Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

Article

A Proposal of Methodology for Designing Big Data Warehouses

Francesco Di Tria, Ezio Lefons and Filippo Tangorra *

Dipartimento di Informatica, Università degli Studi di Bari “Aldo Moro”, Via Orabona 4, I-70125, Bari, Italy
* Correspondence: filippo.tangorra@uniba.it; Tel.:+39-080-544-3281

Abstract: Big Data warehouses are a new class of databases that largely use unstructured and volatile data for analytical purpose. Examples of this kind of data sources are those coming from the Web, such as social networks and blogs, or from sensor networks, where huge amounts of data may be available only for short intervals of time. In order to manage massive data sources, a strategy must be adopted to define multidimensional schemas in presence of fast-changing situations or even undefined business requirements. In the paper, we propose a design methodology that adopts agile and automatic approaches, in order to reduce the time necessary to integrate new data sources and to include new business requirements on the fly. The data are immediately available for analyses, since the underlying architecture is based on a virtual data warehouse that does not require the importing phase. Examples of application of the methodology are presented along the paper in order to show the validity of this approach compared to a traditional one.

Keywords: Big data technology; Business intelligence; Data integration; System virtualization.

1. Introduction

Data warehouses are multidimensional databases used in analytical processing. The traditional architecture is usually based on a multi-level strategy, where (a) the *data source layer* contains heterogeneous databases, which can be internal or external to the information system, (b) the *data warehouse layer* stores synthetic and aggregated data, and (c) the *analytic layer* runs the applications used to perform analysis and to deploy reports and charts for business managers. In this kind of architecture, an ETL (Extraction, Transformation, and Loading) phase is necessary for extracting data from sources, performing both transformation and cleaning operations, and finally feeding the data warehouse [1].

In this context, data warehouses are designed according to opposite approaches: (a) the data-driven approach, which consists in a reengineering of data sources on the basis of the designer’s experience and, then, a multidimensional schema that does not satisfy user requirements might be created; and (b) the requirement-driven one, which aims at considering only the business goals of decision makers, and, then, it may happen that the designer discovers that the needed data are not currently available at the sources when the ETL process is deployed [2].

These solutions are not able to address problems arising in the big data context. In fact, in literature, the issues related to Big Data are always summarized with the 5 *V*s problems, or *Volume*, *Variety*, *Velocity*, *Veracity*, and *Value* [3]. As concerns the field of data warehouse in the context of Big Data, these problems are characterized as follows.

Volume. New data sources, such as social networks and sensor networks, daily generate massive data [4]. So, the traditional multi-level architecture with an ETL phase is impracticable when dealing with Big Data, due to the impossibility to move and store tens of terabytes or more.

Variety. Data generated from the Web are sometimes unstructured, such as tweets or blog posts. These data are of great importance, since they are able to provide valuable feedbacks to companies about user preferences [5, 6]. Traditional design methodologies fail in presence of unstructured data. Indeed, data-driven approaches need well-structured data, since functional dependencies are taken

© 2018 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

into account in the remodeling phase. On the other hand, in requirement-driven approaches multidimensional concepts are mapped against schemas of data sources, when the ETL phase is designed. So, unstructured data containing an useful information of interest, albeit available, may be omitted and not exploited.

Velocity. New data sources arise very frequently and, then, a key factor of success depends on the ability to exploit new available data as soon as possible in order to be competitive [7]. Furthermore, new requirements may emerge as a consequence of the actual business trend. These new requirements affect the current data warehouse that may become obsolete. Therefore, traditional methodologies that present a slow reaction to changes in business requirements and do not allow a fast adjustment to a data warehouse are not suitable.

Further issues related to Big Data are *Veracity*, which refers to data accuracy, and *Value*, which deals with the benefits that can be obtained. Indeed, without an ETL phase, which is responsible to guarantee data quality for decision making, software applications have to apply filtering solutions in order to discard data that may lead to wrong information. Similar solutions are quite expensive in terms of development time and monetary costs. Therefore, designers and managers should carefully consider if a Big Data architecture is the right solution, by evaluating and comparing both costs and benefits in a given business context.

We adopt the term Big Data Warehouse (BDW) as a reference to a data warehouse that is characterized by the 5 *V*s. The solution proposed in literature for the creation of a BDW is the adoption of the Hadoop framework, which is usually used in conjunction with a traditional data warehouse [8]. The result is a complex architecture that determines the movement and the migration of large quantities of data among specialized systems. In the paper, we propose an alternative methodology that addresses Big Data issues and, in particular, focuses on the *Velocity* problem.

The methodology provides solutions to quickly consider further requirements emerged in the lifecycle, on the basis of agile methods that address the frequent changes in user requirements, in order to have the minimum impact on the design process. Furthermore, the design process is largely based on automatic phases that reduce the design efforts and support the designer by avoiding errors and repetitive tasks. On the basis of automatic phases, we are also able to include and to integrate new data sources on the fly, as required by the business goals. So, the data warehouse evolves rapidly in a manner consistent with business needs. The data to be used in the analytical phase are immediately available in the underlying architecture, for the movement of data among systems is avoided and the delays of the importing phase for feeding the data warehouse are discarded. The strategy is based on a virtual data warehouse and a set of software agents to wrap, to manage data sources, and to export data when needed for analytical purposes.

To complete the description, the methodology is classified as hybrid, since it takes into account the best features of traditional methodologies. As a counterpart, the methodology is quite complex because it integrates and reconciles both the requirement and the data oriented approaches [9]. The design methodology is based on two models that organize data both at the conceptual level, for representing multidimensional concepts in a graphical way, and at the logical level, for representing multidimensional schemas in a flexible way in reference to target NoSQL systems.

The paper is structured as follows. Section 2 provides an overview of traditional data warehouses *vs* BDWs. Then, in Section 3 we describe the Big Data Warehouse architecture. In Section 4, we present the conceptual model to define the multidimensional concepts and the logical model to translate those multidimensional concepts into a key-value structure. In Section 5, the complete design methodology is presented, which is illustrated by highlighting the automatic and incremental steps using a working example. Section 6 reports the related work about BDW. Section 7 concludes the paper with our remarks.

2. Data Warehouse *vs* Big Data Warehouse

Traditional data warehouses usually adopt a two-level architecture, where the analytical level is opposed to the data warehouse. However, in the architecture, three physical layers can be observed:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

- *data source layer*, that contains heterogeneous databases, which can be internal or external to the information system;
- *data warehouse layer*, that stores synthetic and aggregated data; and
- *analytic layer*, that runs the applications used to create and to deploy reports and charts by applying OLAP (On-Line Analytical Processing) operators.

In this kind of architecture, an ETL phase is necessary for periodically loading data into the data warehouse according to a refresh strategy [10]. However, there exists a variation to this architecture, where the ETL process does not feed directly the data warehouse but a global and reconciled database, that, on turn, feeds the data warehouse (three-level architecture).

On the other hand, a BDW should adopt a one-level architecture, where only the data source layer and the analytic layer can be observed. To do this, such an architecture presents a virtual BDW, which can be thought as a web-service that exposes a set of metadata derived from the multidimensional modeling and a query language. So, the ETL phase is not executed and, then, there is no need to move tens of terabytes during the feeding process. However, saving storage space and loading time implies higher times in analytical processing, since the BDW is not a physical object and, therefore, all the queries must be redirected to data sources and locally evaluated.

While the multidimensional modeling—executed in the conceptual design—strongly depends on the design methodology [11, 12], the logical design is usually based on

- the relational Model (ROLAP), producing popular star or snow-flake schemas
- multidimensional structures (MOLAP), or
- hybrid solutions (HOLAP).

However, emergent logical models, widely known as non-relational, are utilized in the Big Data context for NoSQL databases: the key-value, column-oriented, graph-oriented, and document-oriented models. They allow a more flexible design, due to their schema-less nature, and present a strong denormalization of data. Furthermore, NoSQL databases are characterized by low answering times and their ability to exploit horizontal scalability [13].

Traditional design methodologies are classified as data-driven or requirement-driven. Data-driven methodologies aim at performing a reengineering of an integrated and well-structured data source; this process is mainly based on the designer's experience, but it can be also supported by algorithms able to detect relationships and to modify data structures. These are known as CASE (Computer Aided Software Engineering) tools. Data-driven methodologies lack of a strategy to manage unstructured data sources that, by definition, are schema-less. Conversely, requirement-driven methodologies do not consider data sources, but rely on business goals solely and, then, solve every problem in the ETL phase. In this case, not only the effort to populate the data warehouse may be a hard task, due to the difficulty to correctly map multidimensional concepts against attributes of data sources, but the designer may also discover that the needed data are not effectively available. Of course, in presence of unstructured data, the mapping strategy cannot be resolved at all. So, the designed data warehouse is completely useless and must be revised.

Since each of these approaches presents both benefits and disadvantages, recent methodologies are hybrid, that is, they are devoted to integrate the different design strategies [14, 15, 16, 17, 18]. These can be classified in pure hybrid methodologies and integration-derived hybrid methodologies. The former group includes all the methodologies that perform the design process considering simultaneously the data sources and the business goals. The latter comprises methodologies that combine and integrate a data-driven approach with a requirement-driven one. On turn, integration-derived hybrid methodologies can be divided into sequential hybrid and parallel hybrid methodologies. In sequential hybrid methodologies, the two stages are executed according to a prefixed order, and the output of the first stage is used as input of the second stage. In parallel hybrid methodologies, the two stages are executed independently and, at the end, the comparison and integration of the schemas coming from the different stages are performed. For these reasons, hybrid methodologies require a lot of effort, when manually executed and, then, the current research is devoted to introduce automatisms to support the designer in the management of

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

several data sources and to integrate new data sources on the fly [19, 20, 21], also using ontologies for automatically solving syntactic and semantic inconsistencies. Another aspect is the adoption of agile approaches in order to quickly consider further requirements and to update the data warehouse accordingly, without performing a complete design process starting from zero [22, 23].

These key factors—automation and agility—are independent and optional issues that are considered also in traditional data warehouses, but these become mandatory in the big data context, because new data sources arise very frequently and new business requirements must be processed to keep pace with competitors and to be leaders in the commercial and financial sectors. The differences between a traditional Data Warehouse and a BDW are summarized in Table 1.

Table 1. Data Warehouse *vs* Big Data Warehouse.

		Data Warehouse	*Big Data Warehouse*
Architecture	*Levels*	two, three	one
	ETL	yes	no
Logical model		ROLAP, MOLAP, HOLAP	non-relational
Methodology	*Strategy*	data-driven, requirement-driven, hybrid	hybrid
	Automation	optional	mandatory
	Agility	optional	mandatory

3. Architecture

The BDW architecture is depicted in Figure 1.

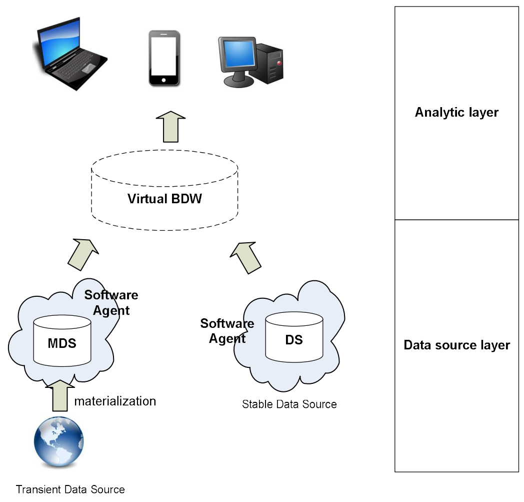


Figure 1. Big data warehouse architecture.

Here, the most important role is played by data sources, where analytical queries are effectively answered. We classify data sources according to two categories: (i) *Transient data sources*. These include all data sources that generate data having a fixed interval of life, such as posts on social networks or log files of servers. Transient data sources need to be materialized, by creating a Materialized Database (MDB), in order to preserve data that can be lost. (ii) *Stable data sources*. These include data sources (DSs) such as databases in any format, such as relational databases, XML files, and Excel documents. These are physical resources, persistent in a storage device.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

MDB and DSs are managed by software agents (SAs) that act as wrappers and aim at performing a pre-processing of data. In detail, they execute the following tasks: (1) *Verifying data quality*. Because of the absence of the ETL phase, the SAs ensure data quality, by correcting errors present in a data source. Whenever errors are impossible to delete, the SAs detect which data can produce valuable information and can be effectively used for decision making. Then, each SA present to the analytic layer only filtered data, those that satisfy quality criteria. (2) *Reducing data cardinality*. Compressing datasets—using techniques such as sampling, histograms, or analytic methods—is a preliminary phase to perform approximate query processing [24], which provides fast answers to analytical queries by creating data synopses. The approximate answers can be affected with a small quantity of error. However, they carry the same information as the exact answers and, therefore, they are widely accepted in decision making processes. The underlying assumption is that an approximate answer is better than the exact answer if the query takes a long time to be completed and the approximation error is low. Analytical queries are always answered loading the opportune data synopses into the central memory and using data synopses instead of accessing a data source. So, SAs compress data sources and quickly return answers to the analytic layer. (3) *Providing an abstract data model*. SAs should also act as wrappers, hiding different data models and providing an abstract data model that agrees with that of the BDW. Therefore, if the BDW is a NoSQL database using a key-value model, then each SA in a data source must provide primitives such as get, put, and delete.

At last, BDW must be accessed by any device, such as mobile devices and tablets. On these platforms, Big Data Analytics software tools allow user to create applications for performing data analyses. Usually, these applications include reports and charts obtained through OLAP operations and score-carding, which is devoted to the definition and the analysis of Key Performance Indicators. Finally, charts and reports are visualized using dashboards. Advanced applications focus on data mining algorithms and what-if analyses, which aim at detecting hidden relationships and regular patterns in datasets. These products are the building blocks for deploying information, knowledge, and experience to be used in the decision making process, in order to improve the business activities of the information system.

3.1. Metadata

Using a virtual data warehouse, all the queries must be mapped onto the data sources. To this end, BDW should be opportunely described by a set of metadata. The metadata can be used by code generators to generate analytical queries and to redirect them onto the data sources.

So, the first aim of metadata is to describe the multidimensional elements of a data warehouse and their relationships. As an example, this is used to perform roll-up and drill-down operations. In web environments, metadata are usually stored in XML files. An example is the metadata representation adopted by Mondrian [25], a popular OLAP application based on the MDX language.

The second aim of metadata is to trace each data source. In this way, it is possible to know, for each multidimensional object (*ie,* cubes and dimensions), which are the data sources that hold data, along their locations (*ie,* the physical parameters, such as the IP address).

4. Models

For the design process, we first rely on a conceptual model that organizes data according to a high level of abstraction, and, then, on a logical model that organizes data in reference to a specific class of target systems.

4.1. Conceptual model

The conceptual model aims at representing multidimensional concepts using a tree-based schema.

Let **G** = (N, E) be a tree, where:

- N = {A_1, A_2, …, A_n} is a set of *n* nodes,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

- E = {(A_i, A_j) | $i \neq j$} ⊂ N×N is a set of oriented edges (from A_i to A_j), and
- A_1 is the root of **G**.

To represent multidimensional concepts, we assume A_1 is the fact. In general, a node represents a concept, while an edge between nodes represents a relationship. The children of the root that are leaf-nodes are measures. The number of outgoing edges, except those towards measures, determines the dimensionality of the fact. The children of the root that are not leaf-nodes are the first hierarchical levels of a dimension and determine the granularity of the fact. So, these nodes are the root of a sub-tree representing a dimension and the set of nodes linked by an edge form a hierarchy.

As an example, a tree for the sales is represented in Figure 2. Here, *sale* is the fact and *quantity* is its measure. This fact can be viewed as a three-dimension cube. The first dimension is *time*, having one hierarchy with three levels: *day*, *month*, and *year*. The second dimension is *product*. This presents two hierarchies: one is composed of levels *product* and *supplier*, the other composed of *product* and *category*. The third dimension is *location*, composed of one hierarchy with two levels: *city* and *region*.

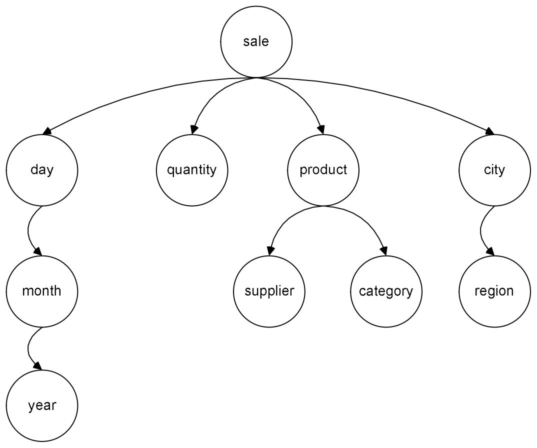


Figure 2. Sale tree.

Sometimes, a tree can degenerate into a graph, if a node presents more than one ingoing edges. For instance, let us consider a further hierarchy *day* → *week* → *year*, besides that *day* → *month* → *year* depicted in Figure 2. Now, the *year* node has two fathers and then the resulting graph presents two linearly-independent paths. Each of these paths is a different hierarchy of the *time* dimension. In the rest of the paper, we use the term *attribute tree* for identifying the graphical representation of the multidimensional concepts, even if a node presents more than one father.

A tree can be refined adding further nodes to hierarchical levels in order to provide descriptive attributes. For example, we may add the node *price* as child of *product* in order to represent the price of a given product. Hereinafter, we assume that each hierarchical level may present zero or more descriptive attributes. Therefore, a descriptive attribute is a leaf-node that is a child of a hierarchical level. This schema can be then remodeled on the basis of traditional operations on graphs in order to obtain a multidimensional schema.

In what follows, in reference to attributes A and B, A and B denote the nodes representing the corresponding attributes, and $A \rightarrow B$ the edge existing from the node A to the node B. In case of branches between nodes A and B simultaneously referring each other (as an example, this happens when a relation has primary key A and alternative key B, or when two relations are mutually referencing each other via the respective foreign keys A and B), the "loop" $A \leftrightarrows B$ is solved algorithmically by a node-splitting and renaming (A, for example). That is, the loop $A \leftrightarrows B$ generates the (sub-)tree $A \rightarrow B \rightarrow A'$.

The basic operations on a graph are:

1. create_root(A), creating the root A;
2. create_node(A), creating the node A;

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

3. delete_node(A), deleting the node A;
4. create_edge(A,B), adding an edge from A to B, and
5. delete_edge(A,B), removing the edge from A to B.

However, complex operations could also be defined:

6. prune(A), removing the A node with all its children;
7. graft(A), removing the A node and adding its children to its parent, and
8. change_parent(A,B,C), for the edge B → C and node A, change parent of C from B to A means:
   (a) delete_edge(B, C), and
   (b) create_edge(A, C).

Accordingly, the four basic operations (*ie*, 2 to 5) defined on the tree correspond respectively to: creating attribute A, deleting attribute A, adding the functional dependency A → B, and removing the functional dependency A → B. Moreover, the change parent operation is very useful to modify hierarchical dimensional levels. Therefore, the basic operations allow performing a reengineering of schemas using a completely data-driven approach. So, if we do not remodel the tree on the basis of the designer's experience and choice only, but we also consider user needs coming from the requirement analysis, then we obtain a hybrid approach. As a further evolution, if we define an algorithm that applies the defined operations on a graph on the basis of a set of constraints derived from the requirement analysis, then we can remodel the tree automatically.

4.2. Logical model

The logical data model chosen for Big Data Warehouses is the key-value non-relational model [13]. According to the key-value representation, the value is a byte sequence uniquely identified by a key, which can be generated by the user or by the system. It is possible to associate a schema to the value in order to create a data structure [26].

We introduce the term *multidimensional document* as a reference to a data structure representing a cube along with its dimensions. In detail, a multidimensional document is a structured and denormalized set of key-value pairs, where the key represents the cube's dimensions, while the value stores the measures and the hierarchies.

As an example, the star schema based on the Relational OLAP (ROLAP) model depicted in Figure 3 can be effectively described by the multidimensional document named *Order* reported in Table 2.

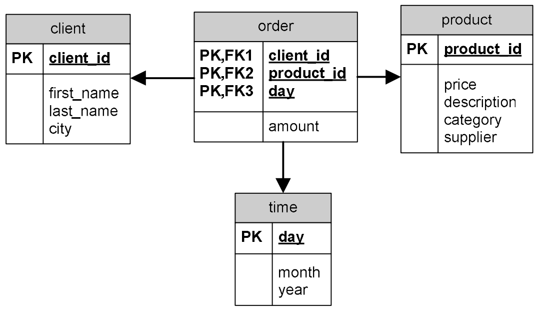


Figure 3. Order star schema.

Table 2. Multidimensional document of Order fact.

Order	
key	**value**

`client_id`	`amount`
`product_id`	`{product_id, price, description, category, supplier}`
`day`	`{client_id, first_name, last_name, city}`
	`{day, month, year}`

Assuming to perform the map of the relational model to the key-value, the key in the multidimensional document is composed of the key(s) of the fact table, while all other attributes are included in the value. (In order to group attributes of the same relation, nested objects can be created.)

It is worth noting that, when dealing with star or snow-flake schemata, a metadata file is always created in order to represent multidimensional concepts. As an example, it is not possible to infer from the schema in Figure 3 that *category* is a hierarchical level of the *product* dimension, while *description* is a simple descriptive attribute of a product and this is not used for aggregation purposes. Similarly, in the nested object `{product_id, price, description, category, supplier}` of the *product* dimension in the multidimensional document, we cannot distinguish a descriptive attribute from a dimension level. So, also in this case, we have to rely on metadata for the correct identification of concept. For the sake of simplicity, the metadata file describing hierarchies is used by OLAP applications for roll-up and drill-down operations, for example.

5. Models

The design methodology we propose is named GRAHAM, which stands for GRaph-based Agile Hybrid Automatic Methodology. GRAHAM is: (i) *agile*, for it adopts an incremental step to quickly modify an existing data warehouse when new business requirements are produced; (ii) *hybrid*, for it considers both data-oriented and requirement-oriented approaches; and (iii) *automatic*, for it uses algorithms to perform the design process in a supervised way, avoiding design errors and repetitive task. As a further consequence, new data sources can be integrated on the fly [16].

The phases of the methodology are depicted in Figure 4. They are classified as follows:

1. *Requirement analysis*, which is based on the *i** framework to represent user requirements;
2. *Source integration*, which produces an integrated schema by reconciling data sources using an ontological approach [27, 28, 29, 30].
3. *Conceptual design*, which represents multidimensional concepts using a tree-based schema;
4. *Logical design*, which is based on the multidimensional documents using the key-value logical model;
5. *Physical design*, which aims at implementing the necessary metadata exposed by the virtual data warehouse.

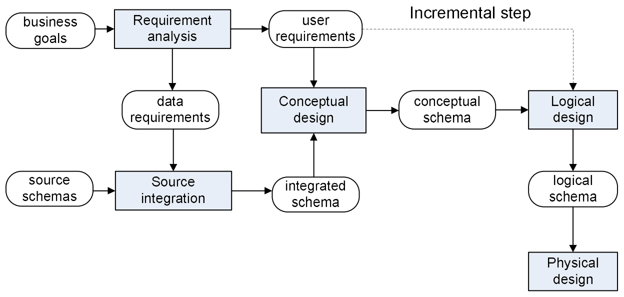


Figure 4. Design Methodology.

5.1 Requirement analysis

In this phase, the needs of the decision makers are investigated. To this aim, we adopt the *i** framework [12], which allows to explicitly represent *business goals* in reference to the *actors*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

considered in the system. In the application of *i** to data warehousing, we observe three main categories of actors: the decision makers, the data warehouse itself, and the data sources. Each actor performs specific *tasks*, in order to achieve his/her/its own goals by accessing *resources*.

To do this, the designer must first produce a model describing the *dependency* relationships among the main actors of the organization, along their own interests. This model is the so-called *strategic dependency* model and aims at outlining how the data warehouse helps decision makers to achieve business goals.

In this high-level model, a decision maker has to achieve a set of goals and needs resources, which must be provided by the data warehouse. On turn, the latter depends on data sources, which can be modeled as a further actor of the system.

Then, each actor in the strategic dependency model is further detailed in a *strategic rationale model* that shows the specific tasks the actor has to perform in order to achieve business goals, which are structured in sub-goals by *means-end* links.

User requirements, alias business goals, are exploded into a more detailed hierarchy of nested goals: (a) *strategic goals*, or high-level objectives to be reached by the actor; (b) *decision goals*, to answer how strategic goals can be satisfied; and (c) *information goals*, to define which information is needed to accomplish a given goal.

From information goals, *information requirements* must be derived. These represent specific tasks to be performed in order to obtain the needed information. In the case of decision makers, the tasks are high-level queries strictly related to multidimensional concepts. Resources are also linked to tasks by decomposition links, in order to provide a context such as measures or aggregation levels. The model elements are summarized in Figure 5.

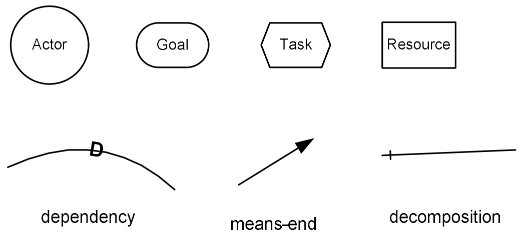
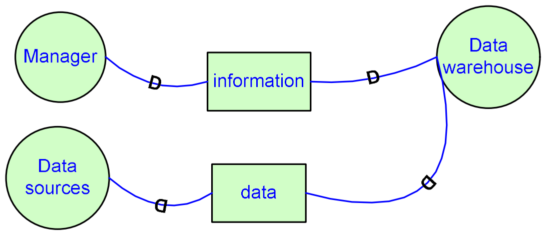


Figure 5. *i** model elements.

Using the information requirements of the strategic rationale model, we extract data requirements, user requirements, and constraints.

In detail, *data requirements* are extracted from the tasks of the *Data sources* actor and used in the source integration phase to establish which data sources are to be considered. In case of unstructured data, these requirements are used to perform a process of structurization and normalization. On the other hand, *user requirements* are extracted from the tasks of the decision makers and used for generating a *workload*, which helps in validating the final conceptual schema before translating it into a logical schema. Finally, *constraints* are extracted from the tasks of the *Data warehouse* actor and used to perform the conceptual design in an automatic way. For these reasons, the methodology is classified as hybrid and automatic.

As an example, Figure 6 is the strategic dependency model that shows the main actors of a system in the commercial context.



Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

Figure 6. Strategic dependency model.

The first actor is the *Manager*, who is a decision maker interested in extracting information from the *Data Warehouse*. On turn, the latter needs to obtain data from the *Data Sources*, in order to accomplish its goal(s). Then, each actor of the strategic dependency model is exploded into a set of nested goals (*see* Figure 7).

As an example, the *Manager* aims at increasing the profits as his/her strategic goal. To do so, s/he needs to increase sales and, maybe, also to decrease costs (not shown in the figure). To satisfy this decision goal, s/he needs information about which are the most important clients and information about the trend of the products on social networks. In order to obtain this information, the decision maker has to do some tasks, such as counting the times each product is cited in a forum or a blog. All the tasks of the decision maker represent a preliminary workload to be used at the end of the design process to validate the final conceptual schema, before translating it into a logical schema. A similar work must be done for the other actors of the system.

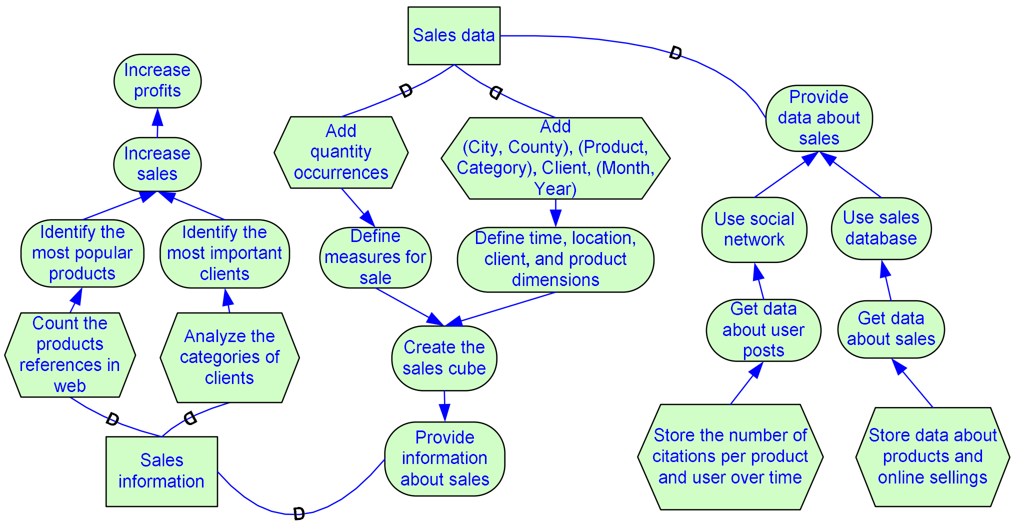

Sales data
Increase profits
Increase sales
Add quantity occurrences
Add (City, County), (Product, Category), Client, (Month, Year)
Provide data about sales
Identify the most popular products
Identify the most important clients
Define measures for sale
Define time, location, client, and product dimensions
Use social network
Use sales database
Count the products references in web
Analyze the categories of clients
Create the sales cube
Get data about user posts
Get data about sales
Sales information
Provide information about sales
Store the number of citations per product and user over time
Store data about products and online sellings
D

Figure 7. Strategic rationale model.

5.2 Source integration

The source integration is devoted to produce a single schema, which reconciles the different data sources, by solving syntactic and semantic inconsistencies. So, the preliminary step is the source analysis, where data sources are understood. In case of structured data, this means understanding the meaning of all the relations and attributes.

On the other hand, when dealing with unstructured data, these can be normalized in several ways, depending on the query to be executed. To make an example, the text of a public post in a forum or in a blog can be used to count the occurrences of the words or, alternatively, to calculate the message length. So, we need some requirements that guide us in the decision about how data should be normalized.

Formally, from the data requirement (which can be thought as a sql query) in the form R[X].[Y], where:

- R is the name of the data requirement,
- X = $\{X_1, \ldots, X_i\}$ the list of the key attributes, and
- Y = $\{X_{i+1}, \ldots, X_k\}$ the list of the numeric attributes to be computed such that X → Y,

we produce the relation $R(X_1, \ldots, X_k)$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

After all the, say n ($n \geq 1$), data sources have been structured and normalized, the designer has to retrieve or produce, for each of them, a conceptual schema S_i ($i = 1, 2, \ldots, n$), along with a data dictionary, storing the description in natural language of the concepts modeled by the database. Then, the integration process proceeds incrementally using an opportune binary operator *integration* that has two conceptual schemas as operands and produces a global conceptual schema G.

Statement. Given the conceptual schemas $S_1, \ldots, S_n$, $n \geq 1$, we assume the following recurrence:

- $G_1 = S_1$, and
- $G_i = \text{integration}(S_i, G_{i-1})$, for $i = 2, \ldots, n$. □

In detail, the integration process of two databases S_i and S_j (or, G_j) is composed of the following steps:

S1. *Ontological representation*. In this step, we consider an ontology describing the main concepts of the domain of interest. If such ontology does not exist, it must be built by domain experts. The aim is to build a shared and reusable *ontology* that does not evolve rapidly over time. So, if such an ontology is built, it is predisposed also for further concepts to be considered when facing new business requirements. □

S2. *Predicate generation*. For each concept in the ontology, we introduce a unary predicate. The output of this step is a set of *predicates*, which represents a vocabulary to build definitions of concepts using the first-order logic. □

S3. *Entity definition generation*. For each entity present in the data sources (and, then, in the intermediate database G_j) and described in the data dictionary, we introduce a definition using ontology-derived predicates and binary relationships. Therefore, an entity definition is a logic-based description of a concept in the database. The output of this step is a set of *entity definitions*. □

S4. *Similarity comparison*. Assuming that similar entities have a very close description, we can detect whether (i) entities that have different names refer to the same concept, and (ii) entities that have equal names refer to different concepts. To do so, we utilize a set of inference rules, the so-called *similarity comparison rules*, to analyze the logic-based descriptions and a metric to calculate the pairwise similarity of entity definitions [31].

In detail, let $S_i(A_{i1}, \ldots, A_{ik})$ and $S_j(A_{j1}, \ldots, A_{jm})$ be two schemas, where A_{tv} denotes an entity of schema S_t. We compare the logic definition, say, L_1 of A_{ih}, $1 \leq h \leq k$, with that, say, L_2 of A_{jq}, $1 \leq q \leq m$. For each comparison, we calculate the output list $L \approx L_1 \cap L_2$ of the ontological concepts shared by those logic definitions and the similarity degree σ computed as

$$\sigma(l, l_1, l_2) = \frac{1}{2}\left(\frac{l+1}{l+l_1+2} + \frac{l+1}{l+l_2+2}\right) \tag{1}$$

where:

- $l = |L|$ or the number of common predicates of L_1 and L_2,
- $l_1 = |L_1 - L|$ or the number of predicates $p \in L_1$ such that $p \notin L_2$, and
- $l_2 = |L_2 - L|$.

The similarity degree σ is then transformed into the interval [0, 1], using α and β, where α is the minimum value σ can assume in the worst case (*ie*, when L_1 and L_2 do not present any common concept), while β is the maximum value σ can assume in the best case (*ie*, when L_1 and L_2 present the highest possible number of common concepts).

Formally, to compute α:

- $l = 0$,
- $l_1 = |L_1|$, and
- $l_2 = |L_2|$,

and to compute β:

- $l = \min(|L_1|, |L_2|)$,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

- $l_1 = |L_1| - |L_2|$, and
- $l_2 = |L_2| - |L_1|$.

When comparing the logical definitions of entities A_{ih} and A_{jq} and calculating their similarity degree σ and shared list L of concepts, one of the following cases can observed:

S4.a. *Equivalent entities.* This holds true if σ=1 (*ie*, $L_1 = L_2$).
S4.b. *Specialization.* A_{ih} and A_{jq} are both *specializations* of a concept present in the ontology if $0 < \sigma < 1$ (*ie*, $L \neq \varnothing$).
S4.c. *Distinct entities.* This holds true if σ=0 (*ie*, $L = \varnothing$). □

S5. *Integrated logical schema generation*. An intermediate *global conceptual schema* $G_u(B_{u1}, \ldots, B_{up})$ is built using the results obtained from the similarity comparison process of $S_i(A_{i1}, \ldots, A_{ik})$ and $S_j(A_{j1}, \ldots, A_{jm})$ and applying some generation rules. If more than one cases listed in step S4 are detected (both cases S4.a and S4.b, for example) only one generation rule is applied; the first one that applies in the list below.
For $s = 1, \ldots, p$, the generation rules are as follows.

S5.a. $B_{us} := A_{ih}$ ($\approx A_{jq}$) if we observe case S4.a. Here, all the distinct attributes are merged in B_{us}.
S5.b. $B_{us} := \{L, A_{ih}, A_{jq}\}$ if we observe case S4.b. Here, all the common attributes are associated to L.
S5.c. $B_{us} := \{A_{ih}, A_{jq}\}$ if we observe case S4.c. Here, all the entities preserve their own attributes.

As concerns the cardinality constraints, we adopt the following solutions. When entities A_i and A_j are merged, we use the minimum of the respective minimum cardinality constraints and the maximum of the respective maximum cardinality constraints for each relationship in which A_i and/or A_j participate. In all the other cases, the original cardinalities are maintained. □

At last, the *global conceptual schema* G_u is translated into an integrated logical schema by applying the well-known mapping algorithms [32]. It is worth noting that both attributes and cardinality constraints are retrieved from the *Data Dictionary* with no elaboration. Only entity definitions are converted into predicates for comparison. The graphical representation of the whole integration process is shown in Figure 8.

When a further schema S_l has to be integrated, the integration process starts from step S3, using the previous result G of step S5 and the schema S_l.

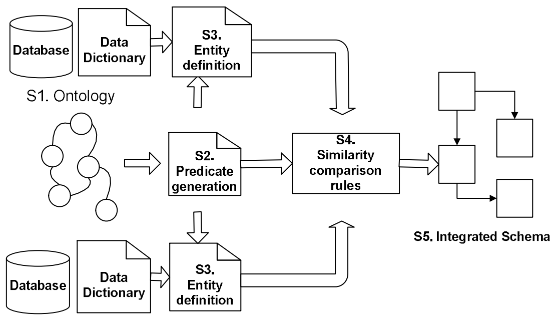


Figure 8. Integration process of (two) databases.

5.2.1. Schema integration example

In traditional data warehouses, *Identifying the most popular products* is an aggregation problem, for we have to count the number of items sold per product using the database internal to the information system. On the other hand, in the Big Data context, such analysis requires a different approach, for we have to use also unstructured data posted in social networks and web-services, in order to consider not only the real-time online sellings but also the products' reputations in forums by users' posts.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

On the basis of the constraints extracted from the tasks of the *Data Sources*, we are enforced to use (a) the internal relational database storing data about sales, clients, and products, updated with data coming also from online databases; and (b) statements posted by users on social networks. The latter is a volatile data source that first needs to be materialized into a set of text files containing users' posts. Unfortunately, these are unstructured data that can be aggregated in several ways. However, the constraints (those labeled as *data requirements*) state that we have to *count the number of citations per product and user over time*, formally *citation[person, product, date].[occurrences]* (*see* the beginning of subsection 5.2). Therefore, we may normalize unstructured data by creating relations *person(<u>user_id</u>, first_name, last_name)*, *product(<u>product_id</u>, name)*, and *citation(<u>person</u>, <u>product</u>, <u>date</u>, occurrences)*.

Through a reverse engineering, we define from data sources the main entities of the domain, in order to compare the concepts. These are shown in Table 3. Notice that at the moment we do not compare relationships. Assuming to compare each entity of the first data source with all the entities of the second database by applying the similarity metrics, we obtain the scores reported in Table 4.

Table 3. Database entities.

Social Network :: Person *Who has an account in the system and interacts with other users*	Person(X) ⇐ socialBeing(X) ∧ individualAgent(X) ∧ has(X,Y) ∧ humanRelationship(Y) ∧ has(X,Z) ∧ userAccount(Z).
Social Network :: Product *An item to be sold, which has a producer and a price*	Product(X) ⇐ temporalThing(X) ∧ partiallyTangible(X) ∧ has(X,Y) ∧ producer(Y) ∧ has(X,Z) ∧ monetaryValue(Z).
Sale :: Client *A person who has an account in the system and buys products*	Client(X) ⇐ socialBeing(X) ∧ individualAgent(X) ∧ has(X,Y) ∧ userAccount(Y) ∧ clientele(Z,X) ∧ supplier(Z).
Sale :: Product *An item to be sold, which has a producer and a price*	Product(X) ⇐ temporalThing(X) ∧ partiallyTangible(X) ∧ has(X,Y) ∧ producer(Y) ∧ has(X,Z) ∧ monetaryValue(Z).
Sale :: Category *Collection of products*	Category(X) ⇐ productType(X).
Sale :: Order *A document sent by an ordering agent to a supplying agent to order some item(s)*	Order(X) ⇐ legalForm(X) ∧ has(X,Y) ∧ supplier(Y) ∧ has(X,Z) ∧ customer(Z) ∧ clientele(Y,Z).

As a comparison example, we consider *Client* and *Person*, that are both composed of six predicates. So, $\alpha=0.125$, for $l=0$, $l_1=6$, and $l_2=6$, while $\beta=0.875$, for $l=6$, $l_1=0$, and $l_2=0$. Making the comparison, we have that $\sigma=0.625$, for $l=4$, $l_1=2$, and $l_2=2$, because these present 4 ontological concepts in common indeed. The mapping of σ from [0.125, 0.875] to [0, 1] is 0.6 (*see* Table 5).

Table 4. Similarity degree.

σ	Client	Product	Category	Order
Person	0.625	0.125	0.229	0.125

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

Product	0.125	0.875	0.229	0.125

Table 5. Normalized similarity degree.

σ	Client	Product	Category	Order
Person	0.6	0	0	0
Product	0	1	0	0

Observing experimental values, the only overlapping concepts are *Person* that maps to *Client* and *Product* that maps to *Product*. In detail, the concept of *Product* is the same in both data sources for σ=1. On the other hand, *Client* and *Person* do not refer to the same concept, because the similarity degree σ<1 and, therefore, they present some differences. However, they show some common concepts for σ>0. So, at a conceptual level, we introduce a hierarchy, where *Client* and *Person* are both specialization of a super-class. This hierarchy is, then, removed in the next logical design and this leads to the definition of the logical schema in Figure 9.

5.3 Conceptual design

Phase 3 uses the tree-based multidimensional model and aims at automatically producing the data warehouse conceptual schema. In fact, this phase can be thought as an expert system [21], able to simulate the behavior and the reasoning of the designer.

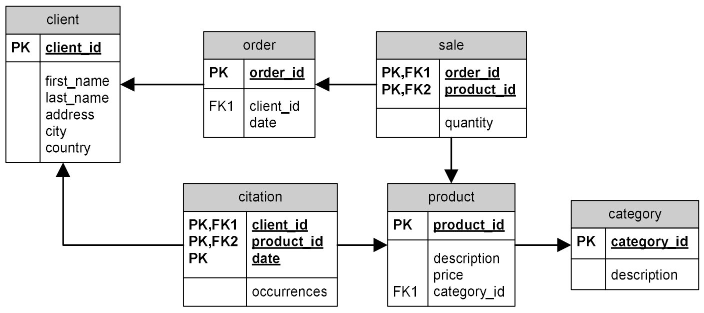


Figure 9. Source integrated schema.

The steps to be performed in this phase are:

1. *identifying* facts present in the integrated logical schema on the basis of the *constraints*;
2. *building* an *attribute tree* for each correctly-identified fact;
3. *remodeling* each attribute tree on the basis of the *constraints*;
4. *validating* the data warehouse conceptual schema, by verifying whether all the remodeled trees agree with the *workload*.

5.3.1. Identifying facts

The main difficulty in this step is to correctly map the multidimensional concepts to the integrated schema. As an example, in a relational schema, the designer must face the problem to identify which relations can be considered as cubes or dimension tables. In our methodology, we initially deal with the correct identification of facts, as these are the starting point to build the initial attribute tree [33].

We consider the facts involved in the constraints coming from the requirement analysis. Given a fact F_1 in a constraint, we choose a candidate fact F_2 in the integrated schema, such that F_2 syntactically corresponds to F_1. For the sake of simplicity, the algorithm for identifying facts considers the cube's name involved in the constraints coming from the requirement analysis and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

scans an integrated schema in order to check if a table of that schema has a name which is a synonym of a fact. If so, that table is marked as a candidate fact table. The rationale is that, for each candidate fact table T, an attribute tree **G** is built starting from T and by navigating through the schema.

5.3.2. Building attribute tree

Assumption 1. Let $R(X_1, \ldots, X_n)$ be a relation, and let **G** = (N, E) be a tree. We assume $X_j \in N$, $\forall j = 1, \ldots, n$. We also assume $(X_i, X_j) \in E$ if X_i is the primary key of R and $i \neq j$. We say that **G** = (N, E) is the *attribute tree* obtained from the relation R, where X_i is the root of **G**. □

On the basis of Assumption 1, the edge (X_i, X_j) indicates that the non trivial functional dependency $X_i \rightarrow X_j$ $(i \neq j)$ holds on R (in this case, established by a primary key constraint). It is worth noting that, for the sake of simplicity, we assume the primary key is composed of only one attribute.

Assumption 2. Let $R(X_1, \ldots, X_n)$ and $S(Y_1, \ldots, Y_m)$ be relations, and let **G** = (N, E) be a tree. We assume $(X_i, Y_j) \in E$ if:

- Y_j is the primary key of the relation S, and
- X_i is a foreign key referencing Y_j. □

On the basis of Assumption 2, an edge can also indicate the presence of a functional dependency established by a foreign key constraint.

Assumption 3. (*Tree Minimization*) Let $R(X_1, \ldots, X_n)$ and $S(Y_1, \ldots, Y_m)$ be relations, and let **G** = (N, E) be a tree. Then, we can use the foreign key constraint *to minimize* the tree G as follows. If

- $X_i \rightarrow X_j$,
- X_j is not (part of) the primary key of R, and
- X_j is a foreign key referencing the primary key Y_s of the relation S,

then the tree **G**' = (N', E') obtained from **G**, where:

- $N' \subseteq N$,
- $(X_i, Y_s) \in E'$,
- $(X_i, X_j) \notin E'$, and
- $(X_j, Y_s) \notin E'$,

is the *minimization* of **G**. □

Assumption 3 reduces the number of nodes, by deleting those redundant. So, when a relation presents a foreign key referencing a primary key, we can keep only that node corresponding to the primary key.

Until now we considered relational schemas composed of one or two relations. Hereafter, we will be concerned with complex schemas composed of several relations. For this end, we stress the point that the topology of the attribute tree depends on the relation taken as the starting point to navigate in the schema.

Assumption 4. Let $R_1(X_{11}, \ldots, X_{1h_1})$, $R_2(X_{21}, \ldots, X_{2h_2})$, …, $R_r(X_{r1}, \ldots, X_{rh_r})$, and $T(Z_1, \ldots, Z_p)$ be r+1 relations. If

- T is the starting relation,
- X_{ij} is the primary key of the relation R_i, $i = 1, \ldots, r$,
- $Z = (Z_{t_1}, \ldots, Z_{t_r})$ is the primary key of the relation T, and
- Z_{t_i} is a foreign key referencing X_{ij}, $i = 1, \ldots, r$,

then the tree **G** = (N, E) is defined so:

- $N = \{T\} \cup \{X_{ij} \in R_i \mid i = 1, \ldots, r\}$,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

- T is the root node of the tree **G**,
- $(X_{ij}, X_{il}) \in E, \forall j \neq l$,
- $(T, X_{ij}) \in E, \forall i = 1, \ldots, r$, and
- $(T, Z_{t_v}) \in E, Z_{t_v} \in T$ and $Z_{t_v} \notin Z$. □

Assumption 4 allows building an attribute tree when a relation representing a many-to-many n-ary relationship is taken as the starting point.

Assumption 5. Let $R_1(X_{11}, \ldots, X_{1h_1})$, $R_2(X_{21}, \ldots, X_{2h_2})$, …, $R_r(X_{r1}, \ldots, X_{rh_r})$, and $T(Z_1, \ldots, Z_p)$ be r+1 relations. If

- R_i is the starting relation, $1 \leq i \leq r$,
- X_{ij} is the primary key of the relation R_i, $i = 1, \ldots, r$,
- $Z = (Z_{t_1}, \ldots, Z_{t_r})$ is the primary key of the relation T, and
- Z_{ti} is a foreign key referencing X_{ij}, $i = 1, \ldots, r$,

then the tree **G** = (N, E) is defined so:

- X_{ij} is the root node of **G**,
- $(X_{ij}, T) \in E$,
- $(T, X_{kj}) \in E$, for $k = t_1, \ldots, t_r$ and $k \neq i$, and
- $(T, Z_v) \in E$, for $v = 1, \ldots, p$ and $v \neq t_1, t_2, \ldots, t_r$. □

Assumption 5 allows building an attribute tree when a relation representing a many-to-many n-ary relationship is encountered while navigating in the schema.

5.3.3. Remodeling attribute tree

In this step, the attribute tree is modified in a supervised way using all the constraints coming from requirements.

We denote with (A, B) :- C the fact that the attribute C can be computed using A and B, that is, C is a derived measure. As an example, (*price*, *quantity*) :- *amount* means that there exists an (algebraic) expression to compute *amount* using *price* and *quantity*.

Let us consider a tree **G** and a constraint coming from the user requirements. A constraint is in the form `<cube>[<dimensions>].[<measures>]`, where

- `<cube>` is the name of the cube
- `<dimensions>` is a list of comma-separated dimensional levels and each dimensions is separated by a semi-colon, and
- `<measures>` is a list of comma-separated measures.

In informal way, we create as many root children as many measures there are in the constraint. Moreover, we add a root child for each first dimensional level in the constraint. In a recursive way, the other dimensional levels are added as children nodes of their own predecessor levels. In general, when we add a (new) node *B* to a parent node *A*, the node *B* is created *ex novo* or it is already present in the tree. In the latter case, the edge between *B* and the old parent of *B* must be deleted and a new edge between *B* and the new parent *A* must be created; this is the so-called *change parent* operation. Furthermore, the function parent(A) returns the parent of node *A*, while root(G) returns the root the tree *G* (*see*, Algorithm 1 for a formal description).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

Algorithm 1. Remodeling attribute tree.

```
Input:
G : graph
C: constraint

Output:
G′ : remodeled G graph

1.  for each measure M in C
2.      if $M \in$ G then
3.          remove $parent(M) \rightarrow M$
4.          add $root(G) \rightarrow M$
5.      else
6.         if (X_1, …, X_n) :- M and $(X_1, …, X_n) \in G$
7.          remove $parent(X_i) \rightarrow X_i$ for $i = 1, …, n$
8.          add $root(G) \rightarrow M$
9.         end if
10.     end if
11. end for
12. for each dimension D in C
13.     for each level $L_j$ in D, for $j = 1, …, m$
14.         if $L_j = L_1$
15.             if $L_j \in G$ then
16.                 remove $parent(L_j) \rightarrow L_j$
17.             end if
18.             add $root(G) \rightarrow L_j$
19.         else
20.             if $parent(L_j) \neq L_{j-1}$
21.                 if $L_j \in$ T then
22.                     remove $parent(L_j) \rightarrow L_j$
23.                 end if
24.                 add $L_{j-1} \rightarrow L_j$
25.             end if
26.         end if
27.     end for
28. end for
29. for each $N \in G$
30.     if $\nexists$ D such that $N \in D$
31.         prune $N$
32.     end if
33. end for
```

5.3.4. Validating conceptual schema

The workload is generated from the tasks of the decision makers, as represented in the strategic rationale model, and is now used in order to perform the validation process against a conceptual schema (*ie*, the remodeled attribute tree). For the sake of simplicity, a workload is a set of queries that produce information to which decision makers are interested. If all the queries of the workload can be effectively executed over the conceptual schema, then such a schema is assumed to be validated and the designer can safely translate it into the corresponding logical schema. Otherwise, the conceptual design process must be revised.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

We define the following issues related to the validation of a conceptual schema in reference to the queries included into the preliminary workload:

- a query involves a cube that has not been defined as such;
- a query requires a measure that is not an attribute of the given cube;
- a query presents an aggregation pattern on levels that are unreachable from the given cube;
- a query requires an aggregation on a field that has not been defined as a dimensional attribute;
- a query requires a selection on a field that has not been defined as a descriptive attribute.

A query is assumed to be validated if there exists at least an attribute tree such that the following conditions hold: (a) the fact is the root of the tree; (b) the measures are the children nodes of the root; (c) for each level in the aggregation pattern, there exists a path from the root to a node X, where X is a non-leaf node representing the level; and (d) for each attribute in the selection clause, there exists a path from the root to a node Y, where Y is a leaf node representing that attribute.

If all the queries are validated, then each attribute tree can be considered as a cube, where the root is the fact, non-leaf nodes are aggregation levels, and leaf nodes are descriptive attributes belonging to a level. So, the conceptual design ends and the designer can transform the conceptual schema into a logical one. On the other hand, if a query cannot be validated, then the designer has to opportunely modify the tree [34]. As an example, if an attribute of the selection clause is not in the tree, then the designer can decide to manually add a further node.

A query of the workload is in the form

```
<cube>[<aggregation_pattern>;<selection_clause>].<measure>
```

where

- `<cube>` is the name of the cube
- `<aggregation_pattern>` is a list of comma-separated dimensional levels used to aggregate data
- `<selection_clause>` is a predicate for selecting data, and
- `<measure>` is the name of the measure.

5.3.5. Validating conceptual schema

From the tasks of the Data Warehouse in Figure 7, we obtain the following constraint, which will be first used to build a preliminary attribute tree and, then, to automatically remodel it:

```
sale[product, category; month, year; city; client].[quantity, occurrences].
```

So, the algorithm first checks for a relation that syntactically agrees with the name of the cube—*sale* in this case. This relation is marked as a candidate fact table and the attribute tree is built with sale as the root and by navigating through the schema. The result is shown in Figure 10.

It is worth noting that this is not a tree formally, since the *client_id* attribute is referenced by different relations and, then, the corresponding node has more than one father.

The same constraint is now used to remodel the attribute tree, by performing a change parent operations and deleting or adding nodes. The rationale is to make numeric attributes, that is measures, root children. The same holds for dimensions. But, while measures are leaf-nodes, a dimension is a sub-tree composed of several dimensional levels, whereas each dimensional level may present zero or more descriptive attributes. The result of the remodeling step is shown in Figure 11.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

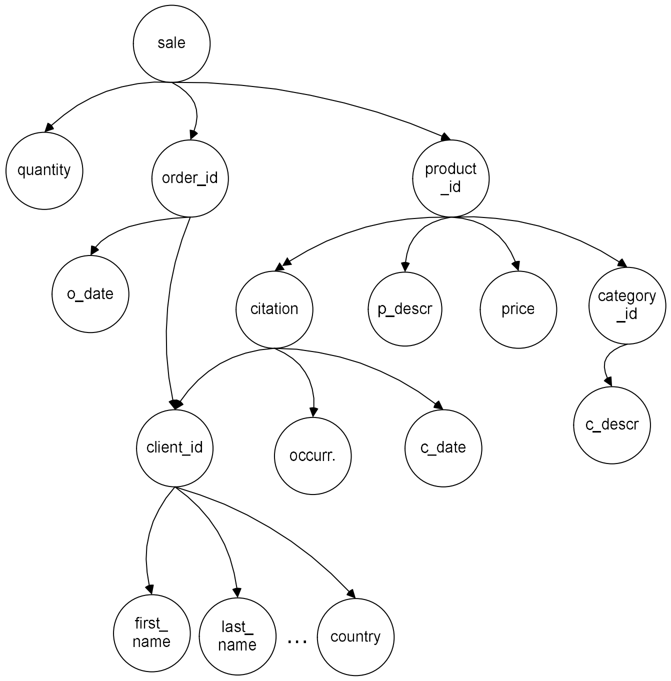


Figure 10. Source attribute tree.

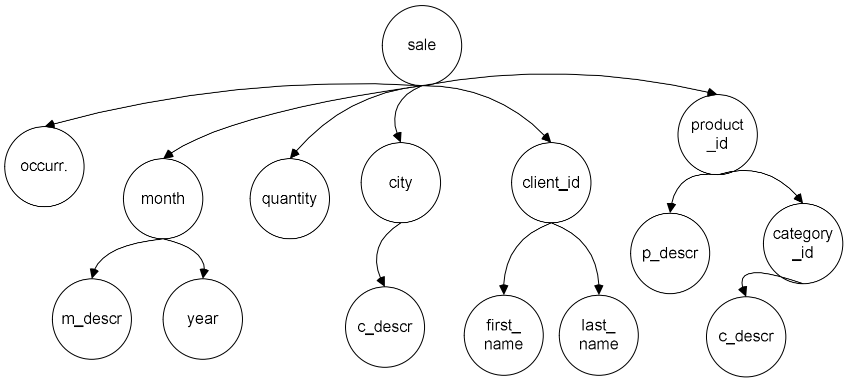


Figure 11. Remodeled attribute tree.

At the end of the remodeling step, the attribute tree representing a cube must be validate against the preliminary workload, in order to check if it can support the execution of all the queries coming from the tasks of the decision makers. If the validation has success, it is possible to safely proceed to the translation of the conceptual schema into a multidimensional document.

As an example, we consider the following query of the preliminary workload:

```
sale[product; year='2016'].occurrences.
```

This can be executed against the remodeled tree for: (i) *sale* is the name of the cube; (ii) the aggregation involves the *product* node which is a dimensional level reachable from the root; (iii) the selection clause involves the *year* node that acts also as a descriptive attribute; (iv) *occurrences* is the child node of the root. Assuming that all the remaining queries can be executed against the tree, then the conceptual schema is validated

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

5.4 Logical design

This phase aims at transforming an attribute tree into a multidimensional document, on the basis of the key-value model, whereas we can associate a data structure to the value. To this end, we use the following translation rules.

Rule 1. For each attribute tree, a multidimensional document must be created. □

This trivial rule is used to define which documents must be created. We assume that the document's name is the root node.

Rule 2. Root children that are leaf-nodes are considered as *measures* and, then, can be tagged as numeric attributes in the value. □

In the value, the measures are numeric attributes represented by leaf nodes linked to the root.

Rule 3. Root children that are non-leaf nodes, except measures, form the key of the multidimensional document. □

Since the number of sub-trees is the dimensionality of the cube, we have to consider the set of root children that are not leaf-nodes nor measures in order to define the key of the document.

Rule 4. For each sub-tree representing a dimension, a nested document must be created. The nested document contains all the attributes of the dimension. □

A sub-tree is the set of nodes having a path to the first dimensional level. These form a dimension that must be included as a nested document reporting all the dimensional and the descriptive attributes. If we create a nested document in a flat way, this corresponds to a star-schema. If we want to create a document equivalent to a snow-flake schema, we have to create nested documents in a recursive way.

The rationale in the logical design is first to create a multidimensional document for each attribute tree. Here, the key is composed by the first dimensional level of each sub-tree—*product_id*, *client_id*, *month*, *city* in this case. The value is a structure composed of measures and a set of nested objects, one for each dimension, which store dimensional levels along their descriptive attributes. The multidimensional document corresponding to the attribute tree in Figure 11 is shown in Table 6. The benefit of adopting the key-value model is a de-normalized dataset, which does not require join operations at all. For example, we may add to the value also an array containing the telephone numbers of the suppliers.

Table 6. Sale multidimensional document.

Sale	
key	**value**
`product_id`	`quantity, occurrences`
`client_id`	`{month, month.description, year}`
`month`	`{city, city.description}`
`city`	`{client_id, first_name, last_name}`
	`{product_id, product.description, category_id, category.description}`

5.4.1 Incremental step

When new requirements emerge, then we have to quickly modify the existing data warehouse by applying agile approaches, in order to release as soon as possible a new version of the data warehouse without performing a complete design process. To this end, we observe that new requirements usually involve the creation of new facts or the addition of measures, dimensions, or hierarchical levels to existing facts. The deletion of elements is the most unusual case, since data warehouses are used to preserve historical data. So, we defined an algorithm that detects if a new

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

constraint requires the creation of a new fact or the modification of an existing one. In case of creation of new facts, the algorithm proceeds with the basic steps of the conceptual design: that is, the identification and the mapping of this concept against the integrated schema and, then, the creation, the remodeling, and the validation of the corresponding attribute tree, which is, at last, transformed into a multidimensional document. Since there is no overlapping among documents, each new fact is independent from the others. This is different from stars/snow-flake schemas, where we may observe shared dimensions and, then, two dimensions may not agree. Moreover, since the steps are completely automatic, this does not represent a drawback for our methodology. In case of modification, we can skip the conceptual design and modify the multidimensional document, using further requirements and the source attribute tree (*note*, not the remodeled one).

In detail, let **G** be an attribute tree, let C be a constraint, and let D be a multidimensional document. In case of addition of an element M of C, we apply the following rules.

Rule 5. If M is an attribute and there exists a non-leaf node *P* in **G** such that (i) M corresponds to *P*, (ii) *P* is not a measure, and (iii) *P* is the first hierarchical level in **G**, then a new element in the key of D is added and a nested document is created. □

This rule allows adding a dimension to an existing document, when a fact requires a further coordinate of analysis, provided that the corresponding attribute exists in the data source.

Rule 6. If M is a measure and there exists a numeric attribute *N* in **G** such that *M* corresponds to *N* or M can be derived from the attribute(s) of **G**, then a new element in the value of D is added. □

This rule allows adding a measure to an existing document, when a fact requires a further numeric attribute, provided that the corresponding attribute exists in the data source. This new measure can be also a derived one. For example, assuming that attributes *quantity* and *price* are present in **G**, we may add *amount* as a part of the value, since it can be derived as *quantity* × *price*.

Rule 7. If M is an attribute and there exists a node *P* in **G** such that (i) M corresponds to *P*, (ii) *P* is not a measure, and (iii) M is an attribute of a dimension K in C, then a further attribute is added to the nested document of K in D. □

This rule allows adding a hierarchical level to an existing document, when a fact requires a further attribute for aggregation purpose, provided that the corresponding attribute exists in the data source.

As an example, we report below a requirement, emerged after the end of the design process and the data warehouse implementation. This new business goal needs a further level of analysis when aggregating sales. This new dimensional level is *country*.

```
sale[product, category; month, year; city, country; client].[quantity, occurrences].
```

In our methodology, first we detect that a multidimensional document *Sale* already exists for the given fact. In reference to the previous requirement, now the *city* dimension must be equipped with the *country* dimensional level. So, we can safely apply **Rule 7**, for: (i) the attribute *country* is present in the source attribute tree, (ii) *country* is not a measure, and (iii) *country* in C must belong to *city* dimension. Consequently, we modify the *sale* multidimensional document by adding an attribute to the nested document of *city*. The new multidimensional document for the *sale* fact is shown in Table 7.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

Table 7. New Sale multidimensional document.

Sale	
key	**value**
`product_id`	`quantity, occurrences`
`client_id`	`{month, month.description, year}`
`month`	`{city, city.description, state}`
`city`	`{client_id, first_name, last_name}`
	`{product_id, product.description, category_id, category.description}`

5.3 Conceptual design

In this phase, the logical schema is implemented by creating a web-service that manages a set of metadata. These can be accessed in order to browse the multidimensional elements, such as measures and dimensions, and to formulate queries. To this aim, the web-service also exposes an interface by means of which a client can interact with a query engine using a query language. Of course, the query language must agree with the adopted logical model. For example, if the logical model is the relational one, then the query language is SQL, while if it is a key-value model, then queries must be expressed using *put* and *get* primitives. Finally, the engine is devoted to translate a query into an access plan, in order to redirect appropriate queries on involved data sources, whose locations and schemas are also registered at the web-service.

When the logical schema is ready, we have two choices: (1) creating a physical data warehouse using a Hadoop-based solution or a NoSQL database, for example Oracle NoSQL, which allows to associate a JSON [35] schema to a value in order to create the data structure and provides simple primitives to store (put), read (get), and delete an attribute of value on the basis of a key [26]; or, (2) creating a virtual data warehouse, as suggested in our architecture. This means defining the necessary metadata.

According to Mondrian's representation, part of the metadata describing the multidimensional document in Table 6 is shown below. The metadata describe a dimension named *Time*, having a hierarchy *Monthly Time*, with two levels: *Month*, and *Year*, each with its own properties.

```
<Dimension name="Time">
    <Hierarchy name="Monthly Time">
        <Level name="Year">
            <Property name="year"/>
        </Level>
        <Level name="Month">
            <Property name="month"/>
            <Property name="description"/>
        </Level>
    </Hierarchy>
</Dimension>
```

If we adopt the first solution, then *Count the product references in Web* (*cfr*, Figure 7) requires first the execution of the ETL phase in order to get the most recent data. This phase takes a time t_1 and the physical data warehouse increases of a space s_1. So, t_1 and s_1 are the costs associated to this solution. Furthermore, it is worth noting that the ETL phase is usually a batch process that may be daily, weekly, or even monthly executed.

If we adopt the second solution, then data are immediately available and we may know the most popular products even every hour, without waiting for the refresh and saving space, for $t_1=0$ and $s_1=0$. As a counterpart, answering times now may be higher, since this solution requires an access to the data sources and an on-line aggregation, while in the physical data warehouse data are always pre-aggregated.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

6. Architecture

In literature, the term Big Data Warehouse (BDW) is usually referred to a data warehouse, built over non-relational databases, giving rise to a hybrid architecture that integrates a traditional data warehouse with technological solutions for managing big data, usually Hadoop [8, 36]. However, there is no standard representation and several scenarios are possible.

The first solution is the use of Hadoop as a staging area, giving the possibility to create a BDW that manages and stores any type of data and also to address scalability problems. From there, using ETL tools, data are moved into a traditional data warehouse for analytical purposes.

In the second solution, Hadoop is primarily used to store big data and, then, to build an autonomous BDW. Conversely, a data warehouse is used to store traditional data sources. Then, both of these are used conjunctly by software tools for analytical processing.

In both the contexts, examples of BDW built over Hadoop are Apache Hive [37] and Apache Tajo [38], which provides a SQL-like interface to manage also unstructured data. In [9], a simplified multidimensional model is adopted to support OLAP operations over Hadoop, obtaining improved performance, regardless of the size of data sets and number of involved dimensions, on the basis of chunk partition and selection methods.

However, at the present, only few works address the issues related to the evaluation of methodological aspects [39] and the impact of big data against the design process. An interesting proposal is the MAD design paradigm [40], which states that data warehouses should be (i) *Magnetic*, for they must attract all the data sources available in an organization; (ii) *Agile*, for they should support continuous and rapid evolution; and (iii) *Deep*, for they must support analyses more sophisticated that traditional OLAP functions. Accordingly, in [41] the authors propose an integration of data warehouse with semantic web technologies to perform an innovative Exploratory OLAP. Here, the data warehouse is integrated at the run time with further data coming from the Web. To this aim, semantic web technologies are first used to discover new data sources and, then, to obtain the underlying data model. The proposed process is called ETQ (Extract, Transform, and Query), which is used in conjunction with the traditional ETL process.

The use of semantic web technologies to provide knowledge about data sources is also proposed in [42], where the authors present a methodology for (a) extracting all the information available in the Web by means of crawlers, and (b) storing all that raw data in the document-oriented database MongoDB. Then, these data are transformed and loaded into a secondary relational database, which integrates a traditional data warehouse, in order to perform several types of analysis, such as sentiment analysis and business intelligence.

5. Conclusions

The common strategy for managing big data in data warehousing is the use of innovative solutions and technologies. The most used technology is the Hadoop framework that provides a file system for physically storing data as key-value pairs in a distributed architecture. This solution is sometimes used together with a traditional architecture.

Our contribution is a different definition of big data warehouse, which is completely virtual for avoiding the movement of huge quantity of data and is designed according to the GRAHAM methodology, based on the key-value logical model at a logical level for NoSQL systems.

Our methodology is hybrid in order to consider both the benefits of traditional approaches, which can be data or requirement driven. Furthermore, it is completely automatic. This is opportune for new data sources are discovered every day and we need to integrate them on the fly by avoiding repetitive tasks. Nonetheless, an agile approach is also considered to quickly process further requirements not emerged in preliminary steps and to address the frequent changes in user requirements. Indeed, these new requirements can affect the current data warehouses that may become obsolete. The use of an ontological approach supports the designer in the integration process, when syntactical and semantic inconsistencies are to be solved, even in presence of unstructured data. Given these aims, our contribution is a methodology that makes data warehouse's evolution fast and coherent with business goals, without delays in analytics phases,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

since data are immediately available just at the end of the design process, thanks to the adoption of a virtual data warehouse that does not require a feeding process.

References

1. Kimball, R.; Ross, M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed.; Wiley, 2013; ISBN: 978-1-118-53080-1. Available online: http://www.essai.rnu.tn/Ebook/Informatique/The Data Warehouse Toolkit, 3rd Edition.pdf (accessed on 08 June 2018).
2. Romero, O.; Abelló, A. A survey of multidimensional modeling methodologies. *International Journal of Data Warehousing and Mining* **2009**, *5*(2), 1-23, DOI: 10.4018/jdwm.2009040101.
3. Madden, S. From databases to big data. *IEEE Internet Computing* **2012**, *16*(3), 4-6, DOI: 10.1109/MIC.2012.50.
4. Chen, M.; Mao, S.; Liu, Y. Big data: a survey. *Mobile Netw Appl* **2014**, *19*(2), 171–209, DOI: 10.1007/s11036-013-0489-0.
5. Russom, P. *Big Data Analytics*; TDWI best practices report, fourth quarter, 2011. Available online: http://www.cloudtalk.it/wp-content/uploads/2012/03/1_17959_TDWIBigDataAnalytics.pdf (accessed on 08 June 2018).
6. Pang, B.; Lee, L. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval* **2008**, *2*(1-2), 1-135, DOI: 10.1561/1500000011. (Now Publishers Inc, 2008, ISBN: 9781601981509).
7. Labrinidis, A.; Jagadish, H.V. Challenges and opportunities with big data. *Proc. of the VLDB Endow.* **2012**, *5*(12), 2032-2033, DOI: 10.14778/2367502.2367572.
8. Mohanty, S.; Jagadeesh, M.; Srivatsa, H. *Big Data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics*. Apress, 2013; ISBN 978-1-4302-4872-9.
9. Song, J.; Guo, C.; Wang, Z.; Zhang, Y.; Yu. G.; Pierson, J.-M. HaoLap: A Hadoop based OLAP system for big data. *Journal of Systems and Software* **2015**, *102*, 167-181, DOI: 10.1016/j.jss.2014.09.024; author-deposited version in: http://oatao.univ-toulouse.fr/13232/1/song_13232.pdf (accessed on 08 June 2018).
10. Kimball, R.; Caserta, J. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley, 2004; ISBN: 978-0-764-56757-5.
11. Golfarelli, M.; Rizzi, S. *Data Warehouse Design: Modern Principles and Methodologies*, McGraw-Hill Osborne, 2009; ISBN: 9780071610407.
12. Mazón, J.N.; Trujillo, J.; Serrano, M.A.; Piattini, M. Designing data warehouses: from business requirement analysis to multidimensional modeling. In REBNITA 2005, Proc. of the 1st Workshop on Requirements Engineering for Business Need and IT Alignment, 2005, pp. 44-53; ISBN: 0 7334 2276 4 University of New South Wales Press.
13. Tiwari, S. *Professional NoSQL*. Wiley, 2011; ISBN: 978-1-118-16780-9.
14. Di Tria, F.; Lefons, E.; Tangorra, F. Academic data warehouse design using a hybrid methodology. *Comput. Sci. Inf. Syst.* **2015**, *12*(1), 135-160, DOI: 10.2298/CSIS140325087D.
15. Di Tria, F.; Lefons, E.; Tangorra, F. Hybrid methodology for data warehouse conceptual design by UML schemas. *Information and Software Technology* **2012**, *54*(4), 360-379, DOI: 10.1016/j.infsof.2011.11.004.
16. Di Tria, F.; Lefons, E.; Tangorra, F. GrHyMM: a Graph-oriented Hybrid Multidimensional Model. In *Advances in Conceptual Modeling: Recent Developments and New Directions*; De Troyer, O., Bauzer, M.C, Billen, R., Hallot, P., Simitsis, A., VanMingroot, H., Eds, Lecture Notes in Computer Science 6999, Springer, 2011, pp. 86-97; ISBN: 978-3-642-24573-2, DOI: 10.1007/978-3-642-24574-9_12.
17. Mazón, J.N.; Trujillo, J. A hybrid model driven development framework for the multidimensional modeling of data warehouses. *SIGMOD Record* **2009**, *38*(2), 12-17, DOI: 10.1145/1815918.1815920.
18. Giorgini, P.; Rizzi, S.; Garzetti, M. GRAnD: a Goal-oriented approach to Requirement Analysis in Data warehouses. *Decision Support Systems* **2008**, *45*(1), 4-21, DOI: 10.1016/j.dss.2006.12.001.
19. Romero, O.; Abelló, A. Automatic validation of requirements to support multidimensional design. *Data & Knowledge Engineering* **2010**, *69*(9), 917-942, DOI: 10.1016/j.datak.2010.03.006.
20. Phipps, C.; Davis, K.C. Automating data warehouse conceptual schema design and evaluation. *Design and Management of Data Warehouses* **2002**, *58*, 23-32, DOI: 10.4018/9781605660103.ch019. Available online: http://ceur-ws.org/Vol-58/phipps-davis.pdf (accessed on 08 June 2018).
21. Di Tria, F.; Lefons, E.; Tangorra, F. Big data warehouse automatic design methodology. In *Big Data Management, Technologies, and Applications*; Hu, W., Kaabouch, N., Eds, IGI Global, Hershey, PA, 2014, pp. 115-149, ISBN: 978-1-4666-4699-5, DOI: 10.4018/978-1-4666-4699-5.ch006.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

22. Collier, K.; Highsmith, J.A. Agile Data Warehousing: Incorporating Agile Principles. Business Intelligence Advisory Service: Executive Report, 2004, 4(12). Available online: https://www.cutter.com/sites/default/files/bia/fulltext/reports/2004/12/Agile%20Data%20Warehousing%20Incorporating%20Agile%20Principles.pdf (accessed on 08 June 2018).
23. Di Tria, F.; Lefons, E.; Tangorra, F. Design process for big data warehouses. In International Conf. on Data Science and Advanced Analytics (DSAA), 2014, pp. 512-518. Electronic ISBN: 978-1-4799-6991-3, DOI: 10.1109/DSAA.2014.7058120.
24. Cormode, G.; Garofalakis, M.; Haas, P.J.; Jermaine, C. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases* **2011**, *4*(1-3), 1-294, DOI: 10.1561/1900000004; Now Publ., 2012, ISBN: 9781601985163. Available online: http://db.cs.berkeley.edu/cs286/papers/synopses-fntdb2012.pdf (accessed on 08 June 2018).
25. Hyde, J. *How to Design a Mondrian Schema*. Resource document. Mondrian Documentation, 2011. Available online: http://mondrian.pentaho.com/documentation/schema.php (accessed on 08 June 2018).
26. Joshi, A.; Haradhvala, S.; Lamb, C. Oracle NoSQL database – scalable, transactional key-value store. In Proc. of the 2nd International Conf. on Advances in Information Mining and Management, 2012, pp. 75-78, ISBN: 978-1-61208-227-1. Available online: http://www.chinacloud.cn/upload/2012-10/12102712502846.pdf (accessed on 08 June 2018).
27. Di Tria, F.; Lefons, E.; Tangorra, F. Ontological approach to data warehouse source integration. In *Information Sciences and Systems*; Gelenbe, E., Lent, R., Eds, Lecture Notes in Electrical Engineering 264, Springer, 2013, pp. 251-259; ISBN: 978-3-319-01603-0, DOI: 10.1007/978-3-319-01604-7_25.
28. Bakhtouchi, A.; Bellatreche, L.; Ait-Ameur, Y. Ontologies and functional dependencies for data integration and reconciliation. In *Advances in Conceptual Modeling: Recent Developments and New Directions*; De Troyer, O., Bauzer, M.C, Billen, R., Hallot, P., Simitsis, A., VanMingroot, H., Eds, Lecture Notes in Computer Science 6999, Springer, 2011, pp. 98-107; ISBN: 978-3-642-24573-2, DOI: 10.1007/978-3-642-24574-9_13.
29. Hakimpour, F.; Geppert, A. Global schema generation using formal ontologies. In *Conceptual Modeling*; Spaccapietra, S., March, S.T., Kambayashi, Y., Eds, Lecture Notes in Computer Science 2503, Springer, 2002, pp. 307-321; ISBN: 3-540-44277-4 DOI: 10.1007/3-540-45816-6_31.
30. Thenmozhi, M.; Vivekanandan, K. An ontology-based hybrid approach to derive multidimensional schema for data warehouse. *International Journal of Computer Applications* **2012**, *54*(8), 36-42, DOI: 10.5120/8590-2343. Available online: https://research.ijcaonline.org/volume54/number8/pxc3882343.pdf.
31. Ferilli, S.; Basile, T.M.A.; Biba, M.; Di Mauro, N.; Esposito, F. A general similarity framework for Horn clause logic. *Fundamenta Informaticae* **2009**, *90*(1-2), 43-66, DOI: 10.3233/FI-2009-0004.
32. Elmasri, R.; Navathe, S.B. *Fundamentals of Database Systems*, 6th ed.; Addison-Wesley, 2011, ISBN: 978-0-136-08620-8.
33. Carmè, A.; Mazón, J.N.; Rizzi, S. A model-driven heuristic approach for detecting multidimensional facts in relational data sources. In *Data Warehousing and Knowledge Discovery*; Pedersen T.B., Mohania, M.K., Tjoa, A.M., Eds, Lecture Notes in Computer Science 6263, Springer, 2010, pp. 13-24, ISBN: 978-3-642-15104-0, DOI: 10.1007/978-3-642-15105-7_2.
34. dell'Aquila, C.; Di Tria, F.; Lefons, E.; Tangorra, F. Logic programming for data warehouse conceptual schema validation. In *Data Warehousing and Knowledge Discovery*; Pedersen T.B., Mohania, M.K., Tjoa, A.M., Eds, Lecture Notes in Computer Science 6263, Springer, 2010, pp. 1-12, ISBN: 978-3-642-15104-0, DOI: 10.1007/978-3-642-15105-7_1.
35. Crockford, D. The application/json Media Type for JavaScript Object Notation (JSON). Network Working Group. Request for Comments: 4627, 2006; available online: http://www.ietf.org/rfc/rfc4627.txt.
36. Krishnan, K. *Data Warehousing in the Age of Big Data*, 1st ed.; Morgan Kaufmann, 2013, ISBN: 978-0-12-405891-0, DOI: 10.1016/C2012-0-02737-8.
37. Thusoo, A.; Sarma, J.S.; Jain, N.; Shao, Z.; Chakka, P.; Anthony, S.; Liu, H.; Wyckoff, P.; Murthy, R. Hive: a warehousing solution over a map-reduce framework. *Proc. of the VLDB Endow.* **2009**, *2*(2), 1626-1629, DOI: 10.14778/1687553.1687609. Available online: http://www.vldb.org/pvldb/2/vldb09-938.pdf (accessed on 08 June 2018).
38. Sakr, S. Big Data Processing Stacks. *IT Professional* **2017**, *19*(1), 34-41, DOI: 10.1109/MITP.2017.6.
39. Di Tria, F.; Lefons, E.; Tangorra, F. Cost-benefit analysis of data warehouse design methodologies. *Information Systems* **2017**, *63*, 47-62, DOI: 10.1016/j.is.2016.06.006.
40. Cohen, J.; Dolan, B.; Dunlap, M.; Hellerstein, J.M.; Welton, C. MAD skills: new analysis practices for big data. In *Proc. of the VLDB Endow*. **2009**, *2*(2), 1481-1492, DOI: 10.14778/1687553.1687576.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 June 2018 doi:10.20944/preprints201806.0219.v1

41. Abelló, A.; Romero, O.; Pedersen, T.B.; Berlanga, R.; Nebot, V.; Aramburu, M.J.; Simitsis, A. Using semantic web technologies for exploratory OLAP: a survey. *IEEE Transactions on Knowledge and Data Engineering* **2015**, 27(2), 571-588, DOI: 10.1109/TKDE.2014.2330822.
42. Ramos, C.M.Q.; Correia, M.B.; Rodrigues, J.M.F.; Martins, D.; Serra, F. Big data warehouse framework for smart revenue management. In *Advances in Environmental Science and Energy Planning*; Mastorakis, N.E., Corbi, I., Eds; Proc. of the 3rd MATREFC '15, 2015, pp. 13-22., ISBN: 978-1-61804-280-4. Available online: http://hdl.handle.net/10400.1/6906 (accessed on 08 June 2018).