

# Continuous and Incremental Data Mining Association Rules using Frame Metadata Model<sup>1</sup>

Joseph Fong and H K Wong  
Department of Computer Science  
City University of Hong Kong, Hong Kong  
Email: [csjfong@cityu.edu.hk](mailto:csjfong@cityu.edu.hk)

Shi Ming Huang  
Information Management Department  
National Chung Cheng University, Taiwan  
Email: [smhuang@mis.ccu.edu.tw](mailto:smhuang@mis.ccu.edu.tw)

## Abstract

Most organizations have large databases that contain a wealth of potentially accessible information. The unlimited growth of data will inevitably lead to a situation in which it is increasingly difficult to access the desired information. There is a need to extract knowledge from data by Knowledge Discovery in Database. Data mining is the discovery stage of KDD whereas association rule is a possible product. It states a statistical correlation between the occurrence of certain attributes in a database table. Such correlation is continuously changing subject to the new updates in the source database. Data mining association rules are often done by computing the association rules for the whole source database. In this paper, we introduce a frame metadata model to facilitate the continuous association rules generation in data mining. A new set of association rules can be derived with the update of the source databases by the data operation function in the frame metadata model. The frame metadata model consists of two types of classes: static classes and active classes. The active classes are event driven, obtaining data from the database when invoked by a certain event. The static classes describe data of the association rule table. Whenever an update occurs in the existing base relations, a corresponding update will be invoked by an event attribute in the method class which will compute the association rules continuously. The result is an active data mining capable of deriving association rules of a source database continuously or incrementally using frame metadata model.

Keywords: active data mining, frame metadata model, association rule, knowledge discovery in Database, metadata

## 1 Introduction

Strategic planning is a vital task of top management in large scale and international companies. Analysis on past marketing experience helps identify customer needs and predict the trends of customers. Against this background, the great interest that is being shown in data mining is understandable. KDD describes the whole process of extraction of implicit, previous unknown and potentially useful knowledge from data. This paper demonstrates the KDD with the basis of adopting statistical techniques in the data mining stage, and to automate the KDD process. Association rule identifies potential relation of data. It can be characterized by Support and Confidence level which describe the degree of association in the rule. A strong association rule will have a large support and high confidence.

The process of keeping the support and confidence up-to-date in response to the changes in the source database is to generate new set of association rules on-line[1]. Deriving association rules can be achieved by full re-computation of the support and confidence of association rule

---

<sup>1</sup> This paper is funded by Strategic Research Grant 7000895 of City University of Hong Kong

continuously or incrementally by computing new record counts in the source database created since last computing date. Such task requires access to the base relations to update the Support and Confidence level in the association rule. In data mining, accessing base relations can be difficult since these relations are distributed across different sources. Moreover, data sources may not support full database functions and querying such sources to compute the view updates might be a cumbersome. The issue of automating the data mining association rule continuously is therefore essential.

Data mining has become major elements of decision support systems. It can derive rules which are refreshed periodically by extracting, transforming, cleaning and consolidating data from several operation data sources. However, we consider business intelligence applications that are characterized by very high data volumes, and that require continuous analysis and mining of the data. For such applications, continuous or incremental association rule generation in data mining according to the new base relation update is necessary.

A methodology is presented here which discovers association rules by computing the data volume of target data according to user requirements. It suggests continuous data mining association rules as a result of the continuous base relation update. Its advantage is to derive the association rules by applying frame metadata model to store the Support and Confidence level of data in an internal association rule (AR) table. The frame metadata model consists of four classes. A header class identifies each fact table. An attribute class defines its attributes' properties. A method class stores its data operations and condition. A constraint class describes each event occurrence. They can be used to implement an event driven active data mining. When an event occurs, it triggers a process in the constraint class, which calls for the operations in the method class for action. Data can be actively updated to generate the association rules for decision support systems. The result is active data mining.

We identify the computing of the Support and Confidence level of data in a source data table as follows:

For example, an association rule below indicates that the data occurrence of  $A_1, A_2, \dots, A_n$  will most likely associate with the data occurrence of  $B$ .

$A_1, A_2, \dots, A_n \Rightarrow B$

The Support and Confidence level of this association is:

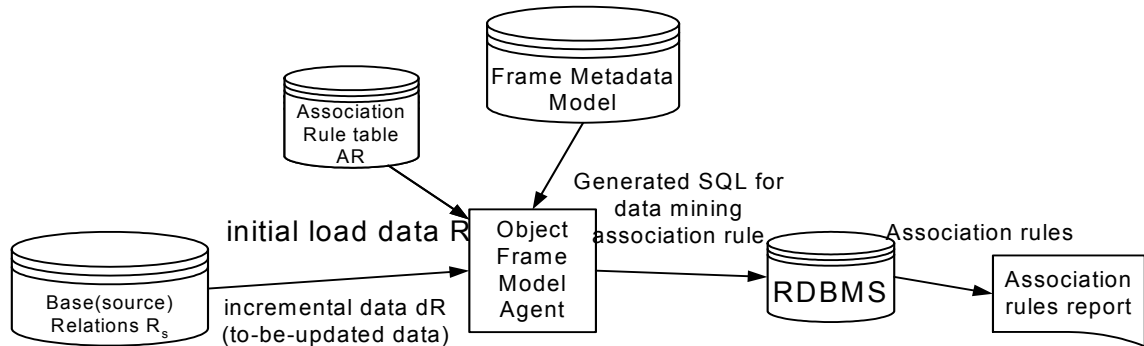
$$\text{Support} = \frac{\text{Record\_Count}(A_1 \cap A_2 \cap \dots \cap A_n \cap B)}{\text{Record\_Count of source data}} * 100\%$$

$$\text{Confidence} = \frac{\text{Record\_Count}(A_1 \cap A_2 \cap \dots \cap A_n \cap B)}{\text{Record\_Count}(A_1 \cap A_2 \cap \dots \cap A_n)} * 100\%$$

The approach is to compute the Support and Confidence level of base relations for data mining by the method class and the constraint class of the frame metadata model for checking the occurrence of the event.

The frame metadata model can help develop active data mining which can update its association rules based on the new data inserted into the base relations. Its objective is to provide a mechanism of synchronizing the update of base relations in source database and its association

rules incrementally. The frame metadata model can transform the data mining association rules process into active data mining as shown in Figure 1.



**Figure 1 Architecture of data mining association rules using frame metadata model**

An Object Frame Model Agent (refer to prototype) automatically calculates Support and Confidence level of the association rule in data mining. Base relations are source data that are loaded into RDB tables in the data mining. Incremental data are data of  $\delta R(dR)$  which are produced from the source relation in order to update association rules after initial loading. Object Frame model Agent is an executor which invokes the constraint class and the method class in the frame metadata model to generate the required SQL transactions for computing the statistical Support and Confidence level figures of association rules.

## 2 Related work

### *On View Maintenance*

The view maintenance problem was studied extensively [2,3] with the recent survey of view maintenance literature. They derived conditions to determine when a derived relation could be updated using its own data and the given updates.

### *On Data mining*

The problem of data mining for association rule has been studied extensively [4,5,6]. In [4], the authors proposed a spectrum of architectural alternative for coupling mining with database system. These alternatives included loose coupling through a SQL cursor interface, encapsulation of mining algorithm in a stored procedure, caching the data to a file system on-the fly and mining, tight-coupling using primarily user-defined functions, etc. The evaluation of the different architectural alternatives showed that the Cache-Mine option was superior in the performance perspectives and the loose-coupling approach incurred a less storage penalty than the SQL-OR and the Cache-Mine approaches. The authors also compared those alternatives based on the qualitative factors.

The work presented in [5] discussed data mining was based on association rules for two numeric attributes and one Boolean attribute. For instance, in a database of bank customers, “Age” and “Balance” were two numeric attributes and “CardLoan” was a Boolean attribute. Taking the pair (Age, Balance) as a point in two dimensional space, an association rule of the form  $((Age, Balance) \in P) \Rightarrow (CardLoan = Yes)$  implied that customers whose ages and balances fell in a

planar region  $P$  tended to use card loan with a high probability. The authors proposed several efficient algorithms that gave optimal association rules for gain, support and confidence.

The algorithms presented in [6] provided an efficient and scalable parallel computation for mining association rules. The Intelligent Data Distribution algorithm efficiently used aggregate memory of the parallel computer by employing intelligent candidate partitioning schema and used efficient communication mechanism to move data among the processors. The Hybrid Distribution algorithm further improved upon the Intelligent Data Distribution algorithm by dynamically partitioning the candidate set to maintain good load balance. The evaluation showed that Hybrid Distribution algorithm scaled linearly and exploited the aggregate memory better and could generate more association rules with a single scan of database per pass

The problem of lack of complete and stepwise case study on the KDD process using the suggested stages by [7] was addressed. The steps in KDD process including defining study scope, data selection, data cleaning, coding, data mining and reporting were demonstrated with examples in the case study.

#### *On Frame metadata model*

[8] investigated architecture of a universal data warehousing for the connectivity of relational and OO data model using an ORDBMS. A frame metadata model was chosen to represent the conceptual and logical schema of the universal data warehouse, which structured an application domain into classes, and its data in relational tables. The universal data warehouse using an ORDBMS offered a relational and an OO view for the data warehouse to accommodate different types of queries efficiently. [9] proposed a frame metadata model approach to integrate existing databases and evolve them to support new database applications. This facilitated an evolutionary approach to integrate existing databases to support new applications.

### **3 Frame metadata model**

A frame metadata model [10] consists of two classes: static classes and active classes. Static classes represent factual data entities and active classes represent rule entities. An active class is event driven, obtaining data from the database when invoked by a certain event. The static class stores data in its own database. Frame metadata model captures the semantics of heterogeneous relational schemas after schema translation. With an object frame model agent, frame metadata model can be processed with an object-oriented view. The frame metadata model handles complex data such as class, constraints and methods. The object agent pre-compiles methods and stores methods as stored procedures, and invokes method and constraint class in the frame metadata model which is a metadata consisting of four main parts: header, attributes, methods, and constraints as follows:

```
Header Class {
    Class_name           // an unique name in all system
    Parents              // a list of superclass names
    Operation            // program call for operations
    Class_type           // active or static class}
Attribute Class {
    Attributes_name      // an attribute in this class
    Class_name           // reference to header class
    Method_name         // a method in this class
```

```

Attributes_type // attribute data type
Association attribute // pointer to another class
Default_value // predefined value
Cardinality // single or multi-valued
Description // description of the attributes}
Method Class {
Method_name // a method component in this class
Class_name // reference to header class
Parameters // Number of arguments for the method
Method_type // return type of method
Condition // the rule conditions
Action // the rule actions}
Constraint Class {
Constraint_name // a constraint component of this class
Class_name // reference the header class
Method_name // constrict method name
Parameters // Number of argument for the method
Ownership // the class name of method owner
Event // triggered event: create update or delete
Sequence // method action time: before or after
Timing // the method action timer}

```

#### 4 Non-stop Data mining association rules

In data mining, source database is continuously updated even though a set of association rules have been derived from the source relation. As a result, the derived association rules will be out-dated soon. In order to maintain the current status of the association rules, we need to update the statistical analysis of the association rules continuously whenever the source relation is being updated. This can be accomplished by frame metadata model as shown below:

*Algorithm for discovering association rules continuously*

```

Begin
  Open source relation;
  Select target attributes for statistical analysis;
  Select associated parameter attributes for statistical analysis;
  Input minimum confidence and support for association rules by the users;
  Compute record count of source relation;
  For each parameter attribute do
  Begin
    Compute record count of parameter attribute;
    Compute record count of target attribute with its associated parameter attribute;
    Compute support = ((target and associated parameter attribute record count) /
                      (source relation record count)) * 100%
    Compute confidence = ((target and associated parameter attribute record count) /
                          (parameter attribute record count)) * 100%
    If computed support ≥ minimum requested support
      And computed confidence ≥ minimum requested confidence
      Then associate rule: parameter attribute(s) => target attribute
    End if
  End
End

```

End

We can implement the algorithm by frame metadata model such that the record count will be computed by the attribute Action in its method class. The operation can be triggered by its constraint class whenever there is an update to the source relation.

For example, given source relation and its target attribute A and parameter attributes B, C, D, E.

Source relation  $R_S$

A	B	C	D	E
a1	b1	c1	d1	e1
...	...	...	...	...
an	bn	cn	dn	en

We can then create an association rule table AR to hold its record count on Date  $Date_1$  as follows:

Association Rule table AR

Date	Rule	Parameter count	Target and parameter count	Association rule
$Date_1$	Rule <sub>1</sub>	Count(B)	Count(B, A)	$B \Rightarrow A$
$Date_1$	Rule <sub>2</sub>	Count(C)	Count(C, A)	$C \Rightarrow A$
$Date_1$	Rule <sub>3</sub>	Count(D)	Count(D, A)	$D \Rightarrow A$
$Date_1$	Rule <sub>4</sub>	Count(E)	Count(E, A)	$E \Rightarrow A$
$Date_1$	Rule <sub>5</sub>	Count(B, C)	Count(B, C, A)	$B, C \Rightarrow A$
$Date_1$	Rule <sub>6</sub>	Count(B, D)	Count(B, D, A)	$B, D \Rightarrow A$
$Date_1$	Rule <sub>7</sub>	Count(B, E)	Count(B, E, A)	$B, E \Rightarrow A$
$Date_1$	Rule <sub>8</sub>	Count(C, D)	Count(C, D, A)	$C, D \Rightarrow A$
$Date_1$	Rule <sub>9</sub>	Count(C, E)	Count(C, E, A)	$C, E \Rightarrow A$
$Date_1$	Rule <sub>10</sub>	Count(D, E)	Count(D, E, A)	$D, E \Rightarrow A$
$Date_1$	Rule <sub>11</sub>	Count(B, C, D)	Count(B, C, D, A)	$B, C, D \Rightarrow A$
$Date_1$	Rule <sub>12</sub>	Count(B, C, E)	Count(B, C, E, A)	$B, C, E \Rightarrow A$
$Date_1$	Rule <sub>13</sub>	Count(B, D, E)	Count(B, D, E, A)	$B, D, E \Rightarrow A$
$Date_1$	Rule <sub>14</sub>	Count(C, D, E)	Count(C, D, E, A)	$C, D, E \Rightarrow A$
$Date_1$	Rule <sub>15</sub>	Count(B, C, D, E)	Count(B, C, D, E, A)	$B, C, D, E \Rightarrow A$

To implement the above example for association rule derivation, we can use frame metadata model to store the record count of the target attribute and its associated attributes as follows:

Class AR in the Frame Metadata model for association table:

Header Class

Class Name	Parents	Operation	Class Type
AR	0	Call Rule <sub>1</sub> ..... Call Rule <sub>15</sub>	Active

Attribute class

Attribute_name	Class_name	Method_name	Attribute_type	Associate_attribute	Default_value	Cardinality	Description
----------------	------------	-------------	----------------	---------------------	---------------	-------------	-------------

Rule	AR		Integer				
Parameter count	AR		Integer				
Rule count	AR		Integer				
Association rule	AR		String				

Constraint class

Constraint_Name	Method_name	Class_Name	Parameter	Ownership	Event	Sequence	Timing
Rule <sub>1</sub>	Rule <sub>1</sub>	AR	δR	Self	Insert	After	repeat
.....	.....	.....	...	...	.....	.....	.....
Rule <sub>15</sub>	Rule <sub>15</sub>	AR	δR	self	Insert	After	repeat

Method class

Method_name	Class_name	Parameter	Method_type	Condition	Action
Rule <sub>1</sub>	AR	R <sub>s</sub> , δR, @A, @B, rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B) ≠null	Update AR set count(B) =count(B)+1 Set count(B,A)=count(B,A)+1 where rule=Rule <sub>1</sub>
Rule <sub>2</sub>	AR	R <sub>s</sub> , δR, @A, @C, rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.C=@C) ≠null	Update AR set count(C) =count(C)+1 Set count(C,A)=count(C,A)+1 where rule=Rule <sub>2</sub>
Rule <sub>3</sub>	AR	R <sub>s</sub> , δR, @A, @D, rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.D=@D) ≠null	Update AR set count(D) =count(D)+1 Set count(D,A)=count(D,A)+1 where rule=Rule <sub>3</sub>
Rule <sub>4</sub>	AR	R <sub>s</sub> , δR, @A,@E, rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.E=@E) ≠null	Update AR set count(E) =count(E)+1 Set count(E,A)=count(E,A)+1 where rule=Rule <sub>4</sub>
Rule <sub>5</sub>	AR	R <sub>s</sub> , δR, @A,@B, @C, Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.C=@C)≠null	Update AR set count(B,C) =count(B,C)+1 Set count(B,C,A) = count(B,C,A)+1 where rule=Rule <sub>5</sub>
Rule <sub>6</sub>	AR	R <sub>s</sub> , δR, @A,@B, @D, Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.D=@D)≠null	Update AR set count(B,D) =count(B,D)+1 Set count(B,D,A) = count(B,D,A)+1 where

					rule=Rule <sub>6</sub>
Rule <sub>7</sub>	AR	R <sub>s</sub> , δR, @A,@B, @E, Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.E=@E)≠null	Update AR set count(B,E) =count(B,E)+1 Set count(B,E,A) = count(B,E,A)+1 where rule=Rule <sub>7</sub>
Rule <sub>8</sub>	AR	R <sub>s</sub> , δR, @A,@C, @D Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.C=@C and δR.D=@D)≠null	Update AR set count(C,D) =count(C,D)+1 Set count(C,D,A) = count(C,D,A)+1 where rule=Rule <sub>8</sub>
Rule <sub>9</sub>	AR	R <sub>s</sub> , δR, @A,@C, @E Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.C=@C and δR.E=@E)≠null	Update AR set count(C,E) =count(C,E)+1 Set count(C,E,A) = count(C,E,A)+1 where rule=Rule <sub>9</sub>
Rule <sub>10</sub>	AR	R <sub>s</sub> , δR, @A,@D, @E, Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.D=@D and δR.E=@E)≠null	Update AR set count(D,E) =count(D,E)+1 Set count(D,E,A) = count(D,E,A)+1 where rule=Rule <sub>10</sub>
Rule <sub>11</sub>	AR	R <sub>s</sub> , δR, @A,@B, @C,@D Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.C=@C and δR.D=@D)≠null	Update AR set count(B,C,D) =count(B,C,D)+1 Set count(B,C,D,A) = count(B,C,D,A)+1 where rule=Rule <sub>11</sub>
Rule <sub>12</sub>	AR	R <sub>s</sub> , δR, @A,@B, @C,@E Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.C=@C and δR.E=@E)≠null	Update AR set count(B,C,E) =count(B,C,E)+1 Set count(B,C,E,A) = count(B,C,E,A)+1 where rule=Rule <sub>12</sub>
Rule <sub>13</sub>	AR	R <sub>s</sub> , δR, @A,@B, @D,@E Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.B=@B and δR.D=@D and δR.E=@E)≠null	Update AR set count(B,D,E) =count(B,D,E)+1 Set count(B,D,E,A) = count(B,D,E,A)+1 where rule=Rule <sub>13</sub>
Rule <sub>14</sub>	AR	R <sub>s</sub> , δR, @A,@C, @D,@E Rule	Tuple	If (Select * from R <sub>s</sub> where δR.A=@A and δR.C=@C and δR.D=@D and δR.E=@E)≠null	Update AR set count(C,D,E) =count(C,D,E)+1 Set count(C,D,E,A) = count(C,D,E,A)+1 where rule=Rule <sub>14</sub>
Rule <sub>15</sub>	AR	R <sub>s</sub> , δR,	Tuple	If (Select * from R <sub>s</sub>	Update AR

		@A,@B, @C,@D, @E Rule		where $\delta R.A=@A$ and $\delta R.B=@B$ and $\delta R.C=@C$ and $\delta R.D=@D$ and $\delta R.E=@E \neq \text{null}$	set count(B,C,D,E) =count(B,C,D,E)+1 Set count(B,C,D,E,A) = count(B,C,D,E,A)+1 where rule=Rule <sub>15</sub>
--	--	--------------------------------	--	--	--

As a result, the Support and Confidence level of each rule in the Association Rule table AR can be computed against user requirement in the following algorithm:

*Implementation algorithm for deriving association rule in table AR*

```

Begin
Let n = total record count;
Let Sm = Minimum support level required by the user;
Let Cm = Minimum confidence level required by the user;
For i = 1 to 15 do
Begin
Let Supporti = (Target and parameter Counti / n ) * 100%;
Let Confidencei = (Target and parameter Counti / Parameter Counti) * 100%;
If Supporti ≥ Sm and Confidencei ≥ Cm
Then Rulei (association rule) is derived
End
End

```

*Discovering association rules incrementally*

For large volume of source relation, the performance of data mining of association rules generation may be slow. We cannot afford updating association rule table continuously. Instead, we have to derive association rules incrementally by storing the record count of the previous computing date into the Association Rule Table and adding the new record counts during the new computing date as follow:

Source relation R<sub>S</sub> with timestamp at Date<sub>1</sub> and Date<sub>2</sub>.

Date	A	B	C	D	E
Date <sub>1</sub>	a1	b1	c1	d1	e1
Date <sub>1</sub>	...	...	...	...	...
Date <sub>1</sub>	an	bn	cn	dn	en
Date <sub>2</sub>	a1	b1	c1	d1	e1
Date <sub>2</sub>	...	...	...	...	...
Date <sub>2</sub>	am	bm	cm	dm	em

We can then create the association rule table AR to hold its record count on Date<sub>2</sub> by changing the date from Date<sub>1</sub> to Date<sub>2</sub> in Association Rule table AR. We can use the same calculation as described in computing association rule continuously, except the total record count of source relation is n + m.

*Discovering association rules on inter-table relationships*

We can also focus the source database of KDD on inter-table dependencies, such as foreign key relationships and inclusion dependencies. In other words, the source database can be multiple relations instead of single relation. In order to apply the same algorithm to discover their association rules, we have to join these tables into a universal relation as

follows:

$$R_s = R_{S1} \bowtie R_{S2} \dots \bowtie R_{Sn}$$

For example, given source relation  $R_{S1}$  and its target attribute  $A$  and parameter attributes  $B, C, D, E$ , and their association parameter attribute  $F$  in source relation  $R_{S2}$ , then the source relation  $R_s = R_{S1} \bowtie R_{S2}$  as follow:

Source relation  $R_{S1}$

<u>A</u>	B	C	D	E
A1	b1	c1	d1	e1
...	...	...	...	...
An	bn	cn	dn	en

Source relation  $R_{S2}$  where \*A is a foreign key

<u>F</u>	G	H	I	J	*A
f1	g1	h1	i1	j1	a1
...	...	...	...	...	...
fn	gn	hn	in	jn	an

Derived source relation  $R_s$  after joining the above two tables are:

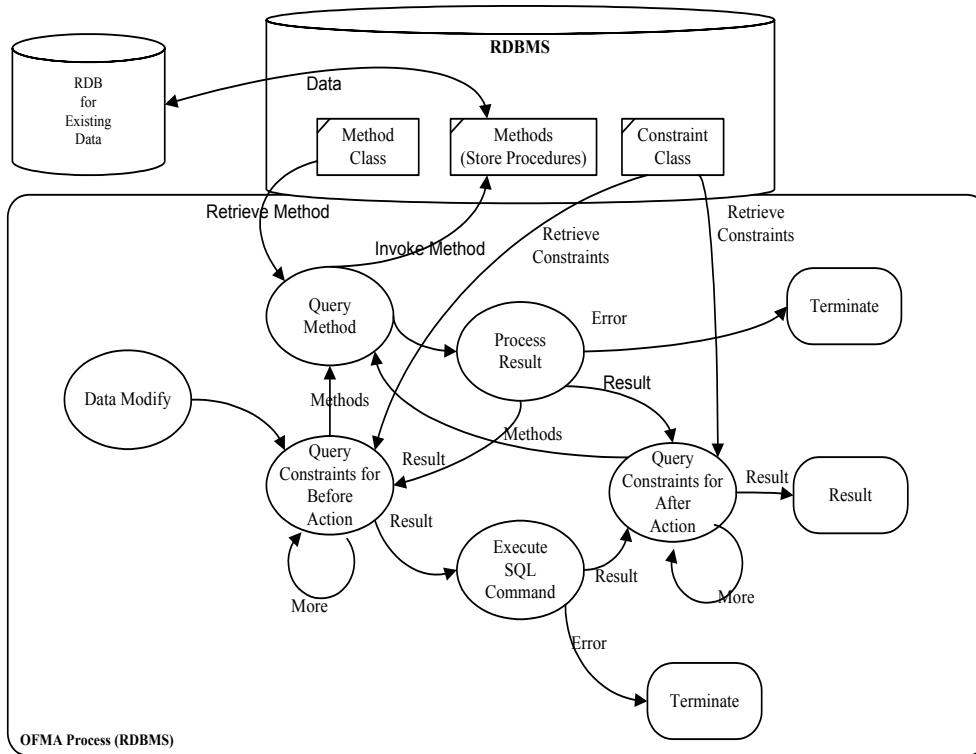
<u>A</u>	B	C	D	E	<u>F</u>
A1	b1	c1	d1	e1	f1
...	...	...	...	...	...
An	bn	cn	dn	en	fn

In real life one can find foreign key relationships between columns whose names and even types are different. In working practice a DBA is frequently confronted with the task of reconstructing foreign key relationships, and needs to make use of various sources of information such as:

- The data dictionary of the database (i.e. metadata)
- Design documentation
- The source-code of the application
- Interview with developers.

## 5 Prototype

A prototype has been implemented for frame object agent model, which is an Application Program Interface developed to invoke the data operation in the frame model for data semantics management. The API process handles the method defined by Constraint class during data modification. The process defined and executed by RDBMS provides triggering event fired on the class. When RDBMS process is invoked, it will check from the Constraint class for any associated method. If method is defined in the Constraint class, the method definition will be queries. The process will invoke the stored procedure defined by the queried method. If the executed result returned by stored procedure violated the rule defined in Constraint class, an error message will be returned to the user (refer to Figure 2).



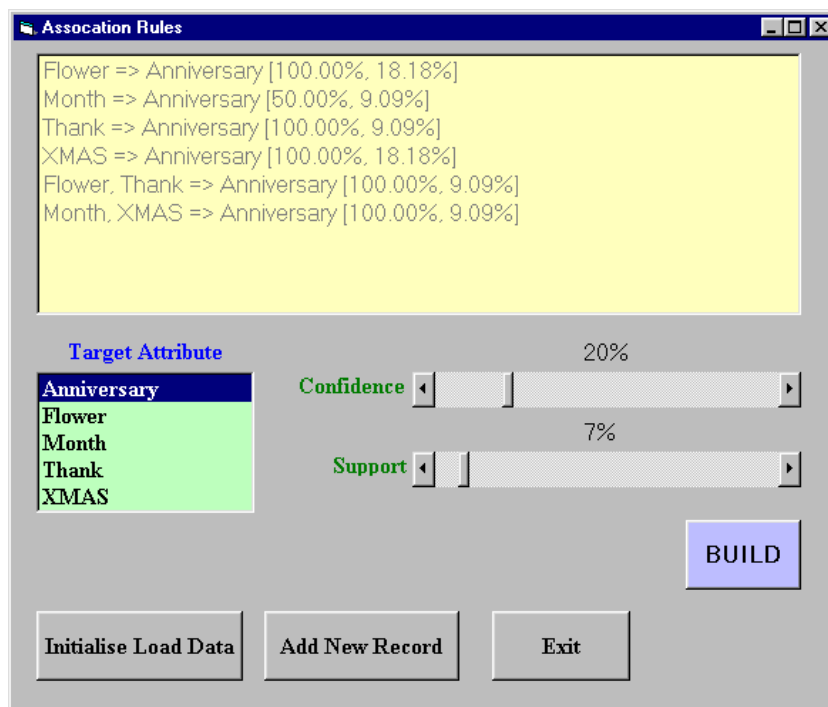
**Figure 2 Data flow of frame object agent model API**

In the following section, we will demonstrate the process of data mining. The situation is that a local church has a donation system to handle donation received from church members and external donors. There are many different donation types. The money collected in each donation type will be used for different purposes. For example, Monthly Donation is used for administration purpose while Christmas Donation supports celebration activities of Christmas in church. Now, suppose the church-in-charge wants to encourage donors to participate in more donation items. Different groups of donors should be identified in order to promote and suggest the donation types for them. The size of the database is about 3000 records.

First, we must define the scope of the study. It will be determined by what kind of knowledge is interested and what kind of the output is expected. For this case, the scope of the study is:

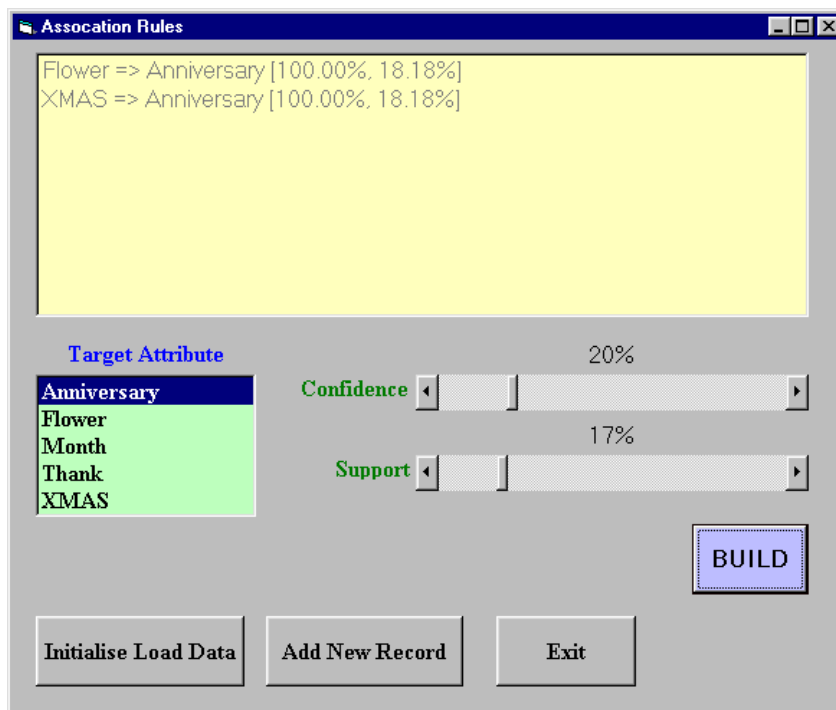
*To identify potential donor on the donation types from existing donation history.*

In Figure 3, we have selected Anniversary as our target attribute. We are interested in the donor who gives anniversary donation. The confidence and support levels are set 20% and 7% which means that we are not interested in sub-groups smaller than 7% of the database, and with that sub-groups we want to find associations that hold for at least 20% of the records.

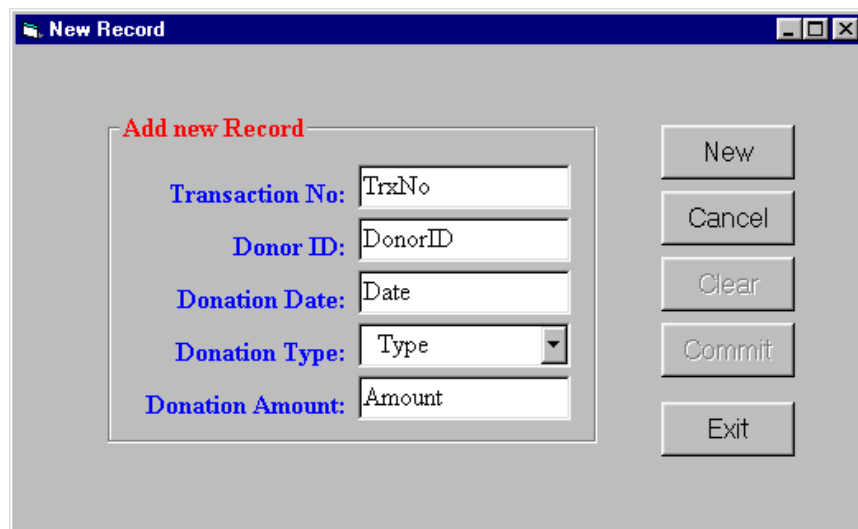


**Figure 3: All significant association rules for the target attribute anniversary (Confidence = 20% and Support = 7%)**

If the Confidence and Support levels change, the association rules become as shown in Figure 4.



**Figure 4: All significant association rules for the target attribute anniversary (Confidence = 20% and Support = 17%)**



**Figure 5: Add new tuple/record API.**

## 6 Conclusion

Data mining in general involves large volume of data such that it is time consuming to process historical data whenever same request is made by the users to generate association rules. This paper presents a solution to the problem by using a frame metadata model to invoke a record count program in a metadata whenever a relevant new record is created in the target database. As a result, data mining association rules can be accomplished online or incrementally without the need of re-computing the historical data record counts. The metadata thus contributes saving of computing time and also the non-stop automation of data mining association rules. The future research of this paper is web mining association rules via the Internet.

## References

- [1] Chen, Q., Hsu, M. and Dayal, U., *A Data Warehouse/OLAP Framework for Scalable Telecommunication Tandem Traffic Analysis*, a report of Software Technology Laboratory, HP Labs, 1501 Page Mill Road, MS 1U4, Palo Alto, CA 94303, USA
- [2] M. Mohania, S. Madria and Y. Kambayashi, *Self-Maintainable Aggregate Views*, Proceedings of the 9<sup>th</sup> International Database Conference, City University of Hong Kong Press, ISBN 962-937-046-8, p.306-317.
- [3] T.Griffin and L. Libkin, *Incremental maintenance of views with duplicates*, Proceedings of the International Conference on Management of Data, 1995.
- [4] Sunita Sarawagi, Shiby Thomas and Rakesh Agrawal, *Integrating Association Rule Mining With Relational Database Systems: Alternatives and Implications*, ACM 1998, pp343-354.
- [5] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita and Takeshi Tokuyama, *Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization*, ACM 199, pp13-23.
- [6] Eui-Hong (Sam) Han, George Karypis and Vipin Kumar, *Scalable Parallel Data Mining for Association Rules*, ACM 1997, pp277-288.
- [7] Pieter Adriaans and Dolf Zantinge, *Data Mining*, Addison Wesley, ISDN 0-201-40380-3.

- [8] Fong, J., Huang, S., *Architecture of a Universal Database: A Frame Model Approach*, International Journal of Cooperative Information Systems, Volume 8, Number 1, March 1999, pp.47-82.
- [9] Fong, J. and Pang, F., *Schema Evolution for New Database Applications: A Frame Metadata model Approach*, Proceedings of Systems, Cybernetics and Informatics, Volume 5, 1999, pp. 104-111.
- [10] Fong, J., Huang, S., *Information Systems Reengineering*, Springer Verlag, ISBN 981-3083-15-8, 1997, pp179-212