Database Support for Data Mining Patterns

Evangelos Kotsifakos, Irene Ntoutsi, and Yannis Theodoridis

Department of Informatics, University of Piraeus,
80 Karaoli-Dimitriou St, GR-18534 Piraeus, Greece
{ek, ntoutsi, ytheod}@unipi.gr

Abstract. The need of extracting useful knowledge from large collections of data has led to a great development of data mining systems and techniques. The results of data mining are known as patterns. Patterns can also be found in other scientific areas, such as biology, astronomy, mathematics etc. Today requirements impose the need for a system that efficiently manipulates complex and diverse patterns. In this work, we study the problem of the efficient representation and storage of patterns in a so-called pattern-base Management System. Towards this aim we examine three well known models from the database domain, the relational, the object-relational and the semi-structured (XML) model. The three alternative models are presented and compared based on criteria like generality, extensibility and querying effectiveness. The comparison shows that the semi-structure representation is more appropriate for a pattern-base.

1 Introduction

Data mining comprises a step of the knowledge discovery process and is mainly concerned with methodologies for extracting knowledge artifacts, i.e. patterns, from large data repositories. Decision trees, association rules, clusters are some well known patterns coming from the data mining area. Patterns can also be found in other areas, such as Mathematics (e.g. patterns in sequences, in numbers, in graphs, in shapes etc.), Geometry, Signal Processing etc. [11]. Nowadays, databases are huge, dynamic, come from different application domains and a lot of different and complex patterns can be extracted from those. In order for someone to be able to exploit the information these patterns represent, an efficient and global (general) Pattern Base Management System (PBMS) for handling (storing / processing / retrieving) patterns is becoming necessary for a lot scientific areas apart from data mining [8]. Scientists of every field have their special needs for pattern creation and management and a PBMS approach would be the solution to the custom-per-problem application that they have to build.

The area of pattern representation and management is recent, and there are only few efforts. PMML [7], SQL/MM [4], CWM [2], JDMAPI [5], PQL [3] are systems developed for storing data mining and statistical patterns. PMML of the data mining Group (DMG) is the most popular approach. Using XML documents it provides a quick and easy way for applications to define predictive models and share these models between PMML compliant applications. PMML defines a variety of specific mining patterns such as decision trees, association rules, neural networks etc. but does not support custom pattern types. PMML version 3.0 provides more patterns and some functions for data preprocessing [7].

The above approaches concentrate mostly on the definition of data mining and statistical models-patterns and the exchange of a set of patterns with specific characteristics between applications rather than on the creation of a general system for the representation and management of different pattern types. Pattern storage and querying techniques as well as pattern-to-data mapping are not among their capabilities.

Recently, two research projects, CINQ [1] and PANDA [10], defined the problem of pattern storage and management and proposed some solutions. CINQ aims at studying and developing query techniques for inductive databases, i.e. databases that store the raw data along with the patterns produced by these data collections [1]. On the other hand, PANDA [10] aims to the definition and design of a PBMS for the efficient representation and management of various types of patterns that arise from different application domains (not only from data mining). Patterns will reside and be managed (indexing, querying, retrieving) in the PBMS just like primitive data reside and are managed in the DBMS. Different types of patterns will be efficiently managed (generality) and new pattern types will be easily incorporated (extensibility) in the PBMS. A very critical decision regarding to the PBMS is whether it should be build from scratch or as an additional layer on top of a DBMS.

The scope of this work is to deal with the problem of pattern representation and storage following the later approach (i.e. working on top of a DBMS). Towards this aim, we examine three well known DBMS approaches: the relational, the object-relational and the semistructured (XML) model.

2 Patterns and Pattern-Bases

We adopt the PANDA project approach as it tries to incorporate all kinds of patterns. The pattern concept is the cornerstone of the PBMS. A *pattern* is a compact and rich in semantics representation of raw data. A *pattern-base* is a collection of persistently stored patterns. A *PBMS* is a system for handling patterns, defined over raw data and organized in pattern-bases, in order to efficiently support pattern matching and to exploit pattern-related operations generating nontrivial information [10]. A PBMS treats patterns just like a DBMS treats raw data.

In order to efficiently manage patterns, a PBMS should fulfill some requirements [10]:

- *Generality*: The PBMS must be able to manage different types of patterns coming from different application domains.
- *Extensibility*: The PBMS must be extensible to accommodate new kinds of patterns introduced by novel and challenging applications.
- *Exploitation of patterns special characteristics:* The PBMS should take into account the special features of patterns so as to improve several operations, like indexing and query processing.

- *Constraint implementation:* The PBMS should implement the constraints defined in the logical pattern model as well as validate patterns in line with these constraints.
- *Reusability:* PBMS must include constructs encouraging the reuse of what has already been defined.

The PANDA consortium has defined a logical model for the PBMS [8], which consists of three basic entities: pattern type, pattern and class defined as follows:

Definition 1. (Pattern Type): A pattern type is a quintuple *pt* = (*n*, *ss*, *ds*, *ms*, *f*), where *n* is the pattern type name, *ss* is the structure schema that describes the structure of the pattern type (in an association rule for example the structure consists of head and body), *ds* is the source schema that describes the dataset from which patterns of this pattern type are constructed, *ms* is the measure schema that defines the quality of the source data representation achieved by patterns of this pattern type and *f* is the formula that describes the relationship between the source space and the pattern space.

An example of the association rule pattern type is presented below:

n: AssociationRule
ss: TUPLE(head: SET(STRING), body: SET(STRING))
ds: BAG(transaction: SET(STRING))
ms: TUPLE(confidence: REAL, support: REAL)
f: head U body $\subseteq$ transaction

Definition 2. (Pattern): A pattern *p*, is an instance of a pattern type *pt*, and has the corresponding values for each component. An example of an association rule pattern, instance of the AssociationRule pattern type defined above, is the following:

pid: 413
s: (head={'Boots'}, body={'Socks', 'Hat'})
d: 'SELECT SETOF(article) AS transaction FROM sales GROUP BY transactionId'
m: (confidence=0.75, support=0.55)
e: {transaction: {'Boots', 'Socks', 'Hat'} $\subseteq$ transaction}

Definition 3. (Class): A class *c*, over a pattern type *pt*, is defined as a triple *c* = (*cid*, *pt*, *pc*) where *cid* is the unique identifier of the class, *pt* is the pattern type and *pc* is a collection of patterns of type *pt*.

A class is defined for a given pattern type and contains only patterns of that type. Each pattern must belong to at least one class. The relationships between the three basic entities of a PBMS, i.e. pattern types, patterns and classes, are shown in the figure below:

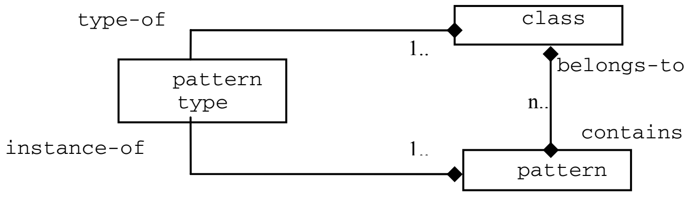

Fig. 1. Relationships between pattern types, patterns and classes

3 Physical Representation in a Pattern-Base

For the representation and storage of the contents of a pattern-base, we examine three traditional DBMS approaches: the relational, the object-relational and the semistructured (XML) model using the entities presented in the previous section.

Next, we present each approach and give some representative queries that point out the advantages and disadvantages of each one. For the implementation we have used Oracle 9i DBMS. This comparison aims to examine the applicability of the logical model in current DBMS technology and is based on qualitative rather than quantitative criteria. The primary goal is to examine whether a PBMS can be built based on the three models presented and which one is the more efficient on supporting the patterns special characteristics.

3.1 Relational Approach

Our main concern during the design and implementation of the pattern-base was to satisfy the three basic requirements of the logical model: generality, extensibility and pattern characteristics exploitation [10]. The relational schema is depicted in Fig 2:

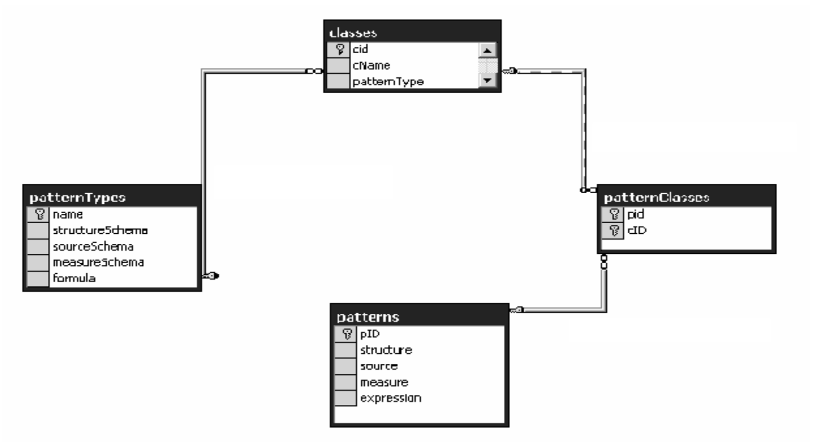

Fig. 2. The relational schema of the pattern-base

Various pattern types are stored in the table *patternTypes*, patterns are stored in the table *patterns* and pattern classes are stored in table *classes*. The table *patternClasses* relates patterns with classes (a class contains one or more patterns of the same class and every pattern belongs to at least one class).

Below we present only some representative queries due to space limitations. The queries will be first described in natural language and then in SQL-like syntax:

RQ1) *Find the structure (respectively, the source, the measure or the expression) of the association rules belonging to class Association_Rule_1.*

```
select patterns.structure from classesr
inner join patternclasses on classesr.cid = patternclasses.cid
inner join patterns on patternclasses.pid = patterns.pid
where (classesr.cname='Association_Rule_1');
```

RQ2) *Find all association rules belonging to class Association_Rule_1 whose structure contains "org" or whose coverage is greater than 0.7.*

```
select pid from classesr
inner join patternclasses on classesr.cid = patternclasses.cid
inner join patterns on patternclasses.pid = patterns.pid
where (classesr.cname='Association_Rule_1') AND
(INSTR(SUBSTR(structure,INSTR(measure,'body'),length(measure)),'ORG')
>0 OR substr(measure,10,instr(measure,'(')-10)>0.007);
```

RQ3) *Return the head and body parts of the structure of patterns that they belong to class Association_Rule_1.*

```
Select Substr(structure,1,instr(structure,'body')-2) as head,
Substr(structure,instr(structure,'body')) as body from classesr
inner join patternclasses on classesr.cid = patternclasses.cid
inner join patterns on patternclasses.pid = patterns.pid
where (classesr.cname='Association_Rule_1');
```

The relational approach is characterized by simplicity and ease of implementation. However, it has a lot of disadvantages that arise from the fact that this approach does not take into account the underlying structure of pattern components (structure, measure, etc.) and treats them as simple texts/ strings. This fact makes querying a complex, time consuming and mostly ineffective process.

3.2 Object-Relational Approach

The object-relational model manages to deal with the basic drawback of the relational model, by defining different objects and attributes for each pattern component and exploiting inheritance. In that way it is less complex and more efficient since querying is simpler.

The basic idea of the object-relational model (a part of it) is depicted in the following schema. At the root of the object relational model stands the *Pattern* entity, which contains generic information about the pattern, such as the pattern identifier, the pattern formula and the pattern source. At the next level of the tree, the *Pattern* is specialized, according to the pattern type it belongs to, for example to association rules patterns, to clusters patterns etc. These entities differ according to their structure and measure components but they also have some attributes in common, those inherited by the *Pattern* entity. For example, object *Association Rule Pattern* contains every attribute from object *Pattern* and it also has the attribute *Structure* that consists of a *head* and a *body*. This object can be further specialized based on the measure component. As it seems in Fig. 3 in the object *Association Rule Pattern 1* the *Measure* component consists of *confidence* and *support*, whereas in the object *Association Rule Pattern 2* the *Measure* component consists of *coverage*, *strength*, *lift* and *leverage*.

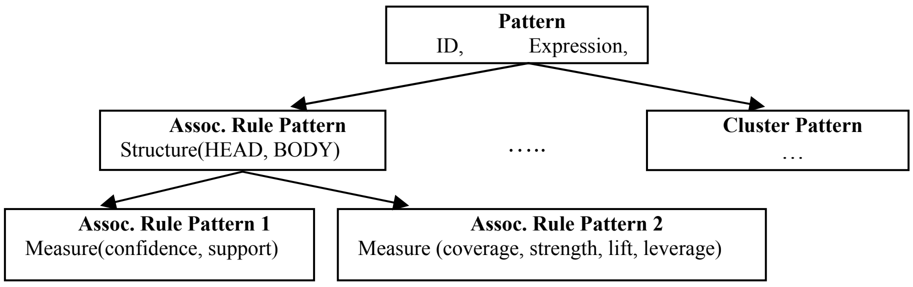

Fig. 3. The basic idea of the object-relational approach

Below we present some representative queries for the object-relational model:

OQ1) *Find the expression (respectively, the ID, the source, the measure or the structure) of patterns.*

```
select expression from hr.tbl_patterns p;
```

OQ2) *Find the body of the structure of association rule patterns*

```
select p.id, value(e) from hr.tbl_patterns p,
table(treat(value(p) as hr.assrule_pattern).structureschema.body) e ;
```

OQ3) *Find the confidence of the measure of association rule patterns.*

```
select p.id, treat(value(p) as
hr.assrule_pattern_1).measureschema.confidence as
confidence from hr.tbl_patterns p;
```

The object-relational approach overcomes some of the relational approach limitations due to the capability of modeling complex entities as objects. It also exploits the similarities among objects through inheritance. The object-relational model is more flexible and efficient from the relational model but it requires exact definition of any new object and of its components.

3.3 Semi-structured (XML) Approach

Unlike traditional databases, in an XML base the format of the data is not so rigid. This property is valuable in our case since patterns come from different application fields having thus different characteristics. For the XML implementation, we have to create an XML schema for each pattern type. Patterns of a specific pattern type will be the XML documents (instances) of the XML schema of this type.

For example, the association rules pattern type is described through the schema “association_rule.xsd” (Fig. 4), whereas the XML document “pattern-association_rules.xml” (Fig. 5) contains patterns of the association rule pattern type schema.

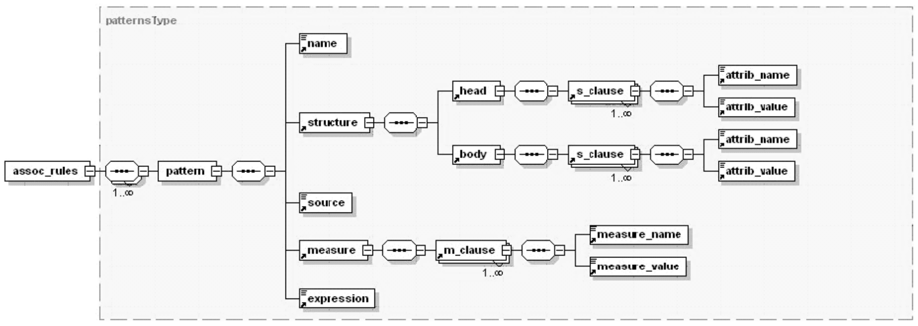

Fig. 4. association_rule.xsd

```
<assoc_rules ptype="association_rule">
            <pattern id="1"> <name>rule 1</name>
              <structure>
                <head>
                  <s_clause>
                    <attrib_name>buys</attrib_name>
                    <attrib_value>scarf</attrib_value>
                  </s_clause>
                </head>
                <body>
                  <s_clause>
                    <attrib_name>buys</attrib_name>
                    <attrib_value>gloves</attrib_value>
                  </s_clause>
                </body>
              </structure>
              <source>SELECT * FROM orders</source>
              <measure>
                <m_clause>
                  <measure_name>support</measure_name>
                  <measure_value>0.35</measure_value>
                </m_clause>
                <m_clause>
                  <measure_name>confidence</measure_name>
                  <measure_value>0.75</measure_value>
                </m_clause>
              </measure>
             <expression>
             {buys="hat",buys="cap",buys="gloves"}
             </expression>
            </pattern>
```

Fig. 5. association_rule.xml

Below we present some representative queries for the XML model in ORACLE XML-SQL syntax:

XQ1) *Find the structure (respectively, the source, the measure or the expression) of the association rule patterns belonging to class "class1".*

```
select
 extract(value(y),'//pattern[@id="'||extract(value(e),
'pid/text()')||'"]/structure') as structures from assoc_rules y,
classes x, TABLE(XMLsequence(extract(value(x),
'class[@name="class1"]//pids/pid'))) e where exists-
Node(value(y),'//pattern[@id="'||extract(value(e),'pid/text()')||'"]/
structure') = 1
```

XQ2) *Return all patterns of a specific pattern type.*

```
select distinct extracValue(value(y),'//pattern[@id="'||
extract(value(e),'pid/text()')||'"]/name/text()') as pattern_name
from assoc_rules y, classes x,
TABLE(XMLsequence(extract(value(x),
'class[@ptype="association_rule"]//pids/pid'))) e
```

XQ3) *Find all the different measures(inside the measure component) of the association rules.*

```
select distinct extractValue(value(r),
'//m_clause/measure_name/text()') as measures from assoc_rules y,
classes x,TABLE(XMLsequence(extract(value(x),'//pids/pid'))) e,
TABLE(XMLsequence(extract(value(y),'//pattern[@id="'||
extract(value(e),'pid/text()')||'"]//m_clause'))) r;
```

XQ4) *Find patterns with the maximum value of the measure lift from patterns belonging to "class1".*

```
select extract(value(aa),'//pattern/name/text()') as pattern_names
from assoc_rules aa, (select max(extractvalue(value(val),'//text()'))
as maximum_lift from assoc_rules a,
TABLE(xmlsequence(extract(value(a),'//m_clause[measure_name="Lift"]
/measure_value'))) val) xxx
where existsNode(value(aa),
'//m_clause[measure_name="Lift"][measure_value="'||
xxx.maximum_lift||'"]') = 1
```

With XML pattern-base, the definition of a new pattern type is easy (extensibility). Furthermore, it is possible to create a proper XML schema for a pattern type, general enough to include every variation of patterns of this type (generality). The XML schema affects also the effectiveness of querying. Queries like "find all the different measures of the association rules", can be easily implemented, unlike the relational and object-relational approaches.

4 A Qualitative Comparison

In this section we present the criteria for the comparison of the three alternative representations and the conclusions we reached.

# 1. Pattern-Base Implementation Complexity

All the three models we presented can be easily implemented. The simplest model is the relational, where both the pattern-base construction and insert operations can be performed in an easy and fast way. The object relational model is slightly more difficult since it requires the definition of different objects for each pattern type (and each of its variations). Insert operations are also more difficult as it should be different for each pattern type and its variations. Finally, the difficulty of the XML model is the fact that its success depends straightforward on the quality (generality) of the XML schema for each pattern type. However, after creating the proper schema insert operations can be easily performed. Furthermore, if this schema is general enough, variations of patterns belonging to a specific pattern type can be easily supported through this pattern type schema.

# 2. Constraint Implementation

The basic constraints imposed by the logical model [8] are the following: (a) every pattern is an instance of one pattern type, (b) every pattern belongs to at least one class, (c) a pattern class should contain only patterns of the same pattern type.
These constraints can be easily implemented in the relational model through the foreign key constraints. In the object relational model these constraints are supported directly by the definition of the pattern type, for example it is impossible to assign a cluster into the association rule pattern type. In the XML model, finally, the implementation of constraints are supported by the DBMS with mechanisms that associate XML documents.

# 3. Pattern Characteristics Exploitation

According to the logical model [8], every pattern consists of five basic components: name, structure, source, measure and formula. However, different pattern types differentiate on some of these components, e.g. in structure or measure. If we exploit the special characteristics of each pattern type we can improve operations like indexing and querying. The relational model does not exploit the underlying structure of patterns as it considers every pattern component as a string. Whereas, both object-relational and XML models take into account the special characteristics of pattern component according to the pattern type.

# 4. Query Effectiveness

The pattern-base does not aim only at the storage of patterns but mainly at their easy management, so the effectiveness of querying is an important criterion. From the representative queries we gave above for each implementation, it is obvious that in the relational model query construction is a complex and time consuming process (it is all about string manipulation formulas). The rest two models exploit the underlying pattern structure, thus queries are expressed more easily.

# 5. Extensibility

Extensibility is the ability to incorporate a new pattern type in the pattern-base; the easier this process is the more extensible the system is. The relational model is very extensible; a new pattern type is simply a new record in the table pattern types. The object-relational model requires the creation of new objects for every new pattern type and its components (the same stands also for the variations of a pattern type). That means that more than one association rule schema maybe required to incorporate the differences in the structure of each association rule. In the XML model a new schema is required for each new pattern type, but on the other hand, since this schema exists and is general enough, variations of patterns of this type can be easily incorporated without any modification.

# 6. Pattern validation

The validity check during insert/ update operations in the pattern-base is critical. With the term validity we mean that each pattern in the pattern-base should follow its pattern type definition. The above criteria is violated in the relational model, whereas it stands for both XML and object relational models because of the XML schemas and the objects' definition respectively.

The conclusions of the evaluation are summarized in the table below:

Table 1. Comparison results

	Relational pattern-base	Object-relational pattern-base	XML pattern-base
Implementation Complexity	High	Medium	High
Constraint Implementation	Yes	Yes	Yes
Pattern characteristics exploitation	No	Yes	Yes
Query effectiveness	Low	Medium	Medium
Pattern validation	No	Yes	Yes
Extensibility	High	Medium	High

From the above table it is clear that the XML pattern-base implementation is the best among the three choices. There are however some points (e.g. query effectiveness) that it is not so efficient. This raises the question whether a special querying language designed exclusively for patterns is needed, like the one proposed in [9].

5 Conclusions and Future Work

Since patterns are compact and rich in semantics representations of raw data [10], they share some common characteristics, but also differentiate according to the type they belong to. Moreover, there are also variations between patterns of the same type.

Patterns nature requires a data-oriented approach whereas traditional databases follow a structure-oriented direction. For the pattern representation problem a semi-structured model is more appropriate than a relational or an object-relational schema. Using XML for the implementation of the pattern-base, we could achieve to build a more complete and general PBMS. There are some points however, where XML suffers such as query efficiency. To deal with efficiency problems a composite model that will be based on both XML and object-relational models should be examined or XML query methods should be developed. Although PMML is an XML-based language and tends to support more and more pattern types, a more general aspect should be adopted. Patterns should be defined per application or scientific area, so the system should be open to user extensions. Pattern querying and data-to-pattern mapping are issues that PMML is not currently taking into account, though important in order to create a more complete PBMS.

References

1. CINQ (Consortium on Discovering Knowledge with Inductive Queries). http://www.cinq-project.org.
2. CWM (Common Warehouse Model). http://www.omg.org/cwm.
3. Information Discovery Data Mining Suite. http://www.patternwarehouse.com/dmsuite.htm.
4. ISO SQL/MM Part 6. http://www.sql-99.org/SC32/WG4/Progression_Documents/FCD/fcd-datamining-2001-05.pdf, 2001.
5. Java Data Mining API, http://www.jcp.org/jsr/detail/73.prt.
6. PANDA (Patterns for Next-generarion Database Systems). http://dke.cti.gr/panda.
7. PMML (Predictive Model Markup Language). http://www.dmg.org/pmml-v3-0.html.
8. Rizzi S., Bertino E., Catania B., Golfarelli M., Halkidi M., Terrovitis M., Vassiliadis P., Vazirgiannis M., Vrahnos E.. Towards a logical model for patterns. *Proc. ER Conference*, Chicago, IL, USA, 2003.
9. Terrovitis M., Vassiliadis P., Skiadopoulos S., Bertino E., Catania B., Maddalena A. Modelling and Language Support for the Management of Pattern-Bases. *Proc. SSDBM Conference*, Santorini, Greece, 2004.
10. Theodoridis Y., Vazirgiannis M., et al. A manifesto for pattern bases. PANDA Technical Report TR-2003-03, 2003. Available at http://dke.cti.gr/panda.
11. Vazirgiannis M., Halkidi M., Tsatsaronis G., Vrachnos E., et al. A Survey on Pattern Application Domains and Pattern Management Approaches. PANDA Technical Report TR-2003-01, 2003. Available at http://dke.cti.gr/panda.