Asian Journal of Information Technology 4 (10): 910-917, 2005

CEV-DW: Creation and Evolution of Versions in Data Warehouse

Shahzad, M.K., J. A. Nasir and M. A. Pasha
Intelligent Information Systems Research Group (RG-IIS)
Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan

Abstract: This study formalizes schema and Version Evolution Operations (VEO), creation and management of multiple versions capable of simulating various alternative business scenarios. Also versioning-algebra (V-algebra) to handle VEO's is also proposed in this study. It is expected that system developed based on proposed solution will not only keep multiple versions but also provide easy way of investigating various versions.

Key words: Data warehouse, multidimensional structures, content changes, schema changes

INTRODUCTION

In today's knowledge oriented society, success of organizations depend upon the degree of correctness in strategic planning and decision-making. Traditional databases are inapt for analysis since their schema is optimized for transactional purpose. Data Warehouse (DW) emerged as an alternative technology but the static structure of its schema puts serious limitations to support any change in its source[1].

The structure and contents of DW reflects real world data stored in DW from source/s and facilitate users to drive analytical results for decision-making. Nevertheless any activity that produces changes to this source/s can generate inconsistent analytical results[1]. Even the diminutive changes may need either evolution or versioning of schema. It is observed that OLTPs are usually independent of each other in their architecture and do not dependent on DW, whereas data warehouse with static schema depends on Transactional Sources (TS), so changes to TS affect DW. Pasha *et al.*[1] has categorized such changes into two types:

Content changes: Insertion /updation/ deletion of records occurred as a result of DML commands on database.

Schema changes: Addition/modification/ dropping of attribute occurred as a result of DDL commands on database.

Several schemas have been developed based on Star-oriented approach but are failed to handle such changes like Star, Star-flake, Snowflake[2] and Semi-Star[1]. Semi-Star has reduced the overhead of schema up-gradations, yet it has not been eliminated completely.

The aim of the study is to formalize and support the concept of alternative versioning and to realize the importance of the concept of simulating business scenarios. This study presents a set of Evolutionary Operations (EO), which results in either evolution or versioning of DW. For handling system dynamics we have supported evolutionary or hierarchical versioning of DS by V-algebra. Changes into the schema are applied to a new version of schema named Child Version explicitly derived from the pervious version, called Parent Version.

To simulate various business alternatives the notion of alternative versioning is used. For this, two types of versions have been used real version and alternative version. Both these version are valid depending upon their valid time called valid-time of real versions and valid-time of alternative versions.

For prototyping, solution is implemented and tested using SQL Sever, Analysis Services and Visual Basic 6.0. For transferring data to fact table DTS package of SQL server 2000 is used. Various alternative business scenarios have also been simulated for testing the flexibility of framework.

ILLUSTRATIVE EXAMPLE

To spell out the problem more effectively let us consider an example of company's sales DW. The company has sale points in multiple cities and the sales are inspected in various locations at certain time. Cities are grouped into administrative regions and products sold are grouped into Categories. The DS of company's data warehouse is given in Fig.1; composed of two types of tables: Fact Tables (FT) and Dimension Tables (DT), which surround the fact table. In the Fig.1 Sales_Fact is the only fact table, surrounded by location, time and product dimensions. The location dimension is composed of two levels: city and region.

Corresponding Author: Shahzad, M.K., Intelligent Information Systems Research Group (RG-IIS), Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan

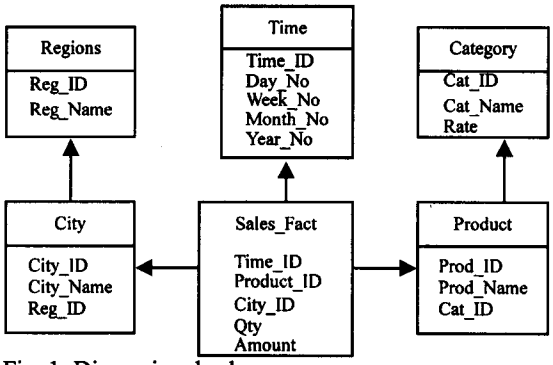

Fig. 1: Dimensional schema

The product dimension also has two levels, Category and Product. Sales_fact table stores the information about quantity of product sold in Qty attribute and the total amount against the sold product in Amt attribute.

Amt is calculated as the product of Rate and Qty of the product. i.e.

Amt = (Sales_Fact . Qty) * (Category . Rate)

Let us further assume that the relevant tables store the following data along with the queries executed to retrieve the results.

Select * from product;

Prod_ID	Prod_Name	Cat_ID
1	P_1	Category1
2	P_2	Category1
3	P_3	Category2

Select * from category;

Cat_ID	Cat_Name	Rate
1	Category1	10
2	Category2	100
3	Category3	1000

Select * from region;

Region_ID	Region_Name
1	Region A
2	Region B

Select * from city;

City_ID	City_Name	Region_ID
1	BWP	1
2	ISL	2
3	KHI	1

Select * from Sales_Fact;

Time_ID	Prod_ID	City_ID	Qty	Amt
1	1	1	65	650
1	2	1	90	900
1	3	2	32	3200
1	1	3	20	2000

For this schema multiple analytical queries can be written, on content and schema changes some of them give consistent results while other does not as explained in the next section.

PROBLEM

DW describes real world that is likely to change with schema and content changes. This example shows problems related to storage, retrieval and proper what-if analysis along with simulating various business scenarios. The problems are:

Firstly, conventional DW schema does not support business dynamics, i.e. multiple data values can be inserted which results in derivation of inconsistent results. e.g. changing the borders of regions may result in cities being moved from one region to another, resulting in inconsistent results.

Assume that boundaries of region are changed in such a way that city KHI is moved from Region A to Region B.

Let us assume a query-computing amount earned in every region before changing the boundaries of region is:

Region_Name	Sum (Amount)
Region A	3550
Region B	3200

And the total amount earned in every region after changing the boundary of region is:

Region_Name	Sum (Amount)
Region A	1550
Region B	5200

Although no change to the database is being made except that region boundary, but inconsistent results are generated. Similar inconsistent results are generated if

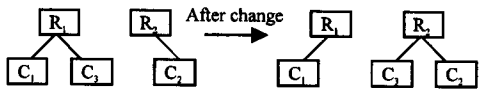

Fig. 2: Changing region of city

region name is changed, city name is changed, product category is changed, product name is changed, and name of product category is changed (Fig. 2).

Secondly, traditional DW schemas do not support any simulating business scenarios. For example assuming that a certain percent of amount earned by a product in a city feeds the total amount, one may investigate: how the total amount of one category would change; if the product category is changed.

Thirdly, adding even a single new attribute in one of the source may require adding this attribute to the DW schema, if one would like to view affect of such attribute in the DW. Also it is possible that this attribute contributes to the calculation of facts, convectional DW schema cannot handle these changes.

In order to handle this kind of changes as well as process and analyze data properly we need a new version of a DW. Before presenting a method to handle mentioned problems, to analyze data properly, evolution of DW schema, creation and management of versions of DW and V-algebra for evolution and versioning, the limitations of the existing approaches are discussed in the next section.

EXISTING APPROACHES

The attempts made to handle changes can be classified into three categories: i) Schema evolution ii) Versions extension iii) Semi-Star Schema. The former approach proposed by Sjoberg *et al.*[3], supports only one MD schema and instance. When change is applied to a schema, it is required to upgrade MD schema, transformation package and multidimensional structures.

In the approach of second category[2], changes are time stamped in order to create temporal versions. However, Golfarelli *et al.*[4] expose their inability to express and process queries that span or compare several temporal versions of data. On contrary, the model and prototype of a temporal DW presented by Blaschaka *et al.*[5] support queries for a particular temporal version of DW or queries that span several version.

Finally, Semi-Star Schema is very suitable for medium dynamic organizations[6]. It has two types of tables to meet the two dimensional requirements, all this is done by sharing the transactional tables for analytical purpose. Due to presence of tera-bytes of data this schema is not suitable for large sized organization.

DIMENSIONAL SCHEMA AND MD-PARADIGMS

MD-Paradigms: For writing V-algebra for schema evolution/versioning, we need formalism that can server as a basis for defining the schema versioning operations and their effects resolution. So, this section contains formal-mathematical notations for MD schema definition. Paradigm defined for MD schema and model of Sjoberg *et al.*[3] has been replaced by different set of attributes to be used for versioning.

Here we assume a universal set U of alphabets to represent versions, its validity time, set of facts, dimension levels, attribute names with granularity level. So paradigm for it can be defined as:

Definition 1: Versioned MD-schema: Each version of MD-schema ($\aleph$) has seven attributes, defined as:

$$\aleph < V_{x,y}, VT_m, F_k, L_j, A_t, \text{gran}, \text{attr} >$$

where,

- $V_{x,y} \subset U$ is finite set of versions of DW schema $\{V_{0,1}, V_{1,1} V_{2,1} \ldots V_{m,n}\}$ where, $V_{x,y} \in U$ for $0 \le x \ge m$ and $0 \le y \ge n$
- VT_m is valid time of real and alternative version calculated by function $\wp$ i.e. $f(VT_m) \rightarrow VT_p \{ VT_r, VT_a \}$ $g(VT_p) \rightarrow S \{0, 1\}$ $\wp(Vt_m) : g \circ f(VT_m) \rightarrow X \{0,1\}$
- $F_k \subset U$ is finite set of fact names $\{F1, F_2, \ldots F_k\}$ where, $F_k \in U$ for $0 \le k \ge m$
- $L_j \subset U$ is finite set of dimension level names $\{L_1, L_2, \ldots\ldots L_n\}$ where, $L_j \in U$ for $0 \le j \ge n$
- $A_t \subset U$ is finite set of attribute names $\{A_1, A_2, \ldots\ldots A_n\}$ where, $A_t \in U$ for $0 \le t \ge n$
- gran: $F \rightarrow 2^L$ is a function that associates a fact with a set of dimensional level names. These dimension levels gran (f) are called the base level of fact f.
- attr: $A \rightarrow F \cup L \cup \{\perp\}$ is a function mapping an attribute either to a fact, to a dimension level or a special $\perp$ symbol which means that this attribute is not connected alt all.

Basic definitions of functions

Definition 1: Versioning function: A function which takes MD-schema as input argument and produces new one due to operation $Oper_a$ on $\aleph_o$. This operation results in $\aleph_n$ MD-schema, mathematically:

$$\aleph_n : Oper_a (\aleph_o)$$

Versioning function either makes changes to same version or creation of new version i.e.

$$J(V_{x,y}) \circledR \begin{cases} V_{evol} & \text{if MD-schema evolutes} \\ V_{vers} = V \cup V_{new} & \text{else schema is versioned} \end{cases}$$

$\aleph_{evol}$ indicates schema evolution, i.e. changes are made to existing schema without maintaining new version.

$\aleph_{vers}$ indicates new version of schema is created and maintained.

Versioning-function is executed as a result of set of formal-operations defined in next section. As versioning function is applied to MD-schema, which results in either evolution of schema or creation of new version. If schema evolutes then Vx,y i.e. version identifier is not changed, but other values may change depending upon change made. Else if new version is created, Vx,y along with other attributes.

Definition 2: Reviving function: A function which takes MD-schema version identifier as input argument and concludes it depending reviving function i.e. changing the values of x and y defined as:

$$Y(V_{x,y}): \begin{cases} \text{if } J(V_{x,y}) = V_{vers} \\ \quad x = x \\ \quad \text{and} \\ \quad y = y \\ \text{Else} \\ \quad x = x.i; \quad \text{for AV} \\ \quad i \text{ is AV identifier} \\ \quad x = x + 1 \quad \text{else} \\ y = \text{parent version identifier} \end{cases}$$

Definition 3: Qualifying function: After taking MD-schema version identifier qualifies it for further use, by changing value of VT_m attribute of identifier is defined as:

$$\rho(Vtm) : g \circ f(VTm) \rightarrow X \{0,1\}$$
$$f(VTm) \rightarrow VTp \{ VTr, VTa \}$$
$$g(VTp) \rightarrow S \{0, 1\}$$

FORMAL OPERATIONS (VERSIONING OPERATIONS)

Inserting fact table: This operation inserts a new fact table to schema, resulting in new connections.

Deleting fact table: This operation deletes a fact table completely from DW.

Insertion level: It extends schema by inserting new dimension level, while affecting the granularity.

Deletion level: It extends schema by deleting dimension level.

Connecting attribute with dimension level: This operation connects attribute with dimension level.

Disconnecting attribute with dimension level: It operation disconnects attribute from dimension level.

Fact insertion: It concludes increased quantity of facts in MD-schema.

Fact deletion: It concludes in decreasing quantity of facts in MD-schema.

Insertion attribute: Inserts an attribute to dimension.

Deletion attribute: Deletes an attribute from dimension.

Connecting attribute to fact: This operation connects existing attribute to a fact.

Disconnecting attribute to fact: This operation disconnects an existing attribute from a fact.

Inserting dimension: This operation inserts a completely new dimension.

Deleting dimension: This operation deletes dimension completely.

All these functions results in evolution of schema-versioning framework with the abilities of handling DW schema under changes.

EVOLUTION AND VERSIONING IN MDS

For evolution method changes to the schema is applied to same schema, with out maintaining the older one. While, according to versioning approach changes to a schema are applied to a new version. This new version of schema named child version is explicitly derived from pervious version called parent version.

For what-if analysis, a decision-maker simulates changes in the real world to create virtual possible scenarios. Alternative versioning can implement these simulations. To handle such scenarios versions have been categorized versions into two different types[7]: Real Version (RV) to handle changes made to source, while Alternative Version (AV) handle simulated changes made by a user to apply what-if analysis.

To reveal schema-versioning status a graphical method is used. The graph portrays all types of versions and their relationship.

Real version: A multi-version data warehouse is composed of a set of parent child versions. Every version of a MV-DW is in turn composed of a schema version and an instance version. The latter stores the set of data consistent with its schema version.

Real case

Sum (Amount)	City_Name	Sum (Amount)
6750	KHI	2000
	BWP	1550
	ISL	3200

Alternative version: DW may also be used for simulating various operation/business scenarios. For example, assuming that a certain percent of amount earned by a product in a city feeds the total amount, one may investigate how the total amount of one category would increase if they move a product from one to another category.

Let us assume a simulation scenario P2 product was moved from category A to B. This change must be applied to the structure of product dimension; consisting in assigning product to a new category i.e. a category foreign key update is required.

Also, one may assume an increase in amount by changing the category of a product. In such a simulating scenario, a new version of fact data will also be created from the previous version. The changes to fact data require computing new values of amount for every affected record in Sales Fact, as the rate of product is changed from 10 to 100, the amount will also change, as it is the product of quantity and rate. In our example, only one record is affected, as it describes sales of P_2. More precisely, the value of amount must be increased according to a new value defined in Category B. Thus, we need a kind of conversion function that would compute new values of facts based on original data.

Having created a new simulation version of a DW, and having converted the fact data, a decision maker may compare the real situation with a hypothetical situation. In both cases the total sum of fines as well as the sum of fines paid in each city is computed, as shown below.

Simulated case

Sum (Amount)	City_Name	Sum (Amount)
9000	KHI	2000
	BWP	3800
	ISL	3200

In the simulation case city BWP would substantially increase the amount. As we have shown in the above examples, data warehouse versions are useful for handling changes in the real world as well as for simulating various business scenarios. Real data warehouse versions are applied in the first case, whereas alternative data warehouse versions are applied in the second case.

Schema versioning graph: To reveal Schema-Versioning status a graphical method with the name Schema-Versioning Graph (SVG) is used. This graph not only portrays RV and AV along with Parent and Child versions but also gives Version-Derivation Relationship (VDR). VDR shows versioning relationship between multiple versions i.e. parent- child, real-alternative and alternative-alternative. In short, SVF keeps track of multiple versions, linearly and alternatively and proposed SVG has the potential to reveal such relationship with their validity time. Validity of versions is dependent on validity period, which is maintained for both types called Valid-Time of real versions (VTr) and valid-Time of Alternative Versions (VTa).

SVG is a two-dimensional graph VTr on horizontal side, while VTa is on vertical side. Each version is shown by a unique number of type Vx,y, where:

x is the counter of the version, which finds an increment of one as soon as new version is maintained,
y represents the x component of its parent version

Real versions are increased horizontally, while Alternative versions increase vertically. Real and Alternative versions are distinguished by the values of x, any value which is after point is the no of the alternative version, other wise it is real version.

Before drawing SVG it is required to give the constructs of this graph.

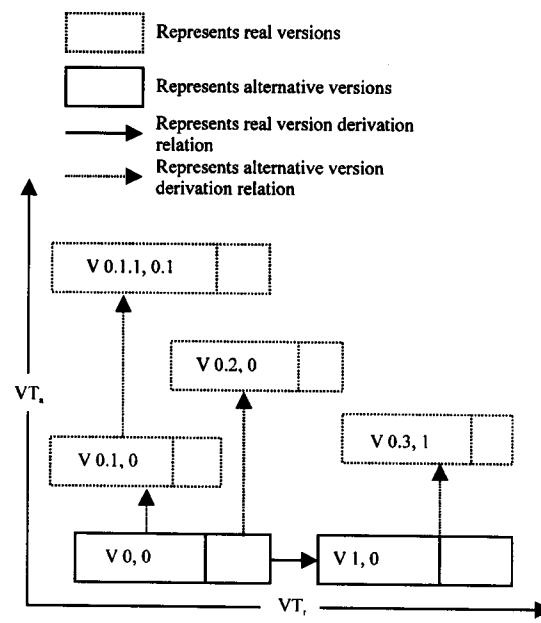

Fig. 2: Schema-versioning graph

The graph shows all the types of versions. V0 is initial real version of a DW. Since this version has no real and alternative as its parent so it is called zero version. Based on zero version, another version V1 is derived.

V-ALGEBRA FOR MULTIPLE VERSIONS

In order to be able to manage problem-creating changes a model for DW has been developed. In this section algebra for is proposed for each problem-creating operation.

Level insertion: It extends schema by inserting new dimension level. The operation increases the set of levels with an isolated element. All this results in changed parameters of MD-Schema.
Set before change

$$\aleph_o < V_{x,y}, VT_m, F_k, L_j, A_t, gran, attr >$$

Set is change due to operation $Oper_{li}$

$$\aleph_n : Oper_{li} (\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}', VT_m', F_k, L_j', A_t, gran', attr' >$$

Where,

$V_{x,y}' : \eth (V_{x,y})$

$$J (V_{x,y}) \circledR \begin{cases} V_{evol} & \text{if MD-schema evolutes} \\ V_{vers} & \text{else schema is versioned} \end{cases}$$

$L_j' : L \cup \{l_{new}\}$
$gran' : gran (f) ; gran (f) : F \rightarrow 2^{L'}$
$Vt_m' : \wp(VT_m)$;
$\wp(Vtm) : g \circ f (VTm) \rightarrow X \{0,1\}$
$f (VTm) \rightarrow VTp \{ VTr , VTa \}$
$g (VTp) \rightarrow S \{0, 1\}$
$attr' : attr (a) ; attr (a) : A \rightarrow F \cup L' \cup \{\perp\}$

In all coming points set before change, changes to version i.e. $V_{x,y}$ and its validity time VT_m will change in the same way as this one.

Level deletion: It extends schema by deleting dimension level. Mathematically, operation removes an element from set of levels. All this results in changes in parameters of MD-Schema.
Set is change due to operation $Oper_{ld}$

$$\aleph_n : Oper_{ld} (\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}', VT_m', F_k, L_j', A_t, gran', attr' >$$

where,

$L_j' : L \cup \{l_{del}\}$
$gran' : gran (f) ; gran (f) : F \rightarrow 2^{L'}$
$attr' : attr (a) ; attr (a) : A \rightarrow F \cup L' \cup \{\perp\}$

Insert attribute: Creates a new attribute without attaching it to a dimension level or fact, as assigning an existing attribute to dimension level or fact is a separate operation i.e. attributed connection.
Set is change due to operation $Oper_{ia}$

$$\aleph_n : Oper_{ia} (\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}', VT_m', F_k, L_j, A_t', gran, attr' >$$

where,

$A_t' : A \cup \{a_{new}\}$
$attr' : attr (a) ; attr (a) : A \rightarrow F \cup L' \cup \{\perp\}$

Delete attribute: Creates a new attribute without attaching it to a dimension level or fact, as assigning an existing attribute to dimension level or fact is a separate operation i.e. attributed connection. Set before change is $\aleph_o$ as shown above.
Set is change due to operation $Oper_{da}$

$$\aleph_n : Oper_{da} (\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}', VT_m', F_k, L_j, A_t', gran, attr' >$$

where,

$A_t' : A \cup \{a_{new}\}$
$attr' : attr (a) ; attr (a) : A \rightarrow F \cup L' \cup \{\perp\}$

Connect attribute to dimension level: Connects an existing attribute a_{new} to a dimension level $l \in L$ and $a_{new} \in A$.
Set is change due to operation $Oper_{cdl}$

$$\aleph_n : Oper_{cdl} (\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}', VT_m', F_k, L_j, A_t, gran, attr >$$

$$attr': \begin{cases} l & \text{if } a = a_{new} \\ attr(a) & \text{else} \end{cases}$$

$$attr(a): A \to F \cup L \cup \{\perp\}$$

Disconnect attribute to dimension level: Disconnects an existing attribute a_{new} from dimension level $l \in L$.
Set is change due to operation $Oper_{dcdl}$

$$\aleph_n: Oper_{dcdl}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, Vt_m, F_k, L_j, A_t, gran, attr >$$

$$attr: \begin{cases} \wedge & \text{if } a = a_{new} \\ attr(a) & \text{else} \end{cases}$$

$$attr(a): A \to F \cup L \cup \{\perp\}$$

Connect attribute to fact: Connects an existing attribute a_{new} to fact $f \in F$.
Set is change due to operation $Oper_{caf}$

$$\aleph_n: Oper_{caf}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, VT_m, F_k, L_j, A_t, gran, attr >$$

where,

$$attr: \begin{cases} f & \text{if } a = a_{new} \\ attr(a) & \text{else} \end{cases}$$

$$attr(a): A \to F \cup L \cup \{\perp\}$$

Disconnect attribute from fact: Disconnects an existing attribute a_{new} to fact $f \in F$.
Set is change due to operation $Oper_{dcaf}$

$$\aleph_n: Oper_{dcaf}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, VT_m, F_k, L_j, A_t, gran, attr >$$

where,

$$attr: \begin{cases} \wedge & \text{if } a = a_{new} \\ attr(a) & \text{else} \end{cases}$$

$$attr(a): A \to F \cup L \cup \{\perp\}$$

Insert fact: This operation extends the MD-model by a new fact. The operation extends the set of facts without attaching dimension levels to this fact Dimensions for this fact have to defined separately.
Set is change due to operation $Oper_{if}$

$$\aleph_n: Oper_{if}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, Vt_m, F_k', L_j, A_t, gran, attr >$$

where,

$$F_k: F_k \cup \{f_{new}\}$$
$$attr: attr(a);\ attr(a): A \to F \cup L \cup \{\perp\}$$

$$gran: \begin{cases} \emptyset & \text{if } f = f_{new} \\ gran(f) & \text{else} \end{cases}$$

$$gran(f): F \to 2^{L'}$$

Delete fact: This operation reduces MD-model by one fact. The fact must not be connected to a dimension ($gran(f_{del}) = \emptyset$) and must not contain any attributes ($attr(a) \neq f_{del}$ for all $a \in A$).
Set is change due to operation $Oper_{df}$

$$\aleph_n: Oper_{df}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, VT_m, F_k, L_j, A_t, gran, attr >$$

where,

$$F_k: F_k \setminus \{f_{del}\}$$
$$attr: attr(a);\ attr(a): A \to F \cup L \cup \{\perp\}$$

$$gran \text{ ®} \begin{cases} \emptyset & \text{if } f = f_{new} \\ gran(f) & \text{else} \end{cases}$$

Insert dimension: Inserts a dimension from a fact.
Set is change due to operation $Oper_{id}$

$$\aleph_n: Oper_{id}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, Vt_m, F_k, L_j, A_t, gran, attr >$$

where,

$$gran: \begin{cases} gran(f) & \text{if } f \neq f_{ins} \\ gran(f) \dot{E} \{l\} & \text{if } f = f_{ins} \end{cases}$$

$$gran(f): F \to 2^{L}$$

Delete dimension: Deletes a dimension from a fact. Set is change due to operation $Oper_{dd}$

$$\aleph_n : Oper_{dd}(\aleph_o)$$

i.e.

$$\aleph_n < V_{x,y}, VT_m, F_k, L_j, A_t, gran, attr >$$

where,

$$gran : \begin{cases} gran(f) & \text{if } f \neq f_{del} \\ gran(f)-\{1\} & \text{if } f = f_{del} \end{cases}$$

$$gran(f): F \rightarrow 2^L$$

CONCLUSIONS

For managing problem-creating changes (originated due to formal operations), evolution/versioning method is proposed with V-algebra. Tool based on it, will not only create and manage multi -versions but also provides a GUI to play with the versions. Similarly, transparent-retrieval of analytical results from multiple versions, documenting multiple versions, data management techniques, versioning in Semi-Star, efficiency of processing are some future research directions.

REFERENCES

1. Pasha, M.A. J.A. Nasir and M.K. Shahzad, 2005. SVF: Schema Versioning Framework for data warehouse. Proceedings of ITAC, Pakistan, 2005.
2. Sapia, C. and G. Hofling, 1999. On schema evolution in multidimensional databases. Proceedings of DaWak99, Italy, 1999.
3. Sjoberg, D., 1993. Quantifying Schema Evolution. Software Technology, 35: 35-54.
4. Golfarelli, M., S. Rizzi and I. Cella, 2004. Beyond data warehousing: What's next in business intelligence? Proceedings of 7th International Workshop on Data Warehousing and OLAP.
5. Blaschaka, M., C. Sapia and G. Hofling, 1999. On schema evolution in multidimensional databases. Proceedings of DaWak99, Italy, 1999.
6. Pasha, M.A., J.A. Nasir and M.K. Shahzad, 2004. Semi-star modeling schema for managing data warehouse consistency. Proceedings of IEEE-NCET, Pakistan, 2004.
7. Bebel, B.E, J. Koncilia, C. Morzy and T. Wrembel, 2004. Creation and management of versions in multiversion data warehouse.