Integrating Active Rules with Data Warehouse Systems

*Shi-Ming Huang andYuan-Mao Hung**

Department of Information Management, National Chung Cheng University,
Ming-Hsiung, Chia-Yi 621, Taiwan
*Department of Computer Science and Engineering, Tatung University, Taiwan
Tel: +886-5-2720411 ext 6403, Fax: +886-5-2721501
E-mail: smhuang@mis.ccu.edu.tw

Abstract

Although many data warehouse systems have been developed in recent years, the practice of data warehouse is still immature. Most of current systems have some limitations in terms of flexibility, efficiency and scalability. Furthermore, the contents of a data warehouse are becoming a large and messy jungle. It is difficult to maintain and analyze the data. Much research has been done in data mart [1; 2; 3; 4] to solve the problems. Unfortunately, the solutions isolate the information into an independent data cube. Users will only retrieve the knowledge from one single angle, but not from a global view. This is due to:

- The presented data warehouse models lack human involvement, particularly in the form of guidance and user control.
- The presented data warehouse systems do not allow users to define the relationship within data marts.

To deal with the above problems, this study intends to design a novel architecture for a data warehouse augmented the active rule technique. The proposed architecture focuses on the uniform metadata model and the active monitor conditions triggered in data warehouse, which employs inference engine to go further analysis or warning.

1. Introduction

Data Warehouse is an architectural construct of information systems that provide users with current and historical decision support information that is hard to access or present in traditional operational databases. It is a cornerstone of the organization's ability to do effective information processing, which, among other things, enable and share the discovery and exploration of important business trends and dependencies that otherwise would have gone unnoticed.

Active rules have been used in databases for several years [5; 6; 7]. The aim of an active database is to perform automatic monitoring of conditions defined over the database state and then to enable the ability to take action (possibly subject to timing constraints) when the state of the underlying database changes (transaction-triggered processing). Most active database rules make definitions like production rules. They use the event-based rule language, in which a rule is triggered by events such as insertion, deletion or modification of data. The form of an active database rule is:

On event
If condition
Then action

Recently, active rules have also been integrated into data warehouse architecture to maintain data consistency in the materialized views [8; 9].

This paper describes a novel architecture for an Active Data Warehouse System (ADWS), which integrate active rule technology and data warehouse technology. First of all, we have to define the ADWS.

Definition: Active Data Warehouse System

Active Data Warehouse System (ADWS) is a data warehouse system, which provides active rule's functionality to monitor the multi-dimensional query events and to go a step further analysis or warning.

The data warehouse will become alive when augmenting with active rules. The active rule is triggered when the event occurs, and once the rule is triggered, the condition is checked. If the condition is satisfied, then the action is executed.

2. System Architecture

This section describes our novel architecture for an ADWS. Figure 1 illustrates the basic architecture of our

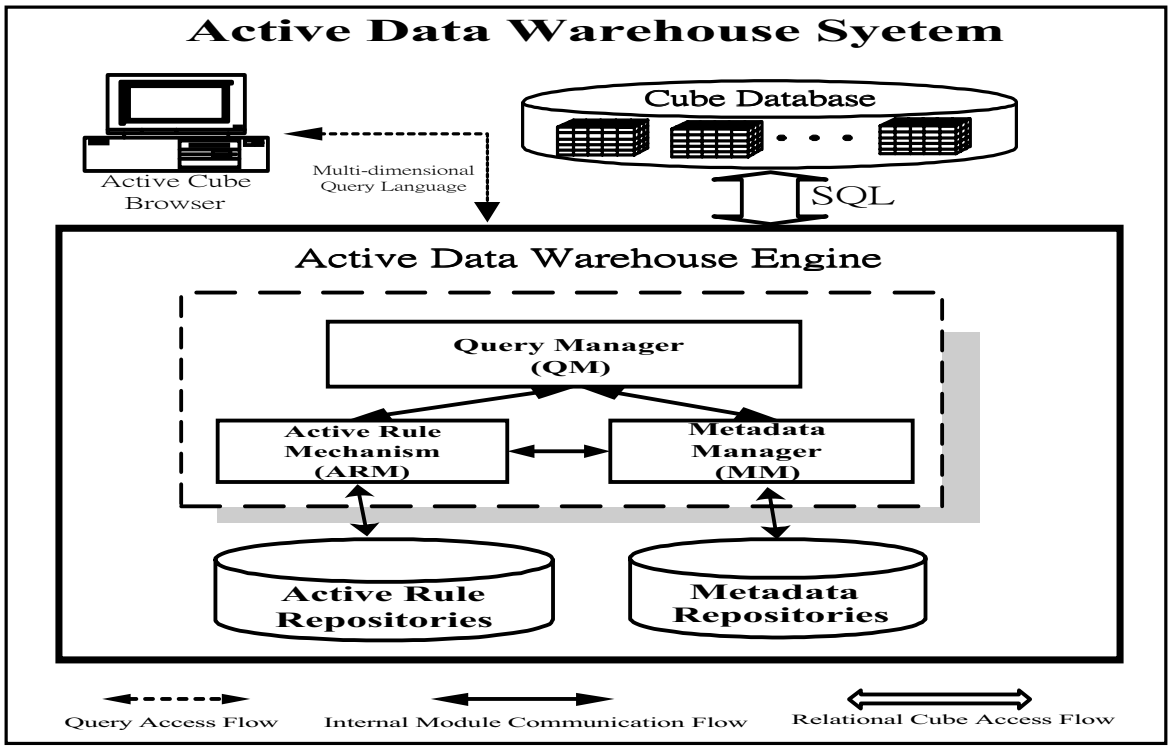


Figure 1: Active Data Warehouse System

system. There are two repositories and three basic process modules in this data warehouse. The two repositories are (1) a metadata repositories and (2) an active rule repositories. The metadata repositories store the translated database schemas of source databases, the global database schema, the star schema of the data warehouse application, and system information of the data warehouse. The active rule repositories store the active rules of data warehouses.

The three basic procedure modules are (1) a Query Manager (QM) module, which provides to analyze multi-dimensional query from active cube browsers, (2) a Metadata Manager (MM) module, which contains the communication mechanism to manage the metadata repositories, and (3) an Active Rule Mechanism (ARM) module, which provides a inference engine to go step triggered rules and inference correctness of result. The detail mechanisms of each module are discussed in the later.

The working environment of this ADWS involves three parts, i.e. the active cube browser as a user interface, a cube database, and an active data warehouse engine. The physical data of the data warehouse is stored in the cube database which is implemented on relational databases. The active cube browser will send a request to active data warehouse engine. The request is represented by our multi-dimensional query language which includes the events information.

The QM module of active data warehouse engine will process the query reguest and translate to SQL. It will also pass the event information to ARM. The ARM may trigger the event information and go a step further analysis or warning.

2.1 Metadata Repositories

2.1.1 Global Database Schema Metadata

Multi-database users view the global schema as the definition of a single database. The heterogeneity of local DBMSs are masked through the global database schema. The global database schema approach requires a total integration at the schema level. In this approach, all local DBSs are translated into EER models. These EER models will be integrated into a reconciled EER model as a global database schema. The meta-data of our global database schema need to represent the EER model. Figure 2 displays ours global database schema metadata by using OMT [13] model.

2.1.2 Star Schema Metadata

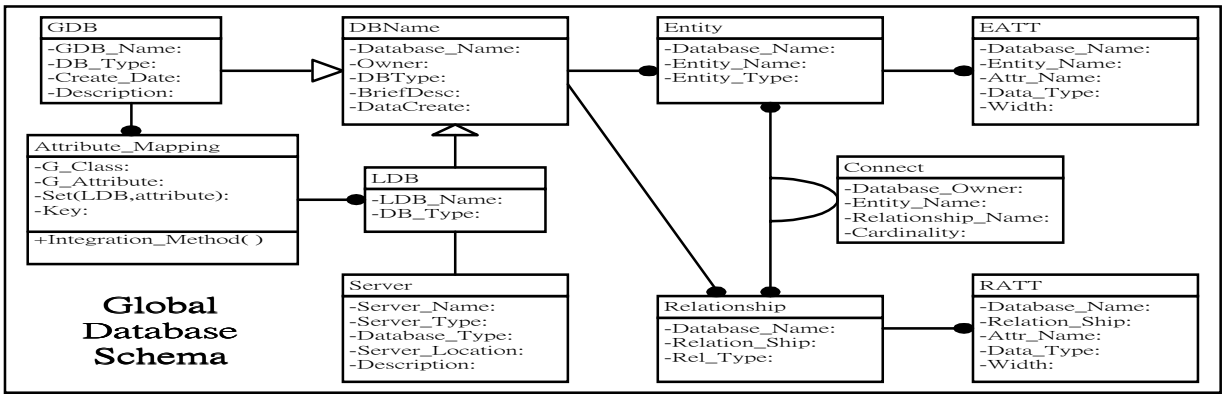


Figure 2: The OMT model of global database schema metadata

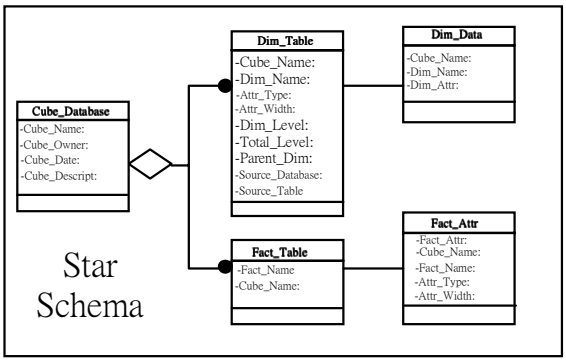


Figure 3: The OMT model of star schema

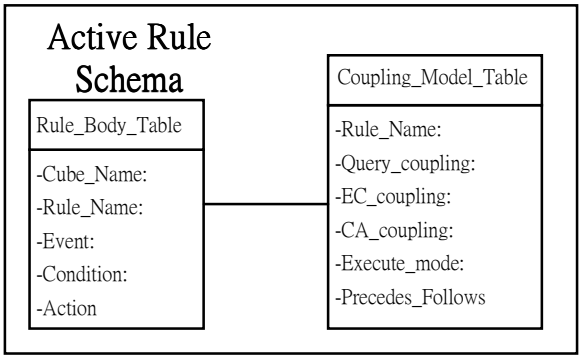


Figure 4: The OMT model of active rule schema

The most popular design technique used to implement a data warehouse is the star schema. The structure of star schema takes advantage of typical decision support queries by using one central fact table for the subject area, and many dimension tables containing de-normalized descriptions of the facts. After the fact table is created, OLAP tools can be uses to pre-aggregate commonly accessed information. Figure 3 displays the OMT model of star schema metadata.

2.2 Active Rule Schema Metadata

There are many useful semantics presented in this active rule schema metadata. The active rule schema can be expressed in two parts: a rule body table and a coupling model table. The rule body table describes the ECA base active rules schema, and the coupling model table describes how the active rules can be integrated into the MDQ. Figure 4 tabulates OMT model of active rule schema.

3. Active Rule Mechanism

Figure 5 depicts the architecture of the active rule mechanism for our ADWS. When *Rule Activation* module receives an event, it will retrieve the related active rules from active rule repository. The *conflict resolution* module will deal with the rule conflict situation. Finally, an active rule will be executed and a new event occurred.

3.1 Data Warehouse Event

Figure 6 shows the event hierarchy, which we have developed to depict the event classification of multi-dimensional query data warehouse. Events can be broadly classified into: I) *Multi-dimensional query events* – events which act as the basic building blocks and for each of which an detector need to be associated and embedded in the system and ii) *data consistency events* – in which case the event is raised by an operation on some piece of structure. (E.g., insert a tuple, update a tuple, update a schema, and drop a schema). The OLAP operators defined in literature [1] and considered in this paper are drill-down, roll-up, push, slice, pull, dice, and select. We have developed to depict the event classification of multi-dimensional data warehouse. Our approaches focus on multi-dimensional query events, which can be classified

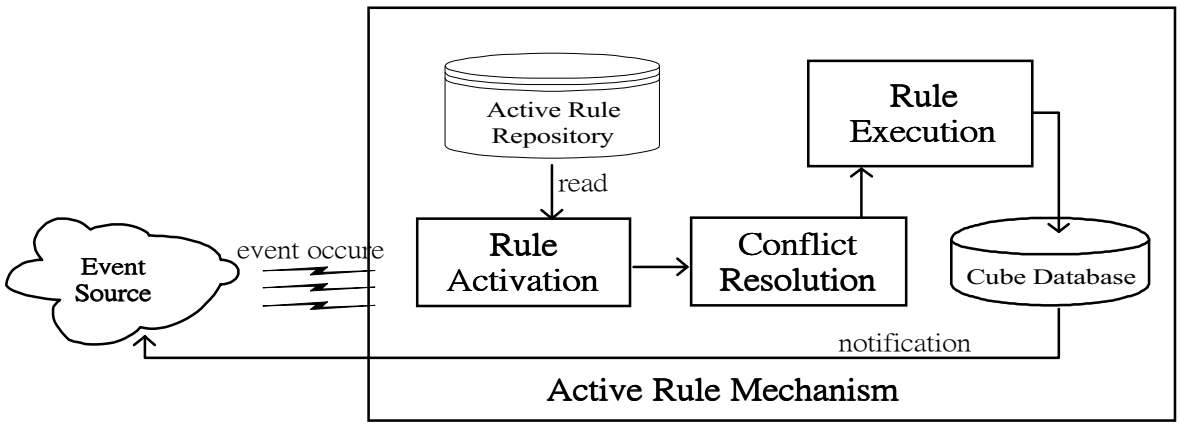


Figure 5: Architecture of Active Rule Inference Engine

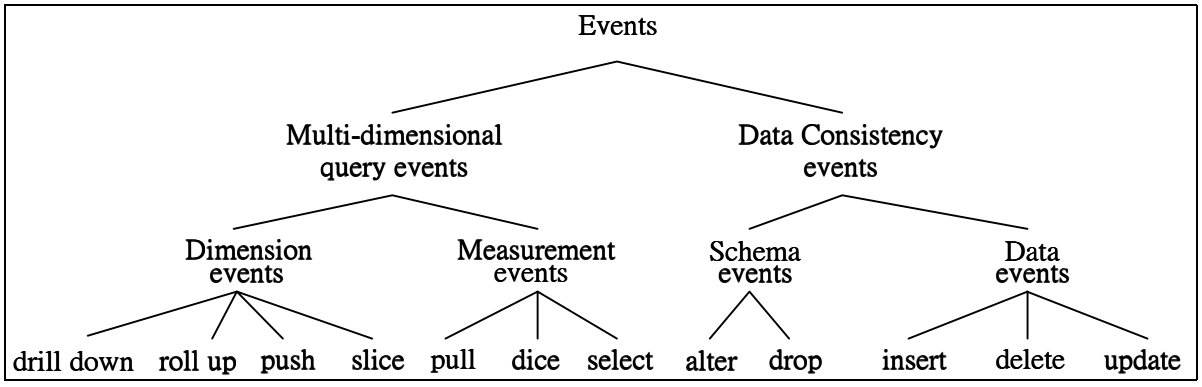


Figure 6: Event Classification

into two categories: *Multi-Dimensional events* and *Measurement events*.

3.2 Active Rule Syntax

Active Data Warehouse Management Systems couple database technology with rule-based programming to achieve the capability of reaction to database stimuli, called events. An ADWS consists of a data warehouse and a set of active rules; the most popular form of active rules is the so-called event-condition-action (ECA) rule, which specifies an action to be executed upon the occurrence of one or more events, provided a condition held. In this section, we provided our active rule syntax, which extended the standard ECA rule.

Figure 7 shows the syntax of our active rule. The syntax has two parts: a rule body and a coupling model. The rule body is to describe the ECA (Event-Condition-Action) base active rules, and the coupling model is to describe how the active rules can be integrated into the database query. The rule body is composed of three main components: a query predicate, an optional condition, and an action. The query predicate controls rule triggering; the condition specifies an additional predicate that must be true if a triggered rule is to automatically execute its action. Active rules are triggered by database state transitions – i.e., by execution of operation blocks. After giving a transition, those rules whose transition predicate holds with respect to the effect of the transition are triggered. By using a coupling model, database designers have the flexibility of deciding how the rule query integrates within the Multi-Dimensional Query (MDQ). There are five different execution attributes to determine semantic of an active rule in our ADWS. They are:

*Query_coupling***:** the execution of a rule is treated as a query in our ADWS, i.e. rule query. If the Query_coupling is set to '**same'**, then the MDQ is committed only when the RQ (Rule Query) and DQ (Data Query) both are committed. If the Query_coupling is set to '**separate'**, then the MDQ committed will only depend on the DQ. It suggests that the *Query_coupling* is set to 'Separate', when the active rule does not have any effect on the DQ. This will enhance the system performance in terms of query execution time.

Rule Body:	***Coupling Model:***
Define Rule <rule name> On <query_predicate> [if <conditions>] then [evaluate query-commalist] execute <action> query_predicate ::= event [,event [,event]] event ::= drill down | roll up | push | slice| pull | dice | select condition ::= query_commalist query_commalist ::= query [,query]* query ::= table_expression	Query_coupling = Same | Separate EC_coupling = Before | After, CA_coupling = Immediate | Deferred, Execute_mode = Repeat | Once, [Precedes <rule_names>] [Follows < rule_names>]

Figure 7: The Syntax of our Active Rule

*EC_coupling***:** defining the execution sequence of the event and condition part for a relational active rule. The '**before'** *EC_coupling* means that the rule condition is evaluated immediately before the DQ is executed. The **'after'** *EC_coupling* means that the rule condition is evaluated after the DQ is in the prepare-to-commit state.

*CA_coupling***:** presenting the execution sequence of the condition and action part for an active rule. The '**immediate'** *CA_coupling* means that the rule action is executed immediately after the rule condition is evaluated and satisfied. The rule action executed after DQ is in the prepare-to-commit state, when *CA_coupling* is specified to '**deferred'**.

*Execute_mode***:** the triggered rule will automatically be deactivated after it is committed, when its *Execute_mode* is specified to '**once'**. On the other hand, the rule is always active if *Execute_mode* is specified to '**repeat'**.

*Precedes_Follows***:** The optional '**precedes**' and '**follows**' clauses are used to induce a partial ordering on the set of defined rules. If a rule r_1 specifies a rule r_2 in its '**precedes**' list, or if r2 specifies r1 in its '**follows**' list, then r_1 is higher than r_2 in the ordering.

3.3 Active Rule Inference Engine

In our ADWS, the rule activation process flow is delineated in the following steps:

Step1: Query coupling evaluation:

If *Query_coupling* is *Separate*, the system will submit the triggered rule to QM (query manager) as a new query. Otherwise, the system will continue the following steps.

Step 2: Event-Condition coupling evaluation-- *Before*

2a. Reasoning rules, which its *EC_coupling* is equal to *Before*.

2b. If the condition evaluation result is true, the following two possible situations may happen.

2b.1. The action part will be executed immediately if its *CA_coupling* is equal to *Immediate*.

2b.2. The action part will be save into a queue if its *CA_coupling* is equal to *Deferred*.

2c. Repeating the steps 2a, 2b until no more rules is reasoned by step 2a.

Step 3: Executing the data query.

Step 4: Executing the queued rules, which are stored by step 2b.2.

Step 5: Event-Condition evaluation--*After:*

5a. Reasoning rules, which its *EC_coupling* is equal to *After.*

5b. If the condition evaluation result is true, there are the following two possible situations.

5b.1. The action part will be executed immediately if its *CA_coupling* is equal to *Immediate*.

5b.2 The action part will be save into a queue if its *CA_coupling* is equal to *Deferred.*

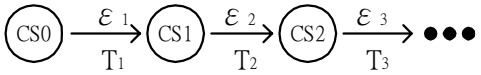
5c. Repeating steps 5a, 5b until no more rules is reasoned by step 5a.

Step 6: Executing queued rules, which are stored by step 5b.2.

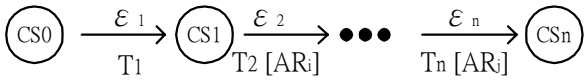
Step 7: Committing the query if and only if all sub-queries are committed.

3.4 Rule execution

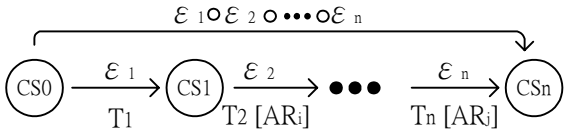
Active rules are activated automatically as a result of database state transitions caused by externally generated operation blocks. Suppose that a stream of operation blocks is submitted for execution. If execution begins in a state CS_0 and active rules are not considered, then the system behaves becomes:



where T_1, T_2, T_3.... are unique transition labels; and ε_1, ε_2, ε_3... are the effects of the transitions. Recall that each transition effect is actually a pair of *[X_DIM, Y_DIM]*. In this section, we assume that externally generated operation blocks correspond to data warehouse query. That is, each state in execution sequence above corresponds to a state in which a query begins execution; the subsequent state corresponds to the point at which the user or application program requests that the query be committed. This one-to-one correspondence between transitions and queries holds only for externally generated operations. When rules are executed, a single query is composed of one externally generated transition followed by some number of rule-generated transitions. Consider now an arbitrary point in rule processing. A state CS_n has been reached, resulting form execution of a user-generated operation block followed by some number of rules:



In state CS_n, rules are triggered based on composite transition effects; the transition tables in their conditions and action are defined accordingly. If a rule AR_k has not yet generated any transitions in the sequence (i.e., AR_k*'s* action has not been executed since the most recent externally generated transition), then AR_k is triggered in state CS_n if its transition predicate is satisfied by the composite effect since state CS_0:



If rule *AR* is indeed triggered and it is chosen for consideration, then its transition tables are based on transition effect $\varepsilon_1 \circ \varepsilon_2 \circ \ldots \circ \varepsilon_n$, current state CS_n, and pre-transition state CS_0. We now give an algorithm of semantics for rule execution. The algorithm is more "procedural" than the previous semantics as described above. This information can be used at any time to determine if the rule is triggered and, if so, to construct the relevant transition tables. Each rule's transition information is modified incrementally as operation blocks are executed and new transitions are created. The main algorithm is given in figure 8, along with its three function calls.

The blow algorithm loops indefinitely as long as there are externally generated operation blocks to be executed. Each iteration of the loop corresponds to a single query containing an externally generated transition followed by consider action and possible execution of some number of rules. With each rule *AR* we associate an attribute

```
Repeat forever
/* Execute externally block, initialize transition information */
Old-state ← current-state();
 ε ← execute-external-block();
[X_DIM,Y_DIM] ← Init-trans-info ( ε , old-state);
For each AR  active-rules () do
  AR.trans-info ← [X_DIM,Y_DIM];
/* Begin rule execution */
AR ← select-eligible-rule( );
While AR ≠ null do
  /* Execute rule block */
  old-state ← current-state();
   ε ← execute-rule-block(AR);
  /* Rule AR gets new transition information */
  AR.trans-info ← Init-trans-info ( ε , old-state);
  /* Modify transition information for all other rules */
  for each AR'  active-rules () do if AR' ≠ AR then
    AR'.trans-info ← modify-trans-info(AR'.trans-info, ε , old-state);
  AR ← select-eligible-rule ()
End while

Function Init-trans-info ( ε , old-state)
  X_DIM ← X_DIMENSION( ε );
  Y_DIM ← Y_DIMENSION( ε );
  Return ([X_DIM, Y_DIM])

Function select-eligible-rule()
  Repeat
    AR ← select-triggered-rule (active-rules());
    If AR ≠ null then
      cond-holds ← check-condition(AR);
    else cond-holds ← false
  Until cond-holds or AR = null;
  Return (AR)

Function modify-trans-info([X_DIM,Y_DIM], ε , old-state)
  X_DIM ← X_DIM ∪ X_DIMENSION( ε );
  X_DIM ← X_DIM ∪ X_DIMENSION( ε );
    Return ([X_DIM, Y_DIM])
```

Figure 8: Rule Execution Algorithm

AR.trans-info. This attribute is a pair of *[X_DIM, Y_DIM]*.

4. Multi-Dimensional Query Language

Data analysis applications look for unusual patterns in data. They categorize data values and trends, extract statistical information, and then contrast one category with another. The active cube browser will send a query to our ADWS. The protocal of the query is our Multi-Dimensional Qrery Language (MDQL) which include the events information.The Query Manager (QM) module of active data warehouse engine will process the MDQL and translate to *SQL*.

4.1 Multi-Dimensional Query Language Syntax

In relational data warehouse systems, users (or application programs) submit streams of multi-dimensional query operations for execution. The operations in a model of system behavior are grouped into various operation blocks. The operation blocks (rather than individual operations) for generality may permit formalism that adapts easily most relational database languages and assumes some familiarity with SQL. So the SQL-like syntax is used to implement the MDQL. That can make

```
SELECT
[Alias.] Select_Item [AS Column_Name]
        [, [Alias.] Select_Item [AS Column_Name] ...]
FROM GlobalTableName/StarSchemaName [Alias]
        [, GlobalTableName [Alias] ...]
Operation_Block := sql-op; seq-op; …….; sql-op
Sql-op := X_DIMENSION | Y_DIMENSION
[X_DIMENSION BY
Column_Name[OLAP_OPERATOR][LEVEL Number]
[, Column_name [OLAP_OPERATOR] [LEVEL Number]...]]
[Y_DIMENSION BY
Column_Name[OLAP_OPERATOR][LEVEL Number]
[, Column_name [OLAP_OPERATOR] [LEVEL Number]...]]
OLAP_OPERATOR =
[RollUp/DrillDown/Push/Pull/Slice/Dice/Select]
[WHERE condition expression]
```

Figure 9: Multi-Dimensional Query Language Syntax

the users to implement the multi-dimension query in the ad hoc application easily. In addition, The SQL like syntax can give user less effort to learn the new syntax. Then the *X_DIMENSION* and *Y_DIMENSION* fields are designed to match for our output format. The query language syntax is illustrated in Figure 9.

Example: Let us consider the data cube represented in Figure 10 and its "multidimensional" view is illustrated. Let us suppose a user is browsing the cube data in countryside level and a new query defined as below.

"Select Vendors for which the total sales is > 37000 units in each County of the North"

Using our MDQL as the following can represent this query as the following:

SELECT County, Vendors, Drink Sales
FROM Sales_Cube
X_DIMENSION: = Roll-Up from Countryside to Region,
Select Region = North,
Drill-Down from Region to County
Y_DIMENSION: = Push Vendors, Pull # of sales, Dice # of sales ≦ 37000

4.2 Active Cube Browser

The active cube browser (see figure 11) provides an OLAP function when users use it to query the data of the cube database. It will occure several events when users interact with the browser. The Active Data Warehouse Engine receives the event and empolys an active rulemechanism to go further analysis and to feedback a warning to the active cube browser. Figure 11 shows that the Active Data Warehouse Engine will detect the occurrence of dimension-drill-down events. Then the rule

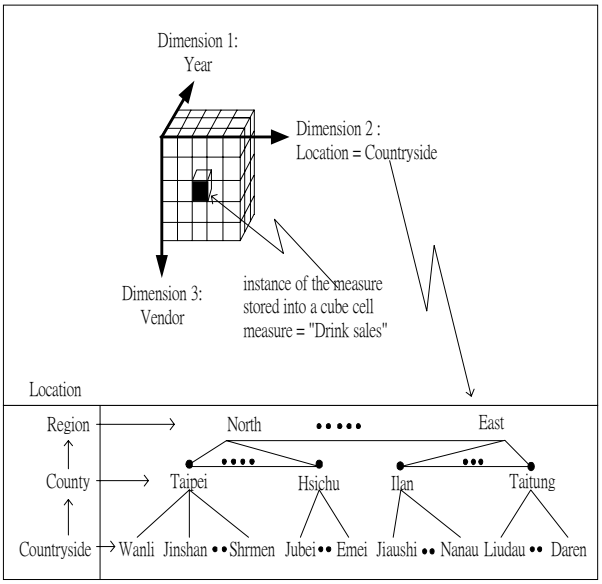


Figure 10: Example of a data cube with hierarchy

of Great_Sales is trigger to decision for next month market and material supply chain relationship.

5. Conclusion

Because of traditional data warehouse systems are passive, they execute queries or transactions only when the users or the application programs explicitly request. This paper incorporates active rules into the data warehouse system. The result of this integration is that the data warehouse becomes active but not passive. It can go further to analyze the data automatically.

Our approach is based on the following major features:

Create the active data warehouse architecture.

Create the relational metadata for global database schema, star schema, and active rule schema to represent the ADWS metadata.

Describe our active rule syntax and its activation mechanism in detailed.

Design a multi-dimensional query language for active cube browser.

The ADWS architecture introduced in this paper can meet the objectives of this study.

Acknowledgement

The work presented in this paper has been supported by the National Science Council, Taiwan, R.O.C., under the Grant No. NSC 89-2213-E-194-041. We deeply appreciate their financial support and encouragement.

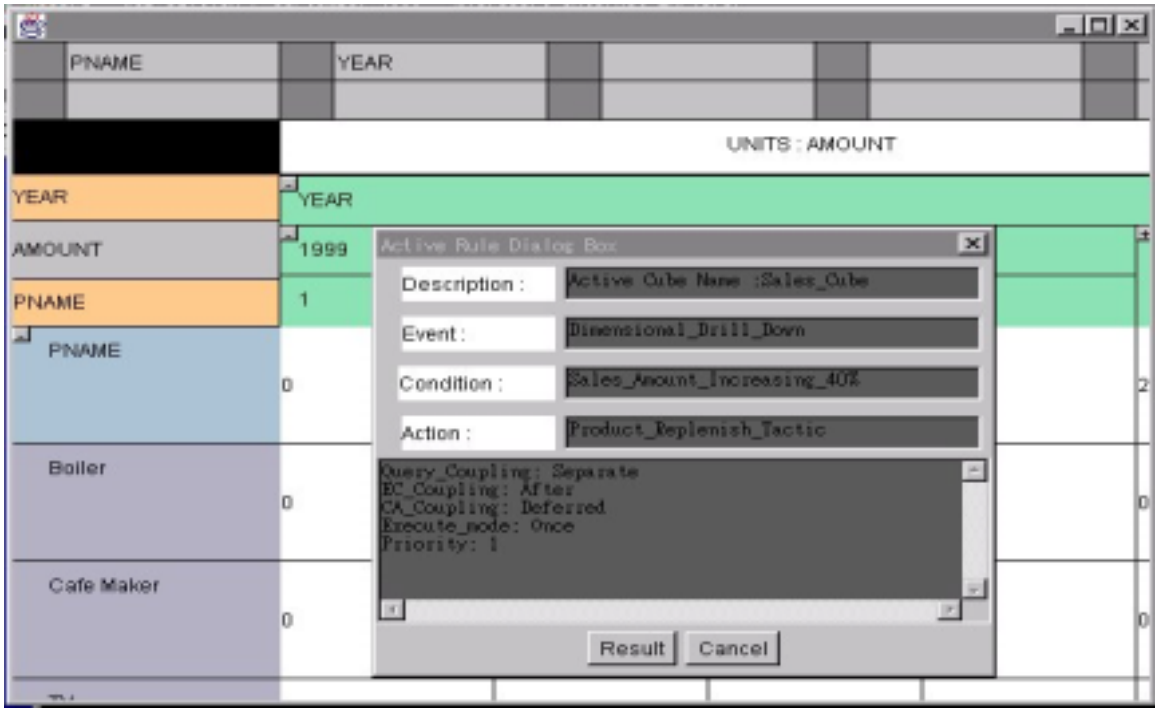


Figure 11: Active Cube Browser

References

[1] Elaheh Pourabbas, and Maurizio Rafanelli, 2000, "Hierarchies and Relative Operators in the OLAP Environment" *ACM SIGMOD*, Vol. 29, Num. 1, March.

[2] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao, 1997, "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Gross-Tab, and Sub-Totals", *Data Mining and Knowledge* Discovery 1, 29-53.

[3] Jiawei Han, Laks V.S. Lakshmanan, and Raymond T. Ng, 1999, " Constraint-Based, Multidimensional Data Mining", *Computer*, Auguest 1999.

[4] E. Baralis and J. Widom, 2000, "Better Static Rule Analysis for Active Database Systems", To appear in *ACM Transactions on Database Systems*, 2000.

[5] Huang Shi-Ming and Huang Chien-Ming, 1998, "A Semantic-Based Transaction Model for Active Heterogeneous Database Systems", The *Proceeding of 1998 IEEE International Conference on Systems,* Man, and Cybernetics, IEEE Press. (to appear)

[6] Paton W. and Diaz Oscar, 1999, "Active Database Systems", *ACM Computing Surveys*, Vol. 31, No. 1.

[7] S.M. Huang and J.K. Liu, 1997, "Developing an Active Heterogeneous Database System", *Proceedings of Intelligent Information Systems*, pp. 415-419.

[8] Adelberg B., Garcia-Molina H., and Widom J., 1997, "The STRIP Rule System for Efficiently Maintaining Derived Data", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp 147-158.

[9] Ceri S., Fraternali P., and Tanca L., 1994, "Automatic Generation of Production Rules for Integrity Maintenance", *ACM Transactions on Database Systems* 19(3): 367-442.

[10] Shi-Ming Huang, Ming-Yi Chen and Joseph Fong, 1999, "A Database Schema Integration Methodology for A Data Warehouse", *9th International Database Conference* (IDC'99), Hong Kong, July, 1999, ISBN 962-937-046-8,pp415-418.

[11] Shi-Ming Huang, Shing-Han Li, and Joseph Fong, 1997, "Translate Relation Database Model Into Extended Entity Relationship Model: A Reverse Engineering Approach", *Tatung Journal*, Vol 26, pp 175-186, ISSN 0379

[12] Shi-Ming Huang, Ming-Yi Chen, and Irene Kawn, 2000, "Conflict Resolution and Reconciliation in Multidatabase Systems", *International Journal of Information and Management Sciences*, Vol. 11, No. 3. (to appear)

[13] James Rumbaugh, Michael Blaha, William Premerlani, Fredrick Eddy, and William Lorensen, 1991, "Object-Oriented Modeling and Designing", by *Prentice-Hall*, Inc. A Division of Simon & Schuster Englewood Cliffs, New Jersey 07632