Chapter 3
A Practical Comparative Study Of Data Mining Query Languages

Hendrik Blockeel, Toon Calders, Élisa Fromont, Bart Goethals, Adriana Prado, and Céline Robardet

Abstract An important motivation for the development of inductive databases and query languages for data mining is that such an approach will increase the flexibility with which data mining can be performed. By integrating data mining more closely into a database querying framework, separate steps such as data preprocessing, data mining, and postprocessing of the results, can all be handled using one query language. In this chapter, we compare six existing data mining query languages, all extensions of the standard relational query language SQL, from this point of view: how flexible are they with respect to the tasks they can be used for, and how easily can those tasks be performed? We verify whether and how these languages can be used to perform four prototypical data mining tasks in the domain of itemset and association rule mining, and summarize their stronger and weaker points. Besides offering a comparative evaluation of different data mining query languages, this chapter also provides a motivation for a following chapter, where a deeper integration of data mining into databases is proposed, one that does not rely on the development of a new query language, but where the structure of the database itself is extended.

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium and Leiden Institute of Advanced Computer Science, Universiteit Leiden, The Netherlands e-mail: hendrik.blockeel@cs.kuleuven.be

Toon Calders
Technische Universiteit Eindhoven, The Netherlands e-mail: t.calders@tue.nl

Élisa Fromont · Adriana Prado
Université de Lyon (Université Jean Monnet), CNRS, Laboratoire Hubert Curien, UMR5516, F-42023 Saint-Etienne, France
e-mail: {elisa.fromont,adriana.bechara.prado}@univ-st-etienne.fr

Bart Goethals
Universiteit Antwerpen, Belgium e-mail: bart.goethals@ua.ac.be

Céline Robardet
Université de Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621, France
e-mail: celine.robardet@insa-lyon.fr

S. Džeroski, *Inductive Databases and Constraint-Based Data Mining,*
DOI 10.1007/978-1-4419-7738-0_3,

3.1 Introduction

An important motivation for the development of inductive databases and query languages for data mining is that such an approach will increase the flexibility with which data mining can be performed. By integrating data mining more closely into a database querying framework, separate steps such as data preprocessing, data mining, and postprocessing of the results, can all be handled using one query language. It is usually assumed that standard query languages such as SQL will not suffice for this; and indeed, SQL offers no functionality for, for instance, the discovery of frequent itemsets. Therefore, multiple researchers have proposed to develop new query languages, or extend existing languages, so that they offer true data mining facilities. Several concrete proposals have been implemented and evaluated.

In this chapter, we consider four prototypical data mining tasks, and six existing data mining query languages, and we evaluate how easily the tasks can be performed using these languages. The six languages we evaluate are the following: MSQL [8], MINE RULE operator [11], SIQL [17], SPQL [2], and DMX [16]. All six are based on extending SQL and have special constructs to deal with itemsets and/or association rules.

The four tasks with which the expressivity of the languages will be tested can all be situated in the association rule mining domain. The tasks are "typical" data mining tasks, in the sense that they are natural tasks in certain contexts, and that they have not been chosen with a particular data mining query language in mind. The four tasks are: discretizing a numerical attribute, mining itemsets with a specific *area* constraint, and two association rule mining tasks in which different constraints are imposed on the rules to be discovered. It turns out that the existing languages have significant limitations with respect to the tasks considered.

Many of the shortcomings of the six languages are not of a fundamental nature and can easily be overcome by adding additional elements to the query languages. Yet, when extending a query language, however, there is always the question of how much it should be extended. One can identify particular data mining problems and then extend the language so that these problems can be handled; but whenever a new type of data mining task is identified, a further extension may be necessary, unless one can somehow guarantee that a language is expressive enough to handle any kind of data mining problem.

While this chapter offers a comparative evaluation of different data mining query languages, this comparison is not the main goal; it is meant mostly as an illustration of the limitations that current languages have, and as a motivation for Chapter 11, where the idea of creating a special-purpose query language for data mining is abandoned, and the inductive database principle is implemented by changing the structure of the database itself, adding "virtual data mining views" to it (which can be queried using standard SQL), rather than by extending the query language.

We dedicate the next section to the description of the chosen data mining tasks. In Section 3.3, we introduce the data mining query languages and describe how they can be used for performing these tasks. Next, in Section 3.4, we summarize the

positive and negative points of the languages, with respect to the accomplishment of the given tasks.

3.2 Data Mining Tasks

Inspired by [3], we verify whether and how four prototypical data mining tasks can be accomplished using a number of existing data mining query languages. To enable a fair comparison between them, we study here data mining tasks which mainly involve itemsets and association rules, as these patterns can be computed by almost all of the surveyed proposals. We also focus on intra-tuple patterns (i.e., patterns that relate values of different attributes of the same tuple), even though some of the languages considered can also handle inter-tuple patterns (which relate values of attributes of different tuples that are somehow connected) [3]. As the precise structure of the patterns that can be found typically also differs between query languages, we will for each task describe precisely what kind of pattern we are interested in (i.e., impose specific constraints that the patterns should satisfy).

For ease of presentation, we will assume that the data table Playtennis2 in Figure 3.1 forms the source data to be mined. The data mining tasks that we will discuss are the following:

- **Discretization task**: Discretize attribute *Temperature* into 3 intervals. The discretized attribute should be used in the subsequent tasks.
- **Area task**: Find all intra-tuple itemsets with relative support of at least 20%, size of at least 2, and area, that is, absolute support $\times$ size, of at least 10. The area of an itemset corresponds to the size of the tile that is formed by the items in the itemset in the transactions that support it. The mining of large tiles; i.e., itemsets with a high area is useful in constructing small summaries of the database [4].

Playtennis2

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	85	High	Weak	No
D2	Sunny	80	High	Strong	No
D3	Overcast	83	High	Weak	Yes
D4	Rain	70	High	Weak	Yes
D5	Rain	68	Normal	Weak	Yes
D6	Rain	65	Normal	Strong	No
D7	Overcast	64	Normal	Strong	Yes
D8	Sunny	72	High	Weak	No
D9	Sunny	69	Normal	Weak	Yes
D10	Rain	75	Normal	Weak	Yes
D11	Sunny	75	Normal	Strong	Yes
D12	Overcast	72	High	Strong	Yes
D13	Overcast	81	Normal	Weak	Yes
D14	Rain	71	High	Strong	No

Fig. 3.1 The data table Playtennis2.

- **Right hand side task**: Find all intra-tuple association rules with relative support of at least 20%, confidence of at most 80%, size of at most 3, and a singleton right hand side.
- **Lift task**: Find, *from the result of the right hand side task*, rules with attribute *Play* as the right hand side that have a lift greater than 1.

While these tasks are only a very small sample from all imaginable tasks, they form a reasonably representative and informative sample. Discretization is a very commonly used preprocessing step. The discovery of itemsets and association rules are common data mining tasks, and the constraints considered here (upper/lower bounds on support, confidence, size, area, lift) are commonly used in many application domains. The fourth task is interesting in particular because it involves what we could call *incremental mining*: after obtaining results using a data mining process, one may want to refine those results, or mine the results themselves. This is one of the main motivating characteristics of inductive databases: the closure property implies that the results of a mining operation can be stored in the inductive database, and can be queried further with the same language used to perform the original mining operation.

3.3 Comparison of Data Mining Query Languages

We include six data mining query languages in our comparison : DMQL [6, 7], MSQL [8], SQL extended with the MINE RULE operator [11], SIQL [17], SPQL [2], and DMX, the extended version of SQL that is included in Microsoft SQL server 2005 [16]. As all these languages are quite different, we postpone a detailed discussion of each language until right before the discussion of how it can be used to perform the four tasks.

3.3.1 DMQL

The language DMQL (Data Mining Query Language) [6, 7] is an SQL-like data mining query language designed for mining several kinds of rules in relational databases, such as classification and association rules. It has been integrated into DBMiner [5], which is a system for data mining in relational databases and data warehouses.

As for association rules, a rule in this language is a relation between the values of two sets of predicates evaluated on the database. The predicates are of the form P(X, y), where P is a predicate that takes the name of an attribute of the source relation, X is a variable, and y is a value in the domain of this attribute. As an example, the association rule “if outlook is sunny and humidity is high, you should not play tennis” is represented in this language by

$$\text{Outlook}(\text{X}, \text{`Sunny'}) \wedge \text{Humidity}(\text{X}, \text{`High'}) \Rightarrow \text{Play}(\text{X}, \text{`No'}),$$

where X is a variable representing the tuples in the source relation that satisfy the rule.

DMQL also gives the user the ability to define a *meta-pattern* (template), which restricts the structure of the rules to be extracted. For example, the meta-pattern

$$\text{P}(\text{X: Playtennis2}, \text{y}) \wedge \text{Q}(\text{X}, \text{w}) \Rightarrow \text{Play}(\text{X}, \text{z})$$

restricts the structure of the association rules to rules having only the attribute *Play* in the right hand side, and any 2 attributes in the left hand side. In addition to the meta-pattern resource, DMQL has also primitives to specify concept hierarchies on attributes. These can be used so as to extract generalized association rules [15] as well as for discretization of numerical attributes.

Next, we present how the tasks described above are executed in DMQL.

Discretization task. In DMQL, a discretization of a numerical attribute can be defined by a concept hierarchy as follows [7]:

```
1. define hierarchy temp_h for Temperature
   on Playtennis2 as
2. level1: {60..69} < level0:all
3. level1: {70..79} < level0:all
4. level1: {80..89} < level0:all
```

By convention, the most general concept, *all*, is placed at the root of the hierarchy, that is, at level 0. The notation ".." implicitly specifies all values within the given range. After constructing such hierarchy, it can be used in subsequent mining queries as we show later on.

Area task. As DMQL was specially designed for extracting rules from databases, the area task cannot be executed in this language.

Right hand side task. The following DMQL query is how intra-tuple association rules with a certain minimum support and minimum confidence are extracted in this language (note that we are not considering the constraint on the maximum size of the rules nor on their right hand sides yet):

```
1. use database DB
2. use hierarchy temp_h for attribute Temperature
3. in relevance to Outlook, Temperature,
                   Humidity, Wind, Play
4. mine associations as MyRules
5. from Playtennis2
6. group by Day
7. with support threshold = 0.20
8. with confidence threshold = 0.80
```

The language constructs allow to specify the relevant set of data (lines 3 and 5), the hierarchies to be assigned to a specific attribute (line 2, for the attribute

Temperature), the desired output, that is, the kind of knowledge to be discovered (line 4), and the constraints minimum support and minimum confidence (lines 7 and 8, respectively), as required by the current task.

DMQL is able to extract both intra- and inter-tuple association rules. For the extraction of intra-tuple rules (as requested by the current task), the group-by clause in line 6 guarantees that each group in the source data coincides with a unique tuple.

Concerning the remaining constraints, although it is possible to constrain the size of the right hand side of the rules using meta-patterns (as shown earlier), we are not aware of how meta-patterns can be used to constrain the maximum size of the rules, as also needed by the current task. An alternative solution to obtain the requested rules is to write two DMQL queries as the one above, the first using the meta-pattern:

$$\mathrm{P(X{:}\ Playtennis2, y)} \wedge \mathrm{Q(X, w)} \Rightarrow \mathrm{V(X, z)(rules\ with\ size\ 3)}$$

and the second, using the meta-pattern:

$$\mathrm{P(X{:}\ Playtennis2, y)} \Rightarrow \mathrm{V(X, z)(rules\ with\ size\ 2).}$$

A meta-pattern can be used in the query above by simply adding the clause "**matching** <meta-pattern>" between lines 4 and 5. We therefore conclude that the right hand side task can be performed in DMQL.

Lift task. In the system DBMiner [5], the mining results are presented to the user and an iterative refinement of these results is possible only through graphical tools. In fact, it is not clear in the literature whether nor how (with respect to the attributes) the rules are stored into the database. For this reason, we assume here that the mining results cannot be further queried and, consequently, the lift task cannot be accomplished in this language.

3.3.2 MSQL

The language MSQL [8] is an SQL-like data mining query language that focuses only on mining intra-tuple association rules in relational databases. According to the authors, the main intuition behind the language design has been to allow the representation and manipulation of rule components (left and right hand sides), which, being sets, are not easily representable in standard SQL [8].

In MSQL, an association rule is a propositional rule defined over *descriptors*. A descriptor is an expression of the form $(A_i = a_{ij})$, where A_i is an attribute in the database and a_{ij} belongs to the domain of A_i. A *conjunctset* is defined as a set containing an arbitrary number of descriptors, such that no two descriptors are formed using the same attribute. Given this, an association rule in this language is of the form $A \Rightarrow B$, where A is a conjunctset and B a single descriptor. The rule "if outlook is sunny and humidity is high, you should not play tennis" is therefore represented in MSQL as

$$(Outlook = \text{'Sunny'}) \wedge (Humidity = \text{'High'}) \Rightarrow (Play = \text{'No'}).$$

MSQL offers operators for extracting and querying association rules: these are called *GetRules* and *SelectRules*, respectively. Besides, it also provides an encode operator for discretization of numerical values.
In the following, we show how the given tasks are executed in MSQL.

Discretization task. MSQL offers an encode operator that effectively creates ranges of values, and assigns integers (an encoded value) to those ranges. The following MSQL statement creates a discretization for the attribute *Temperature*, as required.

```
1. create encoding temp_encoding
          on Playtennis2.Temperature as
2. begin
3. (60,69,1), (70,79,2), (80,89,3), 0
4. end
```

For every set (x,y,z) given in line 3, MSQL assigns the integer z to the range of values from x to y. In this example, the integer 0 is assigned to occasional values not included in any of the specified ranges (see end of line 3). The created encoding, called "temp_encoding", can be used in subsequent mining queries as we show below.

Area task. Similarly to DMQL, MSQL cannot perform the area task, as it was specially proposed for association rule mining.

Right hand side task. As described earlier, MSQL is able to extract only intra-tuple association rules, which, in turn, are defined as having a singleton right hand side. The current task can be completely performed by the operator *GetRules*, as follows.

```
1. GetRules(Playtennis2)
2. into MyRules
3. where length <= 2
4. and support >= 0.20
5. and confidence >= 0.80
6. using temp_encoding for Temperature
```

In line 1, the source data is defined between parentheses. Constraints on the rules to be extracted are posed in the where-clause: here, we constrain the size (length) of the left hand side of the rules, referred to in MSQL as body (line 3), their minimum support (line 4), and their minimum confidence (line 5). Finally, the using-clause allows the user to discretize numerical attributes on the fly. In line 6, we specify that the encoding called "temp_encoding" should be applied to the attribute *Temperature*.

MSQL also allows the user to store the resultant rules in a persistent rule base, although the format in which the rules are stored is opaque to the user. This storage is possible by adding the into-clause to the MSQL query, as in line 2. In this example, the name of the rule base is called "MyRules".

Lift task. As previously mentioned, MSQL offers an operator for querying mining results, which is called *SelectRules*. For example, the following MSQL query retrieves all rules with attribute *Play* as the right hand side, referred to in MSQL as consequent, from the rule base MyRules:

```
1. SelectRules(MyRules)
2. where Consequent is {(Play=*)}
```

The operator *SelectRules* retrieves the rules previously stored in the rule base MyRules (given in parentheses in line 1) that fulfill the constraints specified in the where-clause (line 2). These can only be constraints posed on the length of the rules, the format of the consequent (as in the query above), the format of the body, support or confidence of the rules. The constraint on lift, required by the current task, cannot be expressed in this language, which means that the lift task cannot be completely performed in MSQL.

3.3.3 MINE RULE

Another example is the operator MINE RULE [11] designed as an extension of SQL, which was also proposed for association rule mining discovery in relational databases. An interesting aspect of this work is that the operational semantics of the proposed operator is also presented in [11], by means of an extended relational algebra. Additionally, in [12], the same authors specified how the operator MINE RULE can be implemented on top of an SQL server.

As an example, consider the MINE RULE query given below:

```
1. Mine Rule MyRules as
2. select distinct 1..1 Outlook, Humidity as body,
                   1..1 Play as head,
                   support, confidence
3. from Playtennis2
4. group by Day
5. extracting rules with support: 0.20,
                         confidence: 0.80
```

This query extracts rules from the source table Playtennis2, as defined in line 3. The execution of this query creates a relational table called “MyRules”, specified in line 1, where each tuple is an association rule. The select clause in line 2 defines the structure of the rules to be extracted: the body has schema $\{Outlook, Humidity\}$, while the head has schema $\{Play\}$. The notation “1..1” specifies the minimum and maximum number of schema instantiations in the body and head of the extracted rules, which is referred to as their cardinalities.

The select-clause also defines the schema of the relational table being created, which is limited to the body, head, support and confidence of the rules, the last 2 being optional. In the example query above, the schema of table MyRules consists of all these attributes.

Similar to DMQL, the operator MINE RULE is able to extract both inter- and intra-tuple association rules. For the extraction of intra-tuple rules, the group-by clause in line 4 assures that each group in the source data coincides with a unique tuple. Finally, in line 5, the minimum support and minimum confidence are specified.
Next, we show how the given tasks are performed with MINE RULE.

Discretization task We assume here that the MINE RULE operator has been integrated into a database system based on SQL (as discussed in [12]). Given this, although MINE RULE does not provide any specific operator for discretization of numerical values, the discretization required by this task can be performed by, e.g., the SQL CASE expression below. Such an expression is available in a variety of database systems, e.g., PostgreSQL[1], Oracle[2], Microsoft SQL Server[3] and MySQL[4].

```
1. create table MyPlaytennis as
2. select Day, Outlook,
3. case
   when Temperature between 60 and 69 then '[60,69]'
   when Temperature between 70 and 79 then '[70,79]'
   when Temperature between 80 and 89 then '[80,89]'
   end as Temperature,
4. Humidity, Wind, Play
5. from Playtennis2
```

The query above creates a table called "MyPlaytennis". It is in fact a copy of table Playtennis2 (see line 5), except that the attribute *Temperature* is now discretized into 3 intervals: [60,69],[70,79], and [80,89] (see line 3).

Area task. Similarly to MSQL, MINE RULE was specially developed for association rule discovery. Therefore, the area task cannot be performed with MINE RULE.

Right hand side task. Rules extracted with a MINE RULE query have only the body and head schemas specified in the query. For example, all rules extracted with the example MINE RULE query above have the body with schema $\{Outlook, Humidity\}$ and head with schema $\{Play\}$. To perform this task, which asks for all rules of size of at most 3 and a singleton right hand side, we would need to write as many MINE RULE queries as are the possible combinations of disjoint body and head schemas. On the other hand, since MINE RULE is also capable of mining inter-tuple association rules, in particular single-dimensional association rules[5] [3], an alternative solution to obtain these rules is to firstly pre-process table MyPlaytennis into a new table, by breaking down each tuple t in MyPlaytennis

[1] http://www.postgresl.org/

[2] http://www.oracle.com/index.html/

[3] http://www.microsoft.com/sqlserver/2008/en/us/default.aspx/

[4] http://www.mysql.com/

[5] Single-dimensional association rules are rules that contain multiple occurrences of the same attribute, although over different values.

Fig. 3.2 Table MyPlaytennisTrans: the pre-processed MyPlaytennis data table created before using the MINE RULE operator.

MyPlaytennisTrans

Day	Condition
D1	"Outlook=Sunny"
D1	"Temperature=[80,89]"
D1	"Humidity=High"
D1	"Wind=Weak"
D1	"Play=No"
D2	"Outlook=Sunny"
...	...

into 5 tuples, each tuple representing one attribute-value pair in t (except the primary key). A sample of the new table, called "MyPlaytennisTrans", is depicted in Figure 3.2.

After the pre-processing step, the right hand side task can now be accomplished with the following query:

```
1. Mine Rule MyRules as
2. select distinct 1..2 Condition as body,
                   1..1 Condition as head,
                   support, confidence
3. from MyPlaytennisTrans
4. group by Day
5. extracting rules with support: 0.20,
                         confidence: 0.80
```

Here, the body and head of the extracted rules are built from the domain of the attribute *Condition* (attribute of table MyPlaytennisTrans). The body has cardinality of at most 2, while head has cardinality 1, as requested by this task.

Figure 3.3 shows the resulting table MyRules and illustrates how an association rule, e.g., "if outlook is sunny and humidity is high, you should not play tennis" is represented in this table. [6]

MyRules

body	head	support	confidence
{"Outlook=Sunny","Humidity=High"}	{"Play=No"}	0.21	1
...	...	...	...

Fig. 3.3 The table MyRules created by a MINE RULE query.

[6] For ease of presentation, we adopted here the same representation as in [11]. In [12] the authors suggest that the body and the head itemsets of the generated rules are stored in dedicated tables and referred to within the rule base table, in this case the table MyRules, by using foreign keys.

Lift task. For the execution of the previous task, the table MyRules, containing the extracted rules, was created. Note, however, that the table MyRules contains only the body, head, support and confidence of the rules. Indeed, to the best of our knowledge (see [3, 9, 11, 12]), the supports of the body and head of the rules are not stored in the database for being further queried. As a result, measures of interest, such as lift, cannot be computed from the mining results without looking again at the source data.

Although there is no conceptual restriction in MINE RULE that impedes the execution of this task, we assume here that the lift task cannot be performed using the operator MINE RULE, based on its description given in [3, 9, 11, 12].

3.3.4 SIQL

The system prototype SINDBAD (Structured Inductive Database Development), developed by Wicker et al. [17], provides an extension of SQL called SIQL (Structured Inductive Query Language). SIQL offers new operators for several data mining tasks, such as itemset mining, classification and clustering, and also for preprocessing, such as discretization and feature selection.
In the following, we present how the given tasks are performed in SIQL.

Discretization task. In SIQL, this task can be executed with the following query:

```
1. configure discretization numofintervals = 3
2. create table MyTable as
3. discretize Temperature in Playtennis2
```

In this language, all available operators should actually be seen as functions that transform tables into new tables. The query above, for example, produces a table called "MyTable"(see line 2), which is a copy of table Playtennis2, except that the attribute *Temperature* is now discretized according to the parameter previously configured in line 1. In this example, we discretize the attribute *Temperature* into 3 intervals, as requested by the discretization task.

Area task. For the frequent itemset mining task, SIQL allows the user to specify only the minimum support constraint, as follows:

```
1. configure apriori minSupport = 0.20
2. create table MySets as
3. frequent itemsets in MyTable
```

This query produces the table MySets (line 2) that contains the Boolean representation of the intra-tuple frequent itemsets found in table MyTable (line 3), which was previously created for the discretization task.[7]

[7] Before the mining can start, table MyTable needs to be encoded in a binary format such that each row represents a tuple with as many Boolean attributes as are the possible attribute-value pairs.

Items

item_id	item
1	⟨Outlook, Sunny⟩
2	⟨Humidity, High⟩
...	...

Itemsets

itemset_id	item_id
1	1
1	2
...	...

Supports

itemset_id	support
1	3
...	...

Fig. 3.4 Materialization of SPQL queries.

Observe that, in this language, the attention is not focused on the use of constraints: the minimum support constraint is not posed within the query itself; it needs to be configured beforehand with the use of the so-called configure-clause (line 1). The minimum support constraint is therefore more closely related to a function parameter than to a constraint itself. Additionally, the number of such parameters is limited to the number foreseen at the time of implementation. For example, the constraints on size and area are not possible to be expressed in SIQL. We conclude, therefore, that the area task cannot be executed in this language.

Right hand side task. Although SIQL offers operators for several different mining tasks, there is no operator for association rule mining. This means that this task cannot be executed in SIQL.

Lift task. Due to the reason given above, the lift task is not applicable here.

3.3.5 SPQL

Another extension of SQL has been proposed by Bonchi et al. [2]. The language is called SPQL (Simple Pattern Query Language) and was specially designed for frequent itemset mining. The system called ConQueSt has also been developed, which is equipped with SPQL and a user-friendly interface.

The language SPQL supports a very large set of constraints of different types, such as anti-monotone [10], monotone [10], succinct [13], convertible [14], and soft constraints [1]. Additionally, it provides an operator for discretization of numerical values. Another interesting functionality of SPQL is that the result of the queries is stored into the database. The storage creates 3 different tables, as depicted in Figure 3.4. The figure also shows how the itemset ($Outlook =$ 'Sunny' $\wedge$ $Humidity =$ 'High') is stored in these tables.
Below, we illustrate how the given tasks are executed in SPQL.

Discretization task. In SPQL, this task can be performed as below:

```
1. discretize Temperature as MyTemperature
2. from Playtennis2
3. in 3 equal width bins
4. smoothing by bin boundaries
```

In this example, we discretize the attribute *Temperature* into 3 intervals (bins), as requested by this task, with the same length (line 3), and we also want the bin boundaries to be stored as text (line 4) in a new attribute called "MyTemperature" (specified in line 1).

Area task. In SPQL, the user is allowed to constrain the support and the size of the frequent itemsets to be mined, as follows:

```
1. mine patterns with supp >= 3
2. select *
3. from Playtennis2
4. transaction Day
5. item Outlook, MyTemperature, Humidity, Wind, Play
6. constrained by length >= 2
```

The language allows the user to select the source data (lines from 2 to 5), the minimum absolute support, which is compulsory to be defined at the beginning of the query (line 1), and a conjunction of constraints, which is always posed at the end of the query. In this example, only the constraint on the size (length) of the itemsets is posed (line 6).

SPQL is able to extract both inter- and intra-tuple itemsets. For the extraction of intra-tuple itemsets, line 4 guarantees that each group in the source data corresponds to a unique tuple, while line 5 lists the attributes for exploration.

The constraint on the minimum area (absolute support $\times$ size), however, is apparently not possible to be expressed in this language, since the property support of an itemset cannot be referred to anywhere else but at the beginning of the query. Besides, it is not clear in [2] whether formulas such as support $\times$ length can be part of the conjunctions of constraints that are specified at the end of the queries. On the other hand, note that a post-processing query on the tables presented above, would be an alternative to complete this task. Contrary to SIQL, SPQL also stores the support of the extracted itemsets, which are crucial to compute their area. We therefore conclude that the area task can be accomplished by SQPL, provided that a post-processing query is executed.

Right hand side task. SPQL was specially designed for itemset mining. Consequently, the right hand side task cannot be performed in this language.

Lift task. Given the reason above, the lift task is not applicable.

3.3.6 DMX

Microsoft SQL server 2005 [16] provides an integrated environment for creating and working with data mining models. It consists of a large set of data mining algorithms for, e.g., association rule discovery, decision tree learning, and clustering. In order to create and manipulate the so-called data mining models, it offers an extended version of SQL, called DMX (Microsoft's Data Mining extensions). DMX

is composed of data definition language (DDL) statements, data manipulation language (DML) statements, functions and operators.

In the following, we show how the given tasks can be performed by this language. DMX is able to extract both inter- and intra-tuple patterns. We focus here on the kind of patterns asked by the given tasks.

Discretization task. In DMX, the discretization of numerical values and creation of a data mining model itself can be done synchronously. This is shown below for the accomplishment of the right hand side task.

Area task. In DMX, frequent itemsets cannot be extracted independently from association rules. Nevertheless, the itemsets computed beforehand to form association rules can also be queried after mining such rules. Thus, to compute this task, the following steps are necessary. Firstly, a so-called association model has to be created as follows:

```
1. create mining model MyRules
2. (Day text, Outlook text,
    Temperature text discretized(Equal_Areas,3),
    Humidity text, Wind text, Play text)
3. using microsoft_association_rules
                          (minimum_support = 0.20,
                           minimum_probability = 0.80,
                           minimum_itemset_size = 2)
```

The above DMX query creates a model called “MyRules” that uses the values of the attributes defined in line 2 to generate association rules. The rules are extracted by the algorithm “Microsoft Association Rules”, having as parameters minimum_support, minimum_itemset_size, as required by the current task, and also minimum_probability (the same as confidence, which is set to speed up computation only, as we are just interested in the itemsets). In addition, the user can specify which attributes he or she wants to have discretized, as in line 2. In this example, we specify that the values of the attribute *Temperature* should be discretized into 3 intervals, as demanded by the discretization task.

Having created the model, it needs to be trained through the insertion of tuples, as if it was an ordinary table:

```
1. insert into MyRules
2. (Day, Outlook, Temperature, Humidity, Wind, Play)
3. select Day, Outlook, Temperature,
          Humidity, Wind, Play
4. from Playtennis2
```

When training the model, we explicitly say from where the values of its associated attributes come (lines 3 and 4).

After training the model, it is necessary to query its content in order to visualize the computed itemsets. The content of an association model is stored in the database as shown in Figure 3.5 [16]. It consists of 3 levels. The first level has a single node, which represents the model itself. The second level represents the frequent itemsets

computed to form the association rules. Each node represents one frequent itemset along with its characteristics, such as its corresponding support. The last level, in turn, represents the association rules. The parent of a rule node is the itemset that represents the left hand side of the rule. The right hand side, which is always a singleton, is kept in the corresponding rule node.

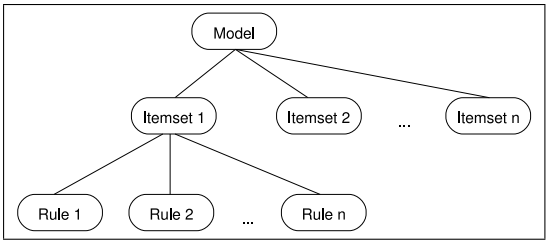

Fig. 3.5 The content of an association model in Microsoft SQL server 2005.

Each node keeps an attribute called "node_type", which defines the type of the node. For example, itemset nodes have node_type equal to 7, while rule nodes have node_type equal to 8. In addition to the attribute node_type, a text description of the itemset is kept in an attribute called "node_description", and its support in "node_support".[8] The description is a list of the items, displayed as a comma-separated text string, as in 'Outlook=Sunny, Humidity=High'. In order to query all itemsets in the model MyRules, along with their corresponding supports, the following DMX query is necessary:

```
1. select node_description, node_support
2. from MyRules.Content
3. where node_type = 7
```

In line 1, we select the text description and support of the rules. In line 2, we specify the model from which the content is to be queried. Finally, as we are only interested in the itemsets, we filter the nodes by their node types in line 3.

Note, however, that the current task asks for itemsets with size of at least 2 and area of at least 10. Therefore, a more complex DMX query is needed. As there is apparently no attribute in an itemset node keeping the size of the corresponding itemset, one needs to compute their sizes by processing the description of the itemset in the attribute node_description. By doing this, the area of the itemsets can also be computed. We assume therefore that this task can only be performed in DMX after the execution of a post-processing query.

Right hand side task. This task can be completely performed in DMX by following the same steps of the last task. Firstly, one needs to create a mining model similar to the one created above, except that here the parameter minimum_itemset_size is replaced with maximum_itemset_size, which is 3 for this task.

After training the model, we query for association rules as below:

[8] The specification of those attributes was found at `http://technet.microsoft.com`.

```
1. select node_description, node_support,
          node_probability
2. from MyRules.Content
3. where node_type = 8
```

For rules, the attribute node_description contains the left hand side and the right hand side of the rule, separated by an arrow, as in 'Outlook=Sunny, Humidity=High → Play = No'. In addition, its support is kept in the attribute called "node_support" and its confidence in "node_probability".[9] Thus, the above DMX query executes the right hand side task.

Lift task. Again, for performing this task, the user has to be aware of the names of the attributes that are kept in every rule node. The lift of an association rule (referred to as importance by [16]) is kept in a attribute called "msolap_node_score", while the characteristics of the right hand side of a rule can be found at a nested table called "node_distribution". [9]

The following DMX query performs the lift task:

```
1. select node_description, node_support,
            node_probability,
            (select attribute_name
            from node_distribution) as a
2. from MyRules.Content
3. where node_type=8
4.   and a.attribute_name = 'Play'
5.   and msolap_node_score >= 1
```

Here, we select from the content of the model MyRules only rules having attribute *Play* as the right hand side (line 4) that have a lift greater than 1 (line 5), just as required by this task. Thus, we can conclude that the lift task can be accomplished by DMX.

3.4 Summary of the Results

We now summarize the results achieved by the proposals presented in this chapter with respect to the accomplishment of the four given tasks. Table 3.1 shows, for each of the proposals, the performed tasks.

Discretization Task. Observe that the discretization task could be executed by all the proposals, although MINE RULE does not offer a specific operator for discretization. This shows that considerable attention is dedicated to pre-processing operations.

[9] The specification of those attributes was found at `http://technet.microsoft.com`.

Table 3.1 Results of each proposal for each task. The symbol $\surd\star$ means that the task was executed only after a pre- or post-processing query.

	Proposals					
	DMQL	MSQL	MINE RULE	SIQL	SPQL	DMX
Discretization task	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Area task					$\surd\star$	$\surd\star$
Right hand side task	$\surd$	$\surd$	$\surd\star$			$\surd$
Lift task				N/A	N/A	$\surd$

Area and Right Hand Side Tasks. From the results of the area and right hand side tasks, two main points can be concluded: firstly, the languages are not flexible enough to specify the kinds of patterns a user may be interested in. For example, MSQL and MINE RULE are entirely dedicated to the extraction of association rules, while SPQL was specially designed for frequent itemset mining. Concerning MINE RULE, the right hand side task could only be executed with a relatively high number of queries or after a pre-processing query (which was the approach we took). As for DMX, although it is able to perform the area task, we observe that there is not a clear separation between rules and itemsets.

The second point is that little attention is given to the flexibility of ad hoc constraints. For example, the constraint on area, which was required by the area task, could not be expressed in any of the proposals that can perform itemset mining. In fact, SPQL and DMX could only accomplish this task after the execution of a post-processing query. Note that the flexibility of these proposals is actually limited to the type of constraints foreseen by their developers; a new type of constraint in a mining operation which was not foreseen at the time of implementation will not be available for the user. In the particular cases of SIQL and DMX, a constraint is more closely related to a function parameter than to a constraint itself.

Lift Task. As for the lift task, we observed that little support is given to post-processing of mining results. Concerning DMQL, we are not aware of whether it considers the closure principle, that is, whether the results can be further queried, as opposed to the other data mining languages.

As for MSQL, although it gives the user the ability to store the mining rules in a rule base, the data space is totally opaque to the user. In other words, the rules can only be queried with the use of the operator *SelectRules*, and with a limit set of available constraints. In the case of MINE RULE, as opposed to MSQL, results are in fact stored in ordinary relational tables, but the format in which they are stored, with respect to the attributes, is not flexible enough. This restricts the number of possible constraints the user can express when querying those results.

Finally, observe that DMX is the only proposal that is able to perform the lift task. On the other hand, the models (and their properties) are stored in a very complex way with this language, making the access and browsing of mining results less intuitive.

3.5 Conclusions

Even though most of the limitations of the languages can be solved by minor extensions to the languages, the need to extend the languages itself is considered a drawback. In summary, we identify the following list of drawbacks noticed in at least one of the proposals surveyed in this chapter:

- There is little attention to the closure principle; the output of a mining operation cannot or only very difficultly be used as the input of another operation. While the closure principle is very important for the expressiveness of SQL, its data mining extensions mostly lack this advantage.
- The flexibility to specify different kinds of patterns and ad-hoc constraints is poor. If the user wants to express a constraint that was not explicitly foreseen by the developer of the system, he or she will have to do so with a post-processing query, if possible at all.
- The support for post-processing mining results is often poor due to a counter-intuitive way of representing mining results. Data mining results are often offered as static objects that can only be browsed or in a way that does not allow for easy post-processing.

In Chapter 11, we describe a new inductive database system which is based on the so-called virtual mining views framework. In addition, we show the advantages it has in comparison with the proposals described here.

Acknowledgements This work has been partially supported by the projects IQ (IST-FET FP6-516169) 2005/8, GOA 2003/8 "Inductive Knowledge bases", FWO "Foundations for inductive databases", and BINGO2 (ANR-07-MDCO 014-02). When this research was performed, Hendrik Blockeel was a post-doctoral fellow of the Research Foundation - Flanders (FWO-Vlaanderen), Élisa Fromont was working at the Katholieke Universiteit Leuven, and Adriana Prado was working at the University of Antwerp.

References

1. Bistarelli, S., Bonchi, F.: Interestingness is not a dichotomy: Introducing softness in constrained pattern mining. In: Proc. PKDD, pp. 22–33 (2005)
2. Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., Trasarti, R.: A constraint-based querying system for exploratory pattern discovery information systems. Information System (2008). Accepted for publication
3. Botta, M., Boulicaut, J.F., Masson, C., Meo, R.: Query languages supporting descriptive rule mining: A comparative study. In: Database Support for Data Mining Applications, pp. 24–51 (2004)
4. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Discovery Science, pp. 77–122 (2004)
5. Han, J., Chiang, J.Y., Chee, S., Chen, J., Chen, Q., Cheng, S., Gong, W., Kamber, M., Koperski, K., Liu, G., Lu, Y., Stefanovic, N., Winstone, L., Xia, B.B., Zaiane, O.R., Zhang, S., Zhu, H.: Dbminer: a system for data mining in relational databases and data warehouses. In: Proc. CASCON, pp. 8–12 (1997)

6. Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: DMQL: A data mining query language for relational databases. In: ACM SIGMOD Workshop DMKD (1996)
7. Han, J., Kamber, M.: Data Mining - Concepts and Techniques, 1st ed. Morgan Kaufmann (2000)
8. Imielinski, T., Virmani, A.: Msql: A query language for database mining. Data Mining Knowledge Discovery **3**(4), 373–408 (1999)
9. Jeudy, B., Boulicaut, J.F.: Constraint-based discovery and inductive queries: Application to association rule mining. In: Proc. ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining, pp. 110–124 (2002)
10. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1**(3), 241–258 (1997)
11. Meo, R., Psaila, G., Ceri, S.: An extension to sql for mining association rules. Data Mining and Knowledge Discovery **2**(2), 195–224 (1998)
12. Meo, R., Psaila, G., Ceri, S.: A tightly-coupled architecture for data mining. In: Proc. IEEE ICDE, pp. 316–323 (1998)
13. Ng, R., Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: Proc. ACM SIGMOD, pp. 13–24 (1998)
14. Pei, J., Han, J., Lakshmanan, L.V.S.: Mining frequent itemsets with convertible constraints. In: Proc. IEEE ICDE, pp. 433–442 (2001)
15. Srikant, R., Agrawal, R.: Mining generalized association rules. Future Generation Computer Systems **13**(2–3), 161–180 (1997)
16. Tang, Z.H., MacLennan, J.: Data Mining with SQL Server 2005. John Wiley & Sons (2005)
17. Wicker, J., Richter, L., Kessler, K., Kramer, S.: Sinbad and siql: An inductive databse and query language in the relational model. In: Proc. ECML-PKDD, pp. 690–694 (2008)