33

Data Mining Query Languages

Jean-Francois Boulicaut[1] and Cyrille Masson[1]

INSA Lyon, LIRIS CNRS FRE 2672
69621 Villeurbanne cedex, France.
jean-francois.boulicaut,Cyrille.Masson@insa-lyon.fr

Summary. Many Data Mining algorithms enable to extract different types of patterns from data (e.g., local patterns like itemsets and association rules, models like classifiers). To support the whole knowledge discovery process, we need for integrated systems which can deal either with patterns and data. The inductive database approach has emerged as an unifying framework for such systems. Following this database perspective, knowledge discovery processes become querying processes for which query languages have to be designed. In the prolific field of association rule mining, different proposals of query languages have been made to support the more or less declarative specification of both data and pattern manipulations. In this chapter, we survey some of these proposals. It enables to identify nowadays shortcomings and to point out some promising directions of research in this area.

Key words: Query languages, Association Rules, Inductive Databases.

33.1 The Need for Data Mining Query Languages

Since the first definition of the Knowledge Discovery in Databases (KDD) domain in (Piatetsky-Shapiro and Frawley, 1991), many techniques have been proposed to support these "From Data to Knowledge" complex interactive and iterative processes. In practice, knowledge elicitation is based on some extracted and materialized (collections of) patterns which can be global (e.g., decision trees) or local (e.g., itemsets, association rules). Real life KDD processes imply complex pre-processing manipulations (e.g., to clean the data), several extraction steps with different parameters and types of patterns (e.g., feature construction by means of constrained itemsets followed by a classifying phase, association rule mining for different thresholds values and different objective measures of interestingness), and post-processing manipulations (e.g., elimination of redundancy in extracted patterns, crossing-over operations between patterns and data like the search of transactions which are exceptions to frequent and valid association rules or the selection of misclassified examples with a decision tree). Looking for a tighter integration between data and patterns which hold in the data, Imielinski and Mannila have proposed in (Imielinski and Mannila, 1996) the concept of inductive database (IDB). In an IDB, ordinary queries can be used to access and manipulate data, while *inductive queries* can be used to generate (mine), manipulate, and apply patterns. KDD becomes

O. Maimon, L. Rokach (eds.), *Data Mining and Knowledge Discovery Handbook*, 2nd ed.,
DOI 10.1007/978-0-387-09823-4_33,

an extended querying process where the analyst can control the whole process since he/she specifies the data and/or patterns of interests. Therefore, the quest for query languages for IDBs is an interesting goal. It is actually a long-term goal since we still do not know which are the relevant primitives for Data Mining. In some sense, we still lack from a well-accepted set of primitives. It might recall the context at the end of the 60's before the Codd's relational algebra proposal.

In some limited contexts, researchers have, however, designed data mining query languages. Data Mining query languages can be used for specifying inductive queries on some pattern domains. They can be more or less coupled to standard query languages for data manipulation or pattern postprocessing manipulations. More precisely, a Data Mining query language, should provide primitives to (1) select the data to be mined and pre-process these data, (2) specify the kind of patterns to be mined, (3) specify the needed background knowledge (as item hierarchies when mining generalized association rules), (4) define the constraints on the desired patterns, and (5) post-process extracted patterns.

Furthermore, it is important that Data Mining query languages satisfy the closure property, i.e., the fact that the result of a query can be queried. Following a classical approach in database theory, it is also needed that the language is based on a well-defined (operational or even better declarative) semantics. It is the only way to make query languages that are not only "syntactical sugar" on top of some algorithms but true query languages for which query optimization strategies can be designed. Again, if we consider the analogy with SQL, relational algebra has paved the way towards query processing optimizers that are widely used today. Ideally, we would like to study containment or equivalence between mining queries as well.

Last but not the least, the evaluation of Data Mining queries is in general very expensive. It needs for efficient constraint-based data mining algorithms, the so-called solvers (De Raedt, 2003,Boulicaut and Jeudy, 2005). In other terms, data mining query languages are often based on primitives for which some more or less ad-hoc solvers are available. It is again typical of a situation where a consensus on the needed primitives is yet missing.

So far, no language proposal is generic enough to provide support for a broad kind applications during the whole KDD process. However, in the active field of association rule mining, some interesting query languages have been proposed. In Section 33.2, we recall the main steps of a KDD process based on association rule mining and thus the need for querying support. In Section 33.3, we introduce several relevant proposals for association rule mining query languages. It contains a short critical evaluation (see (Botta *et al.*, 2004) for a detailed one). Section 33.4 concludes.

33.2 Supporting Association Rule Mining Processes

We assume that the reader is familiar with association rule mining (see, e.g., (Agrawal *et al.*, 1996) for an introduction). In this context, data is considered as a multiset of transactions, i.e., sets of items. Frequent associations rules are built on frequent itemsets (itemsets which are subsets of a certain percentage of the transactions). Many objective interestingness measures can inform about the quality of the extracted rules, the confidence measure being one of the most used. Importantly, many objective measures appear to be complementary: they enable to rank the rules according to different points of view. Therefore, it seems important to provide support for various measures, including the definition of new ones, e.g., application specific ones.

When a KDD process is based on itemsets or association rules, many operations have to be performed by means of queries. First, the language should allow to manipulate and extract

source data. Typically, the raw data is not always available as transactional data. One of the typical problems concerns the transformation of numerical attributes into items (or boolean properties). More generally, deriving the transactional context to be mined from raw data can be a quite tedious task (e.g., deriving a transactional data set about WWW resources loading per session from raw WWW logs in a WWW Usage Mining application). Some of these preprocessing are supported by SQL but a programming extension like PL/SQL is obviously needed.

Then, the language should allow the user to specify a broad kind of constraints on the desired patterns (e.g., thresholds for the objective measures of interestingness, syntactical constraints on items which must appear or not in rule components). So far, the primitive constraints and the way to combine them is tightly linked with the kinds of constraints the underlying evaluation engine or solvers can process efficiently (typically anti-monotonic or succinct constraints). One can expect that minimal frequency and minimal confidence constraints are available. However, many other primitive constraints can be useful, including the ones based on aggregates (Ng *et al*., 1998) or closures (Jeudy and Boulicaut, 2002, Boulicaut, 2004).

Once rules have been extracted and materialized (e.g., in relational tables), it is important that the query language provides techniques to manipulate them. We can wish, for instance, to find a cover of a set of extracted rules (i.e., non redundant association rules based on closed sets (Bastide *et al*., 2000)), which requires to have subset operators, primitives to access bodies and heads of rules, and primitives to manipulate closed sets or other condensed representations of frequent sets (Boulicaut, 2004) and (Calders and Goethals, 2002). Another important issue is the need for crossing-over primitives. It means that, for instance, we need simple way to select transactions that satisfy or do not satisfy a given rule.

The so-called closure property is important. It enables to combine queries, to support the reuse of KDD scenarios, and it gives rise to opportunities for compiling schemes over sequences of queries (Boulicaut *et al*., 1999). Finally, we could also ask for a support to pattern uses. In other terms, once relevant patterns have been stored, they are generally used by some software component. To the best of our knowledge, very few tools have been designed for this purpose (see (Imielinski *et al*., 1999) for an exception).

We can distinguish two major approaches in the design of Data Mining query languages. The first one assumes that all the required objects (data and pattern storage systems and solvers) are already embedded into a common system. The motivation for the query language is to provide more understandable primitives: the risk is that the query

language provides mainly "syntactic sugar" on top of solvers. In that framework, if data are stored using a classical relational DBMS, it means that source tables are views or relations and that extracted patterns are stored using the relational technology as well. `MSQL`, `DMQL` and `MINE RULE` can be considered as representative of this approach. A second approach assumes that we have no predefined integrated systems and that storage systems are loosely coupled with solvers which can be available from different providers. In that case, the language is not only an interface for the analyst but also a facilitator between the DBMS and the solvers. It is the approach followed by `OLE DB` for DM (Microsoft). It is an API between different components that also provides a language for creating and filling extraction contexts, and then access them for manipulations and tests. It is primarily designed to work on top of SQL Server and can be plugged with different solvers provided that they comply the API standard.

33.3 A Few Proposals for Association Rule Mining

33.3.1 MSQL

MSQL (Imielinski and Virmani, 1999) has been designed at the Rutgers University. It extracts rules that are based on descriptors, each descriptor being an expression of the type $(A_i = a_{ij})$, where A_i is an attribute and a_{ij} is a value or a range of values in the domain of A_i. We define a *conjunctset* as the conjunction of an arbitrary number of descriptors such that there are no couple of descriptors built on the same attribute. MSQL extracts propositional rules of the form $\mathscr{A} \Rightarrow \mathscr{B}$, where $\mathscr{A}$ is a conjunctset and $\mathscr{B}$ is a descriptor. As a consequence, only one attribute can appear in the consequent of a rule. Notice that MSQL defines the support of an association rule $\mathscr{A} \Rightarrow \mathscr{B}$ as the number of tuples containing $\mathscr{A}$ in the original table and its confidence as the ratio between the number of tuples containing $\mathscr{A}$ et $\mathscr{B}$ and the support of the rule.

From a practical point of view, MSQL can be seen as an extension of SQL with some primitives tailored for association rule mining (given their semantics of association rules). Specific queries are used to mine rules (inductive queries starting with GetRules) while other queries are post-processing queries over a materialized collection of rules (queries starting with SelectRules). The global syntax of the language for rule extraction is the following one:

```
GetRules(C) [INTO <rulebase name>]
    [WHERE <rule constraints>]
    [SQL-group-by clause]
    [USING encoding-clause]
```

C is the source table and rule_constraints are conditions on the desired rules, e.g., the kind of descriptors which must appear in rule components, the minimal frequency or confidence of the rules or some mutual exclusion constraints on attributes which can appear in a rule. The USING part enables to discretize numerical values. rulebase_name is the name of the object in which rules will be stored. Indeed, using MSQL, the analyst can explicitly materialize a collection of rules and then query it with the following generic statement where <conditions> can specify constraints on the body, the head, the support or the confidence of the rule:

```
SelectRules(rulebase name)
        [where <conditions>]
```

Finally, MSQL provides a few primitives for post-processing. Indeed, it is possible to use Satisfy and Violate clauses to select rules which are supported (or not) in a given table.

33.3.2 MINE RULE

MINE RULE (Meo *et al.*, 1998) has been designed at the University of Torino and the Politecnico di Milano. It is an extension of SQL which is coupled with a relational DBMS. Data can be selected using the full power of SQL. Mined association rules are materialized into relational tables as well. MINE RULE extracts association rule between values of attributes in a relational table. However, it is up to the user to specify the form of the rules to be extracted. More precisely, the user can specify the cardinality of body and head of the desired

rules and the attributes on which rule components can be built. An interesting aspect of MINE RULE is that it is possible to work on different levels on grouping during the extraction (in a similar way as the GROUP BY clause of SQL). If there is one level of grouping, rule support will be computed w.r.t. the number of groups in the table. Defining a second level of grouping leads to the definition of clusters (sub-groups). In that case, rules components can be taken in two different clusters, eventually ordered, inside a same group. It is thus possible to extract some elementary sequential patterns (by clustering on a time-related attribute). For instance, grouping purchases by customers and then clustering them by date, we can obtain rules like $Butter \wedge Milk \Rightarrow Oil$ to say that customers who buy first *Butter* and *Milk* tend to buy *Oil* after. Concerning interestingness measures, MINE RULE enables to specify minimal frequency and confidence thresholds. The general syntax of a MINE RULE query for extracting rules is:

```
MINE RULE <TableName> AS
 SELECT DISTINCT [<Cardinality>]  <Attributes>
                 AS BODY,
                 [<Cardinality>]  <Attributes>
                 AS HEAD
                 [,SUPPORT] [,CONFIDENCE]
 FROM <Table> [ WHERE <WhereClause> ]
 GROUP BY <Attributes> [ HAVING <HavingClause> ]
 [ CLUSTER BY <Attributes>
              [ HAVING <HavingClause> ]]
 EXTRACTING RULES WITH
              SUPPORT:<real>, CONFIDENCE:<real>
```

33.3.3 DMQL

DMQL (Han *et al.*, 1996) has been designed at the Simon Fraser University, Canada. It has been designed to support various rule mining extractions (e.g., classification rules, comparison rules, association rules). In this language, an association rule is a relation between the values of two sets of predicates that are evaluated on the relations of a database. These predicates are of the form $P(X,c)$ where P is a predicate taking the name of an attribute of a relation, X is a variable and c is a value in the domain of the attribute. A typical example of association rule that can be extracted by DMQL is $buy(X,milk) \wedge town(X,Berlin) \Rightarrow buy(X,beer)$. An important possibility in DMQL is the definition of meta-patterns, i.e., a powerful way to restrict the syntactic aspect of the extracted rules (expressive syntactic constraints). For instance, the meta-pattern $buy^{+}(X,Y) \wedge town(X,Berlin) \Rightarrow buy(X,Z)$ restricts the search to association rules concerning implication between bought products for customers living in Berlin. Symbol $+$ denotes that the predicate *buy* can appear several times in the left part of the rule. Moreover, beside the classical frequency and confidence, DMQL also enables to define thresholds on the noise or novelty of extracted rules. Finally, DMQL enables to define a hierarchy on attributes such that generalized association rules can be extracted. The general syntax of DMQL for the extraction of association rules is the following one:

```
Use database ⟨database_name⟩

{Use hierarchy ⟨hierarchy_name⟩
For ⟨attribute⟩ }

Mine associations [as ⟨pattern_name⟩]

[ Matching ⟨metapattern⟩]

From ⟨relation(s)⟩ [ Where ⟨condition⟩]

[ Order by ⟨order_list⟩]
[ Group by ⟨grouping_list⟩] [ Having ⟨condition⟩]

With ⟨interest_measure⟩
Threshold = value
```

33.3.4 OLE DB for DM

OLE DB for DM has been designed by Microsoft Corporation (Netz *et al.*, 2000). It is an extension of the OLE DB API to access database systems. More precisely, it aims at supporting the communication between the data sources and the solvers that are not necessarily implemented inside the query evaluation system. It can thus work with many different solvers and types of patterns. To support the manipulation of the objects of the API during a KDD process, OLE DB for DM proposes a language as an extension to SQL. The concept of OLE DB for DM relies on the definition of Data Mining Models (DMM), i.e. object that correspond to extraction contexts in KDD. Indeed, whereas the other language proposals made the assumption that the data almost have a suitable format for the extraction, OLE DB for DM considers it is not always the case and let the user defines a virtual object that will have a suitable format for the extraction and that will be populated with the needed data. Once the extraction algorithm has been applied on this DMM, the DMM will become an object containing patterns or models. It will then be possible to query this DMM as a rule base or to use it as a classifier. The global syntax for creating a DMM is the following:

```
CREATE MINING MODEL <DMM name>
              (<columns definition>)
              USING <algorithm>
              [(<algorithm parameters>)]
```

For each column, it is possible to specify the data type and if it is the target attribute of the model to be learnt in case of classification. Moreover, a column can correspond to a nested table, which is useful when populating the mining model with data taken in tables linked by a one-to-many relationship. For the moment, OLE DB for DM is implemented in the SQL

Server 2000 software and it provides only two mining algorithms: one for decision trees and one for clustering. However, the 2005 version of SQL server should provide neural network and association rule extractors. This latter one will enable to define minimal and maximal rule support, minimal confidence,and minimal and maximal sizes of itemsets on which the rules are based.

33.3.5 A Critical Evaluation

Let us now emphasize the main advantages and drawbacks of the different proposals. A detailed evaluation of these four languages has been performed on a simple but realistic association rule mining scenario (Botta *et al.*, 2004). We summarize the results of this study and it enables to point some important problems that must be addressed on our way to query languages for inductive databases.

The advantages of the proposed languages is that they are all designed as extensions of SQL. It facilitates the work for database experts and it is useful for data manipulation (or the needed standard queries). They all satisfy the closure property. Indeed, even if all the languages do not systematically provide operators for manipulating extracted rules, it is always possible to access materialized collections of rules using SQL queries. Notice, however, that most of the needed pre-processing or post-processing techniques will need not only SQL queries but also PL/SQL statements. Some languages provide primitives to simplify some typical preprocessing, e.g., the discretization of numerical values. Even if is quite preliminary, it is an important support for the practical use of the association rule mining technique. Finally, the concept of `OLE DB` for `DM` is quite relevant as it enables external providers to plug-in new solvers to the existing systems.

The first major limitation of the proposed languages is the poor support to pre- and post-postprocessing operations. Indeed, they are essentially designed around the extraction step and mainly provide primitives for rule extractions, these primitives being generally fixed, e.g., the possibilities to specify minimal thresholds for a few selected objective measures of interestingness or to define syntactical constraints on the rules. Only `MSQL` and `OLE DB` for `DM` propose restricted mechanisms for discretization. Typical preprocessing techniques for, e.g., sampling or boosting, are not supported. It has been shown that pre-processing processes for KDD are tedious phases for which the use of integrated tools and operators is needed (see, e.g., the MINING MART "Enabling End-User Datawarehouse Mining" EU funded project IST-1999-11993 (Morik and Scholz, 2004)). The lack of primitives for post-processing is also obvious. Only `MSQL` provides a `SelectRules` operator which enables to query rule databases and primitives for crossing-over operations between rules and data. The others rely on SQL and its programming extensions for accessing and manipulating the rules. For instance, using `MINE RULE`, extracted rules are stored in relational tables that have to be queried with SQL. In that case, writing a query which simply returns tuples of a table which satisfy a given rule can be very complex because of SQL mechanisms for handling subset relationships (see (Botta *et al.*, 2004) for examples). Not only the SQL post-processing queries are hard to write but also difficult to optimize given the current state of the art for SQL optimization. A solution can come from query languages dedicated to pattern database manipulations. It is the case of `RULE-QL` (Tuzhilin and Liu, 2002) which extends SQL with operators allowing to access rules components and to specify subset relationships. It is thus easier to write queries that, for instance, select rules that have a left part contained in the consequent of another rule. `RULE-QL` can be seen as a good complement to languages like `MINE RULE`. More generally, some basic research is needed on pattern database querying where patterns can be rules, clusters, classifiers, etc. An interesting work in this direction is done by the PANDA

"Patterns for Next-Generation Database Systems" EU funded Working Group IST/FET-2001-33058 (Theodoridis and Vassiliadis, 2004, Catania *et al*., 2004).

The second main drawback of the proposed languages is that they appear to be quite *ad hoc* proposals. By this term, we mean that they have been proposed on top of some specific algorithms or solvers. The available constraints or conjunction of constraints are the one for which solvers were available at the time of design. When considering the evaluation architecture (described, e.g., for `MINE RULE`), we can see that different solvers cope with specific conjunctions of constraints on the association rules. This is also the case for `DMQL` and `OLE DB` for `DM` proposals, i.e. languages that can extract several types of patterns. For instance, with `DMQL`, each type of rule that

can be extracted is indeed related to a particular solver.

To summarize, primitives are missing and the integration of new primitives by the analyst is not possible. This is obviously due to the lack of consensus on a good collection of primitives. This is true for simple pattern domains like association rules but also for more complex ones. It is interesting to note that the semantics of the association rules for the different query language proposals is not the same. When looking at the details, we can see that even simple evaluation functions like frequency can be defined differently. In other terms, we still lack from a consensus on what is an association rule and what is the semantics of a constrained association rule. The situation is the same for other kinds of patterns, e.g., see the many different semantics for constrained sequential patterns which have been proposed the last 10 years.

We believe that looking for a formal semantics of Data Mining query languages is crucial for the development of the field. Indeed, if we draw a parallel with the development of standard database query languages, we know that (extended) relational algebra have played a major role for their design but also the implementation of efficient query optimizers. The same goal should be taken if we wish to develop Data Mining query languages that are not just "syntactic sugar" on top of solvers. For instance, based on the `MINE RULE` formal semantics, it has been possible to analyze how to optimize queries and also to exploit properties on the relationship between queries. Thanks to data dependencies in the source tables, (Meo, 2003) shows that containment and dominance relations between queries can be used to speed-up the evaluation of new mining queries.

It was one of the main goals of the cInQ "**c**onsortium on knowledge discovery by **In**ductive **Q**ueries" EU funded project IST/FET-2000-26469 to make a breakthrough in this direction. Considering several pattern domains (e.g., association rules, sequences, molecular fragments), they have been looking for useful primitives, new ways to combine them, and not only ad-hoc but also generic solvers for complex inductive queries (e.g., arbitrary boolean expressions over monotonic and anti-monotonic constraints (De Raedt *et al*., 2002)). A simple formal language is sketched in (De Raedt, 2003) to describe both data and pattern manipulations via inductive queries. Some recent contributions to database support for Data Mining are collected in (Meo *et al*., 2004). It contains, among others, extended contributions of the first two workshops organized by the cInQ project.

33.4 Conclusion

In this chapter, we have considered Data Mining query languages issues. To support the whole knowledge discovery process, we need for integrated systems which can deal either with patterns and data. Designing such systems is the goal of the emerging inductive database approach. Following this database perspective, knowledge discovery processes become querying

processes for which query languages have to be designed. On one hand, interesting conceptual, or say abstract, proposals have been made like (Giannotti and Manco, 1999, De Raedt, 2003, Catania *et al*., 2004). On another hand, concrete query languages have been designed and implemented for specific pattern domains, mainly association rules (Han *et al*., 1996, Meo *et al*., 1998, Imielinski and Virmani, 1999, Netz *et al*., 2000). The first approach emphasizes the need for general-purpose primitives and is looking for generic approaches in combining these primitives and designing generic solvers. The second approach is pragmatic: providing an immediate support to practitioners by means of better Data Mining tools. Doing so, the primitives are often tailored to some specific pattern domain, or even some application domain. Ad-hoc solvers are designed for an efficient evaluation of concrete queries. Standards like PMML ((http://www.dmg.org) are also immediately useful for practitioners and software companies. This XML-based language provides a standard format for representing various patterns and this is important to support interoperability between various tools. Let us notice however that it does not provide primitives for pattern manipulation. We strongly believe that both directions are useful on our road towards inductive databases and inductive database management systems.

Acknowledgments

The authors want to thank the colleagues of the cInQ IST-2000-26469 (consortium on knowledge discovery by inductive queries) for interesting discussions on Data Mining query languages. A special thank goes to Rosa Meo for her contribution to this domain and the critical evaluation (Botta *et al*., 2004).

References

R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.

Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Proc. CL 2000*, volume 1861 of *LNCS*, pages 972–986. Springer-Verlag, 2000.

M. Botta, J.-F. Boulicaut, C. Masson, and R. Meo. Query languages supporting descriptive rule mining: a comparative study. In *Database Technologies for Data Mining - Discovering Knowledge with Inductive Queries*, volume 2682 of *LNCS*, pages 27–54. Springer-Verlag, 2004.

J.-F. Boulicaut. Inductive databases and multiple uses of frequent itemsets: the cInQ approach. In *Database Technologies for Data Mining - Discovering Knowledge with Inductive Queries*, volume 2682 of *LNCS*, pages 3–26. Springer-Verlag, 2004.

J.-F. Boulicaut and B. Jeudy. Constraint-based Data Mining. In *Data Mining and Knowledge Discovery Handbook*. Chapter 16.7, this volume, Kluwer, 2005.

J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling KDD processes within the inductive database framework. In *Proc. DaWaK'99*, volume 1676 of *LNCS*, pages 293–302. Springer-Verlag, 1999.

T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proc. PKDD*, volume 2431 of *LNCS*, pages 74–85. Springer-Verlag, 2002.

B. Catania, A. Maddalena, M. Mazza, E. Bertino, and S. Rizzi. A framework for Data Mining pattern management. In *Proc. PKDD'04*, volume 3202 of *LNAI*, pages 87–98. Springer-Verlag, 2004.

L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, 2003.

L. De Raedt, M. Jaeger, S. Lee, and H. Mannila. A theory of inductive query answering. In *Proc. IEEE ICDM'02*, pages 123–130, 2002.

F. Giannotti and G. Manco. Querying inductive databases via logic-based user-defined aggregates. In *Proc. PKDD'99*, volume 1704 of *LNCS*, pages 125–135. Springer-Verlag, 1999.

J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: a Data Mining query language for relational databases. In R. Ng, editor, *Proc. ACM SIGMOD Workshop DMKD'96*, Montreal, Canada, 1996.

T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, November 1996.

T. Imielinski and A. Virmani. MSQL: A query langugage for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.

T. Imielinski, A. Virmani, and A. Abdulghani. DMajor-application programming interface for database mining. *Data Mining and Knowledge Discovery*, 3(4):347–372, 1999.

B. Jeudy and J.-F. Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis*, 6(4):341–357, 2002.

R. Meo. Optimization of a language for Data Mining. In *Proc. ACM SAC'03 - Data Mining track*, pages 437–444, 2003.

R. Meo, P. L. Lanzi, and M. Klemettinen, editors. *Database Technologies for Data Mining - Discovering Knowledge with Inductive Queries*, volume 2682 of *LNCS*. Springer-Verlag, 2004.

R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.

K. Morik and M. Scholz. The Mining Mart approach to knowledge discovery in databases. In *Intelligent Technologies for Information Analysis*. Springer-Verlag, 2004.

A. Netz, S. Chaudhuri, J. Bernhardt, and U. Fayyad. Integration of Data Mining and relational databases. In *Proc. VLDB'00*, pages 719–722, Cairo, Egypt, 2000. Morgan Kaufmann.

R. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. ACM SIGMOD'98*, pages 13–24, 1998.

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

Y. Theodoridis and P. Vassiliadis, editors. *Proc. of Pattern Representation and Management PaRMa 2004 co-located with EDBT 2004*. CEUR Workshop Proceedings 96 Technical University of Aachen (RWTH), 2004.

A. Tuzhilin and B. Liu. Querying multiple sets of discovered rules. In *Proc. ACM SIGKDD'02*, pages 52–60, 2002.