Managing Data Warehouse Traceability: A Life-Cycle Driven Approach

Selma Khouri[1,2(✉)], Kamel Semassel[2], and Ladjel Bellatreche[1]

[1] LIAS/ISAE-ENSMA – Poitiers University, Poitiers, France
{selma.khouri,bellatreche}@ensma.fr
[2] National High School for Computer Science (ESI), Algiers, Algeria
k_semassel@esi.dz

Abstract. Traceability has been used as a quality attribute for softwares for some decades now. Traceability can be defined as the ability to follow the life of software artifacts. Unfortunately, making a DW traceable did not have the same spring as for software systems. Nowadays, DW systems are evolving in a dynamic environment, where DW design become a complex task involving many resources and artifacts. In order to facilitate this task, a design life-cycle has been defined including five main phases. Due to the special idiosyncrasy of DW development, a tailored traceability approach is required. Our proposal in this paper is a novel DW traceability approach, driven by its design life-cycle. This approach covers the *whole* cycle and considers its inter-relationships. This study required (i) the formalization of each life-cycle phase and (ii) the identification of the interactions between and inside these phases. The traceability approach is conducted by two main activities: the *identification* of trace artifacts and links materialized in a traceability model and the *recording* of the model. The approach is illustrated using TPC-H and ETL benchmarks. It is implemented using Postgres DBMS.

Keywords: Data warehouse · Traceability · Design life-cycle · Trace links

1 Introduction

Data warehouses (DWs) are the core components of decision support systems. DW design is a difficult task which involves many actors, tools and artifacts. In order to facilitate this task, DW design follows a design life-cycle including five main phases: requirements definition, conceptual design, logical design, Extract-Transform-Load (ETL) phase and physical design [6]. This cycle has been defined through a long process including three main evolutions [9]: (1) the *vertical* evolution which defined the five design phases. The DW design cycle first considered the logical, ETL and physical phases. This cycle vertically evolved by the addition of the requirements definition phase [10] and then the conceptual design phase [7]. (2) The *internal* evolution defined the steps of each design phase. For example, the requirements definition phase includes four steps: elicitation of

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 199–213, 2015.
DOI: 10.1007/978-3-319-19069-3_13

requirements, design, specification and validation. Each design phase provides an output schema which is *transformed* to another one using some given rules or algorithms. (3) The cycle evolved horizontally due to continuous innovations in data management systems pushed by new requirements. The *horizontal* evolution diversified each design phase by adding new storage schemes (according to given data models like relational, Nosql or even newSQL schemas), database architectures (conventional DB, NoSQL BD, Semantic DB, etc.) and deployment platforms (ex. Centralized, Cloud).

These evolutions let the DW system evolve in a dynamic environment, where new functional or non-functional requirements (like the consideration of new storage schemas) must be incorporated in the system. As instance, the horizontal evolution makes the DW schema evolve very differently from the initial conceptual schema. If we consider the DW conceptual schema presented in Figure 1, this schema can be stored as illustrated in the figure using different layouts (relational, object or Nosql storage layouts). The initial schema goes through successive modifications and the resulting schema may not match with the initial one, structurally and semantically. These transformations are included: *internally* between design elements of each design phase, *vertically* between design elements of the different design phases and *horizontally* between design elements translated using diverse design schemas.

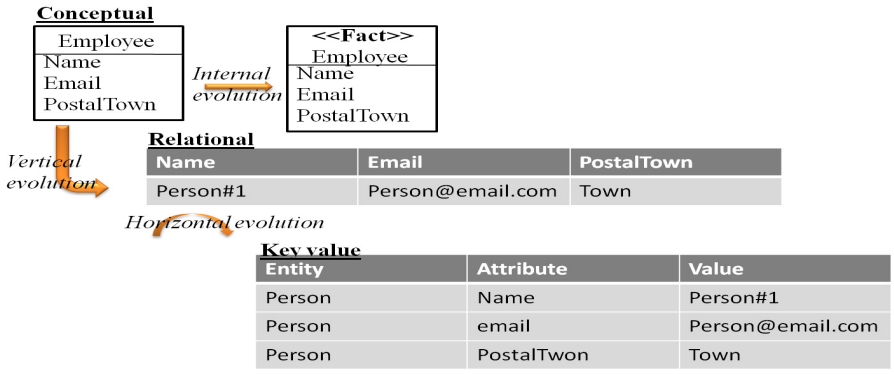


Fig. 1. Evolution links illustrated during DW design cycle

During this complex design process, the relationships between design elements are not recorded and lost, since there is no explicit traceability included. *Tracing* design elements by name matching is also not possible. Consequently, managing traceability become an important challenge for DW design, at the heart of different issues around: the validation of requirements, management of source evolution (at the instance or at the schema level), management of requirements evolution, support of change propagation and limitation of changes impact, decision support about alternative implementations, and more generally

quality management of the DW system [14,18]. Actually, many quality standards such as IEEE Std. 1219, ISO 9000ff, ISO 15504 (SPICE), and SEI CMM/CMMI, recommend traceability as an attribute of software quality [18].

Traceability can be defined as the ability to follow the life of software artifacts [18]. Traceability has been studied first in requirements engineering field for software development [18]. Unfortunately, managing traceability in DW systems did not have the same spring as for software even though it can be seen as a software product. Due to the special idiosyncrasy of DW development, a traceability approach specifically tailored to face several challenges is required [13,14]. Some attempts have been made in order to manage traceability issue in DW design. However, in existing approaches, traceability is studied for some design phases and not for the whole design cycle. This is explained by the lack of a global vision of the design life-cycle, and the different relationships between the design elements. This drawback can be due to the *recent* establishment and consensus on the DW design life-cycle. As instance, most approaches study the traceability only from the sources perspective [4,5], or from the requirements perspective [13]. Our main contribution in this paper, is the proposition of *design life-cycle* driven approach for managing traceability in DW systems. Storing these traces allows an easy forward and backward navigation between design elements. As defined by the ANSI/IEEE Std 830-1993 [2], backward traceability refers to the ability to follow the traceability links from a specific artifact back to its sources from which has been derived. Forward traceability stands for following the traceability links to the artifacts that have been derived from the artifact under consideration. This proposition is motivated by some recent studies which demonstrate the importance of expliciting hidden links between design phases, like [17] which illustrates the usefulness of the implicit links between the ETL model and the DW collected requirements. Managing traceability become possible by:

Identifying the relationships between design elements: each design phase produces in output *schema*, and the navigation between phases is achieved using a design process (usually a set of rule translating an input element into an output element). The identification of design elements and their relationships required a deep study and formalization of the DW design life-cycle. The way in which we propose to achieve traceability is making use of a specific trace component in which information related to the design phase (the design schema) and the translation process between the phases is explicitly stored. Figure 4 illustrates the design artifacts and the traces that should be identified and stored.

Preserving these relationships between all design elements: in a usual design process, only the physical design schema storage is achieved inside the DBMS. Since the DBMS system is the "natural" component for storing the design schema and its meta-data, we propose to extend its meta-model by the other design elements (requirements, conceptual, logical and ETL). In order to keep trace of the design process, the reconciling items between phases (eg. The translation rules) are also stored.

The main contributions of this paper are: **(1)** the formalization of DW design-life cycle, which allowed the identification of design elements that should be

considered. **(2)** An approach for managing traceability in DW systems. The approach is conducted by the DW design life-cycle and considers its three evolution dimensions. **(3)** Implementation of the traceability approach using an ETL benchmark based on the TPC-H benchmark. The trace components are stored in Postgres DBMS implementing the DW, and then retrieved. This implementation shows the feasibility of the traceability approach.

The rest of the paper is organized as follows: section 2 presents a motivating example that will also be used in the implementation section. Section 3 presents the main concepts related to traceability in software product in general, then in DW context. Section 4 presents the design cycle formalization framework. Section 5 illustrates the traceability approach we propose, which includes a traceability model and a set of traceability activities. Section 6 presents the implementation of the approach. Section 6 concludes the paper.

2 Motivating Example

In order to illustrate our proposal, let us define the following example, that uses the ETL benchmark proposed in [15]. This benchmark completes TPC-H[1] with ETL workflows. TPC-H is a decision support benchmark that describes a sales system. The conceptual schema of the benchmark can be defined by a reverse engineering process (as illustrated in[2]). The set of requirements are provided in TPC-H benchmark and defined as business questions. The set of sources and the target DW logical schemas are presented in the following table.

Simitsis et al. provided in [15] a set of ETL workflows that allows to populate the target DW schema with sources data. Each ETL workflow defines an ordered sequence of ETL expressions that are applied to some input elements. Figure 2 presents an example that populate Supplier and Partsupp tables.

In this example, the workflow uses *Partsupp* and *Supplier* source tables as inputs. It includes the following ETL operations: (1) concerning the Partsupp source, surrogate key values are generated for the "part-key" and "suppkey" fields. (2) Then, the "totalcost" field is calculated and added to each tuple. (3) Then, the transformed records are saved and loaded in the table *DW.Partsupp*. (4) Concerning the Supplier source, a surrogate key is generated for the "suppkey" field, (5) and a second activity transforms the "phone" field. (6) Then, the transformed records are saved and loaded in the table *DW.Supplier*.

A traceability approach should, as instance, define the information given by the following TPC-H requirement: "*the requirements identifies customers who might be having problems with the parts that are shipped to them. The requirement lists the customer's name, address, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue. Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all lineitems.*" This requirement has identified the following concepts in the conceptual schema: *Customer* and all its attributes, *LineItem*

[1] www.tpc.org/tpch/

[2] http://www.essi.upc.edu/$\sim$petar/demo.html

and its attributes (l_extendedprice and l_discount). The concept *LineItem* is identified as a fact and *Customer* as a dimension. These concepts are transformed into relations then into tables in the physical level (using corresponding translation rules). The workflows 2 and 5 (named Wishbone and Fork in the benchamrk) are used to populate these tables. The source elements that populated these tables are *Customer* and *LineItem* in source 2. The backward path can also be identified. As instance, the DW *Customer* table has been populated using only *Customer* Table of source1. The requirements that are related to the *Customer* Concept are Requirements 3, 5 and 10 (in the benchmark).

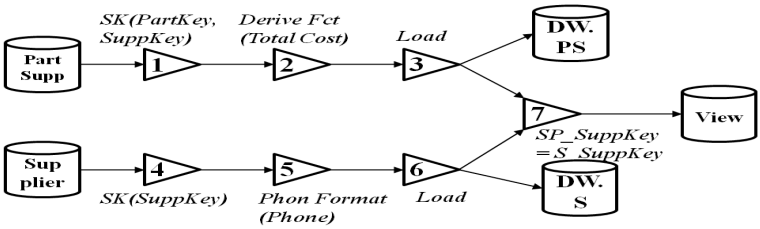


Fig. 2. ETL workflow example

Data Warehouse:

```
PART (rkey s_partkey, name, mfgr, brand, type, size, container,
comment)
SUPPLIER (s_suppkey, name, address, nationkey, phone, acctbal,
comment, totalcost)
PARTSUPP (s_partkey, s_suppkey, availqty, supplycost, comment)
CUSTOMER (s_custkey, name, address, nationkey, phone, acctball,
mktsegment, comment)
ORDER (s_orderkey, custkey, orderstatus, totalprice, orderdate,
orderpriority, clerk, ship-priority, comment)
LINEITEM (s_orderkey, partkey, suppkey, linenumber, quantity,
extendedprice, discount, tax, returnflag, linestatus, shipdate,
commitdate, receiptdate, shipinstruct, ship-mode, comment, profit)
```

Storage House Source:

```
PART (partkey, name, mfgr, brand, type, size, container, comment)
SUPPLIER (suppkey, name, address, nationkey, phone, acctbal, comment)
PARTSUPP (partkey, suppkey, availqty, supplycost, comment)
```

Sales Point Source:

```
CUSTOMER (custkey, name, address, nationkey, phone, acctball,
mktsegment, comment)
ORDER (orderkey, custkey, orderstatus, totalprice, orderdate,
orderpriority, clerk, shippriority, comment)
LINEITEM (orderkey, partkey, suppkey, linenumber, quantity,
extendedprice, discount, tax, returnflag, linestatus, shipdate,
commitdate, receiptdate, shipinstruct, shipmode, comment)
```

3 Related Work

Traceability has been used as a quality attribute for software development. Traceability has started as an area of requirements engineering. It has then been used in model-driven development area. Nowadays, traceability concept is used in its general term, and is seen an instrument to generally follow the whole software development process [18]. Traceability is defined in the IEEE Standard Glossary of Software Engineering Terminology [1] as: (1) the degree to which a relationship can be established between two or more products of the development process. (2) The degree to which each element (like documents, models, or code) in a software development product establishes its reason for existing. A trace is defined in the same standard glossary [90] as a relationship between two or more products of the development process.

3.1 Traceability Meta-Model

A traceability scheme is the basic component of a traceability approach. It helps in this process of recording traces and making them persistent. Different ad hoc traceability meta-models have been proposed in the literature [18]. These models share the same core conception. Currently there is no single standardized traceability meta-model. We thus use the traceability model proposed in [18], which presents the common features of the meta-models found in literature. The model is presented in figure 3.

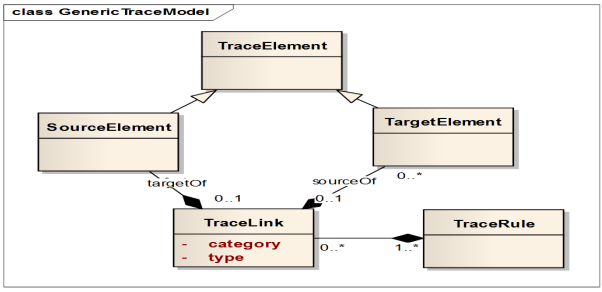


Fig. 3. Generic Traceability Model [18]

The model defines a context used to define which metamodels are used and the values of the transformation parameters (if an automatic transformation was used to create the traces). The trace link can contain metadata defining which rule or operation created the trace. The most important feature of a trace is the links to the model elements or artifacts, which are connected via the trace (source and target elements). Spanoudakis and al. [16] defined eight *categories* of trace links derived from an analysis of related literature: dependency links, refinement, evolution, satisfiability, overlap, conflict, rationalization and contribution.

Two *types* of traceability links are identified [14]: horizontal traceability that links elements belonging to the same project phase or level of abstraction, and vertical traceability links elements belonging to different ones.

3.2 Traceability in DW Design Context

Since DWs development can be conducted as for software products, traceability has been applied for them. In this context, existing studies that manage traceability issue focus on some aspects of DW design, and do not cover the whole design life-cycle.

Among these studies, *Chui et al.* [4] (detailed in [5]) provide an algorithm for tracing the lineage of tuples in set-based aggregate-select-project-join (ASPJ) views in relational DWs. The authors also discuss further optimizations of the tracing procedure in special-case scenarios. This study aim to trace instances of data by means of queries. However it studies traceability from source perspective, it does not support modeling and ignore the impact of user's requirements and the translations between the design phases. *Marotta et al.* [12] propose a DW design approach using a schema transformation approach. The approach identifies the trace as the links (or the path) providing the information about the sequences of primitives that were applied to each DW element starting from a source element. The approach focuses on the traces between the target DW schema and the sources schemas, and the process providing the traces is not detailed. *Trujillo et al.* [13,14] proposed a trace meta model and an approach used for expliciting the traces between the conceptual DW schema obtained from the mapping between sources and requirements. Traceability of the ETL, logical and physical phases are ignored in this study. The traceability links are defined using Query/View/Transformation (QVT) rules and ATLAS Transformation Language. As explained in [18], such frameworks allows technically to record traces as a by-product of the transformation process, but they present issues, like setting up tools and configuring transformations for traceability which is not well supported and is thus also difficult and error-prone. Additionally, these framework focus more on traceability of model elements through consecutive transformation steps, which is not the case of all DW design steps. As instance, the ETL phase does not transform the physical schema but just populates it. Few DW development tools provide some traceability capabilities,but they are usually limited to the ETL and physical phases. We believe that the recent consensus established around the cited design life-cycle including the five phases, allows us to define a global traceability approach covering this cycle. This proposition is furthermore motivated by the necessity of each design phase proved in DW literature, and by the different recent studies that exploit correlations between design phases. As instance, [17] illustrates the usefulness of the implicit links between the ETL model and the DW collected requirements. The availability of design traces become thus mandatory for the achievement of such studies. Our traceability approach is first based on a traceability model, instantiating the meta model (presented previously) for DW design context. This study requires

the formalization of the outputs of each design phase and the links between them, which is presented in what follows.

4 Design Life-Cycle Formalization

The thorough analysis of $\mathcal{DW}$ design literature [8] allows us to propose a formalization of $\mathcal{DW}$ cycle as follows: $< \mathcal{RM}, \mathcal{CM}, \mathcal{LM}, \mathcal{ETL}, \mathcal{PM} >$. Note that the framework formalization must contain the model-based and the process-based facets. We start by the presentation of the model-based aspect :

Requirements Model ($\mathcal{RM}$) : $< Req, Rel, Formalism >$, such as:

- *Req*: is the set of requirements collected from users and validated.
- *Rel*: defines different types of relationships between requirements, such as *conflict, equivalent, require*, etc.
- $Formalism(RM)$: is the formalism used for analyzing the set of requirements (like goal or process oriented formalisms, etc.)

Conceptual Model ($\mathcal{CM}$) : different conceptual models have been used in DW conceptual design, among them E/R, UML class diagram, ontology models and Description logic (DL) formalism. Different studies defined DL as a high level formalism that is able to capture the most popular data *class-based modeling* formalisms presently used in *databases* [3]. We assume that the reader is familiar with the basic concepts of DL formalism. $\mathcal{CM}$ is thus formally defined as follows: $< C, R, Ref, Multidim, Formalism >$

- C: denotes *Concepts* of the model (atomic concepts and concept descriptions).
- R: denotes *Roles* of the model. Roles can be relationships relating concepts to other concepts, or relationships relating concepts to data-values.
- $Ref : C \cup R \rightarrow (Operator, Exp(C, R))$: Ref is a function representing the various correlations between concepts. Operators can be inclusion ($\sqsubseteq$) or equality ($\equiv$). $Exp(C, R)$ is an expression over concepts and roles of $\mathcal{CM}$ using constructors such as union, intersection, restriction, etc.
- $Multidim : C \cup R \rightarrow$ multidim: defines the multidimensional role (fact, dimension, measure, etc) of each concept or role
- $Formalism(CM)$: is the *formalism* followed by the global ontology model like *ER, UML, OWL*, etc.

Logical Model ($\mathcal{LM}$) : $< Construt_{LM}, Formalism(LM) >$

- $Construt_{LM}$: the constructs of the logical data schema. For the relational model, it would be the relations and their attributes.
- $Formalism(LM)$: the used formalism in logical phase (E.g. Relational, Object, etc)

Physical Model ($\mathcal{PM}$) : $< Construt_{PM}, I, Pop, OPS, Formalism(PM) >$

- $Construt_{PM}$: the constructs of the physical data schema. For the relational model, it would be tables and their columns.
- I: the set of instances
- Pop: $Construt_{PM} \rightarrow 2^I$: relates some PM constructs to their instances from the set I.
- OPS: defined optimization structures
- $Formalism(PM)$: the used formalism in the physical design phase (E.g. Relational)

The $\mathcal{ETL}$ *Model* : a conventional ETL process is achieved between the target physical DW schema and physical schemas of sources using some defined mappings relating their elements. The ETL model is thus formally defined by the triplet $< G, S, M >$ [11], such as:

- G: the global schema (the target physical schema as defined in this section).
- S: Each local source S_i is a data repository. We retain in the source:
  - $Construt_{PMS}$: The physical elements of the source.
  - I: the set of instances
  - Pop: $Construt_{PMS} \rightarrow 2^I$: relates some PM constructs to their instances from the set I.
- M: Mappings assertions relate a mappable element of schema $\mathcal{G}$ to a mappable element of schema $\mathcal{S}$, using an *ETL Expression*.

The process-based formalization concerns the different algorithms used during the design process, like the translation algorithms (from one design phase to another) and the ETL algorithm. These algorithms are dependent of the design context. The trace information that need to be identified and stored are related to the way in which specific design elements in each phase are translated into elements of adjacent phases. In DW design, the conceptual schema is defined either from: the sources schemas, or from the requirements schema or after confronting the source and requirements schemas. This confrontation allows identifying the set of relevant concepts that should be stored in the DW. The multidimensional role (fact, dimension, measure, dimension attribute or hierarchy level) of each concept of the conceptual schema is afterward identified. The logical schema (usually relation or multidimensional) is derived from the conceptual schema using some defined rules. The physical schema is derived from the logical schema using some defined rules. The ETL process uses the set of mappings M in order to populate the physical DW schema with data from sources. Consequently, the process-based trace information include:

- The mapping link between the source element and/or the defined requirement and the conceptual element.
- The multidimensional role of each conceptual element.
- The set of rules deriving the logical schema from the conceptual one, and the physical schema from the logical one.

– The specific rule applied for each element.
– The ETL expression used to populate the physical design elements with the identified data (related to source elements).

These trace links are vertical links since they are defined between elements of different phases. Figure 4 illustrates these trace links. Horizontal trace links are also defined. These links are observed in the requirements and conceptual phases: between conceptual elements (Ref function), between the set of requirements (Rel function). Particularly, the links between (source, requirement and conceptual elements) are tagged by their category type, eg. “satisfiable”, “conflict”, etc. The ETL, logical and physical schemas do not store such internal links. Figure 5 illustrates the DW trace model that we propose, based on the design cycle formalization.

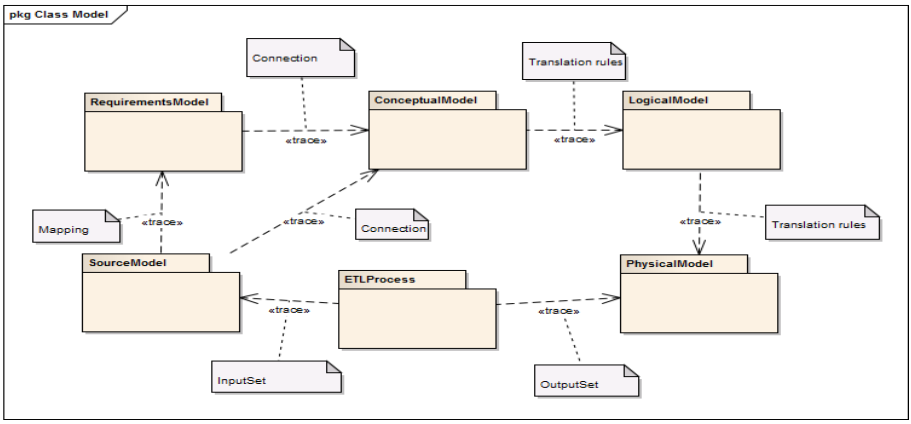


Fig. 4. Vertical Trace links

5 Proposal

This section presents the traceability approach we propose. It includes: the traceability model proposed in the previous section (defining the trace models and links) and the defined activities for recording and using the traces. Basically, there are four activities when working with traces [18]: planning for traceability, recording traces, using them, and maintaining them. These activities are performed as part of the software development process. Unfortunately, there is only little to no guidance for practitioners for applying these approaches into standard procedures [18]. We thus have to propose traceability activities dedicated for DW systems.

5.1 Planing

The outcome of this activity is a traceability model, which is described in the previous section.

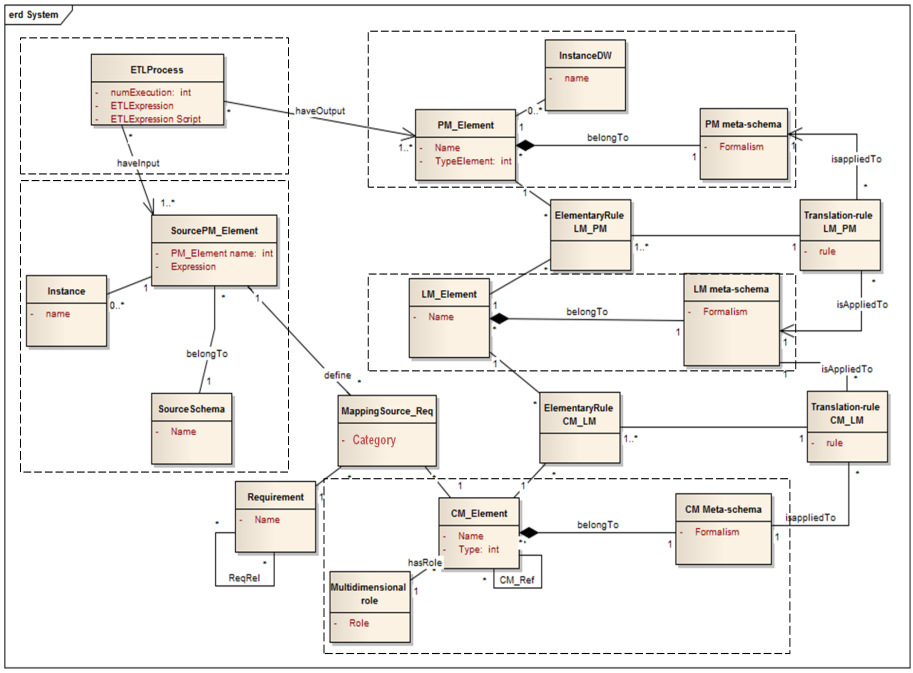


Fig. 5. DW Traceability Model

5.2 Recording

This activity leaves traces on the artifacts and make them persistent. This can be done during the design process or after the actual design process has been finished. It is recommended that traces are recorded immediately to avoid recording imprecise traces. The approach is independent of the formalisms used for each design phase (UML, ER, relational, etc). One just needs to identify the design artifacts and the translation rules used. During the DW design process, the DBMS system is used during the physical design phase, in order to record information about data and meta-data. Two components are recorded: (1) the physical model, which contains the data schema and its instances and (2) the meta-base which contains the physical meta-schema. Since the DBMS is the "natural" component used for recording DW information, we propose to use it in order to record the traceability model. The traceability model we proposed is composed of the different schemas defined in each design phase: the requirements schema, the conceptual, logical, ETL and physical schemas. Since the meta-base is the meta-data repository, it can easily be extended by additional information retracing the origin of the physical model during the whole design cycle. This recording activity is illustrated in figure 6. The instantiation of each model will record the design elements used in each design phase, and their data. This process is illustrated in the next section

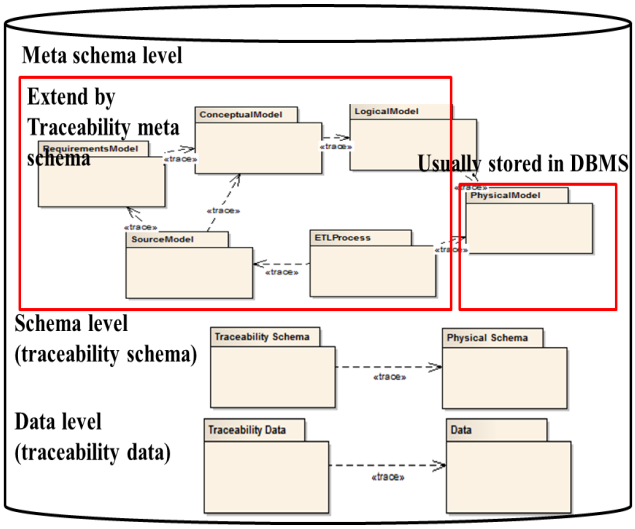


Fig. 6. The recording activity

5.3 Using

This activity retrieves the data describing the traces in order to produce reports or to find relevant information. Since the recording activity is achieved using the DBMS system, the access to data is done using the dedicated query language. Some examples are presented in the next section, where the recording activity is accomplished using Postgres DBMS. Sql query language is thus used to retrieve traceability information. Defined queries and stored procedures have been defined on each design trace in order to easily retrieve trace links.

5.4 Maintaining

The traces must be maintained in case of a structural change of the design process, or if errors in the trace data has been detected. This activity is out of the scope of the current study.

6 Implementation

The objective of this section is to show the feasibility and the advantages of our proposal. We illustrate the approach for the case of ETL traceability. The traceability approach is preceded by the implantation of the ETL workflows of the benchmark. For each ETL operation we execute what is needed in terms of traceability management. This implementation requires three steps: (1) extend the DBMS to allow the manipulation of ETL operations more easily. (2) For each ETL operation, trigger the execution of a series of instructions capturing traceability data and storing them *automatically* in the database. (3) Define queries in order to facilitate the use of the traceability meta-base.

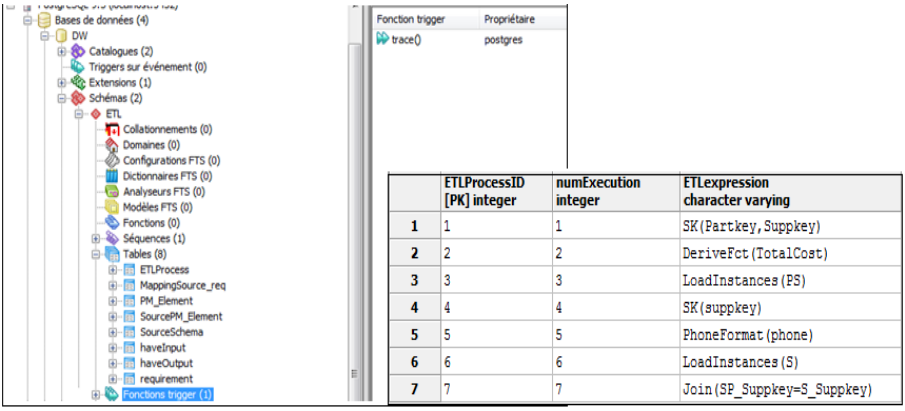
Implementation with PostgreSQL: the implementation of our proposal can be performed on any DBMS. We decided to carry out this task using the free RDBMS PostgreSQL which offers various variability and extensibility features. It also provides the ability to program packages with internal procedures in its own language PL/pgSQL, which turned out to be completely effective.

Extending PostgreSQL to manage the ETL process: the traceability meta-model defined previously is created as an extension of the physical meta-schema of the DBMS. New meta-tables (eg. ETLProcess meta-table) are thus created related to existing "Table" and "Column" of the DBMS.

Managing and storing of traceability data: in order to provide the DBMS new commands that can be used in the ETL process, we need to create new functions for each ETL operation composing the ETL workflow. These operations are implemented using PostgreSQL functions (query language functions witch is written in SQL and procedural language functions written in PL/pgSQL). As instance, in our example, the first ETL operation in the ETL process is the generation of surrogate keys values from the source table Surrogate key SK(Partkey,Suppkey) :

```
CREATE FUNCTION SK(IN source character, IN table1 character, OUT RES boolean)
RETURNS boolean AS
$BODY$
BEGIN
ALTER TABLE source.table1 ADD COLUMN table1Id SERIAL UNIQUE;
END;
$BODY$
LANGUAGE plpgsql VOLATILE
```

Once the function created, its use generates the execution of the defined script. With the same method, we create all the functions of the ETL process such DeriveFct(Total Cost), LoadInstances (PS), etc. Each script is stored in



	ETLProcessID [PK] integer	numExecution integer	ETLexpression character varying
1	1	1	SK(Partkey,Suppkey)
2	2	2	DeriveFct(TotalCost)
3	3	3	LoadInstances(PS)
4	4	4	SK(suppkey)
5	5	5	PhoneFormat(phone)
6	6	6	LoadInstances(S)
7	7	7	Join(SP_Suppkey=S_Suppkey)

Fig. 7. Managing traceability in Postgres

the ETLProcess table (ETLScript attribute). The numExecution order or the operation in the workflow is also stored. Each ETL operation triggers instructions for managing and storing traceability data in the specific tables which instantiate the meta-tables (eg. ETLProcess table). The trigger is associated with the table 'ETLProcess' and executes the function Trace (figure 7) when a new element is inserted. The trace is created with the code:

```
CREATE TRIGGER TRIG_TRACE  AFTER INSERT ON ETL PROCESS FOR EACH ROW
EXECUTE PROCEDURE TRACE();
```

Note that in a conventional design, these traces are usually lost once the DW implemented. this implementation allows their automatic (or semi-automatic) storage during the design process. Now that the database contains traceability information, they can be easily retrieved using SQL queries. For example, if we want to know how the instances of the Partsupp table have been loaded, we can run the query selecting instances of PM_Element ID = 1 (which match to Partsupp) such as:

```
SELECT * FROM ETLProcess, haveOutput
WHERE ETLProcess.ETLProcessID = haveOutput.ETLProcessID
AND haveOutput.PM_ElementID= 1
```

7 Conclusion

We have presented in this paper a DW traceability approach in order to record explicitly the relationships between design elements during DW design life-cycle. The different evolutions of the life-cycle, which make the DW evolve in a very dynamic environment, motivated this study. The approach is based on the cycle formalization that identified the design elements and the interactions between them. The approach includes three activities: (1) planning: which identified the DW traceability model, (2) recording: the DBMS implementing the DW has been used for persisting the traces,(3) using the traces and retrieving them. The approach is illustrated using TPC-H and ETL benchmarks. Postgres DBMS and its extensibility features have been used in order to implement the approach. The main perspectives of this study are: the proposition of a mechanism for automatic trace maintainability, the validation of the approach using different relational and non-relational DBMSs, the proposition of a change propagation process that completes the approach and the automation of the approach in a complete tool that provides a visual support for trace links.

References

1. Radatz, J., Geraci, A., Katki, F.: IEEE standard glossary of software engineering terminology. IEEE Std **610121990**(121990), 3 (1990)
2. IEEE Computer Society. Software Engineering Standards Committee, and IEEE-SA Standards Board. "IEEE Recommended Practice for Software Requirements Specifications." Institute of Electrical and Electronics Engineers (1998)

3. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. In: Logics for Databases and Information Systems, pp. 229–263. Springer, US (1998)
4. Cui, Y., Widom, J.: Practical lineage tracing in data warehouses. In: Proceedings. 16th International Conference on Data Engineering, 2000, pp. 367–378. IEEE (2000)
5. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. The VLDB JournalThe International Journal on Very Large Data Bases **12**(1), 41–58 (2003)
6. Golfarelli, M.: From user requirements to conceptual design in data warehouse design a survey. In: Data Warehousing Design and Advanced Engineering Applications Methods for Complex Construction, pp. 1–6 (2010)
7. Golfarelli, M., Maio, D., Rizzi, S.: The dimensional fact model: a conceptual model for data warehouses. International Journal of Cooperative Information Systems **7**(02n03), 215–247 (1998)
8. Khouri, S.: Cycle de vie smantique de conception de systmes de stockage et de manipulation de donnees. PhD thesis, ENSMA & ESI, October 2013
9. Khouri, S., Bellatreche, L.: Towards a configurable database design: a case of semantic data warehouses. In: Meersman, R., Panetto, H., Dillon, T., Missikoff, M., Liu, L., Pastor, O., Cuzzocrea, A., Sellis, T. (eds.) OTM 2014. LNCS, vol. 8841, pp. 760–767. Springer, Heidelberg (2014)
10. Kimball, R.: The data warehouse toolkit: practical techniques for building dimensional data warehouses. John Wiley & Sons Inc., New York (1996)
11. Lenzerini, M.: Data integration: a theoretical perspective. In: :Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 233–246. ACM (2002)
12. Marotta, A., Ruggia, R.: Data warehouse design: a schema-transformation approach. In : Computer Science Society, SCCC 2002, Proceedings of the 22nd International Conference of the Chilean, pp. 153–161. IEEE (2002)
13. Mat, A., Trujillo, J.: A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses. Information Systems **37**(8), 753–766 (2012)
14. Mat, A., Trujillo, J.: Tracing conceptual models' evolution in data warehouses by using the model driven architecture. Computer Standards & Interfaces **36**(5), 831–843 (2014)
15. Simitsis, A., Vassiliadis, P., Dayal, U., Karagiannis, A., Tziovara, V.: Benchmarking ETL workflows. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 199–220. Springer, Heidelberg (2009)
16. Spanoudakis, G., Zisman, A.: Software traceability: a roadmap. Handbook of Software Engineering and Knowledge Engineering **3**, 395–428 (2005)
17. Theodorou, V., Abelló, A., Thiele, M., Lehner, W.: A framework for user-centered declarative etl. In: Proceedings of the 17th International Workshop on Data Warehousing and OLAP, pp. 67–70. ACM (2014)
18. Winkler, S., Pilgrim, J.V.: A survey of traceability in requirements engineering and model-driven development. Software & Systems Modeling **9**(4), 529–565 (2010)