new index structure developed by us allows to organize effectively OLAP-queries on multidimensional data and can be used for efficient data cubes storage in data warehouses. Finally, the modern trends of the development of database systems lead to use of different divisions of mathematics to data analysis. Within of our concept to data integration, this leads to the use of corresponding content dictionaries of the OPENMath.

2.1 Formal Foundations

The above discussed concept to data integration is based on the following formalisms:

- canonical data model;
- OPENMath objects;
- multidimensional indexes;
- domain element calculus.

Below we will consider these formalisms in detail. As we noted, our approach to data integration is based on the works of the SYNTHESIS group. According to the research of this group, each data model is defined by syntax and semantics of two languages, data definition language (DDL) and data manipulation language (DML). They suggested the following principles of synthesis of the canonical model:

- **Principle of axiomatic extension of data models**

The canonical data model must be extensible. The kernel of the canonical model is fixed. Kernel extension is defined axiomatically. The extension of the canonical data model is formed during the consideration of each new data model by adding new axioms to its DDL to define logical data dependencies of the source model in terms of the target model if necessary. The results of the extension should be equivalent to the source data model.

- **Principle of commutative mappings of data models**

The main principle of mapping of an arbitrary resource data model into the target one (the canonical model) could be reached under the condition that the diagram of DDL (schemas) mapping and the diagram of DML (operators) mapping are commutative.

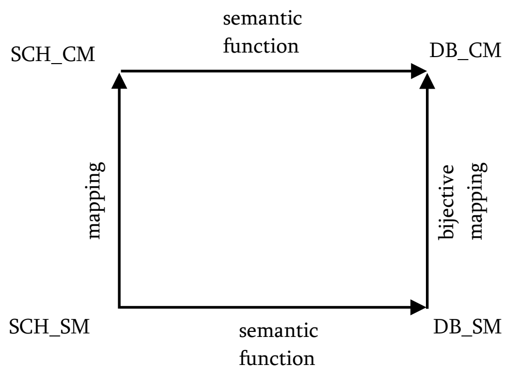

Figure 1 DDL mapping diagram

In Figure 1 we used the following notations: **SCH_CM**: Set of schemas of the canonical data model; **SCH_SM**: Set of schemas of the source data model; **DB_CM**: Database of the canonical data model; **DB_SM**: Database of the source model.

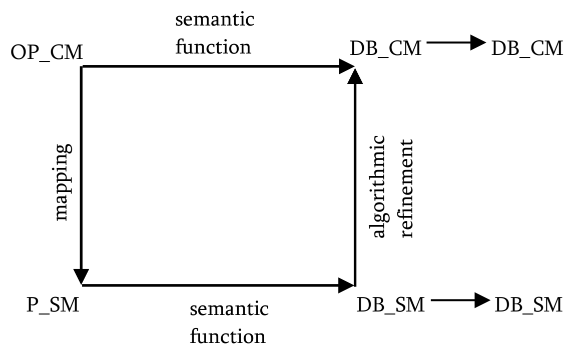

Figure 2 DML mapping diagram

In Figure 2 we used the following notations: **OP_CM**: Set of operators of the canonical data model; **P_SM**: Set of procedures in DML of the source model.

- **Principle of synthesis of unified canonical data model**

The canonical data model is synthesized as a union of extensions.

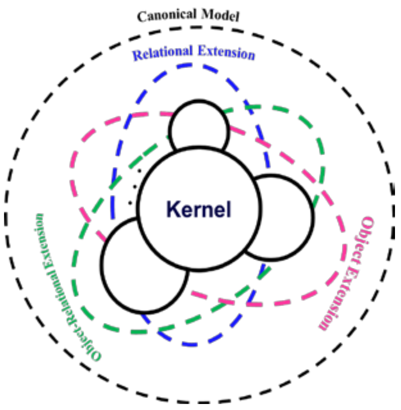

Figure 3 Canonical data model

2.2 Mathematical Objects Representation

The OpenMath is a standard for representation of the mathematical objects, allowing them to be exchanged between computer programs, stored in databases, or published on the Web. The considered formalism is oriented to represent semantic information and is not intended to be used directly for presentation. Any mathematical concept or fact is an example of mathematical object. The OpenMath objects are such representation of mathematical objects which assume an XML interpretation.

Formally, an OpenMath object is a labeled tree whose leaves are the *basic* OpenMath objects. The compound objects are defined in terms of *binding* and *application* of λ-calculus [8]. The type system is built on the basis of types that are defined by themselves and certain recursive rules, whereby the compound types are built from simpler types. To build compound types the following type constructors are used:

- *Attribution*. If *v* is a basic object variable and *t* is a typed object, then a*ttribution* (*v*, *type t*) is a typed object. It denotes a variable with type *t*.
- *Abstraction*. If *v* is a basic object variable and *t*, *A* are typed objects, then *binding* (λ, *attribution* (*v*, *type t*), A) is a typed object.
- *Application*. If *F* and *A* are typed objects, then *application* (*F*, *A*) is a typed object.

The OPENMath is implemented as an XML application. Its syntax is defined by syntactical rules of XML, its grammar is partially defined by its own DTD. Only syntactical validity of the OPENMath objects representation can be provided on the DTD level. To check semantics, in addition to general rules inherited by XML applications, the considered application defines new syntactical rules. This is achieved by means of introduction of content dictionaries. Content dictionaries are used to assign formal and informal semantics to all symbols used in the OPENMath objects. A content dictionary is a collection of related symbols encoded in XML format. In other words, each content dictionary defines symbols representing a concept from the specific subject domain.

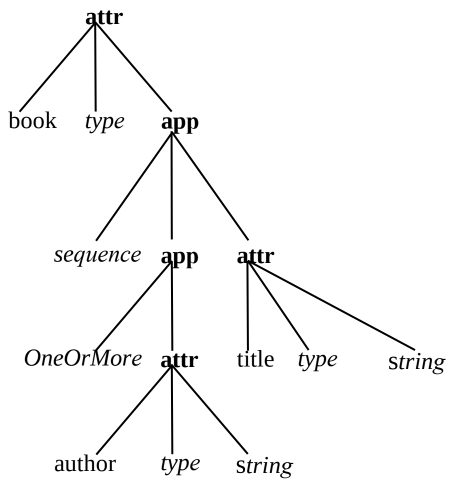

Figure 4 An example of compound object

2.3 Dynamic Indexing Scheme for Multidimensional Data

To support the materialized integration of data during the creation of a data warehouse and to apply very complex OLAP-queries on it a new dynamic index structure for multidimensional data was developed (see more details in [6]). The considered index structure is based on the grid file concept. The grid file can be represented as if the space of points is partitioned into an imaginary grid. The *grid lines* parallel to axis of each dimension divide the space into *stripes*. The number of grid lines in different dimensions may vary, and there may be different spacings between adjacent grid lines, even between lines in the same dimension. Intersections of these stripes form cells which hold references to data buckets containing records belonging to corresponding space partitions.

The weaknesses of the grid file formalism concept are non-efficient memory usage by groups of cells referring to the same data buckets and the possibility of having a large number of overflow blocks for each data buckets. In our approach, we made an attempt to eliminate these defects of the grid file. Firstly, we introduced the concept of the chunk: set of cells whose corresponding records are stored in the same data bucket (represented by single memory cells with one pointer to the corresponding data buckets). Chunking technique is used to solve the problem of empty cells in the grid file.

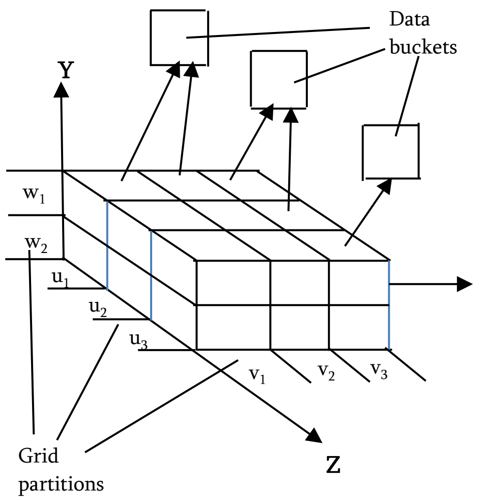

Figure 5 An example of 3-dimensional grid file

Secondly, we consider each stripe as a linear hash table which allows increasing the number of buckets more slowly (for each stripe, the average number of overflow blocks of chunks crossed by that stripe is less than one). By using this technique we essentially restrict the number of disk operations.

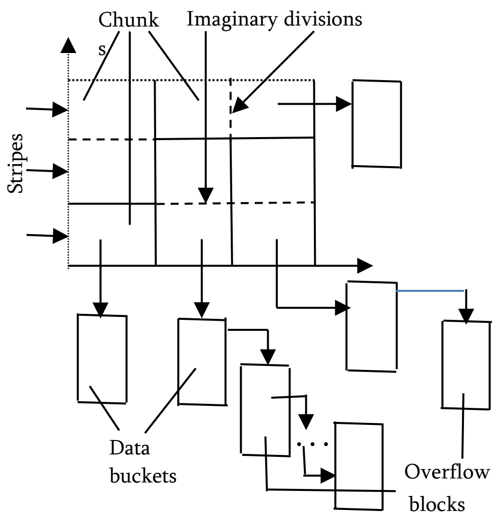

Figure 6 An example of 2-dimensional modified grid file

We perform comparison of directory size by our approach with two techniques for grid file organization proposed in [20]: MDH (multidimensional dynamic hashing) and MEH (multidimensional extendible hashing). Directory sizes for both of these techniques are: $\boldsymbol{O}\left(\boldsymbol{r}^{\mathbf{1}+\frac{\mathbf{1}}{s}}\right)$ and $\boldsymbol{O}\left(\boldsymbol{r}^{\mathbf{1}+\frac{n-\mathbf{1}}{ns-\mathbf{1}}}\right)$ correspondingly, where r is the total number of records, s is the block size and n is the number of dimensions. In our case the directory size can be estimated as $O\left(\frac{nr}{s}\right)$. Compared to MDH and MEH techniques, the directory size in our approach is $\frac{sr^{\frac{1}{s}}}{n}$ and $\frac{sr^{\frac{n-1}{ns-1}}}{n}$ times smaller correspondingly. We have implemented a data warehouse prototype based on the proposed dynamic indexation scheme and compared its performance with MongoDB [26] (see in [17]).

2.4 Element Calculus

In the frame of our approach to data integration as integration model we consider an advanced XML data model. In fact, data model defines the query language [5]. Based on this, to give declarative queries a new query language (domain element calculus) [14] was developed. A query to XML - database is a formula in element calculus language. To specify formulas a variant of the multisorted first order predicate logic language is used. Notice that element calculus is developed in the style of object calculus [10]. In addition, there is a possibility to give queries by means of λ-expressions. Generally, we can combine the considered variants of queries.

3 Extensible Canonical Data Model

The canonical model kernel is an advanced XML data model: a minor extension of the OPENMath to support the concept of databases. The main difference between our XML data model and analogous XML data models (in particular, XML Schema) is that the concept of data types in our case is interpreted conventionally (set of values, set of operations). More details about the type system of the XML Schema can be found in [3]. A data model concept formalized on the kernel level is referred to as *kernel concept*.

3.1 Kernel Concepts

In the frame of canonical data model we distinguish basic and compound concepts. Formally, a kernel concept is a labeled tree whose leaves are basic kernel concepts. Examples of basic kernel concepts are constants, variables, and symbols (for instance, reserved words). The compound concepts are defined in terms of *binding* and *application* of λ-calculus. The type system is built analogously to that in OPENMath.

3.2 Extension Principle

As we noted above the canonical data model must be extensible. The extension of the canonical model is formed during the consideration of each new data model by adding new concepts to its DDL to define logical data dependencies of the source model in terms of the target model if necessary. Thus, the canonical model extension assumes defining new symbols. The extension result must be equivalent to the source data model. To apply a *symbol* on the canonical model level the following rule has been proposed:

Concept $\longleftarrow$ *symbol* ContextDefinition.

For example, to support the concept of *key* of relational data model, we have expanded the canonical model with the symbol *key*. Let us consider a relational schema example:

S = {S#, Sname, Status, City}.

The equivalent definition of this schema by means of extended kernel is considered below:
attribution (S, *type* TypeContext, *constraint*
ConstraintContext)

TypeContext $\longleftarrow$ *application* (*sequence*, ApplicationContext)
ApplicationContext $\longleftarrow$ *attribution* (S#, *type int*),
attribution (Sname, *type string*),
attribution (Status, *type int*),
attribution (City, *type string*))
ConstraintContext $\longleftarrow$ *attribution* (name, *key* S#).

It is essential that we use a computationally complete language to define the context [14]. As a result of such approach, usage of new symbols in the DDL does not lead to any changes in the DDL parser.

3.3 Semantic Level

The canonical model is an XML application. Only syntactical validity of the canonical model concepts representation can be provided on the DTD level. To check semantics the considered application defines new syntactical rules. We define these syntactical rules in content dictionaries.

3.4 Content Dictionaries

The content dictionary is the main formalism to define semantical information about concepts of the canonical data model. In other words, content dictionaries are used to assign formal and informal semantics to all concepts of the canonical data model. A content dictionary is a collection of related symbols, encoded in XML format and fixes the "meaning" of concepts independently of the application. Three kinds of content dictionaries are considered:

- content dictionaries to define basic concepts (symbols);
- content dictionaries to define a signature of basic concepts (mathematical symbols) to check the semantic validity of their representation;
- content dictionary to define a data integration concept.

Supporting the above considered content dictionaries assumes to develop corresponding DTDs. Instances of such DTDs are XML documents. An instance of a DTD of a content dictionary of basic concepts is used to assign formal and informal semantics of those objects. Finally, an instance of a DTD of a content dictionary of a signature of basic concepts contains metainformation about these concepts, and an instance of a DTD of a content dictionary of a data integration concept is a metadata for integrating databases.

3.5 Data Integration Concept

In the frame of our approach to data integration we consider virtual as well as materialized data integration issues within a canonical model. Therefore, we should formalize the concepts of this subject area such as mediator, data warehouse and data cube. We are modelling these concepts by means of the following XML elements: *dbsch*, *med*, *whse* and *cube*.

Mediator. The content of element *dbsch* is based on the kernel *attribution* concept and has an attribute *name*. By means of this concept we can model schemas of databases. The value of attribute *name* is the DB's name. The content of element *med* is based on the elements *msch*, *wrapper*, *constraint* and has an attribute *name*. The value of this attribute is the mediator's name. The element *msch* is interpreted analogously to element *dbsch*. Only note that this element is used during modelling schemas of a mediator. The content of elements *wrapper* and *constraint* is based on the kernel *application* concept. By means of *wrapper* element mappings from source models into a canonical model are defined. The integrity constraints on the level of mediator are the values of the *constraints* elements. It is important that we are using a computationally complete language for defining the mappings and integrity constraints. Below, an example of a mediator for an automobile company database is adduced [5] which is an instance of a content dictionary of data integration concept. It is assumed that the mediator with schema AutosMed = {SerialNo, Model, Color} is integrate two relational sources: Cars = {SerialNo, Model, Color} and Autos = {Serial, Model}, Colors = {Serial, Color}.

```
<cd name = ‘dic’>
 <dbsch name = ‘Source1’>
  <omattr>
   schema definition of Cars
  </omattr>
 </dbsch>
 <dbsch name = ‘Source2’>
  <omattr>
   schema definition of Autos
  </omattr>
  <omattr>
   schema definition of Colors
  </omattr>
 </dbsch>
 <med name = ‘Example’>
  <msch>
   <omattr>
    AutosMed: schema  for mediator is  defined
   </omattr>
  </msch>
  <wrapper>
   <oma>
    <oms name = ‘convert_to_xml’ cd = ‘xml’/>
    <oma>
     <oms name = ‘union’ cd = ‘db’/>
     <omv name = ‘Cars’/>
     <oma>
      <oms name = ‘join’ cd = ‘db’/>
      <omv name = ‘Autos’/>
      <omv name = ‘Colors’/>
     </oma>
    </oma>
   </oma>
  </wrapper>
 </med>
</cd>
```

It is essential that, we use a computationally complete language to model the mediator work.

Data warehouse. As we noted above the considered approach to support data warehousing is based on the grid file concept and is interpreted by means of element *whse*. This element is defined as kernel *application* concept and is based on the elements *wsch*, *extractor*, *grid* and has an attribute *name*. The value of this attribute is the name of the data warehouse. The element *wsch* is interpreted in the same way as the element *msch* for the mediator. The element *extractor* is defined as kernel *application* concept and is used to extract data from source databases. The element *grid* is defined as kernel *application* concept and is based on the elements *dim* and *chunk* by which the grid file concept is modelled. To model the concept of stripe of a grid file we introduced an empty element *stripe* which is described by means of five attributes: *ref_to_chunk*, *min_val*, *max_val*, *rec_cnt* and *chunk_cnt*. The values of attributes *ref_to_chunk* are pointers to chunks crossed by each stripe. By means of *min_val* (lower boundary) and *max_val* (upper boundary) attributes we define "widths" of the stripes. The values of attributes *rec_cnt* and *chunk_cnt* are the total number of records in a stripe and the number of chunks that are crossed by it correspondingly. To model the concept chunk we introduced an element *chunk* which is based on the empty element *avg* and is described by means of four attributes: *id* of type ID, *qty*, *ref_to_db* and *ref_to_chunk*. Values of attributes *ref_to_db* and *ref_to_chunk* are pointers to data blocks and other chunks, correspondingly. Value of attribute *qty* is the number of different points of the considered chunk for fixed dimension. Element *avg* is described by means of two attributes: *value* and *dim*. Values of *value* attributes are used during reorganization of the grid file and contain the average coordinates of points, corresponding to records of the considered chunk, for each dimension. Value of attribute *dim* is the name of the corresponding dimension. To model the concept of dimension of a grid file we introduced an element *dim* which is based on the empty element *stripe* and has a single attribute *name*: i. e. the dimension name.

Data cube. Materialized integration of data assumes the creation of data warehouses. Our approach to create data warehouses is mainly oriented to support data cubes. Using data warehousing technologies in OLAP applications is very important [5]. Firstly, the data warehouse is a necessary tool to organize and centralize corporate information in order to support OLAP queries (source data are often distributed in heterogeneous sources). Secondly, significant is the fact that OLAP queries, which are very complex in nature and involve large amounts of data, require too much time to perform in a traditional transaction processing environment. To model the data cube concept we introduced an element *cube* which is interpreted by means of the following elements: *felement*, *delement*, *fcube*, *rollup*, *mview* and *granularity*. In typical OLAP applications, some collection of data called *fact_table* which represent events or objects of interest are used [5]. Usually, *fact_table* contains several attributes representing dimensions, and one or more dependent attributes that represent properties for the point as a whole. To model the *fact_table* concept we introduced an element *felement* which is based on the kernel *attribution* concept. To model the concept of dimension we introduced an element *delement*. This element is based on the empty element *element* which is described by means of attribute *name*. Value of attribute *name* is the dimension name. The creation of the data cube requires generation of the power set (set of all subset) of the aggregation attributes. To implement the formal data cube concept in literature the CUBE operator is considered [7]. In addition to the CUBE operator in [7] the operator ROLLUP is produced as a special variety of the CUBE operator which produces the additional aggregated information only if they aggregate over a tail of the sequence of grouping attributes. To support these operators we introduced *cube* and *rollup* symbols correspondingly. In this context, it is assumed that all independent attributes are grouping attributes. For some dimensions there are many degrees of granularity that could be chosen for a grouping on that dimension. When the number of choices for grouping along each dimension grows, it becomes non-effective to store the results of aggregating based on all the subsets of groupings. Thus, it becomes reasonable to introduce materialized views.

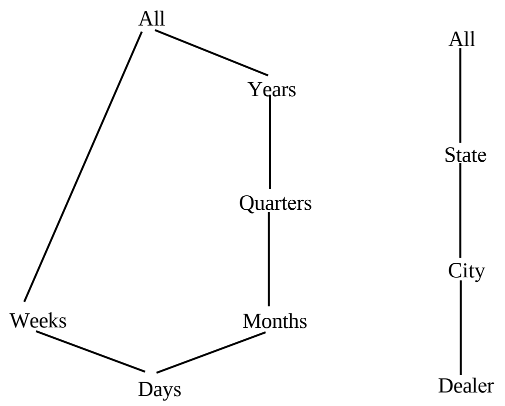

Figure 7 Examples of lattices partitions for time intervals and automobile dealers

Materialized views. A materialized view is the result of some query which is stored in the database, and which does not contain all aggregated values. To model the materialized view concept we introduce an element *mview* which is interpreted by means of an element *view*, and the last is based on the kernel *attribution* concept. When implementing the query over hierarchical dimension, a problem to choose an effective materialized view arises. In other words, if we have aggregated values regarding to granularity Months and Quarters then for aggregation regarding to granularity on Years it will be effective to apply query over materialized view with granularity Quarters. As in [5], we also consider the lattice (a partially ordered set) as a relevant construction to formalize the hierarchical dimension. The lattice nodes correspond to the units of the partitions of a dimension. In general, the set of partitions of a dimension is a partially ordered set. We say that partition P_1 is precedes partition P_2, written $P_1 \leq P_2$ if and only if there is a path from node P_1 to node P_2. Based on the lattices for each dimension we can define a lattice for all the possible materialized views of a data cube which are created by means of grouping according to some partition in each dimension. Let V_1 and V_2 be views, then $V_1 \leq V_2$ if and only if for each dimension of V_1 with partition P_1 and analogous dimension of V_2 with partition P_2 holds $P_1 \leq P_2$. Finally, let *V* be a view and *Q* be a query. We can implement this query over the considered view if and only if $V \leq Q$. To model the concept of hierarchical dimension we introduced an element *granularity* which is based on an empty element *partition*, and the latter is described by means of attribute *name*. The value of attribute *name* is the name of the granularity. Below, an example of data cube for an automobile company database is adduced [5] which is an instance of content dictionary of data integration concept. We consider Sales = {SerialNo, Dealer, Date, Price} as a data cube schema. The considered data cube is implemented on the base of materialized views and is based on three dimensions: Auto, Dealer and Date and has one dependent attribute: Value Set of partitions of dimension Date form a partially ordered set. We are using two granularity elements to represent this set.

```
<cd name = ‘dic’>
 ...
 <cube name = ‘example’>
  <felement>
   <omattr>
    schema definition of Sales
   </omattr>
  </felement>
  <delement>
   <element name = ‘Auto’/>
   <element name = ‘Dealer’/>
   <element name = ‘Date’/>
  </delement>
  <mview>
   <view name = ‘View1’>
    <omattr>
     definition of materialized view Sales1
    </omattr>
   </view>
   <view name = ‘View2’>
    <omattr>
     definition of materialized view Sales2
    </omattr>
   </view>
  </mview>
  <granularity name = ‘Date’>
   <partition name = ‘days’/>
   <partition name = ’months’/>
   <partition name = ‘quarters’/>
   <partition name = ‘years’/>
  </granularity>
  <granularity name = ‘Date’>
   <partition name = ‘days’/>
   <partition name = ’weeks’/>
  </granularity>
 </cube>
</cd>
```

The detailed discussion of the issues connected with applying the query language to integrated data is beyond the topic of this paper. Below the XML-formalization of data integration concept is presented.

<!-- include dtd for extended OPENManth objects -->

```
<!ELEMENT cd (dbsch|med|whse|cube)*>
<!ATTLIST cd name CDATA #REQUIRED>
<!ELEMENT dbsch (omattr)+>
<!ATTLIST dbsch name CDATA #REQUIRED>
<!ELEMENT med (msch,wrapper,constraint*)>
<!ELEMENT msch (omattr)>
<!ELEMENT wrapper (oma)>
<!ELEMENT constraint (oma)>
<!ATTLIST med name CDATA #REQUIRED>
<!ELEMENT whse (wsch,extractor,grid)>
<!ELEMENT wsch (omattr)>
<!ELEMENT extractor (oma)>
<!ATTLIST whse name CDATA #REQUIRED>
<!ELEMENT grid (dim+,chunk+)>
<!ELEMENT dim (stripe)+>
<!ELEMENT stripe EMPTY>
<!ELEMENT chunk (avg)+>
<!ELEMENT avg EMPTY>
<!ATTLIST dim name CDATA #REQUIRED>
<!ATTLIST avg value CDATA #IMPLIED
          dim CDATA #REQUIRED>
<!ATTLIST chunk id ID #REQUIRED
        qty CDATA #REQUIRED
        ref_to_db CDATA #REQUIRED
        ref_to_chunk IDREFS #IMPLIED>
<!ATTLIST stripe ref_to_chunk IDREFS #IMPLIED
                 min_val_CDATA #REQUIRED
                 rec_cnt CDATA #REQUIRED
                 max_val_CDATA #REQUIRED
                 chunk_cnt CDATA #REQUIRED>
<!ELEMENT cube (felement,delement,mview?,
                granularity*)>
<!ELEMENT felement (omattr)>
<!ELEMENT delement (element)+>
<!ELEMENT element EMPTY>
<!ATTLIST element name CDATA #REQUIRED>
<!ELEMENT mview (view)+>
<!ELEMENT view (omattr)>
<!ELEMENT granularity (partition)+>
<!ELEMENT partition EMPTY>
<!ATTLIST view name CDATA #REQUIRED>
<!ATTLIST granularity name CDATA #REQUIRED>
<!ATTLIST partition name CDATA #REQUIRED>
```

Figure 8 DTD for formalization of the data integration concept

4 Conclusion

The data integration concept formalization problems were investigated. The outcome of this investigation is a definition language of integrable data, which is based on the formalization of the data integration concept using a mechanism of the content dictionaries of the OPENMath. Supporting the concept of data integration is achieved by the creation of content dictionaries, each of which contains formal definitions of concepts of a specific area of databases.

The data integration concept is represented as a set of XML DTDs which are based on the OPENMath formalism. By means of such DTDs were formalized the basic concepts of database theory, metadata about these concepts and the data integration concept. Within our approach to data integration, an integrated schema is represented as an XML document which is an instance of an XML DTD of the data integration concept. Thus, modelling of the integrated data based on the OPENMath formalism leads to the creation of the corresponding XML DTDs.

By means of the developed content dictionary of the data integration concept we are modelling the mediator and the data warehouse concepts. The considered approach provides virtual and materialized integration of data as well as the possibility to support data cubes with hierarchical dimensions. Within our concept of data cube, the operators CUBE and ROLLUP are implemented. If necessary, in data integrated schemas new super-aggregate operators can be define. We use a computationally complete language to create schemas of integrated data. Applying the query language to the integrated data is generated a reduction problem. Supporting the query language over such data requires additional investigations.

Finally, modern trends of the development of database systems lead to the application of different divisions of mathematics to data analysis and management. In the frame of our approach to data integration, this leads to the use of corresponding content dictionaries of the OPENMath.

References

[1] Abrial, J.-R.: The B-Book: Assigning programs to meaning. Cambridge University Press (1996)

[2] Briukhov, D. O., Vovchenko, A. E., Zakharov, V. N., Zhelenkova, O. P., Kalinichenko, L. A., Martynov, D. O., Skvortsov, N. A., 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-Infrastructure of Virtual Observatories. Informatics and Applications, 2 (1), pp. 2-34, (2008)

[3] Date, C. J.: An Introduction to Database Systems. Addison Wesley, USA (2004)

[4] Drawar, M.: OpenMath: An overview. ACM SIGSAM Bulletin, 34 (2), (2000)

[5] Garcia-Molina, H., Ullman, J., Widom, J.: Database Systems: The Complete Book. Prentice Hall, USA (2009)

[6] Gevorgyan, G. R., Manukyan, M. G.: Effective Algorithms to Support Grid Files. RAU Bulletin, (2), pp. 22-38 (2015)

[7] Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Tab. In ICDE, pp. 152-159 (1996)

[8] Hindley, J. R., Seldin, J. P.: Introduction to Combinators and λ-Calculus. Cambridge University Press (1986)

[9] Kalinichenko, L. A.: Methods and Tools for Equivalent Data Model Mapping Construction. In EDBT, pp. 92-119, Springer (1990)

[10] Kalinichenko, L. A.: Integration of Heterogeneous Semistructured Data Models in the Canonical One. In RCDL, pp. 3-15 (1990)

[11] Kalinichenko, L. A., Stupnikov, S. A.: Constructing of Mappings of Heterogeneous Information Models into the Canonical Models of Integrated Information Systems. In Proc. of the 12th East-European Conference, pp. 106-122 (2008)

[12] Kalinichenko, L., Stupnikov, S.: Synthesis of the Canonical Models for Database Integration Preserving Semantics of the Value Inventive Data Models. In Proc. of the 16th East European Conference. LNCS 7503, pp. 223-239 (2012)

[13] Luo, C., Hou, W. C., Wang, C. F., Want H., Yu, X.: Grid File for Efficient Data Cube Storage. Computers and their Applications, pp. 424-429 (2006)

[14] Manukyan, M. G.: Extensible Data Model. In ADBIS'08, pp. 42-57 (2008)

[15] Manukyan, M. G., Gevorgyan, G. R.: An Approach to Information Integration Based on the AMN Formalism. In First Workshop on Programming the Semantic Web. Available: https://web.archive.org/web/20121226215425/http://www.inf.puc-rio.br/~psw12/program.html, pp. 1-13 (2012)

[16] Manukyan, M. G.: Canonical Data Model: Construction Principles. In iiWAS'14, pp. 320-329, ACM (2014)

[17] Manukyan, M. G., Gevorgyan, G. R.: Canonical Data Model for Data Warehouse. In New Trends in Databases and Information Systems, Communications in Computer and Information Science, 637, pp. 72-79 (2016)

[18] Nievergelt, J., Hinterberger, H.: The Grid File: An Adaptable, Symmetric, Multikey File Structure. ACM Transaction on Database Systems, 9 (1), pp. 38-71 (1984)

[19] Papadopoulos, A. N., Manolopoulos, Y., Theodoridis, Y., Tsoras, V.: Grid File (and family). In Encyclopedia of Database Systems, pp. 1279-1282 (2009)

[20] Regnier, M.: Analysis of Grid File Algorithms, BIT, 25 (2), pp. 335-358 (1985)

[21] Sharma, S., Tim, U. S., Wong, J., Gadia, S., Sharma, S.: A Brief Review on Leading Big Data Models. Data Science Journal, (13), pp. 138-157, (2014). Doi: http/doi.org/10.2481/dsj.14-041

[22] Stupnikov, S. A.: A Varifiable Mapping of a Multidimensional Array Data Model into an Object Data Model, Informatics and Applications, 7 (3), pp. 22-34 (2013)

[23] Stupnikov, S. A, Vovchenko, A.: Combined Virtual and Materialized Environment for Integration of Large Heterogeneous Data Collections. In Proc. of the RCDL 2014. CEUR Workshop Proceedings, 1297, pp. 339-348 (2014)

[24] Stupnikov, S. A, Miloslavskaya, N. G., Budzko, V.: Unification of Graph Data Models for Heterogeneous Security Information Resources' Integration. In Proc. of the Int. Conf. on Open and Big Data OBD 2015 (joint with 3rd Int. Conf. on Future Internet of Things and Cloud, FiCloud 2015). IEEE 2015, pp. 457-464 (2015)

[25] Zakharov, V. N., Kalinichenko, L. A., Sokolov, I. A., Stupnikov, S. A.: Development of Canonical Information Models for Integrated Information Systems. Informatics and Applications, 1 (2), pp. 15-38 (2007)

[26] MongoDB. https://www.mongodb.org