Knowledge Discovery Query Language (KDQL)

Zakaria Suliman Zubi
Computer Science Department, Faculty of Science, Al-Tahadi University, Sirt, Libya

Abstract: KDD is a rapidly expanding field with promise for great applicability. Knowledge discovery became the new database technology for the incoming years. The need for automated discovery tools caused an explosion in the number and type of tools available commercially and in the public domain. These requirements encouraged us to propose a new KDD model called ODBC_KDD(2) described in [39]. "One of the ODBC_KDD(2) model requirements is the implementation of a query language that could handle DM rules"[40]. This query language called Knowledge Discovery Query Language (KDQL). KDQL is a companion of two major tasks in KDD such as DM and Data Visualization. These requirements motivates us to think for the possibility of joining the two tasks of KDD commonly known as Data Mining (DM) and Data Visualization (DV) together in one single KDD process. Integrating DM and DV requires a new database concept. This database concept is called "i-extended database". I-extended database will be retrieved by the use of KDQL. This I-extended database described in details in [42]. KDQL RULES operations were also theoretically proposed in this paper and some examples were given as well. KDQL RULES are used only to find out the association rules in i-extended database we have.

The development and results of this paper would contribute to the data mining and visualization fields in several ways. The formulation of a set of heuristics for algorithms selection will help to clarify the matching between a specific problem and the set of best-suited algorithms or techniques (i.e. association rules) for solving it. These guidelines are expected to be useful and applicable to real DM projects.

Keywords: Data Mining (DM), Data Mining Query Language (DMQL), Knowledge Discovery in Databases (KDD), Query Optimization (QO), Rule Mining (RM), Association Rules (AR).

1. Introduction of KDQL

The background of KDQL came from the Structure Query Language (SQL) since several extensions to the SQL have been proposed to serve as a data mining query language (DMQL) described in [6, 27]. However, they do not sufficiently address how to visualize query results. We will investigate the requirements for a SQL describing the graphical representation of Knowledge Discovery Query (KDQ) results from the perspective of a large database system. With frequent map output and assesses several SQL extensions with respect to their treatment of the graphical representation. It concludes that the SQL + DM (rules) = is the appropriate form for this task at the user interface. DM (rules) are based on the Association Rules (AR) to interact with a spatial i-extended database. I-extended database or other type of databases such as relational databases can be accessed as well. The association rules will be obtained by the use of KDQL rules can be graphically represented in 2D and 3D charts. The KDQL syntax will be present also in appendix A Moreover; we hope that we will write queries in KDQL syntax in the near future in a demonstrated model.

2. Principles of DMQL rules to interact relational databases
Interacting relational databases is often necessary to specify the interesting set of data that have to be studied, and the kind of rules to be discovered, etc. Moreover, a graphical user interface is helpful for interactive mining of association rules because it facilitates interactively modification of the environment settings, including output styles and formats.

Besides the specification of the kinds of rules that have to be discovered, it is also beneficial to specify the syntactic forms of the rules to be discovered. For example, to find the relationships between the attributes status, gpa and birth place, in relevance to major, for the students born in "Libya", and by using the DMQL described in [12, 40] the structure will be as follows:

\[
\text{discover rules in the form} \\
\text{major}(s : \text{student}, x) \land Q(s, y) \rightarrow R(s, z) \\
\text{from student} \\
\text{where birth place = "Libya"} \\
\text{in relevance to major, gpa, status, and birth place.}
\]

This kind of inclusion of meta-rule forms in the query specification for focusing the search is called meta-rule guided mining [31].

3. Using KDQL to interact I-extended databases

As we know from my recent works described in [39, 41, 42] KDD can be considered as a process that can include steps like forming the data set, data transformations, discovery of patterns, searching for exceptions to a pattern, zooming on a subset of the data, and post-processing some patterns. We describe a comprehensive framework in which all these steps can be carried out by means of queries over i-extended database. I-extended database is a database that in addition to data also contains intentionally defined generalizations about the data. We formalize this concept [42]. The i-extended database consists of a normal database together with a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data. Then, looking for potential query languages built on top of SQL, we will consider association rule mining described in [30]. It is a serious step towards an implementation framework for databases, though it addresses only the association rule mining problem in this stage and perspectives are then discussed.

Data mining sets new challenges to database technology and new concepts and methods are needed for general purpose query languages [5, 36]. A possible approach is to formulate a data mining task as locating interesting sentences from a given logic that are true in the database. Then the task of the user/analyst can be viewed as querying this set, the so-called theory of the database. Formally, given a language \( L \) of sentences (or patterns), the theory of the database \( r \) with respect to \( L \) and a selection predicate \( q \) is the set \( Th(r, L, q) = \{\theta \in L | q(r ; \theta)\} \). The predicate \( q \) indicates whether a sentence of the language is interesting. This definition is quite general: asserting \( q(r, \theta) \) might mean that \( \theta \) is a property that holds, that almost holds, or that defines (in some way) an interesting subgroup of \( r \). This approach has been more or less explicitly used for various data mining tasks (see [9, 35] for a survey and [4] for a detailed study of this setting).

Discovering knowledge from data can be seen as a process containing several steps: understanding the domain, preparing the data set, discovering patterns, post-processing of discovered patterns, and putting the results into use [13]. This is an interactive and iterative process for which many related theories have to be computed: different selection predicates and also classes of patterns might be used. Therefore, a general-purpose query language should enable the user to select subsets of data, but
also to specify and select patterns. It should also support crossing the boundary between data and patterns, e.g., when exceptions to a pattern are to be analyzed or for sophisticated post-processing methods like rule covering [32]. This has motivated the concept of inductive databases, i.e., databases that contain inductive generalizations about the data, in addition to the usual data [5, 36].

The contribution of this paper concerns a formalization of this concept of i-extended database and a first approach for an implementation based on SQL servers. The formalization carries a two part basic message:

(i) a particular inductive database consists of a normal database associated to a subset of patterns from a class of patterns, and an evaluation function that tells how the patterns occur in the data;

(ii) a particular database can be queried (in principle) just by using a straightforward extension of relational algebra, this point of view is also considered in [34].

Searching for solutions based on SQL is motivated by the industrial perspective of relational database mining. A huge amount of work has already been done to provide efficient and portable implementations of SQL, and KDQL architectures between SQL servers and data mining systems. As a starting point, we will apply the KDQL rules operator proposed by the author. These rules could be something like the rules in [7, 8]. The i-extended database and KDQL was defined firstly in an ODBC_KDD(2) model described in [39], and the following figure shows the appearance of both i-extended database and KDQL in the proposed ODBC_KDD(2) model[39].

![Figure 1. Shows the ODBC_KDD(2) Model](image)
4. I-extended databases

The goal of using i-extended database is to describe a data model that makes it possible to view the whole or any part of the KDD process as querying a database structured according to the ODBC_KDD (2) model described in [39]. Thus the database has to contain both data and generalizations about that data. Detailed descriptions of i-extended database defined in [42]. This motivates the following definition (simplified from the one in [8]).

**Schema:** The schema of an i-extended database is a pair \( R = (R, (P_R e, V)) \), where \( R \) is a database schema, \( P_R \) is a collection of patterns, \( V \) is a set of result values, and \( e \) is the evaluation function that defines how patterns occur in the data. This function maps each pair \((r; \theta)\) to an element of \( V \), where \( r \) is a database over \( R \) and \( \theta \) is a pattern from \( P_R \).

**Instance:** An instance \((r; s)\) of an i-extended database over the schema \( R \) consists of a database \( r \) over the schema \( R \) and a subset \( s \subseteq P_R \).

The simple association rule-mining problem has received much attention since its introduction in [29]. The concept of i-extended database is quite general and is not dedicated to this class of patterns. However, for didactic reasons, we use it in our examples.

<table>
<thead>
<tr>
<th>( s_0 )</th>
<th>( e(r_0).f )</th>
<th>( e(r_0).c )</th>
<th>( s_1 )</th>
<th>( e(r_1).f )</th>
<th>( e(r_1).c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B \Rightarrow A )</td>
<td>0.25</td>
<td>0.33</td>
<td>( C \Rightarrow A )</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>( C \Rightarrow A )</td>
<td>0.50</td>
<td>0.66</td>
<td>( C \Rightarrow A )</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>( A \Rightarrow B )</td>
<td>0.25</td>
<td>0.50</td>
<td>( A \Rightarrow B )</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>( C \Rightarrow B )</td>
<td>0.50</td>
<td>1.00</td>
<td>( C \Rightarrow B )</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>( A \Rightarrow C )</td>
<td>0.50</td>
<td>0.66</td>
<td>( A \Rightarrow C )</td>
<td>0.66</td>
<td>1.00</td>
</tr>
<tr>
<td>( B \Rightarrow C )</td>
<td>0.50</td>
<td>0.66</td>
<td>( B \Rightarrow C )</td>
<td>0.33</td>
<td>0.50</td>
</tr>
<tr>
<td>( C \Rightarrow AB )</td>
<td>0.25</td>
<td>1.00</td>
<td>( C \Rightarrow AB )</td>
<td>0.33</td>
<td>1.00</td>
</tr>
<tr>
<td>( B \Rightarrow AC )</td>
<td>0.25</td>
<td>0.50</td>
<td>( B \Rightarrow AC )</td>
<td>0.33</td>
<td>0.50</td>
</tr>
<tr>
<td>( A \Rightarrow BC )</td>
<td>0.25</td>
<td>0.50</td>
<td>( A \Rightarrow BC )</td>
<td>0.33</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( s_2 )</th>
<th>( e(r_2).f )</th>
<th>( e(r_2).c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C \Rightarrow B )</td>
<td>0.50</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Table 1. Patterns in three instances of i-extended database**

*Example 1:* Given a schema \( R = \{A_1, ..., A_n\} \) of attributes with domain \( \{0, 1\} \), and a relation \( r \) over \( R \), an association rule about \( r \) is an expression of the form \( X \Rightarrow B \), where \( X \subseteq R \) and \( B \in R \setminus X \) [29].

Intuitively, if a row of the matrix \( r \) has a 1 in each column of \( X \), then the row tends to have a 1 also in column \( B \). This semantics is captured by frequency and confidence values. Given \( W \subseteq R \),
\[ \text{freq}(W, r) \] denotes the fraction of rows of \( r \) that have a 1 in each column of \( W \). The frequency of the rule \( X \Rightarrow B \) in \( r \) is defined to be \( \text{freq}(X \cup \{B\}, r) \) while its confidence is \( \text{freq}(X \cup \{B\}, r) / \text{freq}(X, r) \). Typically, we are interested in association rules for which the frequency and the confidence are greater than given thresholds. However, we can define i-extended database such that \( P_R \) contains all association rules, i.e., \( P_R = \{X \Rightarrow B| X \subseteq R, B \in R \setminus X\} \). In this case, \( V \) is the set \([0, 1]^2\), and \( e(r, \theta) = (f(r, \theta), c(r, \theta)) \), where \( f(r, \theta) \) and \( c(r, \theta) \) are the frequency and the confidence of the rule \( \theta \) in the database \( r \).

**Queries:** A typical KDD process operates on both of the components of i-extended database. At each stage of manipulating the database \((r, s)\), the user can think that the value of \( e(r, \theta) \) is available for each pattern \( \theta \) which is present in the set \( s \). Obviously, if the pattern class is large, an implementation will not compute all the values of the evaluation function beforehand; rather, only those values \( e(r, \theta) \) that user’s queries require to be computed should be computed. Mining association rules as defined in example 1 is now considered as querying a i-extended database instances of schema \((R, (P_R, e, [0, 1]^2))\).

**Example 2** Assume the dataset is the instance \( r_0 \) in table1. of the schema \( R = \{A, B, C\} \). The i-extended database \( ptb = (r_0, s_0) \) associates to \( r_0 \) the rules on the leftmost table of table 1. We illustrate the selection on tuples \( Q_1 \) and the selection on patterns \( Q_2 \).

1. \((Q_1)\) Select tuples from \((r_0, s_0)\) for which the value for \( A \) is not 0. The result is a new instance \((r_j, s_j)\) where the data part \( r_j \) does not contain the tuple \((0, 1, 1)\), and the pattern part \( s_j \) contains the rules in the second table of table 1, i.e., the rules of \( s_0 \) with updated frequency and confidence values.

2. \((Q_2)\) Select rules from \((r_0, s_0)\) that exceed the frequency and confidence thresholds 0.5 and 0.7, respectively. A new instance \((r_0, s_2)\) is provided where \( s_2 \) contains the rules in the below table of table 1.

An important feature is that operations can be composed due to the closure property: an operation takes an instance of i-extended database and provides a new instance. For instance, the query \( Q_2 \circ Q_1 \) if applied to \((r_0, s_0)\) gives \((r_3, s_3)\), where \( r_3 \) is \( r_1 \) as defined above and \( s_3 \) is reduced to the association rule \( C \Rightarrow A \) with frequency 0.66 and confidence 1.

**KDQL:** Using the above definition for i-extended database it is easy to formulate query language for them. For example, we can write relational algebra queries, where in addition to the normal operations we can also refer to the patterns and the value of the evaluation function on the patterns. To refer to the values of \( e(r, \theta) \) for any \( \theta \in s \), we can think in terms of object-oriented databases: the evaluation function \( e \) is a method that encodes the behavior of the patterns in the data.

For the association rule example, it motivates the notations \( e(r).f \) and \( e(r).c \) when values for frequency and confidence are needed. Furthermore, it is useful to consider that other properties of patterns should be available; as for instance, the values for part of them, their lengths, etc. Following an abstract data type approach, we can consider operations that provide these properties. Hence, continuing example 1, we use body, lbody and head to denote respectively the value of the left-hand side, its length and the value of the right-hand side of an association rule. More generally, specifying i-extended database requires the definition of all these properties.
We now give a few queries by using, hopefully, self-explanatory notations for the simple extension of the relational algebra that fits to our need. Selection of tuple and patterns are respectively denoted by \( \sigma \) and \( \tau \) it is clear from the context, the operation is also applied on a \( i \)-extended database instances, e.g., we write \( \sigma_C ((r, s)) \) to denote \( (\sigma_C (r), s) \).

**Example 3** We now consider association rules in the particular and popular context of the basket analysis problem. Assume data is available in an instance of the schema \( R = \{Tid, Item, Price, Date\} \). \( Tid \) denotes the transaction identifier, \( Item \) the product purchased, \( Price \) its price and finally, \( Date \) the date for this transaction. By \( (r, s) \) we denote \( i \)-extended database for association rules between itemsets, \( s_0 \) denotes the intensionally defined collection of all these rules. Table 2(a) gives a dataset called \( r_0 \) in the result and one sample collection of patterns with their properties and answers in \( r_0 \). Notice that such a collection can typically be stored.

5. **KDQL RULES** operator

In the following, we provide an overview of the **KDQL RULES** operator and then discuss how it can be related to our work with \( i \)-extended database. KDQL rules is a *SQL like* operator which captures most of the association rule mining tasks that have been formulated so far (simple or generalized association rules, association rules with item hierarchies, etc). Moreover, there are quite efficient evaluation techniques that ensure the possibility of solving these DM tasks. It is not possible here to consider all the aspects of such an operator. We introduce it by means of one typical example and refer to [32] for other examples and a complete definition of its syntax and operational semantics. Given the dataset \( r_1 \) as defined in table 2, phase 4 is defined by the **KDQL rules** statement in table 3. The **KDQL rules** operator takes a relational database and produces an SQL3 table in [4], in which each tuple denotes a mined rule. Several possibilities exist to precisely define the input data. Basically, the whole potential of in a nested relation, e.g., an SQL3 table [4].

Consider the following process. First, the user decides to look at association rules derived from \( r_0 \), the dataset for the current month, and he/she wants to prune out all rules that have confidence under 30% or frequency under 5% or more than 7 items (phase 1 in table 2(b)). Then, he/she decides to focus on the rules that hold for the data about the last discount day (say \( Date = 13 \)) and to restrict to 5 the maximum amount of items in the rule (phase 2). Then, he/she wants to eliminate all the patterns that contain item \( D \) in their body. Finally, he/she tries to get association rules that imply expensive items (say \( Price \geq 7 \)). A lower threshold for frequency (say 1%) is considered for phase 4.

Different types of KDD processes could be easily described using the notion of \( i \)-extended database. The key is the closure property, which makes the composition of queries possible [13].
SQL can be used here. The input tables might themselves have been selected using the second WHERE clause. Rules are extracted from groups as defined by a GROUP BY clause (frequency is related to groups and if the clause is missing, any tuple is a group). The schema of the output table is determined by the SELECT clause that defines the structure of the rules (here, BODY, HEAD, SUPPORT and CONFIDENCE).

Sizes of the two components of a rule can be bounded (4 and 1 in our example). The keyword DISTINCT specifies that duplicates are not allowed in these components.

Data is encoded such that one gets all possible couples of itemsets (extracted from the groups) for the body and the head of a rule. It is possible to express mining conditions (first WHERE clause) that limit the tuples involved in this encoding. In our example, the mining condition indicates that Item in the body should not be D. An interesting feature is that DM conditions can be different for body and head, e.g., BODY.price < 7 AND HEAD.price >= 7 indicates that one wants association rules with cheap
products (less than 7) in the body and an expensive product in the head. It is possible to choose the types of the elements in the rules (e.g., Price instead of Item) as well as grouping attributes. This enables the specification of many different mining tasks over the same dataset.

In fact, most of the association rule mining tasks identified in the literature can be specified by means of a KDQL rules statement.

Data and patterns are then a collection of SQL tables. The phases of the simple scenario given in table 2(b) are easily translated into KDQL rules queries as given in table 3. Note that phase 3 is not achieved by means of a KDQL rules statement. Instead, we use a query over the materialization of $s_2$.

Table 3 Phases 1 to 4 of table 2 using KDQL rules

<table>
<thead>
<tr>
<th>Phase 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KDQL RULE</strong> $s_1$ AS SELECT DISTINCT 1..6 Item AS BODY, 1..1 Item AS HEAD, SUPPORT, CONFIDENCE FROM $r_0$ GROUP BY Tid EXTRACTING RULES WITH SUPPORT: 0.05, CONFIDENCE: 0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * AS FROM $s_2$ WHERE D NOT IN BODY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KDQL RULE</strong> $s_2$ AS SELECT DISTINCT 1..4 Item AS BODY, 1..1 Item AS HEAD, SUPPORT, CONFIDENCE FROM (SELECT * AS $r_1$ FROM $r_0$ WHERE Date=13) GROUP BY Tid EXTRACTING RULES WITH SUPPORT: 0.05, CONFIDENCE: 0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KDQL RULE</strong> $s_3$ AS SELECT DISTINCT 1..4 Item AS BODY, 1..1 Item AS HEAD, SUPPORT, CONFIDENCE WHERE BODY.Item&lt;&gt; D FROM (SELECT * AS $r_2$ FROM $r_1$ WHERE Price &gt;=7) GROUP BY Tid EXTRACTING RULES WITH SUPPORT: 0.01, CONFIDENCE: 0.3</td>
</tr>
</tbody>
</table>
The mining algorithms that can not be expressed in terms of SQL queries are activated by the so-called **core operator**. The three main components of the architecture are not so far from the defined in [33] are:

- **Preprocessor**: after the interpretation of a KDQL rules statement, preprocessor retrieves source data, evaluates the mining, grouping, cluster conditions, and encodes the data that will appear in the rules: it produces a set of encoded tables that are stored in the database. These encoded tables are optimized in the sense that mining conditions have been already applied and that unfrequented items do not appear anymore. Practically it has to be defined in the future.

- **Core operator**: The core operator uses these encoded tables and performs the generation of the association rules using known algorithms, e.g., apriori [1]. It then provides encoded rules. Basically, from each pair of **body and head**, elements are extracted to form a rule that satisfy DM conditions and both frequency and confidence criteria. This is a proposed operation and it will be a good challenge to my future work.

- **Post-processor**: At the end of the process, the post-processor decodes the rules and produces the relations containing the desired rules in a table that is also stored in the database. It has to be defined in the future work.

### 6. KDQL in KDD process

The goal of knowledge discovery is to obtain useful knowledge from large collections of data. Such a task is inherently interactive and iterative: one cannot expect to obtain useful knowledge simply by pushing many data to a black box. The user of a KDD system has to have a solid understanding of the domain in order to select the right subsets of data, suitable classes of patterns, and good criteria for interestingness of the patterns. Thus KDD systems should be seen as interactive tools, not as automatic analysis systems.

Discovering knowledge from i-extended database by KDQL should therefore be seen as a process containing several steps:

1. Understanding the domain,
2. Preparing the data set,
3. Discovering patterns (DM),
4. Post-processing of discovered patterns, and
5. Putting the results into use.

See [5] for a slightly different process model and excellent discussion.

The KDD process is necessarily iterative: the results of a DM step can show that some changes should be made to the data set formation step, post-processing of patterns can cause the user to look for some slightly modified types of patterns, etc. Efficient support for such iteration is one important development topic in KDD.

Prominent applications of KDD include health care data, financial applications, and scientific data [37, 3]. In industry, the success of KDD is partly related to the rise of the concepts of data warehousing and on-line analytical processing (OLAP). These strategies for the storage and processing of the accumulated data in an organization have become popular in recent years. KDD and DM can be viewed as ways of realizing some of the goals of data warehousing and OLAP.
7. KDQL algorithms and architecture

7.1 Architecture

In figure 2, we proposed the architecture of the KDQL which consists of a standard SQL query, or an SQL query plus a KDQL rules operation statement. Joining the SQL classical statement and the KDQL rules operation together requires an encoding/decoding operation. Encoding/decoding process will translate the query and send the request to i-extended database and then to a traditional databases via ODBC or JDBC drivers. The encoding/decoding process will get the response from the ODBC or JDBC drivers. The answer will be passed to the visualization process in a table. The visualization process will represent the table in a visualization chart mode. Charts will be appeared in 2D or 3D mode.

The encoding/decoding are part of the query system for formulating data mining queries such as KDQL. The communication between this system and the database can be carried out in ODBC or JDBC.

Searching for patterns and rules in traditional databases or in i-extended database by using KDQL query. KDQL requires some processes such like encoding/decoding, using sequence of SQL statements to capture interesting dataset such as association mining.
7.2 Algorithms

A fairly large class of DM tasks can be described as the search for interesting and frequently occurring patterns from the data. That is, we are given a class $P$ of patterns or sentences that describe properties of the data, and we can specify whether a pattern $p \in P$ occurs frequently enough and is otherwise interesting. That is, the generic data mining task is to find the set:

$$PI(d, P) = \{ p \in P | p \text{ occurs sufficiently often in } d \text{ and } p \text{ is interesting} \}.$$  

A formalism would be to consider KDQL as a language of sentences and view DM as the problem of finding the sentences in KDQL that are “sufficiently true” in the data and furthermore fulfill the user's other criteria for interestingness.

This point of view has either implicitly or explicitly been used in discovering integrity constraints from databases, in inductive logic programming, and in machine learning [20, 37, 21, 22, 9] some theoretical results can be found in [23], and a suggested logical formalism in [10].

While the frequency of occurrence of a pattern or the truth of a sentence can define rigorously, the interestingness of patterns or sentences seems much harder to specify and measure.

A general algorithm for finding $PI(d, P)$ is to first compute all frequent patterns by the following algorithm for finding all frequent patterns, and then select the interesting ones from the output.

**Algorithm 1.** Finding all frequent patterns. Assume that there is an ordering $<$ defined between the patterns of $P$.

1. $C := \{ p \in P | \text{for no } q \in P \text{ we have } q < p \}$; $C$ contains the initial patterns from $P$;
2. **while** $C \neq 0$ do
3. for each $p \in C$
4. find the number of occurrences of $p$ in $d$;
5. $F := F \cup \{ p \in C | p \text{ is sufficiently frequent in } d \}$;
6. $C := \{ p \in P | \text{all } q \in P \text{ with } q \leq p \text{ have been considered already and it is possible that } p \text{ is frequent} \}$;
7. **od**;
8. **output** $F$.

The algorithm proceeds by first investigating the initial patterns with no predecessors in the ordering $< p$. Then, the information about frequent patterns is used to generate new candidates, i.e., patterns that could be frequent on the basis of the current knowledge.

In the next Section we show how this algorithm can be used to solve association mining problems. If line 6 is instantiated differently, hill-climbing searches for best descriptions [24, 37] can also be fitted into this framework. In hill-climbing, the set $C$ will contain only the neighbors of the current "most interesting" pattern.

The generic algorithm suggests a KDQL architecture system consisting of a discovery module and a database management system. The discovery module sends queries to the database, and the database answers. The queries are typically of the form "How many objects in the database match $p$", where $p$ is a possibly
interesting pattern, the database answers by giving the count.

If implemented naively, this architecture leads to slow operations. To achieve anything resembling the efficiency of tailored solutions, the database management system should be able to utilize the strong similarities between the queries generated by the discovery module.

The view of KDQL as locating frequently occurring and interesting patterns from data suggests that KDQL can benefit from the extensive research done in the area of combinatorial pattern matching (CPM); see, e.g., [38]. One can even state the following CPM principle of KDQL:

It is better to use complicated primitive patterns and simple logical combinations than simple primitive patterns and complex logical form.

8. Association rules algorithms

In this Section, we discuss using KDQL algorithms to discover DM problems such as association rules where the above algorithm can be used.

Given a schema \( R = \{A_1, \ldots, A_p\} \) of attributes with domain \( \{0,1\} \), and a relation \( r \) over \( R \), an association rule [12] about \( r \) is an expression of the form \( X \rightarrow B \), where \( X \subseteq R \) and \( B \in R \setminus X \). The intuitive meaning of the rule is that if a row of the matrix \( r \) has a 1 in each column of \( X \), then the row tends to have a 1 also in column \( B \).

Examples of data where association rules might be applicable include the following:

- A student database at a university: rows correspond to students, columns to courses, and a 1 in entry \((s, c)\) indicates that the student \( s \) has taken course \( c \).

- Data collected from bar-code readers in supermarkets: columns correspond to products, and each row corresponds to the set of items purchased at one time.

- A database of publications: the rows and columns both correspond to publications, and \((p, p') = 1\) means that publication \( p \) refers to publication \( p' \).

- A set of measurements about the behavior a system, say exchanges in a telephone network. The columns correspond to the presence or absence of certain conditions and each row correspond to a measurement: if entry \((m, c)\) is 1, then at measurement \( m \) condition \( c \) was present.

Given \( W \subseteq R \), we denote by \( s(W, r) \) the frequency of \( W \) in \( r \); the fraction of rows of \( r \) that have a 1 in each column of \( W \). The frequency of the rule \( X \rightarrow B \) in \( r \) is defined to be \( s(X \cup \{B\}, r) \), and the confidence of the rule is \( s(X \cup \{B\}, r) = s(X, r) \).

In the discovery of association rules, the task is to find all rules \( X \rightarrow B \) such that the frequency of the rule is at least a given threshold \( \sigma \) and the confidence of the rule is at least another threshold \( \theta \). In large retailing applications the number of rows might be \( 10^6 \) or even \( 10^8 \), and the number of columns around 5000. The frequency threshold \( \sigma \) typically is around \( 10^{-2} \) to \( 10^{-4} \). The confidence thresholds can be hundreds of thousands of association rules. (Of course, one has to be careful in assigning any statistical significance to findings obtained from such methods.)

Note that there is no predefined limit on the number of attributes of the left-hand side \( X \) of an association rule \( X \rightarrow B \), and \( B \) is not fixed, either. This is important so that unexpected
associations are not ruled out before the processing starts. It also means that the search space of the rules has exponential size in the number of attributes of the input relation. Handling this requires some care for the algorithms, but there is a simple way of pruning the search space.

We call a subset \( X \subseteq R \) frequent in \( r \), if \( s(X, r) \geq \sigma \). Once all frequent sets of \( r \) are known, finding the association rules is easy. Namely, for each frequent set \( X \) and each \( B \in X \) verify whether the rule \( X \setminus \{B\} \Rightarrow B \) has sufficiently high confidence.

How can one find all frequent sets \( X \)? This can be done in a multitude of ways [5, 15, 29, 25, 2, 11]. A typical approach [5] is to use that fact that all subsets of a frequent set are also frequent. A way of applying the framework of Algorithm find all frequent patterns is as follows.

First find all frequent sets of size 1 by reading the data once and recording the number of times each attribute \( A \) occurs. Then form candidate sets of size 2 by taking all pairs \( \{B, C\} \) of attributes such that \( \{B\} \) and \( \{C\} \) both are frequent. The frequency of the candidate sets is again evaluated against the database. Once frequent sets of size 2 are known, candidate sets of size 3 can be formed, these are sets \( \{B, C, D\} \) such that \( \{B, C\}, \{B, D\}, \) and \( \{C, D\} \) are all frequent. This process is continued until no more candidate sets can be formed.

As an algorithm, the process is as follows.

**Algorithm 2.** Finding frequent sets for association rule.

1. \( C := \{\{A\} \mid A \in R\} \); 
2. \( F := \emptyset \); 
3. \( i := 1 \); 
4. while \( C \neq \emptyset \) do 
5. \( F' := \) the sets \( X \in C \) that are frequent; 
6. \( \text{add } F' \text{ to } F \); 
7. \( C := \) sets \( Y \) of size \( i+1 \) such that each subset \( W \) of \( Y \) of size \( i \) is frequent; 
8. \( i := i+1 \); 
9. \( \text{od} \);

The algorithm has to read the database at most \( K + 1 \) times, where \( K \) is the size of the largest frequent set. In the applications, \( K \) is small, typically at most 10, so the number of passes through the data is reasonable.

A modification of the above method is obtained by computing for each frequent set \( X \) the sub relation \( r_X \) consisting of those rows \( t \subseteq r \) such that \( t[A] = 1 \) for all \( A \in X \). Then it is easy to see that for example \( r_{\{A,B,C\}} = r_{\{A,B\}} \cap r_{\{B,C\}} \). Thus the relation \( r_X \) for a set \( X \) of size \( k \) can be obtained from the relations \( r_X \) and \( r_{X'} \), where \( X' = X \setminus \{A\} \) and \( X'' = X \setminus \{B\} \) for some \( A, B \in X \) with \( A \neq B \). This method has the advantage that rows that do not contribute to any frequent set will not be inspected more than once. For comparisons of the two approaches, see [31, 33, 28].

The algorithms described above work quite nicely on large input relations. Their running time is approximately \( O(NF) \), where \( N = np \) is the size of the input and \( F \) is the sum of the sizes of the sets in the candidate collection \( C \) during the operation of the algorithm [10]. This is nearly linear, and the algorithms seem to scale nicely to tens of millions of examples. Typically the only case when they fail is when the output is too large, i.e., there are too many frequent sets.

The methods for finding frequent sets are simple: they are based on one nice but simple observation (subsets of frequent sets must be frequent), and use straightforward implementation techniques.

A naive implementation of the algorithms on top of a relational database system would be easy: we need to pose to the database management system queries of the form " What is \( s(\{A_1, \ldots, A_k\}, r) \)?", or in SQL.
select count(*) from r t
where t[A_1] = 1 and ... and t[A_k] = 1

The number of such queries can be large: if there are thousands of frequent sets, there will be thousands of queries. The overhead in performing the queries by an ordinary DBMS would probably be prohibitive.

The customized algorithms described above are able to evaluate masses of such queries reasonably efficiently, for several reasons. First, all the queries are very simple, and have the same general form, thus there is no need to compile each query individually. Second, the algorithms that make repeated passes through the data evaluate a large collection of queries during a single pass. Third, the algorithm that builds the relations \( r_X \) for frequent sets \( X \) use the results of previous queries to avoid looking at the whole data for each query.

Association rules are a simple formalism and they produce nice results for binary data. The basic restriction is that the relation should be sparse in the sense that there are no frequent sets that contain more than about 15 attributes.

Namely, the framework of finding all association rules generates typically at least as many rules as there are frequent sets, and if there is a frequent set of size \( K \), there will be at least \( 2^K \) frequent sets.

The information about the frequent sets can actually be used to approximate fairly accurately the confidences and supports of a far wider set of rules, including negation and disjunction [12].

As an example, consider the simple case of mining for association rules in a course enrollment database. The user might say that he/she is interested only in rules that have the "Data Management" course on the left-hand side. This restriction can be utilized in the algorithm for finding frequent sets:

9. Sampling the results of KDQL

DM is often difficult for at least two reasons: first, there are lots of data, and second, the data is multidimensional. The hypothesis or pattern space is in most cases exponential in the number of attributes, so the multidimensionality can actually be the harder problem.

A simple way of alleviating the problems caused by the volume of data (i.e., the number of rows) is to use sampling. Even small samples can give quite good approximation to the association rules [5, 11] or functional dependencies [18] that hold in a relation. See [17] for a general analysis on the relationship between the logical form of the discovered knowledge and the sample sizes needed for discovering it.

The problem with using sampling is that the results can be wrong, with a small probability. A possibility is to first use a sample and then verify (and, if necessary, correct) the results against the whole data set. For instances of this scheme, see [11, 18]; also the generic algorithm can be modified to correspond to this approach. We give the sample-and-correct algorithm for finding functional dependencies.

**Algorithm 3.** Finding the keys of a relation by sampling and correcting.

**Input.** A relation \( r \) over schema \( R \).

**Output.** The set of keys of \( r \).

**Method.**

1. \( s := \text{a sample of } r \);  
2. \( K := \text{keys}(s) \);  
3. **while** there is a set \( X \in K \) such that \( X \) is not a key of \( r \) do  
4. \( \text{add some rows } u,v \in r \) with \( u[X] = v[X] \) to \( s \);  
5. \( K := \text{keys}(s) \);  
6. **od**;  
7. **output** \( K \).
10. KDQL by examples

In this Section, we introduce our KDQL operator using KDQL rules, showing its application to mining problems based on a practical case. The practical case is an extended database and classical database collecting purchase data of a food market. When a customer buys a set of products (also called items), the whole purchase is referred to as a transaction having a unique identifier, a date and a customer code.

<table>
<thead>
<tr>
<th>tr.</th>
<th>Customer</th>
<th>item</th>
<th>date</th>
<th>price</th>
<th>q.ty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cust1</td>
<td>milk</td>
<td>12/17/95</td>
<td>140</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>cust1</td>
<td>corn flaks</td>
<td>12/17/95</td>
<td>180</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>cust2</td>
<td>bread</td>
<td>12/18/95</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>cust2</td>
<td>cheese</td>
<td>12/18/95</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>cust2</td>
<td>coke</td>
<td>12/18/95</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>cust1</td>
<td>coke</td>
<td>12/18/95</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>cust2</td>
<td>bread</td>
<td>12/19/95</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>cust2</td>
<td>coke</td>
<td>12/19/95</td>
<td>300</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4 The Purchase table for a food-market

Association rules in literature, association rules were introduced in the context of the analysis of purchase data, typically organized in a way similar to that of the purchase table.

A rule describes regularities of purchased items in customer transactions. For example, the rule.

\{ cheese, coke \} \Rightarrow bread

States that if cheese and coke are bought together in a transaction, also bread is bought in the same transaction. In this association rules, the body is a set of items and the head is a single item. Note that the rule \{cheese, coke\} \Rightarrow cheese, is not interesting because it is a tautology: in fact if the head is implicated by the body the rule does not provide new information. This problem has the following formulation:

KDQL RULE Associations AS
SELECT DISTINCT 1..n item AS BODY,
DATA rules AS
SELECT DISTINCT 1..1 item AS HEAD,
SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY transaction
EXTRACTING RULES WITH
SUPPORT: 0.1,
CONFIDENCE: 0.2

The KDQL RULE operator produces a new table, called association, where each tuple corresponds to a discovered rule. The SELECT clause defines the structure of rules: the body is defined as a set of items whose cardinality is any positive integer as specified by 1..n; the head is defined as a set containing one single item, as specified by 1..1. The annotations 1..n and 1..1 are optional in the syntax of Appendix A this cardinalities are assumed by default when they omitted. The DISTINCT keyword states that no replications are allowed inside body or head. This keyword is mandatory because rules are meant to point out the presence of certain kind of items, independently of the number of their occurrences.
Furthermore, the **SELECT** clause indicates that the resulting table has four attributes: **BODY**, **HEAD**, **SUPPORT** and **CONFIDENCE**.

The **KDQL RULE** operator inspects data in the Purchase table grouped by transaction, as specified by the **GROUP BY** clause. Table 5 shows the purchase table after the grouping. Rules are extracted from within groups, their support is the number of groups satisfying the rules divided by the total number of groups, and their confidence is the number of groups satisfying the rule divided by the number of groups satisfying the body.

The clause **EXTRACTING RULES WITH** indicates that the operator produces only those rules whose support is greater than or equal to the minimum support and the confidence is greater than or equal to the minimum confidence. In this case, we have a minimum threshold for support of 0.1 and for confidence of 0.2. Table 6 shows the resulting **associations** table; observe that if we change the minimum support to 0.3, we then loose almost all rules of table 6 except those having 0.50 as support.

Variants of association rules several variants of the basic case of simple association rules are possible, in the following, we discuss them.

If we are interested only in extracting rules from a portion of the source table instead of the whole table, a selection on the source table is necessary.

<table>
<thead>
<tr>
<th>tr.</th>
<th>Customer</th>
<th>Item</th>
<th>date</th>
<th>Price</th>
<th>q.ty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>customer1</td>
<td>milk</td>
<td>12/17/95</td>
<td>140</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>customer1</td>
<td>corn_flaks</td>
<td>12/17/95</td>
<td>180</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>customer2</td>
<td>bread</td>
<td>12/18/95</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>customer2</td>
<td>cheese</td>
<td>12/18/95</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>customer2</td>
<td>coke</td>
<td>12/18/95</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>customer1</td>
<td>bread</td>
<td>12/19/95</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>customer2</td>
<td>coke</td>
<td>12/19/95</td>
<td>300</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 5 The Purchase table grouped by transaction**

<table>
<thead>
<tr>
<th>BODY</th>
<th>HEAD</th>
<th>S.</th>
<th>C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{milk}</td>
<td>{corn_flaks}</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>{corn_flaks}</td>
<td>{milk}</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>{bread}</td>
<td>{cheese}</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>{bread}</td>
<td>{coke}</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>{cheese}</td>
<td>{bread}</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>{cheese}</td>
<td>{coke}</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>{coke}</td>
<td>{bread}</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td>{coke}</td>
<td>{cheese}</td>
<td>0.25</td>
<td>0.33</td>
</tr>
<tr>
<td>{bread,cheese}</td>
<td>{coke}</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>{bread,coke}</td>
<td>{cheese}</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>{cheese,coke}</td>
<td>{bread}</td>
<td>0.25</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 6 The associations table containing association rules valid for data in purchase table**
Similarly to the classical SQL FROM clause, in our KDQL it is possible to specify an optional WHERE clause associated to the FROM clause. This clause creates a temporary table by selecting tuples in the source table that satisfy the WHERE clause, then, rules are extracted from this temporary table. For example, if we are interested only in purchases of items that cost no more than $150, we write:

**KDQL RULE Associations AS**
**SELECT DISTINCT 1..n item AS BODY, 1..1 item AS HEAD, SUPPORT, CONFIDENCE FROM Purchase WHERE price <= 150 GROUP BY transaction**
**EXTRACTING RULES WITH SUPPORT: 0.1, CONFIDENCE: 0.2**

If rules must be extracted only from within groups with a certain property, it is possible to use the classical SQL HAVING clause associated to the GROUP BY clause. Inside this clause, either aggregate functions (such as COUNT, MIN, MAX, AVG) or predicates on the grouping attributes can be used. For instance, if we like to extract rules from purchases of no more than six items, we write:

**KDQL RULE Associations AS**
**SELECT DISTINCT 1..n item AS BODY, 1..1 item AS HEAD, SUPPORT, CONFIDENCE FROM Purchase**
**GROUP BY transaction**
**HAVING COUNT (*) <= 6**
**EXTRACTING RULES WITH SUPPORT: 0.1, CONFIDENCE: 0.2**

With the KDQL RULE operator it is possible to group the source table by whichever attributes; this fact changes the meaning of extracted rules. For example, if the Purchase table were grouped by customer instead of the usual transaction, rules would describe regularities among customers, independently of the purchase transactions. Thus, we analyze the customer behavior without paying attention to the transactions in which items are purchased. We will give a simple example but we are not going to focus on clustering rules in this paper; it will be a future work. The problem is formalized as follows:

**KDQL RULE Customer Associations AS**
**SELECT DISTINCT item AS BODY, 1..n item AS HEAD, SUPPORT, CONFIDENCE FROM Purchase**
**GROUP BY customer**
**HAVING COUNT (*) <= 6**
**EXTRACTING RULES WITH SUPPORT: 0.1, CONFIDENCE: 0.2**

In [28] the case of association rules is extended to generalized association rules, i.e. rules with an arbitrary number of elements in the head. Our operator treats also this case, by means of a different specification for the cardinality of the head, that becomes 1..n instead of 1..1.

**KDQL RULE General Associations AS**
**SELECT DISTINCT item AS BODY, 1..n item AS HEAD, SUPPORT, CONFIDENCE FROM Purchase**
**GROUP BY transaction**
**EXTRACTING RULES WITH SUPPORT: 0.1, CONFIDENCE: 0.2**

10. Condensed KDQL representations

We remarked in Section 6 that KDD is an iterative process. Once a DM algorithm has been used to discover potentially interesting patterns, the user
often wants to view these patterns in different ways, have a look at the actual data, visualize the patterns, etc. A typical phenomenon is also that some pattern \( p \) looks interesting, and the user wants to evaluate other patterns that closely resemble \( p \). In implementing such queries, caching of previous results is obviously useful. Still, having to go back to the original data each time the user wants some more information seems somewhat wasteful. Similarly, in the KDQL algorithms presented in Section 9.7 the frequency and interestingness of each pattern are verified against i-extended database. It would be faster to look at some sort of short representation of the data.

Given a data collection \( d \in D \), and a class of patterns \( P \), a condensed representation for \( d \) and \( P \) is a data structure that makes it possible to answer queries of the form “How many times does \( p \in P \) occur in \( d \)” approximately correctly and more efficiently than by looking at \( d \) itself.

A simple example of a condensed representation is obtained by taking a sample from the data: by counting the occurrences of the pattern in the sample, one gets an approximation of the number of occurrences in the original data. Another, less obvious example is given by the collection of frequent sets of a 0-1 valued relation [12]: the collection of frequent sets can be used to give approximate answers to arbitrary Boolean queries about the data, even though the frequent sets represent only conjunctive concepts. The data cube [14] can also be viewed as a condensed representation for a class of queries. Similarly, in computational geometry the notion of an “approximation” [19] is closely related.

Developing condensed representations for various classes of patterns seems a promising way of improving the effectiveness of KDQL algorithms. Whether this approach is generally useful is still open.

11. Visual representation of the KDQL rules

For effective and interactive mining of i-extended database knowledge, visual representation of both data and knowledge has become an important issue for further study. Some interesting techniques can been developed for visual representation of association rules at a simple concept level using 2D and 3D dimensional feature tables such as bar charts, pie charts, or multi-dimensional curves [16]. There are also some interesting studies on the visual representation of association rules, such as using arrow width to represent the strength of rule implication [39], etc. visualizing the result table must focus on extracting right knowledge data rules and avoiding the redundant rules. However, the visual representation of rules at multiple concept levels is still largely an open issue for further study.

11.1 Removal of redundant rules

Discovering the knowledge's by the use of association rules at i-extended database or at the relational databases levels, similar rules could be generated at different concept levels in DBMS. Some of these rules can be considered as redundant and can be eliminated from the knowledge base to avoid the inclusion of many superfluous rules. An interesting measurement for the generation of only interesting rules at mining association rules proposed in [28] is as follows. In principle, a rule is considered redundant if it does not convey additional information and is less general than another rule. More specifically, a rule is considered interesting only if its support is more than \((r, s)\) times the expected value or its confidence is more than \((r, c)\) times the expected value, where \(r\) is a user specified minimum interest ratio. Efficient
methods have been developed to discover interesting (i.e., non redundant) association rules with non redundant values [28]. It seems that a similar measurement can be developed in mining other methodology rules at any concept levels. The judgment, detection and removal of redundant rules at mining various kinds of rules are an important issue for further study.

11.2 Data & Knowledge Querying

One of the reasons attributed to the great success of relational database technology has been the existence of a high-level, declarative, query language, which allows an application to express what conditions must be satisfied by the data it needs, rather than having to specify how to get the required data. Given the large number of patterns that may be mined, there appears to be a definite need for a mechanism to specify the focus of the analysis. Such focus may be provided in at least two ways. First, constraints may be placed on the database (perhaps in a declarative language) to restrict the portion of the database to be mined for, e.g. [39]. Second, querying may be performed on the knowledge that has been extracted by the mining process, in which case a language for querying knowledge rather than data is needed. An SQL like querying mechanism has been proposed for the KDQL as well details of which are provided in latter.

11.3 Visualizing KDQL results

The information visualization is a conjunction of a number of fields such as data mining, cognitive science, graphic design, and interactive computer graphics. Information visualization attempts to use visual approaches and dynamic controls to provide understanding and analysis of multidimensional data. The data may have no inherent 2D or 3D semantics and may be abstract in nature. There is no underlying physical model and much of the data in databases is of this type. The role of information visualization first acts as an exploratory tool, useful for identifying subsets of the data, structures trends and outliers may be identified, statistical tests tend incorporate isolated instances into a broader model as they attempt to formulate global features and then there is no requirement for an hypothesis, but the techniques can also support the formulation of hypotheses if wanted.

12 Interacting KDQL with I-extended databases

After we apply DM rules into KDQL empirically it will act like DMQL [12]. We will join the results to a visual mode such as in SQL+D [5], the SQL+D could be an alternative for visualizing the result in KDQL at this stage. In KDQL mode, we will use the classical SQL query language to formulate the query in the program at this level, but sooner or later. The program will understand the syntax of KDQL and my future challenge is to make it works. KDQL mode has the advantage that they can be understood easier than complex ones, and therefore they can provide valuable insight to analysts in order to understand i-extended database. The demonstration of the program is still under constriction and we hope it will be approved in the near future.
13 Conclusion

In this Paper we described the KDQL RULES operation and their four phases and how can the KDQL interact i-extended databases? The architecture of the KDQL was given, some remarkable algorithms such as association rules were shown, and examples were given as well. In this Paper there were some problems, which were not solved yet, and one of my future challenges is to solve them. One of these problems could be the applying of KDQL RULES syntax operation in the constricted demonstrated program instead of the classical SQL statement.

References


[34] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. SIGMOD'98, 13-24, Seattle, Washington.


[41] Zakaria S. Zubi, Knowledge Discovery in Remote Access Database, Ch. 8,9, PhD dissertation, Debrecen University, Hungary, 2002.

Appendix A
KDQL Syntax

A.1 KDQL syntax

This appendix presents the proposed syntax of the KDQL statement rules. In the syntax, square brackets denote the optionally. Productions based on the classical SQL syntaxes are as follow.

A.1.1 Denotations:
- `<FromList>` denotes the standard SQL clauses `FROM`.
- `<WhereClause>` denotes the standard SQL clauses `WHERE`.
- `<TableName>` denotes identifiers such as table names.
- `<AttributeName>` denotes identifiers such as attribute names.
- `<AttributeList>` denotes a list of attributes names to be identifier.
- `<Number>` denotes a positive integer.
- `<real>` denotes real numbers.

```
<KDQL_RULES_OP> ::= KDD RULES <TableName> AS SELECT DISTINCT <BodyDescr>, <HeadDescr>
[,<SUPPORT>] [,<CONFIDENCE>]
[WHERE <WhereClause>]
FROM <FromList> [WHERE <WhereClause>]
GROUP BY <Attribute> <AttributeList>
[HAVING <HavingClause>]
{{[CLUSTER BY <Attribute> <AttributeList> (It could be a future work)]}
[HAVING <HavingClause> ]}
EXTRACTING RULES WITH SUPPORT :<real> ,
CONFIDENCE :<real>
```

```
<Body_Description_KDQL>:=
[
<Cardinality_Sheap>] <AttributeName> <AttrList> AS BODY
/* default cardinality sheap for the Body: 1..n */
```

```
<Head_Description_KDQL>:=
[
<Cardinality_Sheap>] <AttributeName> <AttrList> AS BODY
/* default cardinality sheap for the Head: 1..1 */
```

```
<Cardinality_Sheap> ::= <Number> .. (<Number> | n)
```