DMQL: A Data Mining Query Language for Relational Databases *

Jiawei Han Yongjian Fu Wei Wang Krzysztof Koperski Osmar Zaiane
Database Systems Research Laboratory
School of Computing Science
Simon Fraser University, B.C., Canada V5A 1S6
E-mail: {han, yongjian, weiw, koperski, zaiane}@cs.sfu.ca

Abstract

The emerging data mining tools and systems lead naturally to the demand of a powerful data mining query language, on top of which many interactive and flexible graphical user interfaces can be developed. This motivates us to design a data mining query language, DMQL, for mining different kinds of knowledge in relational databases. Portions of the proposed DMQL language have been implemented in our DBMiner system for interactive mining of multiple-level knowledge in relational databases.

1 Introduction

Data mining is a promising field with flourishing R & D activities and successful systems reported recently [17, 5]. Since the tasks and applications of data mining are broad and diverse, it is expected that various kinds of flexible, interactive user interfaces for data mining will emerge.

We believe that the development of successful data mining systems may resemble that of relational systems which have been dominating the database system market for decades. Although there are many different graphical user interfaces in commercial relational systems, its underlying "core" relational query language sets a solid foundation for research and development of relational systems, facilitates information exchange and technology transfer, and promotes commercialization, broad application, and wide acceptance of the technology. In this sense, the success of the relational systems should be credited in part to the standardization of relational query languages, which was done at the early stage in the development of the field [20]. The recent standardization activities in database systems, such as the work related to SQL-3, OMG and ODMG [3], show again the importance of a standard database language in the development and commercialization of future database systems.

This motivates us to examine what should be the primitives of a data mining language. The current data mining R & D activities show that data mining covers a wide spectrum of tasks, from data summarization to mining association rules, data classification, or finding some specific patterns. This makes the design of a comprehensive data mining language a challenging task. Currently, there are many graphical user interfaces for different tasks of data mining. However, we feel that it is important to understand the underlying mechanisms of different data mining methods and construct a general data mining language.

In this paper, we examine the general philosophies which influence the design of such a data mining query language and present step-by-step a tentatively designed Data Mining Query Language, DMQL. The design cannot be claimed complete by any standard. However, it may serve as an interesting example for further discussion.

2 Design of a data mining language: philosophy

The philosophy of data mining may strongly influence the design of a data mining language. We have the following philosophical considerations which will serve as guidelines in our design of a data mining language.

1. The set of data relevant to a data mining task should be specified in a data mining request.

   Since a user may be interested in any portion of data in a database, a data miner should be able to work on any specific subset of data. This implies that a data miner may take a database query as a subtask to first retrieve the relevant set of data before data mining. If a user cannot identify precisely the relevant set of data, a superset of the data can be collected, and certain mechanisms can be developed to help identify or rank the relevant set

*Research was supported in part by the grant NSERC-A3723 from the Natural Sciences and Engineering Research Council of Canada, the grant NCE:IRIS/PRECARN-HMI-5 from the Networks of Centres of Excellence of Canada, and grants from BC/Advanced Systems Institute, the MPR Teltech Ltd., and the Hughes Research Laboratories.

of data and/or attributes. Therefore, a data mining language should take a query language as its subtask in the specification.

2. The kinds of knowledge to be discovered should be specified in a data mining request.

   Ideally, one may expect that a knowledge discovery system will perform interesting discovery autonomously without human instruction or interaction. However, since mining can be performed in many different ways on any specific set of data, huge amounts and different kinds of knowledge may be generated by unguided, autonomous discovery, whereas much of such discovered knowledge could be out of user's interest. Thus, we propose *command-driven data mining*, which specifies both the potentially relevant set of data and the kinds of knowledge to be discovered. This leads to guided discovery of desired kind of knowledge on a relevant set of data and represents constrained search for the desired knowledge.

3. Background knowledge could be generally available for data mining process.

   Discovery may be performed with the assistance of relatively strong background knowledge (such as conceptual hierarchy information, etc.) or with little support of background knowledge. The discovery of conceptual hierarchy information itself can be treated as a part of a knowledge discovery process. However, the availability of relatively strong background knowledge not only improves the efficiency of a discovery process but also expresses user's preference for guided generalization, which may lead to an efficient and desirable generalization process.

4. Data mining results should be able to be expressed in terms of generalized or multiple-level concepts.

   Without concept generalization, discovered knowledge is expressed in terms of primitive data (data stored in the databases), often in the form of functional or multivalued dependency rules, primitive level association rules, or integrity constraints. On the other hand, with concept generalization, discovered knowledge can be expressed in terms of concise, expressive, and high-level or multiple-level abstraction, in the form of generalized rules or generalized constraints, and be associated with statistical information. Obviously, it is often desirable for large databases to have rules expressed at the concept levels higher than the primitive ones.

5. Various kinds of thresholds should be able to be specified flexibly to filter out less interesting knowledge.

   A user may like to flexibly and/or interactively specify various kinds of thresholds which can be used to select desired, interesting rules and filter out less interesting ones in data mining.

3 Mining different kinds of rules

Based on the above considerations, a data mining query language, DMQL, has been designed in our DBMiner project for mining several kinds of knowledge in relational databases. It consists of the specifications of four major primitives in data mining: (1) the set of data in relevance to a data mining process, (2) the kind of knowledge to be discovered, (3) the background knowledge, and (4) the justification of the interestingness of the knowledge (i.e., thresholds).

The first primitive, the set of relevant data, can be specified in a way similar to that of a relational query, which is to be used to fetch the set of relevant data from the database.

The second primitive, the kind of knowledge to be discovered, may include generalized relations, characteristic rules, discriminant rules, classification rules, association rules, etc., which are detailed as follows.

1. A generalized relation is a relation obtained by generalizing from a large set of low level data. A generalized relation can then be used for extraction of different kinds of rules or be viewed at high concept levels from different angles.

2. A characteristic rule is an assertion which characterizes a concept satisfied by all or most of the examples in the class undergoing examination (called the target class). For example, the symptoms of a specific disease can be summarized by a characteristic rule.

3. A discriminant rule is an assertion which discriminates a concept of the class being examined (the target class) from other classes (called contrasting classes). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others.

4. A classification rule is a set of rules which classifies the relevant set of data, which is usually obtained by first classifying the data (i.e., obtaining a preferred classification scheme) and then returning a set of rules associated with each class or subclass. For example, one may classify diseases and provide the symptoms which describe each class or subclass.

5. An association rule describes association relationships among a set of data (patterns). For example,

one may discover a set of symptoms frequently occurring together with certain kinds of diseases and further study the reasons behind it.

The third primitive, the background knowledge, is a set of concept hierarchies or generalization operators which provide corresponding higher level concepts and assist generalization processes. Primitives for the specification of concept hierarchies will be discussed in Section 4.

The fourth primitive, the interestingness or significance of the knowledge to be discovered can be specified as a set of different mining thresholds depending on the kinds of rules to be mined, which will be examined in more detail in Section 3.4.

3.1 Syntax of DMQL for mining different kinds of rules

DMQL adopts an SQL-like syntax to facilitate high level data mining and natural integration with relational query language, SQL.

The DMQL language is defined in an extended BNF grammar, where "[ ]" represents 0 or one occurrence, "{ }" represents 0 or more occurrences, and words in sans serif font represent keywords, as shown below.

```
⟨DMQL⟩ ::=
    use database ⟨database_name⟩
    {use hierarchy ⟨hierarchy_name⟩ for ⟨attribute⟩}
    ⟨rule_spec⟩
    related to ⟨attr_or_agg_list⟩
    from ⟨relation(s)⟩
    [where ⟨condition⟩]
    [order by ⟨order_list⟩]
    {with [⟨kinds_of⟩] threshold = ⟨threshold_value⟩
        [for ⟨attribute(s)⟩]}
```

In ⟨DMQL⟩, "use database ⟨database_name⟩" directs the mining task to a specific database "⟨database_name⟩", and the optional statement, "use hierarchy ⟨hierarchy⟩ for ⟨attribute⟩", assigns ⟨hierarchy⟩ to a particular attribute ⟨attribute⟩ (otherwise, a default hierarchy is used). The statement, ⟨rule_spec⟩, is the specification of the kind of rules to be discovered. For discovering different kinds of rules, the rule specification should be in different formats, which will be presented in detail in the following sections. The related-to statement, "related to ⟨att_or_agg_list⟩", selects a list of relevant attributes and/or aggregations for generalization. The "from" and "where" clauses, "from ⟨relation(s)⟩ [where ⟨condition⟩], form an SQL query to collect the set of relevant data. The "order by" clause simply specifies the order of rows to be printed. The "with-threshold" statement specifies various kinds of thresholds.

For rule specification, the following kinds of rules are considered in DMQL.

1. Data generalization.

```
⟨rule_spec⟩ ::=
  generalize data [into ⟨relation_name⟩]
```

2. Mining characteristic rules.

```
⟨rule_spec⟩ ::=
  find characteristic rules [as ⟨rule_name⟩]
```

3. Mining discriminant rules.

```
⟨rule_spec⟩ ::=
  find discriminant rules [as ⟨rule_name⟩]
  for ⟨class_1⟩ with ⟨condition_1⟩
    from ⟨relation(s)_1⟩
  in contrast to ⟨class_2⟩ with ⟨condition_2⟩
    from ⟨relation(s)_2⟩
  { in contrast to ⟨class_i⟩ with ⟨condition_i⟩
    from relation(s)_i }
```

4. Data classification and mining classification rules.

```
⟨rule_spec⟩ ::=
  find classification rules [as ⟨rule_name⟩]
    [according to ⟨attributes⟩]
```

5. Mining association rules.

```
⟨rule_spec⟩ ::=
  find association rules [as ⟨rule_name⟩]
```

3.2 Specification of interestingness and thresholds

A data mining task may need to specify a set of thresholds to control its data mining process, including guiding an induction process, constraining search for interesting knowledge, testing the interestingness or significance of the discovered knowledge, etc. This requires the introduction of the fourth set of primitives, a set of data mining thresholds, in DMQL.

Different kinds of rule mining may need to specify different kinds of thresholds which can be categorized into at least three classes, as follows.

1. *Significance threshold.* It indicates that there should exist at least some reasonably substantial evidence (support) of a pattern in the data set in order to warrant its presentation. In mining association rules, this threshold is called the *minimum support* [1], and the patterns passing this support threshold are called *large* (or *frequent*) *data items*; whereas in mining characteristic rules, it is called *noise threshold* [7], and the patterns which cannot pass this threshold are treated as noise.

2. *Rule confidence threshold.* It indicates that the probability of B under condition of A in rule $A \rightarrow B$, i.e., $Prob\{B|A\}$, must pass this threshold to make sure that the implication relationship is reasonably strong [1].

3. *Rule redundancy threshold.* It indicates that the rules to be presented are not similar to those already there [19].

The syntax of the threshold specification is as below,

```
with [⟨kinds_of⟩] threshold = ⟨threshold_value⟩
    [for ⟨attribute(s)⟩]
```

where ⟨kinds_of⟩ can be *support, confidence, noise, redundancy*, etc., which will be shown in later examples. Threshold values can be set and modified interactively using similar language primitives.

3.3 Examples of mining different kinds of rules

Example 3.1 Let us examine a university database with the following schema.

```
student(name, sno, status, major, gpa, birth_date,
    birth_place, address)
course(cno, title, department)
grading(sno, cno, instructor, semester, grade)
```

A few data mining query examples are presented in DMQL as follows. Notice that the statements like "use database university_database" is omitted in the example queries.

1. Mining characteristic rules. Query, (q_1), is to find the general characteristics of the graduate students in computing science in relevance to attributes *gpa, birth_place* and *address*, for the students born in Canada.

```
(q1) : find characteristic rule
       related to gpa, birth_place, address, count(*)%
       from student
       where status = "graduate" and major = "cs"
          and birth_place = "Canada"
       with noise threshold = 0.05
```

This data mining query will first retrieve data from the database using a transformed SQL query, where the high level constants "Canada" and "graduate" are transformed into low level primitive concepts in the database according to the provided (default) concept hierarchy for each attribute. The algorithm for finding characteristic rules [7] is then executed with the data generalized to high level for manipulation and presentation. The set of generalized data grouped according to the high level concept values of the attributes *gpa, birth_place* and *address* are presented, associated with the corresponding *count*(∗)% (i.e., the count of tuples in the corresponding group in proportion to the total number of tuples). The noise threshold 0.05 means that a generalized tuple taking less than 5% of the total count will not be included in the final result.

2. Mining discriminant rules. Query (q_2) is to find the discriminant features to compare graduate students versus undergraduate students in computing science in relevance to attributes *gpa, birth_place* and *address*, for the students born in Canada.

```
(q2) : find discriminant rule
       for cs_grads with status = "graduate"
       in contrast to cs_undergrads
          with status = "undergraduate"
       related to gpa, birth_place, address, count(*)%
       from student
       where major = "cs" and birth_place = "Canada"
```

This data mining query will first retrieve data into two classes, "cs_grads" and "cs_undergrads", using a transformed SQL query which maps the high level constants in (q_2) into low level ones. The algorithm for finding discriminant rules [7] is then executed for data mining and result manipulation.

3. Mining classification rules. Query (q_3) is to classify students according to their gpa's and find their classification rules for those majoring in computing science and born in Canada, with the attributes *birth_place* and *address* in consideration.

```
(q3) : find classification rules for cs_students
       according to gpa
       related to birth_place, address
       from student
       where major = "cs" and birth_place = "Canada"
```

This query will first collect the relevant set of data, and then execute some data classification algorithm, such as [14, 21] to classify students according to their gpa's and present each class and its associated characteristics.

4. Mining association rules. Query (q_4) is to find strong association relationships for those students majoring in computing science and born in Canada, in relevance to the attributes *gpa, birth_place* and *address*.

```
(q4) : find association rules
       related to gpa, birth_place, address
       from student
       where major = "cs" and birth_place = "Canada"
       with support threshold = 0.05
       with confidence threshold = 0.7
```

This query will first collect the relevant set of data and then execute an association mining algorithm,

such as [1] or [8], to find a set of interesting association rules. The support and confidence thresholds are specified (otherwise using default values) for mining strong rules. □

4 Specification of concept hierarchies

Concept hierarchy (or lattice) provides useful background knowledge for expressing data mining results in concise, high-level terms. Concept hierarchies can be specified based on database attribute relationships, particular grouping operations, etc. Any given concept hierarchies should be able to be adjusted dynamically based on current data distributions. Moreover, hierarchies for numerical attributes should be able to be constructed automatically based on data distributions.

DMQL handles concept hierarchy specifications as follows.

1. Hierarchy specification at the schema level. Concept hierarchies can be specified at the schema level based on database attribute relationships since such relationships may exist in the attribute semantics. For example, $address(num, street, city, province, country)$ may indicate that *city* is more general than *street* but less general than *province* and *country*. Similarly, $date(day, month, year)$ forms naturally a *built-in* concept hierarchy for generalization.

   DMQL specifies concept hierarchy at the schema level in the following syntax.

   ⟨define_concept_hierarchy⟩ ::=
   define hierarchy for ⟨attr_name⟩
   [(⟨hier_name⟩)]: ⟨attr_set⟩ < ⟨attr_set⟩

   An example of such is shown below.

   define hierarchy for address:
   {city, province, country} < {province, country}

2. Hierarchy specification by set grouping. Some hierarchical information can be specified by concept grouping which explicitly shows that one group of concepts is at a level lower than another. The syntax adopted in DMQL is as follows.

   ⟨define_concept_hierarchy⟩ ::=
   define hierarchy for ⟨attr_name⟩
   [(⟨hier_name⟩)]: ⟨constant_set⟩ < ⟨constant_set⟩

   An example of such is shown below.

   define hierarchy for address: {B.C., Alberta,
   Manitoba, Saskatchewan} < {Western_Canada}
   define hierarchy for address: {Western_Canada,
   Central_Canada, Maritime_Provinces} < {Canada}

3. Modification of concept hierarchies. A given hierarchy should be modifiable via some simple statements. One may insert a term as a subordinate concept of a superordinate one in a hierarchy or delete a term from it. The syntax adopted in DMQL is as follows.

   ⟨modify_hierarchy⟩ ::=
   insert ⟨concept_name⟩ under ⟨concept_name⟩
   to hierarchy [(⟨hier_name⟩)] for ⟨attr_name⟩
   | delete ⟨concept_name⟩ under ⟨concept_name⟩
   from hierarchy [(⟨hier_name⟩)] for ⟨attr_name⟩

   An example of such is shown below.

   delete ⟨Manitoba⟩ under Western_Canada
   from hierarchy for address
   insert ⟨Territories⟩ under Canada
   to hierarchy for address

4. Primitives for concept hierarchy adoptions. There could be multiple concept hierarchies for a set of attributes. A hierarchy stored in a system may not always suit a particular data mining task well. Moreover, concept hierarchies may not always be provided by specification of attribute relationships or set groupings. Thus, it is often necessary to provide primitives to choose a hierarchy other than the default one from a set of available ones, dynamically adjust a hierarchy, or automatically generate a hierarchy based on the statistics of the relevant set of data. DMQL specifies these options in the following syntax.

   ⟨hierarchy_adoption⟩ ::=
   use hierarchy ⟨hier_name⟩ for ⟨attr_name⟩
   | display hierarchy [⟨hier_name⟩] for ⟨attr_name⟩
   | dynamically adjust hierarchy [⟨hier_name⟩]
   for ⟨attr_name⟩
   | generate hierarchy [⟨hier_name⟩] for ⟨attr_name⟩

   Some examples are shown below.

   use hierarchy climate_regions for address
   dynamically adjust hierarchy for address
   generate hierarchy for gpa

5 Interactive data mining and graphical user interfaces

5.1 Interactive mining of multiple-levels of knowledge

Although a data mining task can be specified flexibly using the primitives discussed above, it is difficult to predict the mining results at the time of query submission. Thus, interactive refining of a mining task or mining results becomes essential for effective mining.

Interactive refining of a mining task often requires easy modification of a query condition, thresholds,

relevant attributes, selected hierarchies, or letting a hierarchy be dynamically adjusted, etc. Such tasks should be accomplished conveniently by a graphical user interface although they can be specified (but not so conveniently) using DMQL language primitives.

For interactive refining of data mining results, one should display the results using rule visualization tools [12] or in different output forms, including generalized relations, projected statistical tables, bar charts, pie charts, curves, surfaces, quantitative rules, etc. This process may be helped by report writers or graphical display softwares. DMQL provides the following primitives for displaying results in different forms.

⟨result_displaying⟩ ::= display in ⟨result_form⟩

where the ⟨result_form⟩ could be projected statistical tables, bar charts, curves, etc. when appropriate.

Moreover, with the availability of concept hierarchies, knowledge can be expressed at different levels of abstractions. Interactive mining should facilitate the discovered knowledge to be viewed at different concept levels and from different angles. This can be accomplished by transforming data mining results conveniently with "*roll-up*" and "*drill-down*" operations [11]. Notice that *roll-up* can be done by climbing up the concept hierarchy of an attribute or dropping some attribute(s); whereas *drill-down* by stepping down the concept hierarchy of an attribute or adding some attributes. DMQL provides the following primitives for traversing through different levels of abstractions.

⟨multi-level_manipulation⟩ ::= up on ⟨attr_name⟩
| down on ⟨attr_name⟩
| add ⟨attr_name⟩
| drop ⟨attr_name⟩

For example, one may "up on address" or "drop city" to generalize the mining results.

5.2 From DMQL to flexible GUIs

As discussed above, the goal of designing the DMQL language is to provide necessary primitives for data mining engines to work on. However, a data mining user may prefer to use flexible GUIs to interact with a data miner for fruitful and convenient data mining. In current relational technology, SQL provides the "core" language of relational systems on top of which various GUIs have been constructed. Similarly, a data mining query language may serve as a "core language" for data mining system implementations, on top of which various kinds of GUIs should be developed for effective data mining.

Based on our experience, a data mining GUI may consist of the following functional components.

1. Data collection, which is an interface for collecting the relevant set of data. Such an interface will be very much like a GUI for construction of relational queries.

2. Presentation of data mining results, which is an interface for displaying data mining results in different forms, including tables, graphics, charts, curves, visualization of data mining results, etc.

3. Manipulation of data mining primitives, which may include dynamic adjustment of various kinds of thresholds, selection, display, and modifications of concept hierarchies, or modification of other mining requests or conditions, etc.

4. Interactive multi-level mining, which may include roll-up or drill-down of data mining results on any selected attributes.

5. Other miscellaneous information, which may include on-line manuals, help, indexed search, debugging, and many other interactive graphical facilities.

We have designed and developed some data mining GUIs on both the UNIX and PC platforms in our DBMiner system [9]. By examination of many data mining products or prototypes, it seems difficult to set up a standard on data mining GUIs. However, classification of GUI primitives and exploration of their underlying mechanisms are quite useful at designing a good data mining query language.

6 Discussions and conclusions

We designed and developed a preliminary version of a data mining query language, DMQL, for effective data mining in relational databases. Portions of the language have been implemented in our DBMiner system [9] for interactive data mining. Such a language also serves as a base for further development of GUIs for interactive mining of multiple-level knowledge.

There are many issues which need further studies in this direction, which are listed as follows for discussion.

1. Besides mining characteristic, discriminant, classification, and association rules, DMQL needs to specify primitives for other data mining tasks, including data clustering [15, 4, 23], and mining data evolution rules or sequential patterns [2], deviation rules [16], rules characterized by some specific patterns [18, 6], etc. It is necessary to work out such language primitives for effective communication with a data mining system. Moreover, it is not clear to us whether one could construct a more uniform syntactic framework than the current DMQL to specify primitives for mining all kinds of knowledge.

2. Graphical user interfaces have been popularly used in data mining. It is unclear whether a standard can be worked out for designing a "core" data mining GUI. Such a "core", if possibly agreed by different researchers and developers, may facilitate software development, system communication, and standardization. Moreover, it seems that some GUI primitives, such as pointing to a particular place in a curve or graph, are difficult to be specified using a text-based data mining query language like DMQL. It is not clear whether we should eventually design a "core" GUI-based language to substitute an SQL-like data mining language.

3. It is reasonably easy to design a data mining language for data mining in relational databases. It is a great challenge to design languages for knowledge mining in other kinds of databases, such as transaction databases [1], object-oriented databases [10], spatial databases [13], multimedia databases, legacy databases, global information systems [22], etc. With the emerging activities for data mining in these databases, the design of data mining languages for such mining tasks may become an important issue in future research.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[3] R.G.G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Databases, Rev. Ed.* Addison-Wesley, 1994.

[4] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. on Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[5] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[6] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int'l Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.

[7] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[8] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[9] J. Han, Y. Fu, K. Koperski, G. Melli, W. Wang, and O. R. Zaïane. *Knowledge Mining in Databases: An Integration of Machine Learning Methodologies with Database Technologies.* Canadian Artifical Intelligence, October 1995.

[10] J. Han, S. Nishio, and H. Kawano. Knowledge discovery in object-oriented and active databases. In F. Fuchi and T. Yokoi, editors, *Knowledge Building and Knowledge Sharing*, pages 221–230. Ohmsha, Ltd. and IOS Press, 1994.

[11] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996.

[12] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int'l Conf. on Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[13] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int'l Symp. on Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.

[14] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. 1996 Int. Conference on Extending Database Technology (EDBT'96)*, Avignon, France, March 1996.

[15] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[16] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[17] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

[18] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.

[19] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[20] M. Stonebraker. *Readings in Database Systems, 2ed.* Morgan Kaufmann, 1993.

[21] L. Winstone, W. Wang, and J. Han. Multiple-level data classification in large databases. In *submitted for publication*, March 1996.

[22] O. R. Zaïane and J. Han. Resource and knowledge discovery in global information systems: A preliminary design and experiment. In *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD'95)*, pages 331–336, Montreal, Canada, Aug. 1995.

[23] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996.