Benitez Guerrero, Edgard; Collet, Christine; Adiba, Michel
The WHES approach to data warehouse evolution
e-Gnosis, núm. 2, 2004, p. 0
Universidad de Guadalajara
Guadalajara, México

Available in: http://www.redalyc.org/articulo.oa?id=73000211
THE WHES APPROACH TO DATA WAREHOUSE EVOLUTION

El enfoque WHES en la evolución de los depósitos de datos

Edgard Benitez-Guerrero¹,², Christine Collet¹, Michel Adiba¹

ebenitez@lania.mx / Christine.Collet@imag.fr / Michel.Adiba@imag.fr

Recibido: noviembre 12, 2003 / Aceptado: enero 30, 2004 / Publicado: febrero 18, 2004

Abstract. This paper describes the WHES system that enables the creation and evolution of data warehouses. WHES supports a data warehouse evolution model and its associated language MDL. Using a multidimensional approach, this allows one to define and modify the warehouse schema (even in the presence of data) independently of the database model used to implement it. WHES also provides tools to populate the warehouse with data extracted from multiple heterogeneous and autonomous sources. WHES is implemented on top of a relational DBMS as a set of interconnected JavaBeans components manipulating XML data.

Keywords: Computer Science, WHES system, data warehouses, language MDL.

Resumen: El artículo describe el sistema WHES, que permite la creación y evolución de depósitos de datos. El sistema WHES soporta un modelo de evolución de depósito de datos y de su lenguaje asociado: el MDL. La utilización de un enfoque multidimensional permite definir y modificar el esquema de depósito (incluso con la presencia de datos) independientemente del modelo de base de datos utilizado para implementarlo. El WHES provee asimismo de herramientas para poblar el depósito con datos extraídos de múltiples fuentes autónomas y heterogéneas. El WHES es implementado sobre un DBMS relacional como un grupo de componentes JavaBeans interconectados manipulando datos XML.

Palabras clave: Computación, sistema WHES, depósitos de datos, lenguaje MDL.

Introduction

A data warehouse (DW) [12] is a collection of historical data, built by gathering and integrating data from several sources, which supports decision-making processes. On-Line Analytical Processing (OLAP) [7] applications provide users with a multidimensional view of the DW and with the tools to manipulate it. Measures of interest (e.g. sales) are examined according to different axes (e.g. product, store and day of sale). Axes are part of dimensions that organize them into a hierarchy. The Time dimension, for instance, is organized as a hierarchy involving days at the lower level, and months and years at higher levels. In this context, querying is aimed to aggregate measures at various granularities. DWs are often implemented using multidimensional or relational databases. A relational implementation typically employs star schemas (or variations thereof), where a fact table containing the measures references a set of dimension tables.

A common assumption is that, once created, the schema of a DW is not (physically or logically) modified. However, it needs to evolve over time, and this for several reasons [19]: (i) it is often difficult to design a schema in a single phase, making necessary to adopt an incremental approach, (ii) periodical revisions are needed to complete the design or to correct errors, and (iii) new user requirements arise. If these modifications are not integrated, the DW becomes incomplete or obsolete and then users’ information needs are not satisfied anymore.
Existing solutions to the schema evolution problem proposed for traditional relational and object-oriented databases ([1, 15, 14], for instance) are only partially suited for the DW context. This is because they do not consider specific DW characteristics, such as the multidimensional nature of data. As consequence, the data warehouse administrator (DWA) is responsible of ensuring that a schema modification reflects the intended semantics. In addition, there is no standard way for DWAs to express their schema evolution requirements. Finally, after modifying a warehouse schema, the DWA needs to check the impact of the modification on warehouse internals (aggregated data and indexes), on the refreshing infrastructure, and on existing OLAP applications. There is currently a lack of conceptual tools (models, languages), software infrastructures and mechanisms to gracefully evolve data warehouses. Our work [3, 2] is aimed to fill this gap.

This paper describes the WareHouse Evolution System (WHES) prototype, which enables the creation and evolution of data warehouses. WHES supports a data warehouse evolution model and its associated Multidimensional Data definition Language (MDL). Using a multidimensional approach, this allows one to define and modify the warehouse schema independently of the database model used to implement it. WHES also provides tools to populate the warehouse with data extracted from multiple heterogeneous and autonomous sources.

The remainder of this paper is organized as follows. Section 2 introduces the warehouse evolution model and the MDL language. Section 3 presents the architecture of WHES and its main functions. Section 4 discusses related work. Finally, Section 5 concludes and introduces future research.

Warehouse Evolution Model and Language

This section describes our warehouse evolution model (a multidimensional data model and a set of related schema evolution operators) and the MDL language.

Multidimensional Data Model

Several multidimensional data models have been proposed (see [17] for a survey). The model presented here is an adaptation of [6]. Its concepts are dimension and cube, and for each of them, the terms schema and instance are defined.

![Figure 1](image-url). (a) Schema of the Product dimension and (b) schema of the Sales cube
A dimension schema has a name and a set of levels. They represent data domains at different granularities and are organized into a hierarchy by a rollup relation. Associated to a level, there is a (possibly empty) set of properties. A dimension instance (or simply, a dimension) is a set of rollup functions relating values (members) of different levels. A rollup function associates a member \( e \) of a level \( l \) with a member \( e' \) of a level \( l' \) such that \( l \) rolls up to \( l' \) in the schema of the dimension. Figure 1(a) shows the schema of the Product dimension which has three levels: the product’s \textit{code}, the product’s \textit{category} and \textit{department}. Associated with the code level, one can find as property the product \textit{name}.

A cube schema has a name, a set of axes (where an \textit{axis} is a dimension level) and a set of measures. Figure 1(b) shows the schema of the Sales cube. Its axes are \textit{code} (level of the Product dimension) and \textit{day} (level of the Time dimension), and its measure is \textit{quantity}. A cube instance (or simply, a cube) is a set of cells (tuples) associating a collection of members, each one belonging to an axis of the cube schema, with one or more measure values.

The collection of cubes and dimensions forms a multidimensional database. The schema of this database is composed by the collection of schemas of its dimensions and cubes.

**Evolution Operators**

We propose 16 operators to modify multidimensional schemas (see Table 1 and Table 2). Each of them takes as input and produces as output a multidimensional database (see [2] for details).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDimension</td>
<td>Creates a dimension</td>
</tr>
<tr>
<td>DropDimension</td>
<td>Removes a dimension</td>
</tr>
<tr>
<td>RenameDimension</td>
<td>Renames a dimension</td>
</tr>
<tr>
<td>RenameLevel</td>
<td>Renames a level</td>
</tr>
<tr>
<td>AddLevel</td>
<td>Adds a level</td>
</tr>
<tr>
<td>DeleteLevel</td>
<td>Removes a level</td>
</tr>
<tr>
<td>AddProperty</td>
<td>Adds a property</td>
</tr>
<tr>
<td>DeleteProperty</td>
<td>Removes a property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCube</td>
<td>Creates a cube</td>
</tr>
<tr>
<td>DropCube</td>
<td>Removes a cube</td>
</tr>
<tr>
<td>RenameCube</td>
<td>Renames a cube</td>
</tr>
<tr>
<td>RenameMeasure</td>
<td>Renames a measure</td>
</tr>
<tr>
<td>AddAxis</td>
<td>Adds an axis</td>
</tr>
<tr>
<td>DeleteAxis</td>
<td>Removes an axis</td>
</tr>
<tr>
<td>AddMeasure</td>
<td>Adds a measure</td>
</tr>
<tr>
<td>DeleteMeasure</td>
<td>Removes a measure</td>
</tr>
</tbody>
</table>

Evolution operators for dimensions include CreateDimension and DropDimension. RenameDimension and RenameLevel change, respectively, the name of a dimension and the name of a level. AddProperty adds a property \( p \) to a level \( l \), while DeleteProperty removes the property \( p \) from the level \( l \). AddLevel adds a level \( l \) to which an existing level \( l' \) rolls-up. Figure 2(a) shows the addition of the \textit{brand} level to the Product dimension (note that now the \textit{code} level rolls-up to this level). Finally, DeleteLevel removes a level \( l \) and its properties from a dimension.
To express cube evolution, we have CreateCube and DropCube. RenameCube and RenameMeasure change the name of a cube and the name of a measure, respectively. AddMeasure adds a new measure \( m \) to a cube \( c \), while DeleteMeasure removes the measure \( m \) from a cube \( c \). AddAxis adds an axis to a cube. For instance, one can add the city level (of a Store dimension) as an axis of the Sales cube (see Figure 2(b)). DeleteAxis eliminates an axis from a cube.

**The MDL language**

MDL provides expressions to define and modify multidimensional schemas. Its main aspects are described in the following. [2] gives a complete description.

![Figure 3](image-url) MDL expressions describing (a) the schema of the Product dimension, (b) the schema of the Sales cube, and their respective modifications ((c) and (d))

The expression create dimension allows one to define: (i) the name of the dimension, (ii) each level (clause level) and its properties (clause property) and (iii) the roll-ups (clause rollup) between the levels. Figure 3(a) shows the expression to create the Product dimension. The expression create cube allows one to define: (i) the name of the cube, (ii) its axes (clause axis) and (iii) its measures (clause measure). Figure 3(b) shows the expression to create the Sales cube.

MDL provides expressions to drop obsolete dimensions and to modify existing ones. Figure 3(c) shows the expression to add the brand level to the schema of the Product dimension. The code level rolls up to this new level. The default member of the brand level is ”ABC”. MDL also provides expressions to drop obsolete cubes and to modify existing ones. Figure 3(d) shows the expression to add the city axis to the Sales cube. The default member of this axis is ”Grenoble”.

© 2004, e-Gnosis [online], Vol. 2, Art. 11

---

**Figure 2.** (a) Schema of the Product dimension before and after adding the brand level. (b) Schema of the cube Sales before and after adding the city axis.
The WHES System

To validate our warehouse evolution model and language, we prototyped the WHES system. It ensures the creation and the evolution of a DW, both at schema and instance levels. It also provides tools to populate the DW with data extracted from several autonomous and heterogeneous sources. WHES is implemented on top of a relational DBMS as a set of JavaBeans components manipulating XML [18] data.

Architecture

Figure 4 shows the overall WHES architecture. The Administration Tool verifies the validity of the MDL expressions given as input by the DWA and enables him/her to manually initiate the DW population process. The Coordinator performs several tasks: (a) it processes the requests for the initial creation of the schemas of cubes and dimensions, which are stored in a local Metadata Repository, (b) it handles the requests for the modification of existing cubes and dimensions, and (c) it controls the DW population process.

The Warehouse Wrapper translates the multidimensional schema into a relational one (which is then created by the underlying DBMS) and multidimensional data into a relational format to load it into the warehouse. The Integrator creates cubes and dimensions (with data extracted from the sources) which are then stored in a Multidimensional Cache. Once created, dimensions and cubes are exported to the warehouse by the Warehouse Wrapper. The Integrator interacts with an XML Data Mediation System (formed by a mediator and a set of wrappers, each of them associated to a source) to extract source data. Due to space limitations, this paper focuses on MDL expression processing. The details of the warehouse population process can be found in [2].

---

1 This system enables to query (using XML-QL [9]) the individual sources as if they were a single global, virtual XML source.
MDL Expressions Processing

As previously said, the DWA uses MDL expressions to describe a multidimensional schema and then WHES translates it into a relational (star) one. For that, WHES implements a set of translation rules, corresponding to a simple, one-to-one mapping between the concepts of the two data models. For example, the translation of the schema of the Product dimension gives as result the relation schema Product[id_code,code,category,department,name], where id_code is an artificial key and the other attributes correspond to a level or to a property of the dimension. Translation rules for cubes are similar.

The DWA also uses MDL to express his/her needs concerning schema evolution at the multidimensional level and then WHES propagates the changes to the relational level. For that WHES implements a set of propagation rules stating that if the multidimensional schema is modified then the relational schema has to be modified accordingly, and that its associated data has to be adapted as well. For example, if one adds the brand level to the Product dimension, the attribute brand has to be added to the schema of the Product relation which needs to be recomputed as ε brand='ABC' (Product)*. See [2] for details.

Related Work

[10, 11] propose operators to modify and update dimensions. We took their operators to add and delete levels as the starting point of our work, and extended this study to consider cube evolution. Our proposal is similar to [4, 5] that also propose a set of schema evolution operators. The difference with ours is their “granularity”: theirs can be considered as “low level” operators, in the sense that each of them adds or removes specific elements (a level, a property, a measure, etc.) of a schema. The problem is that schema consistency may be lost between two successive evolutions. Indeed, one can create isolated levels and attributes, cubes without measures, and cyclic roll-up relationships. To avoid this, we designed “high level” operators ensuring schema consistency between evolutions.

MDL has syntactic similarities with other languages. SQL is extended in [16] with expressions to create, delete or modify “dimension schemas”, which informs the Oracle 9i DBMS that a set of relations must be considered as being a single dimension. The MDX language [13] offers the create cube command to define cube schemas. Note that, even if MDL is syntactically similar to these languages, the intended usage is different. The ”dimension schemas” of [16] are used to rewrite a query in terms of existing materialized views. In contrast, MDL is a “meta” data evolution language, used by DWAs to communicate with WHES.

* ε z=v(R) [8] introduces a new attribute z into the schema of the relation R. Each tuple t of R is extended with a new value v, taken by the attribute z in t.
Conclusion

This paper described the WHES system that enables the creation and evolution of data warehouses. WHES supports a data warehouse evolution model and its associated evolution language MDL. Using a multidimensional approach, this allows one to define and modify the warehouse schema (even in the presence of data) independently of the database model used to implement it. A limitation of WHES is that it physically modifies the multidimensional schema and the relational one. Thus, historical data can be lost, disturbing existing OLAP applications. Our future work includes exploring other schema evolution approaches, such as the version-based approach [14], that might avoid these disadvantages.

Acknowledgements

The authors wish to thank the anonymous referees for their useful comments. Mr. Benitez-Guerrero was supported by the Ph.D. Scholarship Program of the CONACYT of the Mexican Government.

References