Using Ontologies and Requirements for Constructing and Optimizing Data Warehouses

L. Bellatreche*, S. Khouri*, I. Boukhari* and R. Bouchakri*
* LIAS/ENSMA Poitiers University Futuroscope, France
bellatreche@ensma.fr, (selma.khouri, ilyes.boukhari, rima.bouchakri)@ lisi.ensma.fr

Abstract - Developing database (DB) and data warehouse (DW) applications pass through three main phases imposed by the ANSI/SPARC architecture: conceptual modeling, logical modeling and physical modeling. Some research efforts add a new ontological level above the conceptual one. This architecture has created two main actors whose presence is mandatory to ensure the success of applications: "conceptual designer" for conceptual and logical levels and "database administrators" (DBA) for physical level. Note that some administration tasks need some inputs from conceptual phase. Unfortunately, interaction between these two actors is negligible. Recently, some research and industrial efforts identify a highest cost of DBA and propose tools (advisors) to replace them, in order to ensure what we call zero-administration. The main limitation of these tools is their robustness. In this paper, we propose a new human resource management for database applications. Instead of replacing DBA, we claim to delegate some DBA tasks to conceptual designers. These tasks are usually those having inputs user requirements that may be translated to SQL queries. First, we propose a user make user requirements persistent into DWs. An analysis of requirements is given to identify SQL queries that may be used for physical design phase. Finally, a selection of indexes based on user requirements is presented and evaluated using star schema benchmark.

I. INTRODUCTION

DBs and DWs usually follow the ANSI/SPARC architecture, where three different levels of schemas exist: (i) external schemas describing the different external views of the data, (ii) the conceptual schema describe all data items and relationships between them and (iii) the internal schema contain definitions of the stored records, access methods and indexes, etc. This architecture creates two types of actors: conceptual designers and administrators. The first actors generally act based on user requirements. The analysis of these requirements allows designers to generate conceptual models including entities, attributes and cardinalities. Another aspect of this analysis is the identification of treatments needed for the target application. By analyzing these treatments, we can identify three main categories: (i) treatments that can be easily answered by SQL queries, (ii) treatments that may be answered by programming languages and (iii) treatments that may be partially answered by SQL queries. Database administrators (DBA) otherwise, need to select relevant optimization structures (e.g., partitioning, indexing, materialized views, etc.) during the physical design. This selection is usually based on the most frequently asked queries that can be identified from the conceptual modeling phase. Recently, we are moving towards truly zero-administration of database systems in which there is not the possibility for a DBA to check the recommendations of physical design tools before applying them. These tools are developed to reduce the cost of administrators. Developing such tools may suffer from robustness of the proposing recommendations. Reference [4] shows the risk of using such tools. Our vision is that, instead of replacing the task of administrators, we propose to delegate some administration tasks (such as selection of optimization structures) to conceptual designers. To do so, user requirements analysis needs to be well established.

On the other hand, a new extension of ANSI/SPARC architecture has been proposed introducing an ontological layer. Ontologies are actually seen as conceptual models defining a whole domain via a logical theory. We propose in this paper to revise this extended architecture by giving designers a more important role and by making user requirements persistent in the BD/DW structure. We focus in this paper on DW design as a case study. The paper is structured as follows. Section 2 presents the related work. Section 3 presents the background needed. Section 4 presents our methodology for designing DWs following the revisited version of ANSI/SPARC. Section 5 validates our proposal by performing new type of optimization: requirements-driven optimization. Section 6 concludes the paper.

II. RELATED WORK

Requirements engineering (RE) plays a crucial role in information systems development to reduce the risk of failure. Requirements for a DW system determine its functional behavior and all needed enterprise information. Requirements analysis in DW design literature can differ according to the object analyzed. We distinguish: Process driven analysis that analyzes requirements by identifying business processes of the organization, User driven analysis that identifies requirements of each target user and unifies them in a global model, and Goal driven analysis [8] that identifies goals and objectives that guide decisions of the organization at different levels. It is recognized by different authors that goal-driven analysis provides a suitable definition of user requirements especially for decisional applications. These requirements are either functional or non functional. Non-functional requirements are defined as constraints of the system (such as security, performance, flexibility, etc).

In addition to requirements, DW systems must integrate the set of available sources of the organization. Most of current DW design methods follow a mixed
design defining the DW model from both data sources and user requirements. As for traditional information system, these methods design DW systems following ANSI/SPARC specification (Fig. 1). This vision assumes that the conceptual model does not exist and have to be defined. With the emergence of domain ontologies, many DB and DW design methods used existing ontologies models as a first level of specification. Reference [12] proposed a method similar to Chen methodology for designing DBs by adding a semantic layer using linguistic ontologies. Reference [6] proposed an extension of ANSI/X3/SPARC architecture, illustrated in figure Fig.2.(a), giving ontologies their place during design of DB/DW. The conceptual model in this architecture is just a view of an ontological model. Ontologies have been used in DW methods for providing an integrated and unified view of data sources [11], and also for identifying, formalizing, and reasoning on user requirements as we proposed in [9, 10].

In the first version of ANSI/X3/SPARC, and also in the revisited version with ontologies, two main actors are present. Requirements specification and conceptual design step are supervised by designers. Logical design step generally relies on defined translation rules that are supported by many case tools (like Power AMC of Sybase or Rational Rose). Physical design step is supervised by the DBA. As for the logical design step, physical step is being automated by administration tools like Oracle SQL Tuning Advisor or SQL Server Advisor. These tools are provided to assist the DBA and sometimes to replace it. In this case, robustness of administration tools must be proven [4]. In this situation, the role of designers is strengthened while the role DBA is weakened. Designers rely for their work only on user requirements. They represent the most important information for the final system. For the case of DW systems for example: (i) requirements are saved a posteriori as queries in most DBMS (like Oracle or SQL Server) for optimization purposes, (ii) user’s preferences, which are considered as non-functional requirements, are exploited for personalization and recommendation processes like in [7], (iii) quality of the DW can be measured according to its capacity to fulfill the different requirements and goals like in [13]. Unfortunately no trace of requirements is currently saved in DBs or DWs systems once developed.

To solve this issue, we propose a new version of ANSI/SPARC architecture (see Fig. 2.(b)) that: (i) gives the designer a bigger role who is present all along design phases and (ii) makes requirements persistent within the BD/DW structure. In this architecture, we try to reduce the gap between logical and physical design phase to help designers in their design tasks. We take as an example of physical design task, the ‘index selection problem’. We assume that this task can be performed by designers during logical phase using user requirements. Usually, DB or DW indexes are defined relying on a set of frequent queries and a set of constraints (space storage, update cost, etc). Those queries are obtained only once the DB/DW is defined and used during a certain period. During this warm stage, where no statistical information is available, requirements can be used in order to define some relevant optimization structures.

### III. BACKGROUND

We review in this section, relevant concepts to understand our methodology which are: Ontologies and Ontologies for Databases.

#### A. Ontologies: Formal model

Ontologies have been introduced in DW design field first to provide an integrated view of data sources. It is recognized that an effective approach to deal with integration and interoperability is the use of common standards. Many integration applications used domain ontologies as a reference model. We proposed and analyzed in [1] different integration scenarios where a domain ontology integrates a set of semantic sources. We assume in our method this integrating ontology existing as a reference model. We proposed and defined a core ontology model that is common to all ontology formalisms [1]:

\[
\text{Ontology} = \langle C, P, \text{Sub}, \text{OIL}, \text{DAML and OWL} \rangle
\]

where:

- \( C \): is the set of classes describing the concepts of a given domain.
- \( P \): is the set of properties describing the instances of \( C \).
- \( \text{Sub} \): \( C \rightarrow 2^C \) is the subsumption function, which associates each class \( C_i \) to its direct subsumed classes. Two subsumption relationships are introduced in our framework: (i) \( \text{OOSub} \): describing the usual subsumption of inheritance relationship, where the whole set of applicable properties is inherited. (ii) \( \text{OntoSub} \): describing a subsumption relationship without inheritance, where only a part of applicable properties may be

![Figure 1. ANSI/SPARC architecture for DBs/DWs](image-url)
OntoSub can be seen as the operator of ontological modularity.

- **Applic:** \( C \rightarrow 2^P \) is a function that associates to each ontology class, the properties that are applicable for each instance of this class.

### B. Ontologies for databases

The amount of data described by ontologies can be huge, especially since the emergence of ontologies in various domains such as E-commerce or Semantic Web. As a consequence a scalable storage system is required for ontologies and its associated instances. We call Ontology Based Data Base (OBDB) a database that stores data together with the ontologies defining their semantics. Many OBDBs have been proposed in the literature following different architectures. We proposed in [3] a new architecture of OBDBs called OntoDB, where a new part “the meta-schema” is introduced. OntoDB architecture is composed of 4 parts as illustrated in Fig. 3. Parts 1 and 2 are traditional parts available in all DBs, namely the meta-base part that contains the system catalog and the data part that contains instance data. Part 3 “ontology” allows representing ontologies in the database. The part 4 “meta-schema” is specific to OntoDB and records the ontology model into a reflexive meta-model. The presence of the meta-schema part offers flexibility of the ontology part, since it is represented as an instance of the meta-schema. We use the meta-schema in this paper, in order to store user requirements. Specific languages have been designed to query ontologies and their instances such as SPARQL or RdQL. The OntoQL language has been proposed in OntoDB project.

### IV. PROPOSED METHODOLOGY

We present our methodology according to the steps described in the new version of ANSI/SPARC architecture (Fig 2.b). We start by defining the requirements model proposed and its connection to the ontology model. The conceptual model defined is then translated into a relational logical model. We then present the final ontology-based data warehouse making requirements persistent. The example we follow all along the method is based on SSB Star Schema Benchmark (ftp://www.cs.umb.edu/~poneil/StarSchemaB.PDF) benchmark specification for DWs. The conceptual model of this benchmark is considered as our domain ontology. This conceptual model is obtained from a normalized version of SSB relational model by a reverse engineering process. Fig. 5.(C) presents the ontology model in a UML class diagram. We consider business questions provided in the specification of SSB as user requirements.

#### A. Ontology and user requirements models

The requirements model we propose follows a goal-driven approach that has been used in DW context. A goal is an objective that the system under consideration should achieve. After analyzing works of goal-oriented literature, we propose our Goal-oriented model following the GQM definition (Goal/Question/Metric): “A goal is defined for an object, for a variety of reasons, with respect to various
models of quality, from various points of view, relative to a particular environment”.

Fig. 4.(b) presents the goal model we defined, as an UML class diagram. This model is composed of a main class Goal-Model that stores all Goals. Let us consider the first requirement of SSB specification Goal_1: This requirement measures the revenue increase from eliminating various ranges of discounts in given product order quantity intervals shipped in a given year. The revenue is equal to (extendedprice*discount). This goal is described by the following properties: goal identifier (Id), name (Goal_1), description, purpose (measure) and a priority (mandatory or optional goal). Three reflexive relationships between goals are distinguished: (i) Influence relationships (positive or negative), (ii) ParentChild relationships decomposing a general goal into sub-goals and (iii) GoalRelation representing relations of conflicts, refinement, containment and equivalence between goals. A goal is characterized by three classes related to Goal class by aggregation relationships, that we call coordinates of the goal: a Metric measuring the satisfaction of a goal (extendedprice*discount), a Result to analyze (Revenue increase), and Parameters influencing the goal (Discount, Quantity, Year). Each goal involves one or more actors interacting with the system to fulfill the goal (eg. Sales manager for Goal_1).

Users’ goals are expressed at the ontological level where we defined a mapping between coordinates of each goal (Metric, Result and Parameter) and the properties of the domain ontology. Designer picks from the ontology the most relevant properties to define coordinates of each goal. Those goals will be used to define the conceptual view of the DW. Fig. 4.(a) presents a fragment of the ontology meta-model connected to our goal model. Concretely speaking, meta-classes like class Goal has been created in the ontology meta-model. Coordinates of each goal are defined as object-properties which domain is the meta-class Goal, and which range is the meta-class Property.

B. Conceptual view

A conceptual view of the DW (that we call DW ontology (DWO)) is defined by extracting concepts and properties of the ontology used to express user goals. Three scenarios are possible: (1) DWO = DO: the DO corresponds exactly to users’ goals, (2) DWO ⊆ DO: the DWO is extracted from the DO using OntoSub operator and covers all requirements, (3) DWO ⊊ DO: the DO does not fulfill the entire users’ requirements. The designer extracts the fragment of the DO corresponding to requirements and can locally enrich it with new concepts and properties. DWO is then annotated with multidimensional concepts. DW design must follow the multidimensional paradigm which organizes data into central concepts called Facts representing the subject analyzed, and Dimensions representing different perspectives of analyzing the fact. Facts are generally linked to their corresponding dimensions by means of (1,n) relationships.

C. Logical level

A set of transformation rules is used to translate the conceptual model into a logical relational model. Several works in the literature proposed methods for translating ontologies described in a given formalism (PLIB, OWL, RDF) to a relational or object-relational representation. We use the set of translation rules proposed in the context of OntoDB project [3].

In our case study scenario (ontology model and requirements), the DWO was equal to the DO. The relational model defined (Fig. 6) has one main fact Order having the following dimensions Customer, Supplier, Brand and Time. Each dimension is divided into levels. Note that this schema covers totally the relational model of SSB benchmark (tables and attributes) after de-
normalization (i.e., regrouping dimension levels into one dimension table). We propose in [10] a case tool implementing these design steps.

D. Definition of the ontology-based DW

The DW we define is validated using OntoDB OBDB. OntoDB introduces the meta-schema of the ontology that we extend by the goal model in order to keep a persistent trace of requirements during the DW life cycle (Fig. 7(b)).

This extension is done using OntoQL statements (Fig. 7(a)). This solution satisfies perfectly the new ANSI/SPARC architecture we revisited. The ontology layer is represented through the domain ontology. A conceptual view of the DW is materialized by storing the DWO. The requirements model is stored, on which designers rely for their different design tasks at the conceptual, logical, and physical design phases.

V. VALIDATION: REQUIREMENTS-DRIVEN OPTIMIZATION

We propose here to define index optimization structures from the set of specified requirements. The purpose of this step is to help the designer identifying relevant optimization structures basing on the only resource he possesses: users’ requirements. The traditional index selection problem (ISP) is formalized as follows:

Having on the following assumptions:

1. \( I = \{I_1, ..., I_n\} \) set of candidate indexes (obtained from selection predicates of queries).
2. \( Q = \{Q_1, ..., Q_m\} \) set of queries.
3. \( S \) storage space allocated for indexes.

The resolution of the problem consists in finding an index configuration such that: the execution cost of queries is minimal, and the space allocated for storing indexes does not exceed \( S \).

We adapt this problem by considering user requirements instead of queries that are not yet available at this design step. Many algorithms have been proposed to solve ISP. We follow the algorithm we proposed in [2] for selecting bitmap join indexes (BJI). We start by translating the set of requirements stored in the goal model proposed above (Fig. 4(b)), into queries. Note that this translation is natural since an OLAP query corresponds to the definition of a goal presented above, expressed using SQL syntax. This translation is done using plugin Accelio available within Eclipse environment. Accelio provides templates for translation from Model to Model and from Model to text. We use Model to Text translation to translate each goal instance to an SQL query. We define the following translation algorithm, applied to each goal instance:

```
Create Entity #GoalModel (#Property (#Domain String, #Name String, #Date Date), #collects REF (#Goal))
Create Entity #Object (#its_properties REF (#Property) Array)
Create Entity #Result (#its_properties REF (#Property))
Create Entity #Metric (#its_properties REF (#Property))
Create Entity #Parameter (#its_properties REF (#Property) Array)
Create Entity #Metric(#its_properties REF (#Property) Array)
Create Entity #Goal (#Property (#Id Int, #Name String, #Description String, #Context String, #Purpose String), #defined_for REF (#Object), #has REF (#Result), #Measured_by REF (#Metric), #analyze REF (#Parameter), #Concern REF (#Actor))
Create Entity Functional_Goal UNDER #Requirement
Create Entity NonFunctional_Goal UNDER #Requirement
```

Figure 6. OBDW: Extended OBDB of type III by the requirements model
SELECT Properties used as (Result or Metric) in the goal model
FROM Properties used as (Objects) in the goal model
WHERE Properties used as (Parameter) in the goal model

We adapted algorithm proposed in [2] by considering queries obtained from user requirements. The algorithm is based on the following inputs: workload of queries (extracted from requirements), a given storage space, selection attributes and data of the cost model. Those data are: size of the fact table, attributes cardinalities, size of a fact tuple, size of a page system, frequency of execution of each query. All these data can be estimated at the logical level. We consider for the frequency of execution of each query the number of actors formulating the corresponding requirement. For simplicity reasons, we assumed that frequency is equal to 1 for each requirement. The algorithm proposed a set of indexes that we compared with the real set of queries (workload) provided by SSB benchmark. Table 1 summarizes obtained results and shows this comparison. We notice that the set of indexes generated from the set of requirements, covers the set of indexes generated from queries of the benchmark. We demonstrated here that some relevant optimization structures can be proposed by designers from the logical level.

VI. CONCLUSION

DBs and DWs usually follow the ANSI/SPARC architecture during their design, consisting in three layers: conceptual level assumed by designers, logical level based on a set of translation rules and physical level assumed by DBA. This design levels relies on elicitation and analysis of user requirements. In this current architecture, two issues are noticed: (i) the role of DBA is constantly minimized by the introduction of administrator tools, (ii) user requirements represent the most valuable information on which designers rely for their task but no trace of those requirements is stored in the final DB/DW structure. ANSI/SPARC architecture has already been extended by the introduction of an ontological level. Due to their similarity with conceptual models, many DB and DW methods have used ontologies as a first level of specification. We proposed a new ANSI/SPARC version and a methodology for reconsidering the role of designers and making requirements persistent. We propose a requirements-driven optimization process as validation. The final DW materializes the requirements model, the conceptual DW and the logical model. As perspectives, we project to study new optimization structures based on requirements, to consider other types of requirements like non functional requirements and to study the evolution of requirements on the DW management.

REFERENCES