http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (2): 250-259, 2006
ISSN 1812-5638

Transparent Querying Multiple-versions of Data Warehouse

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

Abstract: Data Warehouse (DW) offers static structured multidimensional schemas at logical level, which does not has the ability to manage the changes that the data warehouse schema undergoes over time in response to evolving business requirements. Although multiple versions of data warehouse and use of multi-version query language is suggested for handling such problems, yet managing large number of DW versions makes queries construction more complex and unmanageable task. In this study we have developed a transparent-retrieval-method using query-confinement method for managing multiple versions of a DW.

Key words: Multi-version data warehouse, transparency, querying, confinement

INTRODUCTION

Data Warehouse (DW) integrates autonomous and heterogeneous external data sources to make information available in integrated form for analytical processing, decision-making and data mining applications. A distinctive feature of DWs is that of storing historical data, hence, the objective data-sources may change with the passage of time. These changes can be categorized as[1]:

- 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.

The impact of these changes on DW can be handled in two different ways: (i) Evolution method, (ii) Versioning method. The first approach advocates for updating a schema by transforming data from old data warehouse schema into a new schema. Only the latest version of DW is being maintained[2-5]. Whereas, the second approach advocates for maintaining multiple versions of DW. In this approach versioning can be done implicitly by temporal extension[6,7] or explicit by physically storing different versions of DW.

Schema Versioning Framework (SVF) handles address this issue by applying changes to new version called Child Version which is explicitly derived from pervious version called Parent Version. For what-if analysis, a decision-maker virtually creates possible scenarios by simulating the changes occur in real world. These scenarios are represented by two different types of versions: (i) Real Version (RV) that handles changes made to source; (ii) Alternative Version (AV) that handles simulated changes made by a user to apply what-if analysis[8]. SVF also offers a user-friendly graphical tool for managing various versions of DW (Fig. 1). However, it does not address the core issue related to the retrieval of data from multiple versions of DW.

This study presents a novel approach for handling multiple versions retrieval problem. Our approach is based on transparent retrieval from multi-version of DW. The versioning transparency can be provided at three levels: (i) Source Transparency (ii) Selection Transparency and iii) Projection transparency. In our approach versions are made transparent to users by developing multi-version

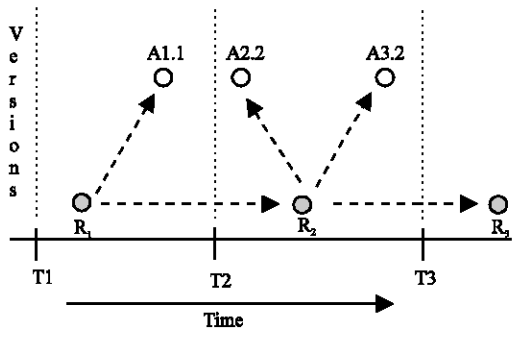

Fig. 1: Schema versioning graph

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

integrated logical schema. The new schema will hide the versions detail from the end user and allows them to write complex queries for a DW with multiple versions. For executing such queries we have proposed transparency query processing layers emphasizing on decomposition method at confinement layer. This study also presents the implementation of synthetic analyzer, which is used for the development of multi-version integrated logical schema and querying transparently multiple versions.

PROBLEM DEFINITION

It is never targeted to maintain multiple DW versions, but consistent and reliable required retrieval for what-if analysis is the real task. In multi-version DW data of interest is usually distributed among several versions and user being unaware of data location may want to query complete DW. Multi-version retrieval problem was addressed by making extensions to conventional query language[9]. This approach is suitable for retrieval from smaller number of versions, but by this approach query-writing becomes very complex with the passage of time due to increased number of versions. In fact it was found approximately impossible to write queries for large number of real and alternative versions. Also, Morzy and Wrembel[9] gives some prototype limitations, which are removed in the study.

To solve these issues it is required to propose and build a conception that can support query writing independent of versions their types and quantity. So the conception of transparency is proposed in this study, with its types and execution layers.

EXISTING APPROACH

The only approach given for retrieval from multiple real and alternative versions makes extension to traditional SQL language[9]. The query language with proposed extensions have functionality of traversing several versions; compare their results and then merging them.

Disadvantage of conventional approach includes the restricted querying scopes in term of versions i.e. only the set of three querying possibilities were given, that are: i) Querying only current DW version ii) Query the set of real DW versions iii) Or querying set of selected alternative DW versions. Also in this approach retrieval from multiple real and alternative has not been made possible.

Some more limitations given by Morzy and Wrembel[9] are: 1) All predicates of the select command apply to all the DW version pointed to in the version from and version in clause. 2) The query parser is unable to infer appropriate version of interested from the where clause. 3) The results of each version is not merged into a single one.

Given disadvantage gives rise to the need for transparency conception. In transparent querying multi-version DW user need not to keep information about multiple DW versions, in spite a superset schema of the entire versions is used for querying multi-version DW.

BASIC DEFINITIONS

These definitions will further be used for defining transparency and query execution phases.

Definition 1-multi-version DBMS: Multi-version DBMS for Multi-version DW Is a software system that permits the creation and management of multi-version data warehouse and provide transparency for retrieval.

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.

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

$\aleph_{evol}$ 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. 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:

$$\psi(V_{x,y}) \begin{cases} \text{If } \vartheta(V_{x,y}) = V_{vers} \\ \quad \begin{cases} x = x \\ \& \\ y = y \end{cases} \\ \text{Else} \\ \quad \begin{cases} x = x.i; & \text{for AV.} \\ i \text{ is AV identifier} \\ x = x+1 & \text{else} \end{cases} \\ y = \text{parent version identifier.} \end{cases}$$

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:

$$\wp(VTm) : g \circ f(VTm) \to X\ \{0,1\}$$
$$f(VTm) \to VTp\ \{VTr, VTa\}$$
$$g(VTp) \to S\ \{0, 1\}$$

TRANSPARENCY AND ITS TYPES

Transparency refers to separation of high-level semantics of a system from lower-level implementation issues. In other words, a transparent system hides the implementation detail from users. Users do not require detail low-level information multi-version retrieval.

The advantage of a fully transparent multi-version DBMS is the high level support that it provides for development of complex multi-version applications. It is obvious that we would like to make all DBMS fully transparent to provide ease to users. Fully transparent access means that the users can pose the query without paying any attention to versions and let the system worry about resolving these issues.

Three types of transparencies can be provided while querying multi-version DW, which are: Source, Analysis and Projection transparency.

Source transparency: Data stored in different versions are transparent to users. i.e. user will retrieve data from multiple versions independent of the existence of source in versions, but it should be present in at least one of the versions. For example, consider two versions, V1, V2 such that table Ti belong to only one version. In this case users need not to keep information about individual versions. In spite of it user will write query independent of the existence of table i.e. retrieve the data of Ti.

Analysis transparency: In analysis transparency user will write query independent of predicate-attribute present in one or more versions, but it should be present in at least one version. For example, consider two versions V1, V2 such that table Ti belong to both versions, at the same time A1 is present in V1.T1 but not in V2.T1. In this case user will write query independent of predicate-attribute presence in versions. So for retrieval user will query like, select from tables with predicate.

Projection transparency: For projection transparency, user should write the query independent of existence attribute existence in one or more versions. For example, consider two versions V1, V2 such that table Ti belong to both versions, at the same time A1 is present in V1.T1 but not in V2.T1. In this case user will write query independent of projection-attribute presence in versions. So for retrieval user will query such that, select attributes from Tables.

Advantages of transparency: In distributed multi-version DW transparency increases many fold advantages, which are:

1. Users do not need to know about existence of multiple versions of DW.
2. Query is written independent of tables present in one or more versions.
3. It also hides information that data resides in specific versions or in all versions.
4. Query writing becomes simple even in the presence of larger number of versions.

Users need not to mention about the attribute-predicate presence or absence in versions.

REFERENCE ARCHITECTURE FOR MULTI-VERSION DWH

Architecture: It includes tools for extracting data from multiple operational databases and external sources; for cleaning, transforming and integrating this data; for loading data into the data warehouse and for periodically refreshing the warehouse to reflect updates at the sources. To handle business dynamics multiple version of DW are required to be maintained. These can be real and alternative.

In addition to the warehouse catalog, MV-Catalog contains MV-ILS and information about various real and alternative versions with formal operations which evolutes versions. Data in the warehouse is stored and managed by one or more servers, which present multidimensional views of data to a variety of front end tools: query tools, report writers, analysis tools and data mining tools etc.

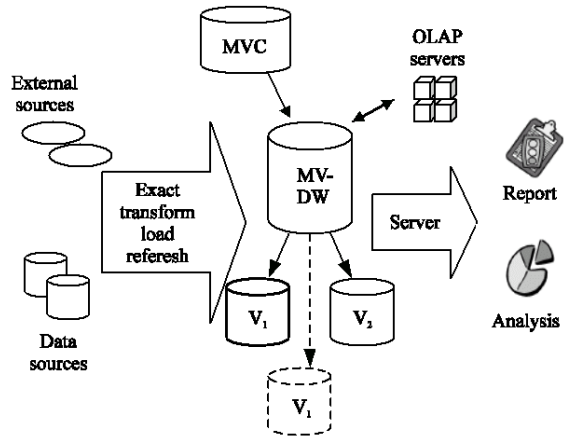

Fig. 2: Multi-version data warehouse architecture

Multi-version integrated logical schema (MV-ILS): As the name indicates, MV-ILS is a logically integrated schema that shows union of all schemas of all versions. In other words, the core of MV-ILS is a user facilitator for querying multiple versions of DW. Using which, instead of keeping the versions information user need to know about MV-ILS. For confinement purpose MV-catalog contains necessary versioning information (that are operations). This information helps in confining global query on multiple versions to local query on individual version. For efficient access of confinement-rules and better processing of confinement phases, these rules are loaded in main memory.

MV-CATALOG

The feature of MV-DBMS is the recognition of an integrated catalog to hold data about the schema versions, rules, applications and so on. The dictionary is expected to be accessible to administrators as well as to the DBMS. A data dictionary or catalog is a repository of information describing the data in the database; it is the data about the data or meta-data. Typically MV Catalog stores:

- Names types and sizes of data items.
- Names of relationships.
- Integrity constraints on data.
- Name of authorized users who have access to data.
- External, conceptual and internal schemas and mapping between schemas.
- Conceptual schemas of each version.
- Multi-version integrated logical schema.
- Metadata of integrated schema.
- Confinement rules, to be used for query confinement
- Heterogeneity between versions.
- Statistics about the objects.

QUERY EXECUTION PHASES

Query processing layers play a vital role in data manipulation for SQL query Ozsu and Patric[10] has given the query processing layers for distributed database, the layers are: Query decomposition, data localization, global optimization, local optimization. These layers are functional for retrieval in distributed databases but are not functional for multiple versions of data sources.

So we have proposed a new layer query confinement for transparent retrieval on multiple versions. The details of confinement layer and process are given here along with all the possible cases. So the layered architecture for transparent query processing (Fig. 3).

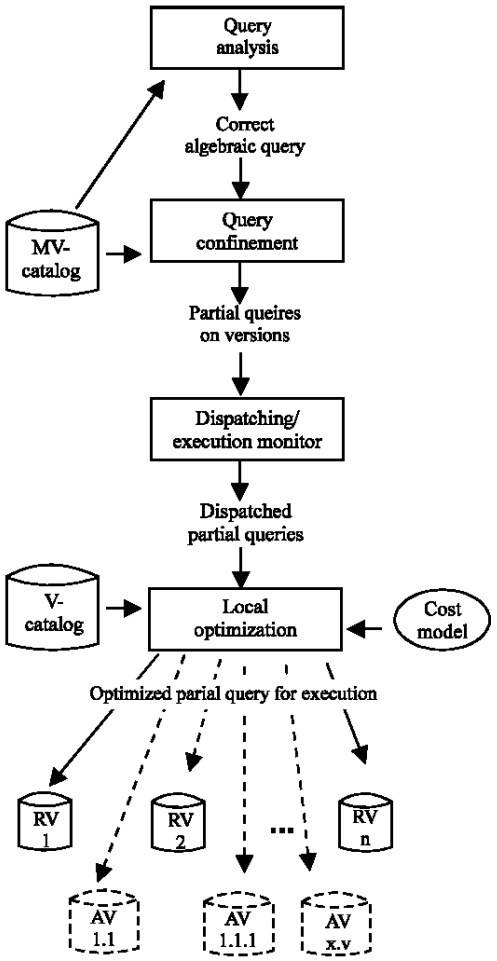

Fig. 3: Transparent query processing phases

QUERY CONFINEMENT LAYER

Query confinement layer transforms the generic SQL query on multiple-versions of data source, into respective partial queries i.e. mono-version queries executable on individual version. MV-catalog will support query-confinement process by maintaining set of information regarding versioning, reviving and version-evolution operations[3]. Dispatcher/Execution-Monitor, dispatches and then monitors mono-version queries to relative real or alternative version for execution purpose.

This study mathematically highlights all cases of confinement process. For simplicity, we consider two real versions V1 & V2 of MV-DW with following set of tables:

V1 = {T1, T2 ... Tn} & V2 = {T1, T2, ... Tm}
(Where n and m are given table number for version V1 and V2.)
The simple schemas of table T1 of version V1 and V2 has following set of attributes:
V1.T1 = {Com, ... M}
V2.T1 = {Com, ... F}
& Com = {A1, A2, ... Ak}
Com is set of common attributes of Table T1 of V1 and V2.

Case 1: Source transparency: Here we elaborate source transparency with complete set of prospects.

i) One version source
Let Ti be any source table where:
Ti $\in$ V1 and Ti $\notin$ V2

Transparent user query for multiple versions
Select * from Ti

Confinement layer will produce following partial queries
Select * from V1.Ti

ii) Two versions Source
Now, let Ti be any source table where:
Ti $\in$ V1 and Ti $\in$ V2

Transparent user query for multiple versions
Select * from Ti

Confinement layer will produce following partial queries
Select * from V1.Ti
$\oplus$
Select * from V2.Ti

In the same way confinement for source transparency can be done for n number of versions.

Case 2: Projection transparency: Here, we elaborate projection transparency with complete set of prospects.

i) Extra attribute in one version
Consider M attribute where
M $\notin$ V1 and M $\in$ V2
Also Com is common set of attributes in version V1 and V2.

Transparent user query for multiple versions
Select Com, M from Ti

Confinement layer will produce following partial queries
Select Com from V1.Ti
$\oplus$
Select Com, M from V2.Ti

Same is the case for attribute F where F belong to V2 but not V1

ii) Distinct attributes in versions
Consider M, F attribute where
M $\notin$ V1 , M $\in$ V2
& F $\notin$ V2 , F $\in$ V1
Also Com is common set of attributes in version V1 and V2.

Transparent user query for multiple versions Select Com, M, F from Ti

Confinement layer will produce following partial queries
Select Com, F from V1.Ti
$\oplus$
Select Com, M from V2.Ti

Case 3: Selection transparency: Here we elaborate selection transparency with complete set of prospects.

i) Common attribute predicate
Let Ti be any table such that
Ti $\in$ V1 and Ti $\in$ V2
And Com $\in$ V1.Ti and V2.Ti
Where Com is set of common attributes.

Transparent user query for multiple versions
Select Com from Ti where Com = 'value'

Confinement layer will produce following partial queries:

Select Com from V1.Ti where Com = 'value'

⊕

Select Com from V2.Ti where Com = 'value'

ii) Single Distinct attribute predicate
Let Ti be any table such that
Ti ∈ V1 and Ti ∈ V2
And M ∈ V1.Ti and M ∉V2.Ti

Transparent user query for multiple versions.
Select Com from Ti where M = 'value' and Com = 'value2'

Confinement layer will produce following partial queries:
Select Com from V1.Ti where M = 'value' and Com = 'value2'

⊕

Select Com from V2.Ti where Com = 'value2'

iii) Multiple Distinct attribute predicate
Let Ti be any table such that
Ti ∈ V1 and Ti ∈ V2
And M ∈ V1. Ti and M ∉V2. Ti
And F ∈ V2. Ti and F ∉V1. Ti

Transparent user query for multiple versions
Select Com from Ti where M = 'value' and Com = 'value2' and F = 'value3'

Confinement layer will produce following partial queries:
Select Com from V1.Ti where Com = 'value2 and M = 'value'

⊕

Select Com from V2.Ti where Com = 'value2' and F = 'value3'

EXECUTION MONITORING

This layer with twofold responsibilities dispatches partial queries to their relevant real or alternative versions and monitors query execution. For optimal execution it is divided into parts: i) Global Execution Monitor (GEM) ii) Local Execution Monitor (LEM).

GEM using L2G algorithm coordinates with LEMs to produce consistent and reliable results within small time. If any of the LEMs rejects mono-version query for which further is either not possible or not suitable, GEM will stop all the participating LEM for query execution. Here we are presenting L2G coordination algorithm for monitoring query execution.

L2G Algorithm: This is simple and elegant algorithm to ensure the atomicity and consistency of retrieval.

L2G Coordinator:

```
Declare Votes { Var1, Var2, Var3... Var n }
        /*where n is number of participating version
        which is equal to number of LEMs. */
    msg : Message
    ev: Event

Cast [ Votes { LE1(Var1), LE2(Var2), ... LEn(Var n)} ,
Query ]
    /* cast the vote to LEMs */
Wait (ev)
    /* wait for event to occur*/
Check if Vote = 'Vote-Abort'
        /* for each vote check value if single abort,
        informed by setting variable to 0*/
    msg (Abort {LE1, ... LEn } )
    No Result.
    END
    /* all processing for Q abort, no result */
else
Maintain (result)
    /*maintain all results and wait for all responses */
if All (Votes) = 'Vote-Complete'
    /* found if all variables set to 1 */
Compile (result)
Result.
END
```

L2G Participants:

```
    Wait-until( Receive [ Vote, Query ]
        /* receive vote for processing with query*/
if result = complete
        /* if result is completed*/
Cast ('Vote-Complete')
    /* cast vote that result is complete*/
else
Cast ('Vote-Abort')
    /* cast vote that result is not complete by setting
    variable to 0*/

Ack-to-msg (Abort LEk)
    /* query aborted
```

Finally, each partial query is optimized using the local schema catalog of each version. Local optimization uses cost model, V-catalog and centralized optimization algorithms for optimization, as used in centralized databases.

SYNTHETIC ANALYZER FOR MV-DW RETERIVAL

Here we presents the ongoing implementation of Synthetic Analyser (S-analyser) for transparent retrieval

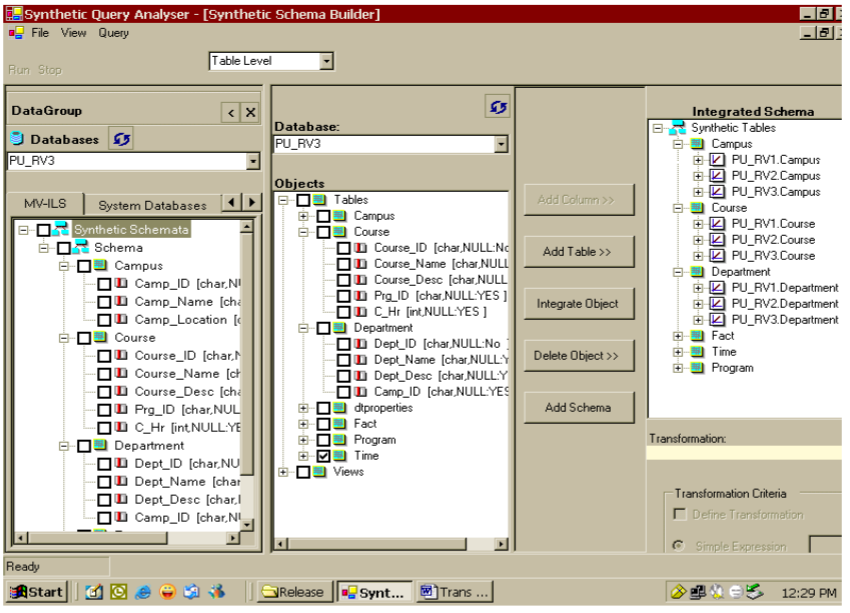

Fig. 4: Synthetic MV-ILS Builder

from DW versions developed in C# DotNet. S-analyzer with the abilities of working as a front-end application for multiple DBMS provides transparency to the users. Also, S-analyzer has various components for better administrating, managing and maintaining tasks of heterogeneous DW versions. S-analyzer has two main components: 1) Synthetic MV-ILS Builder 2) Synthetic Query Analyzer.

Synthetic MV-ILS Builder: The main window interface of builder is shown in Fig. 4, with three pans to be used for logical integration and development of MV-ILS. It is required to access all possible versions and its schema for logical integration. So for version-accessing purpose a combo box is provided above the middle pane.

Schema-viewer is shown in the left windowpane to have a look at MV-ILS. It can be build by integrating any number of versions; it can also be called as an enterprise view of n versions. In-short MV-IIS provides: 1) Single view of all the versions, independent of their heterogeneity. 2) Interface for handling independent versions.3) Give tables- their attributes and data-types for better querying. 4) It is also accessible for reference in the S-query analyzer for ease of end-user. 5) Provide transparency to the users for querying MV-DW. 6) Once MV-ILS has been designed, views over it can be defined for users.

Let V be the set of versions, While M_v is the MV-ILS for all the versions, then mathematically, let V be set of n versions.

V = {V1, V2, V3, …. Vn}
and
V1 = {T1, T2, T3 … Tn}
V2 = {T1, T2, T3 … Tm}
….
Vn= {T1, T2, T3 … Tk}
; where Ti is table name.
Then

$$M_v = \sum_1^n U\ (Vi)$$

Integration schema details are provided in the right pan called Version-navigator. It allows us to: 1) Give the mapping of all the mono-version tables to the global table. 2) Global tables and their elements. 3) Provide foundation for the MV-ILS. 4) Facilitate adding deleting of global Tables.

Synthetic query analyzer: Second important component of S-analyzer is Synthetic-query analyzer (SQA) (Fig. 5), to be used for the transparent retrieval. The input query to it, is to be called transparent query (generic query) on multiple versions, which is further transformed to confined queries on mono-version called Mono-Version

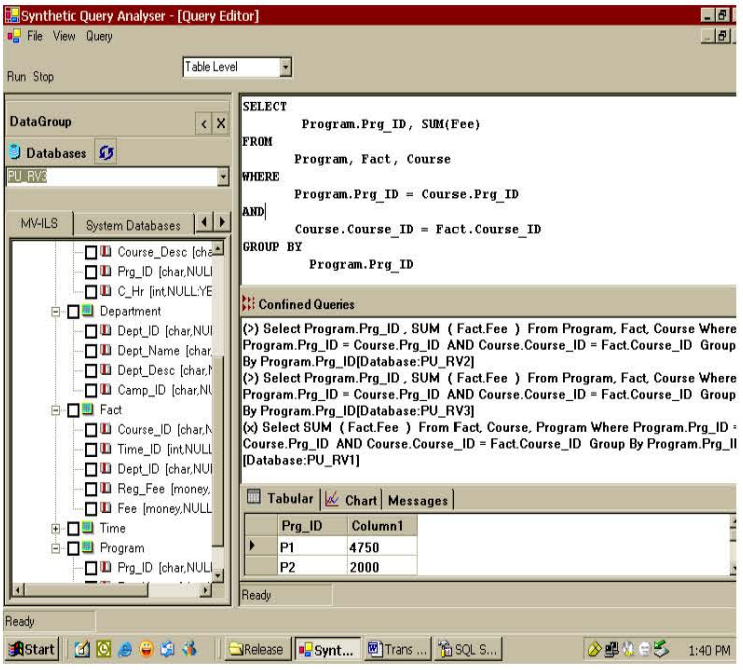

Fig. 5: Using aggregates in SQ-analyser

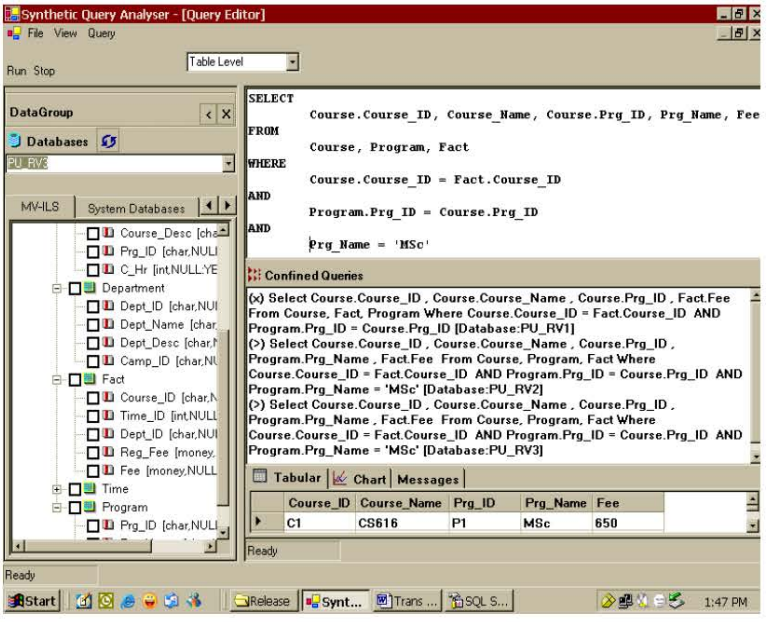

Fig. 6: Confinement for selection transparency

Query (MVQ). Schema-viewer is accessible in SQA for better query writing. SQA also provides a Querying Interface (QI) for writing input queries Standard-SQL transparent to versions.

SQA performs the following steps on the user query for transparent retrieval on multi-versions.

- At first, user query is analyzed for its semantics and syntax correctness.
- A multi-version query expressed by a user is decomposed to the set of independent partial queries (MVQ), each one for its version.
- Each MVQ is then evaluated for its correction, i.e. queries are rejected for which further processing is not possible as shown in Fig. 6 by (X) symbol, resulting in saving the execution time, cost and resources.

- Every confined query is then executed using its appropriate version.
- The results of every confined query are then merged to a single unit for presentation.

SQA also contains sub-components that are: 1) Querying Interface (QI), to be used for query writing purpose 2) Result Visualizer (RV) (Data Grid/Chart), is used to represent the results in grid and graphical format. 3) Query Confinement Module (QCM), which confines and displays mono-version queries for verification.

TRANSPARENT RETRIEVAL FOR HETROGENEOUS DW SCHEMATA--- CASE STUDY

Multiple DW versions can have same or different dimensional schemas (DS). Two versions are homogeneous if: 1) Both have same amount of tables i.e. DT and FT's. 2) The tables are identical i.e. amount of attributes and types of attributes. Versions can also be heterogeneous if either their table amount is not identical or if their structure is not identical, i.e. one table or its attribute is present in version while not in others.

The S-analyzer presented here, is tested by applying a case study of University of the Punjab (PU) data warehouse. PU is a multi-campus university with over 150 departments. Each campus contains more than one department, while various degree programs are offered in each department, also relevant courses are offered in each program. For simplicity we consider smaller portion of the PU-DW. Consider PU-DW is composed of two types of Tables Fact Tables (FT) and Dimension Tables (DT), which surround the FT. In this case fact is the only fact table, surrounded by course and department dimensions. The course dimension is composed of only one level, ie. Course. While the dimension of department also have two levels Department and Campus. The fact is the only fact table, with the information about the quantity of Fee collected.

PU-DW has three departments: UCIT, GIS, Geographic Dept. UCIT presented in old campus while, while GIS and Geographic Dept are in new campus of the university. GIS shifted from New Campus to Old Campus due to its stronger collaboration with UCIT. Although no change is being made except the change of campus, but inconsistent analytical results are produced[1]. Consider, with the evolution of DS of DW RV_1, RV_2 and RV_3 are produced. The schemata of all the version are assumed to be:

PU_RV1 = {Campus, Time, Department, Course, Fact}
Campus = {Camp_ID, Camp_Name, Camp_Loc}
Time = {Time_ID, Date, Week_No, Month_No, Year}
Department = {Dept_ID, Dept_Name, Dept_Desc, Camp_ID}
Course = {Course_ID, Course_Name, Course_Desc}
Fact = {Course_ID, Time_Id, Dept_ID, Reg_Fee, Fee}

The dimensions like campus, time and department remain same while changes are occurred to that of course and program.

PU_RV2 = {Campus, Time, Department, Course, Progam, Fact}
Course = {Course_ID, Course_Name, Course_Desc, Prg_ID}
Program = {Prg_ID, Prg_Name, Prg_Desc}

The dimensions like campus, time, program and department remains same while changes are occurred to that of course.

PU_RV3 = {Campus, Time, Department, Course, Progam, Fact}
Course = {Course_ID, Course_Name, Ch_Hr, Course_Desc, Prg_ID}

Administrators are required to the build a MV-ILS using SMV-ILS builder, as shown in Fig. 3. This schema shows union of all the versions will facilitate user in writing transparent queries. So the above case MV-ILS becomes:

PU_MV-ILS = {Campus, Time, Department, Course, Program, Fact}
Campus = {Camp_ID, Camp_Name, Camp_Loc}
Time = {Time_ID, Date, Week_No, Month_No, Year}
Department = {Dept_ID, Dept_Name, Dept_Desc, Camp_ID}
Course = {Course_ID, Course_Name, Prg_ID, Course_Desc}

Program = {Prg_ID, Prg_Name, Prg_Desc}
Fact = {Course_ID, Time_Id, Dept_ID, Reg_Fee, Fee}

For retrieval consider, a user who inspects amount of fee collected in each program. The query written for such purpose is given in Fig. 4. SQA firstly, evaluates query for

its semantic and syntactic correctness using MV-Catalog. Then, QCM confines query for mono-version, in this case three MVQs are produced for version. After execution of the queries the result is merged in shown. In this case user is provided with Source Transparency, i.e user has written its query independent of the existence of dimension table in any of the version.

Also consider the example of Selection transparency, for what-if analysis it is required to retrieve course and program information but with predicate of M.Sc as their program. Although in RV_1, program is not present but users need not to keep a look at evoluting versions. Figure 5 shows the query and results after retrieval from all the versions. The queries, which are executed and retrieve results are executed and monitored by the execution monitor. The responsibility of execution monitor is to keep the execution information about the queries, in case of bouncing of single query it will stop the execution of all other MVQs.

Querying alternatives: A few queries for what-if analysis have been used and verified above. Here are the set of possibilities that have also been tested and verified for S-analyzer evaluation purpose. The types of queries are:

- Two versions source, i.e. one dimension or fact table present in one or more than one versions. e.g. Campus and time are two dimensions that are present in all versions
- Distinct attribute/s in one or more versions. i.e. although versions have same dimension and fact tables but the attributes are not same, e.g. in course dimension Ch_Hr is distinct attribute.
- Common attribute predicate, i.e. the predicate given in the query is to be applied in all versions. e.g. any predicate which is in all the versions.
- Distinct attribute predicate, i.e. the predicate used in query is on the attribute which is either in not present in all the versions. e.g. The predicate to be applied on the Ch_Hr attribute of the course.

CONCLUSIONS

Multiple versions of DW are required to be maintained, to meet the dynamic analytical requirements. For better retrieval a conception of querying transparency is proposed for MV-DW, in which transparency can be provided at source, analysis and projection level. This study presents the modifications to conventional query processing layers by adding confinement layer to it.

S-analyzer is given which implements proposed layers for transparent retrieval; also analyzer has been tested by PU case study of heterogeneous versions. The empirical result demonstrates that transparency provides ease of use for querying multi-version sources.

REFERENCES

1. Pasha, M.A., J.A. Nasir and M.K. Shahzad, 2004. Semi-star schema for managing data warehouse consistency. Proceedings of IEEE-NCET'04.
2. Morzy, T. and R. Wrembel, 2003. Modeling a multi-version data warehouse: A formal approach. Proceedings of ICEST'03.
3. Blaschka, M., C. Sapia and Hofling, 1999. On schema evolution in multi-dimensional databases. Proceedings of DaWak'99.
4. Hurtado, C.A. and A.O. Mendelzon, 1999. Maintaining data cubes under dimension updates. Proceedings of ICDE'99.
5. Hurtado, C.A., A.O. Mendelzon and A.A. Vaisman, 1999 Updating OLAP dimensions. Proceedings of DOLAP'99.
6. Eder, J. and C. Koncilia, 2001. Changes of dimension data in temporal data warehouse. Proceedings of DaWaK'01.
7. Eder, J. and T. Morzy, 2002. The COMET metamodel for temporal data warehouses. Proceedings of 14th CAISE'02.
8. Morzy, T., R. Wrembel and Eder. 2004. Creation and management of versions in multiversion data warehouse. Proceedings of ACM Symposium on Applied Computing.
9. Morzy, T. and R. Wrembel, 2004. On querying versions of multi-version data warehouse. Proceedings of 7th ACM International Workshop on Data Warehousing and OLAP.
10. Ozsu, T.M. and P. Valduriez, 1999. Principles of Distributed Database Systems. 2nd Edn., Prentice Hall Publishers, pp: 198-235.