A Semi-Automatic Approach to Handle Data Warehouse Schema Evolution using Ontology

M. Thenmozhi1, K. Vivekanandan2, G. Gayathree3
1Pondicherry Engineering College/Assistant Professor, Department of Computer Science and Engineering,, Puducherry-14, India
Email: thenmozhi@pec.edu
2Pondicherry Engineering College/Professor, Department of Computer Science and Engineering, Puducherry-14, India
3Pondicherry Engineering College/PG Student, Department of Computer Science and Engineering, Puducherry-14, India

Abstract—In recent years, the number of digital information storage and retrieval systems has increased immensely. Data warehousing has been found to be an extremely useful technology for integrating such heterogeneous and autonomous information sources. Data within the data warehouse is modelled in the form of a star or snowflake schema which facilitates business analysis in a multidimensional perspective. The data warehouse schema is derived from these sources and business requirements. Due to changing business needs the information sources not only change their data but as well change their schema structure. In addition to the source changes the business requirements for data warehouse may also change. Both these changes results in data warehouse schema evolution. Existing approaches either deal with source changes or requirements changes in a manual way and the changes to the data warehouse schema is carried out at the physical level. This may induce high maintenance costs and complex OLAP server administration. As ontology seems to be promising technique in data warehousing research, in this paper we propose an ontological approach to automate the evolution of a data warehouse schema. Our method assists the designer of the data warehouse to handle evolution at the conceptual level and after validation the physical schema is constructed automatically.

Index Terms— Data warehouse evolution, Data warehouse schema changes, Data source evolution for data warehouse, Requirements evolution for data warehouse.

I. INTRODUCTION

A data warehouse (DW) is a large centralized database that stores data integrated from multiple, heterogeneous data sources. Data integrated in a DW are analyzed by the On-Line Analytical Processing (OLAP) applications to discover business trends, patterns and anomalies as well as exploring the hidden dependencies between data [2]. The multiple data sources, integrated into a DW can change in the course of time. The business requirements of a DW also change frequently. Changes in data sources or business requirements result not only in changes to the DW workload but can invalidate existing schemata of a DW. Hence it is necessary to propagate the changes to the schema structure of the DW [16]. As a consequence of this, the DW evolves. Evolution in the DW can be categorized into three different approaches such as schema evolution, schema versioning and view maintenance. Schema evolution involves in transferring data from old
schema and updating it in a new schema [18]. Schema versioning consists in keeping the history of all versions by temporal extension or by physical storing of different versions [3]. Unlike Schema evolution, schema versioning keeps track of changes by keeping different versions of a given DW. View maintenance aims at maintaining different materialized views according to the changes made in the source and requirements [5]. In existing works, the structural changes for DW schema during evolution are carried out manually at the physical level. This places a huge burden over the DW administrator and increases maintenance cost. Hence to overcome the limitations in DW schema evolution process it is necessary to provide a solution to automate the evolution task.

Ontology seems to be a promising solution for DW design and helps to handle several design issues [14] [15] [19]. In this paper, we propose a conceptual approach for handling requirement and data source changes for DW evolution using ontology. The changes in the requirements specified by the stakeholders as well as the developers are analyzed and later on incorporating them into the DW by performing appropriate additions, deletions and updates. In a similar manner any structural changes in the underlying data source are captured and propagated to the DW schema.

This paper is organized as follows. In section 2 we discuss preliminaries of DW multidimensional modeling, DW requirements and Ontology. Section 3 we give an overview of evolution works in the DW domain. In section 4 we present the details of proposed work and illustrate them using an example. Section 5 provides the discussion of the proposal. Finally, section 6 concludes the paper and enumerates our perspectives.

II. PRELIMINARIES

A. Multidimensional Modeling

Building a DW requires adopting design and implementation techniques completely different from those underlying information systems. It is widely accepted that DW organizes data in the form of multidimensional model. The multidimensional modeling involves generating a star schema for DW [7] [13]. The star schema consists of a central fact table consisting of numerical fields called measures and related to a set of dimension tables through referential integrity constraints. When dimension tables are normalized and dimension hierarchies are formed, the resulting schema is called as the snowflake schema.

B. DW Requirements

Successful DW design needs to be based upon not only data sources but also requirements in order to adequately represent the information needs of DW users. Requirement analysis for DW differs when compared to traditional software system. Here the requirement analysis tasks should involve analyzing the goals, resources, and rules that affect the DW structure. For modeling the information requirements, a UML profile for i* modeling framework [21] [6] as given in Figure 1 has been widely adopted by the DW community.

The i* profile is represented by means of various UML meta-classes on the left side of the Figure 1. On the right hand side of the Figure 1 the i* profile for DWs, is shown which is based on a classification of the different kinds of goals that decision makers expect to fulfill with the DW. The Strategic goal represents the main objectives of the business process (for example, “increase no. of customers”). Decision goals represent the information on how a strategic goal is achieved (for example, “determines some kind of promotion”). Information goals represent information on how decision goals be achieved in terms of information required” (for example, “analyze customer profile”). From the information goals information requirements are derived based on which contexts and measures are identified. Contexts represent the dimensions by which a DW fact is analyzed and measures are the attributes of DW fact by which the business is measured. The profile reuses the previous stereotype Goal, as we can see in the Figure 1.

C. Ontology Definition

Ontology is most commonly defined as “a formal, explicit specification of a shared conceptualization”. An Ontology describes the knowledge in a domain in terms of classes and properties, as well as relationships between them [8]. An ontology language is a formal language used to encode the ontology so that it can be shared and processed by machines. Web Ontology Language (OWL) is an ontology language recommended by W3C. It is a formal markup language for describing ontological data and for sharing information [4]. In the proposed approach the decision of using the ontology based approach, instead of using another technology for example a UML-based approach, lies in the fact that ontologies are machine processable and provides an elegant way to perform reasoning that is required for the automation of the evolutionary task.

The OWL features used in our approach are shown in Table 1.
III. RELATED WORKS

In [9] the different challenges of change management were addressed for which a framework analyzing the impact of the changes was proposed. The challenges discussed in the proposed work enable effective combination of changes in the requirements and on the architecture design. It seems to be the foundation of designing and developing effective DW systems in case of evolution. A prototype based on versioning, called the multi-version trajectory data warehouse (MVTDW) was proposed in [17]. The atomic and complex set of Trajectory DW (TDW) schema changes was identified. Few constraints were defined to assure integrity between versions and some algorithms were applied in case of the schema and instance changes on the TDW versions. An approach based on a multi version DW was presented in [3]. The DW version was used for incorporating structural changes in external data sources as well as changes to a DW schema resulting from changing user requirements. DW versions were also applied to create alternative business scenarios for predicting the future. In [20], the authors presented a conceptual requirement-oriented framework for data warehouse evolution. It analyzes the changes in the requirements specified by the stakeholders as well as the developers and later on incorporates them into the warehouse by performing appropriate additions, deletions and updates. Moreover, it consists of a module that cleans redundant or dirty data by employing certain cleaning algorithms. Comprehensive metadata support has also been provided to enhance the framework operation. The main contribution of [18] is the definition of a model supporting the multiversional implementation schema and implementation schema of separate data cube instances. The model can dynamically self-improve its design satisfying both changes in the query workload as well as changes to the implementation schema. A new derivation procedure is introduced and it is applied for evolving and improving the old to a new implementation schema state of the same implementation schema instance. Furthermore, the procedure is applied to the multiversional implementation schema consisting of two separate schema instances. These existing approaches mainly work on the physical level of the DW hence it requires lots of maintenance effort. To overcome the issues in existing works, the proposed approach works at the conceptual level to manage the DW evolution.
### Table 1 - OWL Features

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Class</td>
<td>Classes represents the concepts of the domain being modeled</td>
</tr>
<tr>
<td>C₁≡ C₂</td>
<td>Equivalent</td>
<td>States that two classes are equivalent</td>
</tr>
<tr>
<td>C₁ ⊑ C₂</td>
<td>subClassOf</td>
<td>Creates class hierarchies</td>
</tr>
<tr>
<td>C₁ ∩ C₂</td>
<td>disjointWith</td>
<td>State that two classes C₁ and C₂ are disjoint</td>
</tr>
<tr>
<td>C₁ ⊔ C₂</td>
<td>unionOf</td>
<td>The union of two classes</td>
</tr>
<tr>
<td>P</td>
<td>Property</td>
<td>To represent attributes of concepts and relationships between concepts</td>
</tr>
<tr>
<td>dom(P)</td>
<td>Domain</td>
<td>Specifies the class(-es) to which the property belongs to</td>
</tr>
<tr>
<td>rang(P)</td>
<td>Range</td>
<td>Specifies the class(-es) to which the value of the property belongs to</td>
</tr>
<tr>
<td>∀P, C</td>
<td>allValuesFrom</td>
<td>To restrict the range of the property when applies to specific class</td>
</tr>
<tr>
<td>≥nP, ≤nP</td>
<td>Min/Max Cardinality</td>
<td>Specifies the min/max cardinality of the property</td>
</tr>
</tbody>
</table>

### IV. Ontology Based DW Schema Evolution Management

In this section we propose a formal approach for managing the DW schema evolution using ontology. When data source and requirement evolves the DW schema need to be updated in order to provide up-to-date information to users. Before making structural changes to the existing DW physical schema our proposed work provides a method for updating the conceptual representation of the schema. Given information about a data source, requirements and DW schema as well as the changes in the data source or requirements, our method produces the updated version of the DW schema at the conceptual level. Figure 2 represents the steps to be followed in our approach. By making use of the ontological representation of our inputs we facilitate the automation (semi-automation) of the evolution task. First the changes occurred at data source or requirements are extracted from the corresponding ontology representation. Next the type of change and the entity affected by the change are derived and the change is propagated to the DW schema based on the mapping between data source and DW ontology. The updated DW ontology is validated by verifying its consistency. Finally from the DW ontology the new version of DW physical schema is constructed automatically in the underlying database.

#### A. Input Representation

In order to automate the DW schema evolution process we represent our input such as data source, DW and business requirement in ontology format.

**Ontology for Data Source and DW:** Applying the reverse-engineering approach we define the conceptual model of existing data sources and DW system. A semantic mapping method is adapted to facilitate the transformation of data sources and DW system from a relational model into an OWL based structure. The ontology can be constructed using protégé tool [10]. The mapping from database meta-data such as E-R model or UML model to ontology could be done using the mapping rules as given below:

1. The database table is mapped to an ontology class.
2. If a database table is related to another, then the two tables are mapped to classes with parents-child relationship.
3. If a database table is related to two tables, then the table is divided into two transferred classes.
4. The primary key is mapped to a data type property of the ontology.
5. The foreign key is mapped to an object property of the ontology.

The data source ontology (DSO) is represented as DSO = \( \{C, DP, OP\} \). Where, C is the set of classes, DP is the set of data property and OP is the set of object properties of the ontology.

The DW ontology (DWO) is represented as DWO = \( \{FC, DC, MDP, BOP, AC\} \). Where FC is the set of Fact classes, DC is the set of Dimension classes, MDP is the set of Measure data property, BOP is the set of
Business process object property such as domain and range properties, cardinality constraints, disjointness etc., and AC is the set of action class used for aggregation functions used for each fact.

**Ontology for Business Requirements**: We assume that a formal requirement analysis for the given domain has been carried out earlier and the requirements based on i* modeling framework is available. In order to capture the changes in requirements we represent it in a formal way using requirement ontology. The requirement ontology (RO) is represented as \( RO = \{ \text{SG}, \text{DG}, \text{IG}, \text{IR}, \text{CC}, \text{MDP} \} \). Where, SG is the set of Strategic goal, DG is the set of Decision goal, IG is the set of Information goal and IR is the set of information requirements represented as classes in the ontology. CC is the set of Context represented as classes, MDP is the set of Measure represented as data property.

**Example**: In order to explain the steps of our approach we make use of the sample schema for data source and DW given in Table 2. The corresponding DSO and DWO constructed using the above steps for the given schemas are shown in Figure 3. For the given sales domain the requirements based on a goal oriented approach for DW is represented using RO is given in Figure 4.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Tables</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
<td>s_customers</td>
<td>cid, name, country, city, street</td>
</tr>
<tr>
<td></td>
<td>s_orders</td>
<td>oid, cid, date, amount, price</td>
</tr>
<tr>
<td>DW</td>
<td>t_customers</td>
<td>cid, firstName, lastName, address</td>
</tr>
<tr>
<td></td>
<td>s_orders</td>
<td>oid, cid, date, amount, price</td>
</tr>
</tbody>
</table>

**Mapping**

Data Source to DW

**Capturing Change**

From Data Source and Requirements

**Change Representation**

Type of change Element changed

**Propagation**

From Data Source and Requirements to DW

**Validation**

Consistency and Correction of DW Schema

**Implementation**

DW ontology to DW Physical Schema

Figure 2. Proposed Approach

Figure 3. Data Source and DW Ontology
B. Deriving Mapping

Once the inputs such as a data source, requirements and DW are represented using ontology the next step is to find the mapping between data source and DW elements. This mapping helps to identify how DW elements are related to data source elements and propagate the changes over data warehouse. As the DW is populated with data obtained from several data sources through ETL (extract, load and transform) operations an ETL mapping exists for the DW under operation. Using the ETL mapping one can manually make changes over the DW schema structure at the physical level. Further, update the ETL mapping and ETL operations as required. As our approach aims to automate (semi-automate) the evolution task, we use a conceptual representation of the data source and DW systems. Thus we need to derive the mapping and relationship between them. The mapping phase aims at defining the correspondence between the two concepts of DSO and DWO. Here, the fact class in DWO is mapped to a corresponding class in DSO. The measure data property of DWO is mapped to equivalent data property from DSO. Similarly, dimension class and dimension data property are mapped to the corresponding class and data property respectively in DSO.

One of the mapping methods is to compute the similarity for both ontologies. In our approach we use WordNet [12] based similarity measure to find the mapping between classes and data properties. Reasoning over the descriptions provided by the mapping finds the relationship. The equivalence, subsumption and consistency are the three basic ontology reasoning tasks [11]. Our work focus on only structural changes of DW schema and ETL evolution is out of scope this paper and may be considered as future work in this area. Table 3 represents the possible relationship between the concepts and the corresponding ETL operations.

Example: The mapping results of DSO and DWO for Customer class in DWO and it corresponding properties for the given example is shown in Table 4. The graphical representation of the mapping and the corresponding ETL operations is given in Figure 5.

C. Change Representation

The mapping produced in the previous step is used to handle DW evolution. In order to propagate the changes over DWO we need to find the type of change and concept changed. In this section we present the set of evolution operators to represent the type of change and concept changed. The three possible changes that occur are Addition, Deletion and Rename. The DW elements such as Fact, Dimension, Measures etc., are subject to change. Their equivalent ontology concepts in DWO need to be changed accordingly. Performing a change over the DWO may require additional changes to be executed over the ontology. For example, addition of a new dimension i.e., class to the DWO requires addition of its data property and object property. The type of change, element changed and additional changes are given in Table 5.
TABLE III. RELATIONSHIP AND ETL OPERATIONS BETWEEN DSO AND DWO

<table>
<thead>
<tr>
<th>Relationship using Ontology Reasoning</th>
<th>ETL Operations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent</td>
<td>Load</td>
<td>Retrieves entities from a source element</td>
</tr>
<tr>
<td>Subsumed</td>
<td>Split</td>
<td>Extract parts of entities from a source element</td>
</tr>
<tr>
<td>Subsumed</td>
<td>Merge</td>
<td>Merges entities from several source elements</td>
</tr>
<tr>
<td>Subsumed</td>
<td>Join</td>
<td>Joins entities from several source elements</td>
</tr>
<tr>
<td>Equivalent</td>
<td>Filter</td>
<td>Selects entities based on a specified value</td>
</tr>
<tr>
<td>Equivalent</td>
<td>Aggregate</td>
<td>Aggregates the value of an attribute</td>
</tr>
<tr>
<td>Equivalent</td>
<td>Store</td>
<td>Stores entities target element</td>
</tr>
</tbody>
</table>

TABLE IV. RELATIONSHIP BETWEEN DSO AND DWO FOR CUSTOMER CLASS

<table>
<thead>
<tr>
<th>DSO</th>
<th>DWO</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Customer</td>
<td>Equivalent</td>
</tr>
<tr>
<td>Name</td>
<td>FirstName</td>
<td>Subsumed</td>
</tr>
<tr>
<td>Name</td>
<td>LastName</td>
<td>Subsumed</td>
</tr>
<tr>
<td>Street</td>
<td>Address</td>
<td>Subsumed</td>
</tr>
<tr>
<td>City</td>
<td>Address</td>
<td>Subsumed</td>
</tr>
<tr>
<td>Country</td>
<td>Address</td>
<td>Subsumed</td>
</tr>
</tbody>
</table>

Figure 5. Mapping between DSO and DWO
D. Propagation of changes to DW Schema

The next step of our approach is to use the mapping and evolution operators discussed in the previous steps to propagate the changes over the DW Schema which is represented as DWO. The following subsections discuss the steps to handle requirements evolution and data source evolution.

Handling Requirements Evolution: This section deals with the details of requirement evolution over DW schema. The requirements stated by the various stakeholders and developers frequently change. The changes may due to the reasons like ambiguous or insufficient requirements, changes in requirements in later stages of DW environments, generation of new requirements due to technological advances, updates, etc. For example, in order to increase the analysis axis of a business it may be required to add a new dimension to the existing data warehouse. Following are the steps to be followed in our approach in order to handle changes in the requirements:

1. Update new requirements in the requirement ontology.
2. Check the existence of the concept in the data source ontology.
3. Find the type of change and the type of concept changed.
4. If the change is the addition of a concept depending on the type of concept apply evolution operators for addition as given in Table 5.
5. If the change is the deletion of a concept depending on the type of concept apply evolution operators for deletion as given in Table 5.
6. If the change is rename of a concept depending on the type of concept apply evolution operators for rename as given in Table 5.
7. Update the dependent concepts such as aggregate functions.
8. Check the mapping to resolve any inconsistency for the recent change.
9. Check for consistency of the new ontological version of DW schema using ontology reasoner.
10. Get designer approval and update mapping between the new versions of DSO and DWO.

<table>
<thead>
<tr>
<th>Type of Change</th>
<th>DW Schema Elements</th>
<th>Equivalent Ontology Concept Changed</th>
<th>Elementary Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>Table (Fact, Dimension)</td>
<td>Class</td>
<td>Add Data Property</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add Object Property</td>
</tr>
<tr>
<td></td>
<td>Attributes (Measures, Descriptive)</td>
<td>Data Property</td>
<td>Add Property Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add Property Range</td>
</tr>
<tr>
<td></td>
<td>Relationship (Primary Key, Foreign Key)</td>
<td>Object Property</td>
<td>Add Property Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add Property Range</td>
</tr>
<tr>
<td>Deletion</td>
<td>Table (Fact, Dimension)</td>
<td>Class</td>
<td>Delete Data Property</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete Object Property</td>
</tr>
<tr>
<td></td>
<td>Attributes (Measures, Descriptive)</td>
<td>Data Property</td>
<td>Delete Property Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete Property Range</td>
</tr>
<tr>
<td></td>
<td>Relationship (Primary Key, Foreign Key)</td>
<td>Object Property</td>
<td>Delete Property Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete Property Range</td>
</tr>
<tr>
<td>Rename</td>
<td>Table (Fact, Dimension)</td>
<td>Class</td>
<td>Rename Class (If required)</td>
</tr>
<tr>
<td></td>
<td>Attributes (Measures, Descriptive)</td>
<td>Data Property</td>
<td>Rename Data Property (If required)</td>
</tr>
<tr>
<td></td>
<td>Relationship (Primary Key, Foreign Key)</td>
<td>Object Property</td>
<td>Rename Object Property (If required)</td>
</tr>
</tbody>
</table>

Handling Data source Evolution: Individual information sources are often autonomous hence it may change without being controlled from a higher data integration layer such as a DW. Following are the steps to be followed in order to handle changes in the data source:
1. Retrieve the changes made in the original data source.
2. Apply the changes over the data source ontology.
3. Find the type of change and the type of concept changed from data source ontology.
4. If the change is the *addition* of a concept in the source then check the requirements and get designer approval to carry out the changes over DW ontology. Apply evolution operators for addition as given in Table 5.
5. If the change is the *deletion* of a concept in the source then check the mapping to find the corresponding mapped DWO concept. Depending on the type of concept apply evolution operators for deletion as given in Table 5.
6. If the change is *rename* of a concept in the source then check the mapping to find the corresponding mapped DWO concept apply evolution operators for rename as given in Table 5.
7. Update the dependent concepts such as aggregate functions.
8. Check the mapping to resolve any inconsistency for the recent change.
9. Check for consistency of the new ontological version of DW schema using ontology reasoner.
10. Get designer approval and update mapping between the new versions of DSO and DWO.

*Example:* Handling source evolution: For the given scenario we assume that a new attribute called *Age* is added to the s_customers table of the data source. This attribute is added as a data property to the s_customers class of DSO. Here the type of change is the *addition* and the concept changed is data property. After checking the requirement using RO, we apply the evolution operation for data property addition over the DWO as given in Table 5. Now *Age* is added as a data property to the s_customers class of DWO. Domain and Range values for *Age* are added as well. As the new change has not introduced any inconsistency we update the mapping. Figure 6 shows the resultant mapping between DSO and DWO after change propagation. The ETL designer can now update the ETL operations. If no further changes are introduced the DWO can be saved as new ontological DW Schema version.

![Figure 6. Mapping between DSO and DWO after Change Propagation](image)

**E. Physical DW schema construction**

The final phase involves in construction of physical DW schema as a separate version. Once changes are applied to the DWO, the OWL structure can be transformed into an equivalent relational model [1]. For the automatic construction of DW in the underlying relational database from DWO we use the following steps:
1. A table is created for every class in the ontology, with the same name as the OWL class.
2. An attribute representing the primary key is added into this table.
3. For all data type properties the domain class and the range class are obtained.
   i. An attribute is created with the same name as the data type property.
   ii. The attribute is added to the table which is mapped to the domain class, and the attribute data type is obtained from the range of the data type property.
4. For an object type property,
   i. A foreign key relationship is created between the table that maps the domain class and the table that maps the range class.
   ii. An attribute is added to the table that maps the domain class and the foreign key constraint is added to this attribute as well.
5. Attributes are added to the tables that map OWL subclass, through an inheritance statement referring to their parent table.

The new version of DW schema can be maintained using an existing schema version management approach. Using updated ETL operations the new version can be populated from the underlying data source.

V. DISCUSSIONS

Within the DWH context, most of the DW schema evolution techniques mainly focus on changes at the code or implementation level [9] [7] [18] [3]. Handling schema evolution at the physical level increases the maintenance cost. Moreover, in existing approaches the relations between requirements, multidimensional design and underlying data sources were not focused. Hence they do not provide a method of impact analysis of requirements and data source evolution over the DW schema. In this paper, we summarized the research effort aiming at supporting graceful evolution of data warehouse schema at the conceptual level using ontology. This helps to solve the existing issues in this context. As currently the proposed work is under implementation results validating the proposal would be the immediate future work.

VI. CONCLUSION

Challenging issues for DW solutions arise due to the dynamicity of the information sources to be integrated as well as changing business requirements. The existing works on DW evolution such as schema versioning and schema evolution mainly concentrate on changing the schema structure at the physical level. In this paper, we have proposed an ontology based approach to handle evolution of the DW schema at the conceptual level. The ontological representation of the data source, requirements and DW helps us to provide automation (semi-automation) of evolution task. Here we make use of the mapping and evolution operators to propagate the changes over DW schema. We present the steps for propagating data source and DW changes. Finally we derive the updated version of the DW physical schema from the ontological version.

Future work includes handling ETL evolution along with DW schema evolution.

REFERENCES


