Knowledge Discovery in Remote Access Databases

Doctor (Ph.D) Thesis

Zakaria Suliman Awad Zubi

Debrecen University

May 2002

Ezen értekezést a Debreceni Egyetem TTK Matematika és Számítástudományok doktori iskola Informatika programja keretében készítettem a Debreceni Egyetem TTK doktori (PhD) fokozatának elnyerése céljából.

Debrecen, 2002 / . . . ./ . . . . . .

Zakaria Suliman Awad Zubi

Tanúsítom, hogy ***Zakaria Suliman Awad Zubi*** doktorjelölt 1998- 2002 között a fent megnevezett Doktori Iskola Informatika programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Az értekezés elfogadását javasolom.

Debrecen, 2002/ . . . /. . . . . .

Dr.Fazekas Gábor

Abstract

Data mining is an emergent field, whose main goal is to discover useful patterns hidden in large databases. Because of its interdisciplinary nature, there is a wide variety of techniques and methods that come from diverse disciplines, such as statistics, database, machine learning, knowledge representation, and visualization. The Knowledge Discovery in Databases (KDD) process is modeled as an iterative process composed of several phases, each of which contains many obstacles, open problems, and research questions needing to be investigated and resolved. These reflect the current limitations of both humans and machines to generate, analyze, and interpret knowledge from large databases. To improve the data mining process requires strong theoretical and empirical research, that involves, mainly, the creation of better interfaces to database systems, new strategies to simplify the pre-processing stage, optimization and tuning of inductive learning algorithms and creation of a proposed new ones, and better techniques to interpret and evaluate patterns produced by data mining methods.

In this dissertation work, we will focus on two research problems within the KDD process: algorithm selection and algorithm engineering. Currently, the selection of a data mining algorithm that performs well in solving a data mining problem is rather subjective and it may lead to users and data analysts to make wrong decisions about the most appropriate technique for the problem being solved, or they may spend significant amount of time and effort trying to apply a technique that is not best suited to the problem. Thus, this course of research will introduce a set of heuristics to guide the user in the selection of the most appropriate methods for searching for patterns in a data set, for a particular problem or data mining task. In addition, other issues arise when the selected data mining algorithm is applied to a training dataset to induce a model. A model to data mining could be one of the remote access KDD models. One of these models which we used to call it ODBC _ KDD (2), was proposed by us. The methodology of this model began when an end user submitted a query. This query will be reconstructed to be what we used to call it knowledge discovery query language (KDQL). To meet the KDD process requirements the classical user query must have some extra parameters or rules to extract the hidden information or patterns in the databases. Many data mining algorithms rely on several parameters that the user must set, and that significantly affect the quality of generated patterns. Generating these patterns requires logical investigation in the form of data mining to be able to find out the association rules that we used to discover or mine. These rules help us to discover many associations in one particular database. Association rules can drive us to understand the behavior of our databases. The requirement of discovering the association rules in our databases leads us to think for a strong query language that could express more complex questions then the classical SQL. Such type of languages is called data mining query language (DMQL). Commonly, the user is forced to explore a huge parameter space without clues about which parameter settings are more convenient to induce an appropriate model for the dataset being explored. Also, when the induced model is used to predict new cases, it is fundamental that the model be represented in such form that the user can understand how the model is really working in making decisions, and then exploring alternative models based on the query language and also to the databases that have to be retrieved. According to the databases we implement a database concept called *i-extended database*. The main aim of this is to extract all the useful information from classical databases and store it a standard form to make it

suitable for establishing the knowledge discovery query language (KDQL). Regarding to the data mining query language we implement a data mining query language named as knowledge discovery query language (KDQL). The syntactic of *KDQL* came from the *Structured Query Language (SQL)* since several extensions to the *SQL* have been proposed to serve as a *data mining query language* (DMQL) described in DMQL Chapter 7. 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* to interact i-*extended database*. I-extended database can access to other type of databases such as relational databases. The association rules will be obtain by the use of KDQL rules, and then graphically represented in a 2D and 3D charts. The KDQL syntax will presented also in the appendix A, without a practical use. We also provide some practical scripts from the KDQL program by displaying some retrieving results with charts of four different types. Visualization result can significantly presented in 2D or 3D in forms such as: pie, bar, line and points charts.

To my father Suliman Zubi,

my mother Memona Bouseaf,

my wife Emaan Zubi

my kids Yahia, Mohamed and Suliman.

Acknowledgments

For contributions to the form and content of this thesis, my knowledge and my sanity, my first thanks are to my wife Emaan without whom this work wouldn't have been presented as it is. I am really indebted to Emaan for her everlasting understanding and making the last steps of writing this thesis enjoyable.

I am very grateful to my parents for their continuous moral support and encouragement. I hope I will make them proud of my achievements, as I am proud of them.

I wish to express my deep gratitude to my senior supervisor Prof. Dr.Arató Mátyás and to my real supervisor and mentor Dr.Fazekas Gábor. I thank him for accepting me as a Ph.D student under his supervision. I also thank Dr. Fazekas Gábor for continuous encouragement, confidence and support, reviewing the text of this thesis, and for sharing with me his knowledge and love of this field. I am sure Dr.Fazekas Gábor will always be my mentor and an example for perseverance and hard work. As an advisor, he taught me practices and showed me directions I will use in my academic career.

I am very thankful to Dr.Kormos Janos, my teacher and friend, for his insightful comments and advice. His observations and suggestions for improvement were very assuring, especially in the field of statistic and object oriented.

My gratitude and appreciation also goes to Dr. Bajalinov Erik for the frequent enthusiastic and constructive discussions regarding the programming in Delphi.

My deepest thanks to Dr.Varga Katalin and Dr.Várterész Magdolna for serving as examiners of my thesis.

I would also like to thank the many people in our department, supports and faculty, for always being helpful over the years. I also thank my friends in the institute of Mathematics and Informatics.

I would like also to thank the Libyan Security of Education for there financial supports through the Libyan Embassy at Budapest. My also thanks go to Mr. Basheer Nassain the Libyan student advisor and Mr. Khalid Zintaney the head of the financial office in the Libyan Embassy, Budapest for there understanding regarding my financial support.

Contents

List of Tables

Chapter 5

Chapter 7

Chapter 8

Chapter 9

List of Figures

Part I

Preface

Chapter 1
Overview

1.1 Introduction

Looming atop a wide variety of human activities are the menacing profiles of ever-growing mountains of data. These mountains grew as a result of great engineering successes that enabled us to build devices to generate, collect, and store digital data. With major advances in database, technology came to the creation of huge efficient data stores. Advances in computer networking have enabled the data glut to reach anyone who cares to tap in. Unfortunately, we have not witnessed corresponding advances in computational techniques to help us *discover the* accumulated data. Without such developments, we risk missing most of what the data have to offer.

Be it a satellite orbiting our planet, a medical imaging device, a credit-card transaction verification system, or a supermarket's checkout system, the human at the other end of the data gathering and storage machinery is faced with the same problem: *What to do with all this data?* Ignoring whatever we cannot analyze would be wasteful and unwise. Should one choose to ignore valuable information buried within the data, and then one's competition may put them to good use; perhaps to one's detriment. In scientific endeavours, data represents observations carefully collected about some phenomena under study, and the race is on for who can explain the observations best. In business endeavours, data captures information about the markets, competitors, and customers. In manufacturing, data captures performance and optimization opportunities, and keys to improving processes and troubleshooting problems.

The value of raw data is typically predicated on the ability to extract higher-level information: information useful for decision support, for exploration, and for better understanding of the phenomena generating the data. Traditionally, humans have done the task of analysis. One or more analysts get intimately familiar with the data and with the help of statistical techniques provide summaries and generate reports. In effect, analysts determine the *right* queries to ask and sometimes even act as sophisticated query processors. Such an approach rapidly breaks down as the volume and dimensionality of the data increase. Who could be expected to "understand" millions of cases each having hundreds of fields? To further complicate the situation, the data grow and change at rates that would quickly overwhelm manual analysis (even if it were possible). Hence, tools to aid in at least the partial automation of analysis tasks are becoming a necessity. These tools could be Knowledge Discovery in Database (KDD), or Data Mining (DM).

1.2 Data Mining (DM) and Knowledge Discovery in Databases (KDD)

1.2.1 Motivation

Data Mining (DM) is not new to statistician; it is a term synonymous with *data dredging or fishing* and has been used to describe the process of trawling through data in the hope of identifying patterns, but for us Data Mining is the fitting model to extract patterns from the databases. *Knowledge Discovery in Database* (*KDD*)

is the overall process of discovering useful knowledge from databases, but also KDD is the nontrivial process of identifying valid novel, potentially useful, and ultimately understandable patterns in databases. A clear distinction between Data Mining and Knowledge Discovery is drawn under their conventions(see figure 1.1 for the process); the Knowledge Discovery process takes the raw results from Data Mining and carefully and accurately transforms them into useful and understandable information. This information is not typically retrievable by standard techniques but is uncovered through the use of KDD or DM techniques. A detailed description of figure 1.1 can be found in [122].

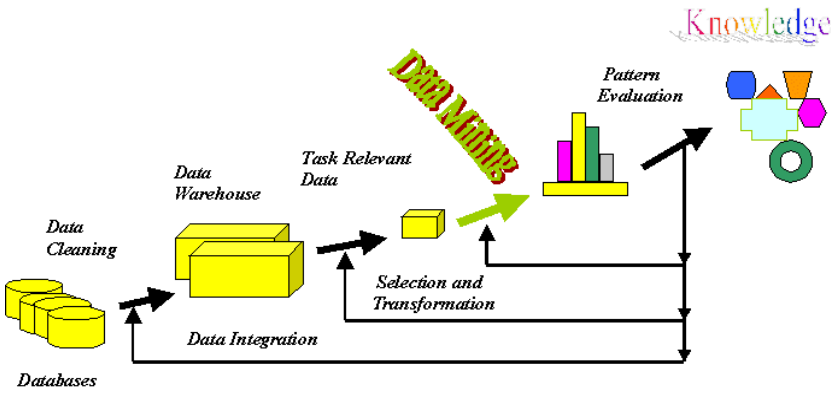

Figure 1.1 KDD and DM process

Steps in KDD process can be defined as follows.

1.Learning the application domain (i.e. relevant prior knowledge and goals of application).
2.Gathering and integrating of data.
3.Cleaning and preprocessing data (i.e. may take 60% of effort!).
4.Reducing and projecting data (i.e. find useful features, dimensionality/variable reduction,…etc).
5.Choosing functions of DM (i.e. summarization, classification, regression, association, clustering,…etc).
6.Choosing the mining algorithm(s).
7.DM: search for patterns of interest.
8.Evaluating results.
9.Interpretation: analysis of results (i.e. visualization, alteration, removing redundant patterns, …etc).
10. Use of discovered knowledge.

KDD steps also can be merged into two classes such as
data cleaning + data integration = data pre-processing;
data selection + data transformation = data consolidation.

In fact, KDD process is based on the extensive use of methods developed in more traditional topics, such that *Artificial Intelligence*, *Machine Learning*,

Mathematical Statistics, *Database Management*, *Visualization*, *Pattern Recognition*, *Decision Support,* etc, see figure 1.2.

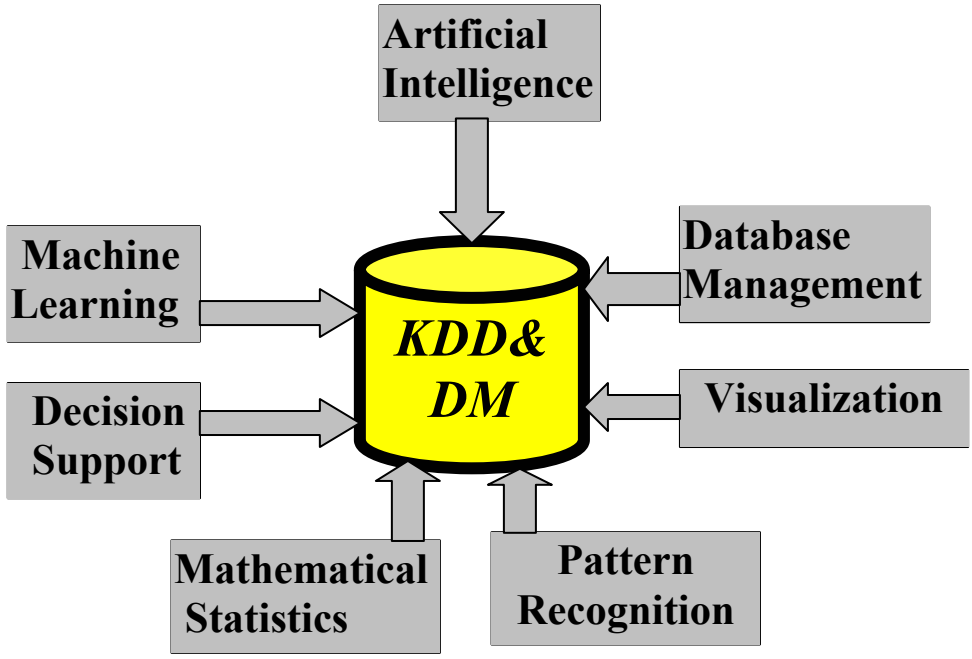

Figure1.2 KDD & DM shared with several topics

A clear definition for DM and KDD could be the extraction of interesting *(non-trivial, implicit, previously unknown and potentially useful*) information or patterns from data in large databases. Alternative name such as mining databases led to knowledge extraction, data/pattern analysis, data archeology, data dredging, information harvesting, business intelligence, etc.

1.2.2 History of DM

The first appearance of DM was in 1990-2000s after the RDBMS and advance data model was established in 1980s. Relational data model and relational DBMS was implemented in 1970s, although the first occurrence of data collection and database creation was in 1960s. In theses day's knowledge discovery in databases (KDD) is concerned with extracting useful information from databases. The term *Data Mining* has historically been used in the database community and in statistics. We take the view that any algorithm that enumerates patterns from, or fits models to data is a *Data Mining algorith*m. We further view Data Mining to be a single step in a larger process that we call the KDD process. KDD process is a process that includes *Data Warehousing, Target Data Selection, Cleaning, Preprocessing, Transformation and Reduction*, *Data Mining, Model Selection (or Combination), Evaluation and Interpretation*, and finally consolidation and use of the extracted "*knowledge*." Hence, Data Mining is but a step in this iterative and interactive process.

KDD's goal, as stated above, is very broad, and can describe a multitude of fields of study. Statistics has been preoccupied with this goal for over a century.

Therefore, we have many other fields including database systems, pattern recognition, artificial intelligence, data visualization, and a host of activities related to data analysis (see figure 1.2). So why has a separate community emerged under the name “KDD”?

The answer: new approaches, techniques, and solutions have to be developed to enable analysis of large databases. We faced with massive data sets, traditional approaches in statistics and pattern recognition collapse. For example, a statistical analysis package (e.g. *K-means clustering* in our favorite *Fortran library*) assumes data can be “loaded” into memory and then manipulated. What happens when the data set will not fit in main memory? What happens if the database is on a remote server and will never permit a particular scan of the data? How do I sample effectively if I am not permitted to query for a stratified sample because the relevant fields are not indexed? What if the data set is in a multitude of tables (relations) and can only be accessed via some hierarchically structured set of fields? What if the relations are sparse (not all fields are defined or even applicable to any fixed subset of the data)? How do I fit a statistical model with a large number of variables?

The open problems are not restricted to scalability issues of storage, access, and scale. For example, a problem that is not addressed by the database field is one that *Usama Fayyad* [1] uses to call it the “*query formulation problem*”: *what to do if one does not know how to specify the desired query to begin with?* For example, it would be desirable for a bank to issue a query at a high level: “give me all transactions whose likelihood of being fraudulent exceeds 0.75.” It is not clear that one can write a SQL query (or even a program) to retrieve the target. Most interesting queries that arise with end-users of the data are of this class. KDD provides an alternative solution to this problem. Assuming that certain cases in the database can be identified as “fraudulent” and others as “known to be legitimate”, then one can construct a *training sample* for a Data Mining algorithm, let the algorithm build a predictive model, and then retrieve records that the model triggers on. This is an example of a much needed and much more natural interface between humans and databases. Issues of inference under uncertainty search for patterns and parameters in large spaces, and so on are fundamental to KDD.

Data mining queries derive their name from the similarities between searching for valuable information in a large database and mining rocks for a vein of valuable ore. Both imply either sifting through a large amount of material or ingeniously probing the material to exactly pinpoint where the values reside. It is, however, a misnomer, since mining for fish in sea is usually called "*fish mining*" and not "*water mining*", thus by analogy, data mining should have been called "*Knowledge Mining*" instead. Nevertheless, data mining became the accepted customary term, and very rapidly a trend that even overshadowed more general terms such as *Knowledge Discovery in Databases* (*KDD*) that describe a more complete process [7]. Other similar terms referring to *data mining are: Data Dredging, Knowledge Extraction, and Pattern Discovery*.

While these issues are studied in many related fields, approaches to solving them in the context of large databases are unique to DM and KDD.

1.2.3 Importance of DM and KDD

If we take into account that what *Kurt Thearling* [2] say's then we can say that *Data Mining* is the process of searching through databases for interesting statistical relationships that would be hard to find any other way.

For example, traditionally someone selling cookware might try various marketing techniques aimed at young women, but this would merely be a guess as to what works. DM of a customer database might uncover a surprising relationship such as: middle aged men who buy luxury soap and discount foods respond to promotions for cookware. This information would be useful for targeting future promotions.

In potential applications, database analysis and decision support are used for market analysis and management to target marketing, customer relation management, market basket analysis, cross selling and market segmentation. Risk analysis and management is used for forecasting, customer retention, improved underwriting, quality control and competitive analysis. Fraud detection and management is used to detect insurance companies. Also other potential applications were defined for text mining (i.e. news group, email, documents) and web analysis or intelligent query answering.

1.2.3.1 Steps in DM and KDD

A typical DM project would go through the following steps:

1. Chose an issue. For example, *"we're losing existing customers"*.
2. Gather large amounts of data from sources such as customer service logs and customer demographic databases. This data might already be collected. If not, you will need to start collecting customer data as part of your normal business operations. It is also possible to buy overlay data from third parties such as estimated income levels based on zip codes.
3. Transform the data into the format needed for the DM application.
4. Use DM software applications to uncover patterns in the database.

DM articles sometimes give the impression that the whole process automatically, almost magically, finds valuable insights. While DM automates a lot of time-consuming work, a smart analyst is still required and statisticians do not need to worry that they'll be reduced to flipping burgers any time soon.

1.2.3.2 Typical applications for solving common problems

The vast majority of DM and KDD applications are for marketing purposes. Another area where DM is used is fraud and risk management. One of the biggest problems in DM is clean data. Vendors and academics have debated the merits of various algorithms but those differences are minor compared to the problems companies have with data. For example, statistically there is a

big difference between a null value (no response) and zero yet it is not uncommon for null values to show up in the data as zeros.

Organizing the data so that a DM application can also be a large project. Simply having the data on some system is no guarantee that it will be easy to put it into a form that the DM application can use [4].

1.2.3.3 How do we know if our company can benefit from DM?

Most companies using DM have large customer bases, 100,000 customers or more, and have many data about these customers. DM is more important when customers are fickle and the company needs to work hard to find ways to attract and retain them [5].

1.2.4 Appearances of DM

DM techniques are the result of a long process of research and product development. This evolution began when business data was first stored on computers, continued with improvements in data access, and more recently, generated technologies that allow users to navigate through their data in real time. DM takes this evolutionary process beyond retrospective data access and navigation to prospective and proactive information delivery. DM is ready for application in the business community because it is supported by three technologies that are now sufficiently matured: Massive data collection, powerful multiprocessor computers, and DM algorithms.

DM appears in marketing analysis, in where are the data sources for analysis? (i.e. credit card transactions, loyalty cards, discount coupons, customer complaint calls, plus (public) lifestyle studies). Target marketing also is used in marketing analysis for instance, to find clusters of "model" customers who share the same characteristics: interest, income level, spending habits, etc. Using determine customer purchasing patterns over time is another for DM in marketing analysis to identify the conversion of single to a joint bank account: marriage, etc. Cross-market analysis is also a common use of DM in marketing analysis to find associations/co-relations between product sales or prediction based on the association information. There is other marketing analysis applications were DM appears these applications are customer profiling, identifying customer requirements, provides summary information.

Corporate analysis and risk management also uses DM in Finance planning and asset evaluation (i.e. cash flow analysis and prediction, contingent claim analysis to evaluate assets, cross-sectional and time series analysis). In Corporate analysis and risk management we use DM in resource planning (i.e. summarize and compare the resources and spending) and in competition to monitor competitors and market directions, group customers into classes and a class-based pricing procedure and set pricing strategy in a highly competitive market.

In fraud detection and management DM was used in several applications such as widely used in health care, retail, credit card services, telecommunications (phone card fraud), etc. The DM approach in fraud detection and management was the

use of the historical data to build models of fraudulent behavior and then DM will help to identify similar instances. For instances, (1*) auto insurance*: detect a group of people who stage accidents to collect on insurance. (2) *money laundering*: detect suspicious money transactions (US Treasury's Financial Crimes Enforcement Network). (3) *medical insurance*: detect professional patients and ring of doctors and ring of references. Other uses of fraud detection are detecting inappropriate medical treatment (i.e. Australian Health Insurance Commission identifies that in many cases blanket screening tests were requested (save Australian $1m/yr). Detecting telephone fraud is another use of fraud detection for instances, (1) telephone call model: destination of the call, duration, and time of day or week. (2) analyze patterns that deviate from an expected norm and (3) British Telecom identified discrete groups of callers with frequent intra-group calls, especially mobile phones, and broke a multimillion dollar fraud. Fraud detection also has been used in retail (i.e. Analysts estimate that 38% of retail shrinks is due to dishonest employees).

Commercial databases are growing at unprecedented rates. A recent *estimated time of arrival group* and known also as *ETA group* survey of *data warehouse* (DW) projects found that 19% of respondents are beyond the 50-gigabyte level, while 59% expect to be there by second quarter of 1996 in some industries, such as retail, these numbers can be much larger. The accompanying need for improved computational engines can now be met in a cost effective manner with parallel multiprocessor computer technology [6]. DM algorithms embody techniques that have existed for at least 10 years, but have only recently been implemented as mature, reliable, understandable tools that consistently outperform older statistical methods. In the evolution from business data to business information, each new step has built upon the previous one. For example, dynamic data access is critical for drill through in data navigation applications, and the ability to store large databases is critical to DM. This will allow new business questions to be answered accurately and quickly.

1.2.5 Tools for KDD and DM

As a former developer of DM and KDD Tools, it will be understood how difficult it is to create applications that are relevant to business users. Much of the DM community comes from an academic background and has focused on the algorithms buried deep in the bowels of the technology. But algorithms are not what business users care about. Over the past few years the technology of DM has moved from the research lab to Fortune 500 companies, requiring a significant change in focus. The core algorithms are now a small part of the overall application, being perhaps 10% of a larger part, which itself is only 10% of the whole. That being said, the focus of this article is to point out some areas in the remaining 99% that need to be improved upon [15]. This is one of a current top ten list:

1. *Database integration*: No flat files. One more time: No flat files. Not supporting database access (reading and writing) via ODBC or native methods is just plain lazy. Companies spend millions of dollars to build DW to hold their data and DM applications must take advantage of this. Besides saving significant manual effort and storage space, relational integration

allows DM applications to access the most up-to-date information available. Today many of the leading DM vendors have heard this message but there is stillroom for improvement [7].

2. *Automated model scoring*: Scoring is the unglamorous workhorse of DM. It does not have the sexiness of a neural network or a genetic algorithm but without it, DM is useless. (There are some DM applications that cannot score the models that they produce to this list is like building a house and forgetting to put in any doors.) At the end of the day, when your DM tools have given you a great predictive model, there's still a lot of work to be done. Scoring models against a database is currently a time consuming, error prone activity that has not been given the consideration that it is due. When someone in marketing needs to have a database scored, they usually have to call someone in IT and cross their fingers that it will be done correctly. If the marketing campaigns that rely on the scores are run on a continuous (daily) basis, this means a lot of phone calls and lot of manual processing. Instead, the process that makes use of the scores should drive the model scoring. Scoring should be integrated with the driving applications via published API's (a standard would be nice but it's probably too soon for this) and run-time-library scoring engines [121]. Automation will reduce processing time, allow for the most up-to-date data to be used, and reduce error [8].
3. *Exporting models to other applications:* This is really an extension to the *automated* model scoring; once a model has been produced, other applications (especially applications will drive the scoring process) need to know that they exist. Technologies such as OLE automation can make this process relatively straightforward. It's just a matter of adding the *"export"* button on the DM user interface and creating a means to extend the export functionality by external applications. Exporting models will then close the loop between DM and the applications that need to use the results (scores). Besides exporting the model itself, it would be useful to include summary statistics and other high-level pieces of information about the model so that the external application could incorporate this information into its own process.
4. *Business templates:* Solving a business problem is much more valuable to a user than is solving a statistical modeling problem. This means that a cross-selling specific application is more valuable than a general modeling tool that can create cross-selling models. It might be simply a matter of changing terminology and a few modifications to the user interface but those changes are important. From the user's perspective, it means that they don't have to stretch very far in order to take their current understanding of their problem and map it to the software they are using.
5. *Effort Knob:* Users do not necessarily understand the relationship between complex algorithm parameters and the performance that they will see. As a result, the user might naively change a tuning parameter in order to improve modeling accuracy, increasing processing time by an order of magnitude. This is not a relationship that the user can (or should) understand. Instead, a better solution is to provide an "effort knob" that allows a user to control global behavior. Set it to a low value and the system should produce a model quickly, doing the best it can give the limited amount of time. On the other hand, if it is set to the maximum value the system might run overnight to produce the best model possible. Because time and effort are concepts that a

business user can understand, an effort knob is relevant in a way that tuning armatures are not.

6. *Incorporate financial information:* DM does not operate in a vacuum. The results of the DM process will drive efforts in areas such as marketing, risk management, and credit scoring. Each of these areas is influenced by financial considerations that need to be incorporated in the DM modeling process. A business user is concerned with maximizing profit, not minimizing RMS error. The information necessary to make these financial decisions (costs, expected revenue, etc.) is often available and should be provided as an input to the DM application.
7. *Computed target columns:* In many cases the desired target variable does not necessarily exist in the database. If the database includes information about customer purchases, a business user might only be interested in customers whose purchases were more than one hundred dollars. Obviously, it would be straightforward to add a new column to the database that contained this information. But this would probably involve database administrator and IT personnel, complicating a process that is probably complicated already. In addition, the database could become messy as more and more possible targets are added during an exploratory data analysis phase. The solution is to allow the user to interactively create a new target variable. Combining this with an application wizard described later, it would be relatively simple to allow the user to create computed targets on the fly.
8. *Time series data:* Much of the data that exists in DW has a time based component. A year's worth of monthly balance information is qualitatively different than twelve distinct non-time-series variables. DM applications need to understand that fact and use it to create better models. Knowing that a set of variables is a time-series allows for calculations to be done that make sense only for time series data: trends, slopes, deltas, etc. These calculations have been in use manually by statisticians for years but most DM applications cannot perform them because time-series data is considered as a set of unrelated variables.
9. *Use vs. view:* DM models are often complex objects. A decision tree with four hundred nodes is impossible to fit on a high-resolution video display, let alone be understood by a human viewer. Unfortunately most DM applications do not differentiate between the model that is used to score a database and the model representation that is presented to users. This needs to be changed. The model that is presented visually to the user does not necessarily have to be the full model that is used to score data. A slider on the interface that visualizes a decision tree could be used to limit the display to the first few (most important) levels of the tree. Interacting with the display would not have an effect on the complexity of the model but it would simplify its representation. As a result, users would be able to interact with the system to provide only the amount of information they can comprehend [9].
10. *Wizards:* Not necessarily a must-have, application wizards can significantly improve the user's experience. Besides simplifying the process, they can help prevent human error by keeping the user on track.

Some of current KDD tools are listed as follows.

1. *Science:* SKYCAT: used to aid astronomers by classifying faint sky objects.
2. *Marketing:* AMEX: used customer group identification and forecasting. Claims 10%-15% increase in card usage.
3. *Investment: Many use Few tell*: LBS Capitol Management: uses and expert system/neural network to manage $600 million portfolio. Results outperform market.
4. *Fraud Detection:* HNC Falcon, Nestor Prism: credit card fraud detection FAIS: US Treasury money laundering detection system.
5. *Manufacturing:* CASSIOPEE: a trouble-shooting system used in Europe to diagnose 737 problems by deriving families of faults by clustering.
6. *Telecommunications:* TASA (Telecommunications Alarm-Sequence Analyzer): locates patterns of frequently occurring alarm episodes and represents the patterns as rules.
7. *Data Cleaning:* MERGE-PURGE: used by Washington State to locate and remove duplicate welfare claims.
8. *Sports:* ADVANCED SCOUT: helps NBA coaches analyze data to organize and interpret game data ==> player selection and team management.
9. *Information Retrieval:* Intelligent Agents have been designed to navigate the internet and return information pertinent to some non-trivial query.

To use DM or KDD approaches we need to call data from a remote side. To access these databases we should use a tool which can access to any databases. This tool is *Open Database Connectivity* (ODBC).

1.3 Open Database Connectivity (ODBC)

1.3.1 Motivation of ODBC

The Open Database Connectivity (ODBC) interface allows applications to access data from remote database management systems (DBMS). The interface permits maximum interoperability (a single application can access diverse back-end databases). Application developer can develop, compile, and ship an application without targeting a specific DBMS product. Users can then add modules called database drivers that link the application to their choice of database management systems. The ODBC interface defines libraries of ODBC function calls that allow an application to connect to a DBMS, execute SQL statements, and retrieve results [11].

1.3.2 ODBC history

ODBC is based on Call-Level Interface and was defined by the *SQL Access Group*. *Microsoft* was one member of the group and was the first company to release a commercial product based on its work (under Microsoft Windows). ODBC 1.0 was released in September 1992. ODBC is not a Microsoft standard (as many people believe). ODBC drivers and development tools are available now for Microsoft Windows, Unix, OS/2, and Macintosh. There exist other types of standards for the same purpose (c.f. JDBC).

1.3.3 Importance of ODBC

Application programs (*word processor, data access & retrieve tool, etc*.) perform processing by passing SQL statements to and receiving results from the ODBC driver manager. Driver manager with a dynamic link library (DLL) that loads specific drivers on behalf of an application. Driver is a DLL element that processes ODBC function calls received from the driver manager, submitting the resultant SQL requests to a specific data source, and returns results to the application. Driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS. Data Source consists of a DBMS, the operating system the DBMS runs on, and the network (if any) used to access the DBMS.

1.3.4 Appearances of ODBC

ODBC is a set of database serve as an *application programming interface* (API) provided by the database system vendors for their client programs to use. The benefit of using the simple set of APIs is that the client programs can be database system independent. This is a noble concept since it prevents the developers from having to write one program for Oracle, one for SQL server, and one for access, and many others for other database systems of the customers' choice. ODBC drivers refer to the software that implements those defined API's. The client programs make call through the drivers. The drivers in turn translate the calls into database system's native database requests. There are different levels of ODBC standard compliance to allow vendors to provide a subset of all defined services and still be considered standard compliant. Client programs can first inquire about whether a particular service is available before committing the applications to support certain database features [11].

1.3.5 ODBC Drivers, Servers and Tools

1.3.5.1 XML and ODBC

ODBC is a standard for accessing data from SQL databases. *XML* is a standard for content data interchange and business-to-business integration we can go to the MERANT site *(www.merant.com)* for information about an ODBC driver for XML data sources [119]. Also the *intelligent systems research group* (ISRG) presented the ODBC2XML shareware product that generates XML documents from ODBC data sources.

1.3.5.2 Heterogeneous Data Replication

ODBC is a useful solution for replicating data across heterogeneous SQL data sources. In *syware* site (www.syware.com) we can find more detailed information about heterogeneous SQL data sources [120].

1.3.5.3 Drivers and Vendors

There are dozens of vendors and hundreds of ODBC drivers. For popular DBMS products, there are often multiple vendors with drivers. There are also drivers for *oracle database managements system* (ODBMS), *oracle relational database managements system* (ORDBMS), and non-relational data such as *lotus notes* and *indexed sequential access method* (ISAM) files. If we are doing serious ODBC development we need serious tools. We may be able to trace tools even if you aren't writing directly to the ODBC API. Syware is a suite of several ODBC tools that are useful for purposes such as logging SQL and ODBC functions, replaying scripts, and validating an application's use of ODBC.

1.3.5.4 ODBC and JDBC servers

ODBC requires the installation of a driver manager, one or more drivers, a cursor library, and INI files. This is often installed on top of an infrastructure that includes protocol stacks, client libraries and network libraries. Administering ODBC or JDBC for a large number of clients can be a problem. To simplify administration, several vendors have built ODBC and JDBC servers. By using a data access server, we can reduce the resource requirements on the client. The client of such a server requires only a single driver and single network transport. Vendors of these servers often can provide drivers for both ODBC and JDBC clients. For instances, *OpenLink, Simba Technologies,* and *Merant* are middleware vendors who have server-side products that support ODBC and JDBC clients [11].

1.3.5.5 Developing a Driver

If we are developing an ODBC driver and we don't have an optimizing SQL engine, then we've probably noticed there isn't one in the ODBC *software development kits* (SDK). Developing a driver is a much simpler task if we start with SDK driver. In syware we can find information about different SDK drivers such as *Dr.DeeBee driver SDK, Simba SDK* and *Open Access SDK* [120].

1.4 Query Language and SQL

1.4.1 Motivation of SQL

The query language is a source language consisting of procedural operators that invoke functions to be executed. A query is a method by which you obtain access to a subset of records from one or more tables that have attribute values satisfying one or more criteria. There are a variety of ways to process queries against databases. One of the processing queries is SQL. Structure Query Language SQL is an industry-standard language for creating, updating and, querying relational database management systems.

1.4.2 History of SQL

SQL was developed by IBM in the 1970s for use in System R. It is the de facto standard as well as being an *International Organization for Standardization* (ISO) and *Americans National Standards Institute* (ANSI) standard. It is often embedded in general purpose programming languages. The first SQL standard, in 1986, provided basic language constructs for defining and manipulating tables of data; a revision in 1989 added language extensions for referential integrity and generalized integrity constraints. Another revision in 1992 provided facilities for schema manipulation and data administration, as well as substantial enhancements for data definition and data manipulation. Development is currently underway to enhance SQL into a computationally complete language for the definition and management of persistent, complex objects. This includes: generalization and specialization hierarchies, multiple inheritance, user defined data types, triggers and assertions, support for knowledge based systems, recursive query expressions, and additional data administration tools [10]. It also includes the specification of *Abstract Data Types* (ADT), object identifiers, methods, inheritance, polymorphism, encapsulation, and all of the other facilities normally associated with object data management.

1.4.3 Importance of SQL

SQL is the de facto standard language that does much more than asking questions. SQL used to manipulate and retrieve data from relational databases. SQL enables a programmer or database administrator to modify a database's structure, change system security settings, add user permissions on databases or tables, query a database for information and update the contents of a database. With SQL we can also create tables, add data, delete data, splice data together, trigger actions based on changes to the database, and store your queries within your program or database.

1.4.4 Appearances of SQL

SQL is a language which can be used to make queries and other requests of many database managers using a variety of connection vehicles. SQL is not a procedural language, and it is missing many of the things which are familiar to programmers who know other languages. SQL allow us to store, retrieve and query tabular data without having to rely on any external database engine or package. We use SQL to access and manipulate the contents of any databases.

1.4.5 Tools of SQL

In this Section we will introduce some of the popular implementations of SQL, each of which has its own strengths and weaknesses. Where some implementations of SQL have been developed for PC use and easy user interactivity, others have been developed to accommodate very large databases.

1.4.5.1 Microsoft access

Microsoft access is a PC-based DBMS, to illustrate some of the examples in this text. Access is very easy to use and we can use *graphical user interface* (GUI) tools or manually enter our SQL statements.

1.4.5.2 Personal Oracle7

We use Personal Oracle7, which represents the larger corporate database world, to demonstrate command line SQL and database management techniques. (These techniques are important because the days of the standalone machine are drawing to an end, as are the days when knowing one database or one operating system was enough). We can write SQL statements to entered the Oracle's SQL *Plus tool. This tool then returns data to the screen for the user to performs the appropriate action on the database.

Most examples are directed toward the beginning programmer or first time user of SQL. We begin with the simplest of SQL statements and advance to the topics of transaction management and stored procedure programming. The ORDBMS is distributed with a full complement of development tools. It includes *Delphi, C++* and *Visual Basic* language library (Oracle Objects for *object linking and embedding* (OLE)) that can link an application to a *personal oracle database*. It also comes with graphical tools for database, user, and object administration, as well as the *SQL *loader utility*, which is used to import and export data to and from oracle.

1.4.5.3 Microsoft Query

Microsoft Query is a useful query tool that comes packaged with Microsoft's Windows development tools, Visual C++, and Visual Basic. It uses the ODBC standard to communicate with *i-extended database* (it will be described later in Chapter 8). Microsoft Query passes SQL statements to a driver, which processes the statements before passing them to a database system.

1.4.5.4 Open Database Connectivity (ODBC)

ODBC is a functional library designed to provide a common Application Programming Interface (API) to i-extended database systems. It communicates with the database through a library driver, just as Windows communicates with a printer via a printer driver. Depending on the database being used, a networking driver may be required to connect to a remote database. The unique feature of ODBC (as compared to the Oracle or Sybase libraries) is that none of its functions are database vendor specific. For instance, we can use the same code to perform queries against a Microsoft Access table or an Informix database with little or no modification. Once again, it should be noted that most vendors add some proprietary extensions to the SQL standard, such as *Microsoft's and Sybase's Transact SQL and Oracle's PL/SQL* [11].

We should always consult the documentation before beginning to work with a new data source. ODBC has developed into a standard adopted into many products, including *Visual Basic, Visual C++, FoxPro, Borland Delphi,* and *PowerBuilder*. As always, application developers need to weigh the benefit of using the emerging ODBC standard, which enables you to design code without regard for a specific database, versus the speed gained by using a database specific function library. In other words, using ODBC will be more portable but slower than using the Oracle7 or Sybase *libraries.*

Until recently, if you weren't working on a large database system, you probably had only a passing knowledge of SQL. With the advent of client/server development tools (such as Visual Basic, Visual C++, ODBC, Borland's Delphi, and Powersoft's PowerBuilder) and the movement of several large databases (Oracle and Sybase) to the PC platform, most business applications being developed today require a working knowledge of SQL.

1.4.6 How KDD tools can access databases today?

DM is a term for the computer implementation of a timeless human activity: It is the process of using automated methods to uncover trends, patterns, and relationships from accumulated electronic traces of data. DM or Knowledge Discovery, as it is sometimes called lets us exploit an enterprise data store by examining the data for patterns that suggest better ways to produce profit, savings, higher quality products, and greater customer satisfaction.

With the release of *SQL server 7.0* in fall 1998, Microsoft stepped squarely into the maturing area of *decision support* and *business intelligence*. SQL server 7.0's with *on-line analytical processing* (OLAP) services provides a widely accessible, functional, and flexible approach to OLAP and multidimensional cube data query and data manipulation. SQL server 2000 extends OLAP services capabilities, incorporating DM algorithms in its renamed analysis services.

1.4.6.1 Origins of DM

To mine data, we need access to data, so it's no coincidence that DM developed at the same time as DW did. As computer power and database capability grew through the late 1900s, people began to see that data wasn't simply a passive receptacle, useful only in performing billing or order entry functions. People could also use data in a more proactive role to provide predictive value in guiding their businesses forward. This notion led to the development of a new breed of computer systems that went beyond running the business (as early computer applications did) to informing and analyzing the business. These new systems were sometimes called decision support systems or *executive information systems* (EIS). These systems were designed to harness growing computing power and improved GUIs to provide ad hoc analytical reports that could slice and dice data in novel ways and went well beyond earlier notions of static reporting. Slicing and dicing data drilling down into detailed reports or zooming up to a 10,000 foot "*big picture*" view required special ways of organizing data for decision making. This need gave rise to the data warehouse. The term DW was virtually unknown in 1990. Ten years later, DW

has become a multibillion dollar business of capturing and organizing data to provide a proactive analytical (versus operational) environment that uses data in defining and guiding business activity. As DW matured, decision support and EIS gave way to the more general concepts of business intelligence in DM.

1.4.6.2 Concepts of business intelligence in DM

Business intelligence involves organizing data along various potential dimensions of analysis so that you can cross-reference and display any view of data say sales results from within any number of other potential dimensions say region or product line. The ability to move up and down dimensions lets you drill down into detail or zoom up for a more general view. The ability to show variations in data along various dimensions often, many dimensions simultaneously provides multidimensional reporting capability in real-time. This general approach to manipulating data became known as OLAP that is, processing data for analytical purposes instead of operational purposes. The term online refers to having the analytical data continuously available. OLAP takes advantage of a DW by making data continuously available in a form that supports analytical decision-support tasks. The distinguishing characteristic of OLAP is the preprocessing, indexing, and storage of data in various dimensional representations to quickly deliver the various dimensional views business intelligence requires. However, business intelligence OLAP tools might not find all the patterns and dependencies that exist in data. OLAP cubes are appropriate for a limited amount of data exploration, involving major variations according to critical and known business dimensions. But when the dimensions change as the business changes or when you're exploring novel situations, DM can be an extremely flexible and powerful complement to OLAP.

DM solutions are perfectly suited for shifting through hundreds of competing and potentially useful dimensions of analysis and associated combinations. All DM algorithms have built in mechanisms that can examine huge numbers of potential patterns in data and reduce the results to a simple summary report. The business intelligence OLAP and DM approaches to reporting on data belong together and are synergistic when deployed together. Microsoft recognized this synergy after it released SQL server 7.0 and began a development program to migrate DM capabilities into the SQL server 2000 release.

The most common DM techniques are decision trees, neural networks, cluster analysis, and regression. In preparing to release SQL server 2000 and commerce server, Microsoft developed a substantial DM infrastructure and core DM algorithms to carry out decision tree and cluster analysis DM tasks. As part of the DM infrastructure, Microsoft created the OLE DB for DM specification, an extension of OLE DB for OLAP that defines the DM infrastructure and interfaces that expose DM models and algorithms to DM consumers. OLE DB for DM serves as a standard that external product vendors can use for delivering their DM functionality in the Microsoft environment. Other common DM technique is known as *Data Mining Suite* [3].

1.4.6.3 The DM development approach

The DM and exploration group at Microsoft, which developed the DM algorithms in SQL server 2000, describes the goal of DM finding "*structure within data*". As defined by the group, structures are revealed through patterns, which are relationships or correlations (co-relations) in data. So the DM and exploration group has captured the essence of DM. Correlations produce patterns or associations that show the structure of data. Structure, when placed in a business context, can drive a business model or even drive the business and improve its effectiveness in the marketplace. The DM and exploration group model of DM is to deliver indicators of data structure through extensions of the data query process. Traditionally, you construct a query to retrieve particular information fields from a database and to summarize the fields in a particular fashion. A DM query is different from a traditional query in the same way that a DM model is different from a traditional database table. In a DM query, we specify the question that we want to examine (e.g., gross sales or likeliness to respond to a targeted marketing offer), and the DM query processor returns to the query station the query results in the form of a structural model that responds to the question.

1.4.6.4 The central object of Microsoft's DM

Implementation in SQL server 2000 is the DM model. The DM and Exploration group built several query wizards to facilitate the process of creating and interacting with the DM model so that end users need no query syntax. The OLE DB for DM specification provides interfaces that can be accessed directly from a client application, however, so both end users and third-party applications can access DM directly through query processing. The query creates a DM model to predict or classify age based on other attributes in the data set, such as gender, product name, or product type and quantity. Here, the client executes a CREATE statement that is similar to a CREATE TABLE statement. A full description of the language for creating and manipulating a DM model is contained in the OLE DB for DM specification, see (http://www.microsoft.com/) for more description.

Although the wizard driven interface is the primary mechanism for accessing SQL server 2000's DM query engine, clients and third party applications can access DM models by using an OLE DB command object. After a DM model structure is built (either by wizard or directly), it is stored as part of an object hierarchy in the analysis services directory. The patterns or structure within the data are stored in summary form with dimensions, patterns, and relationships so that the predictive or classification power of the data will persist regardless of what happens to the original row level data that the model is based on

1.5 Data Visualization in DM

1.5.1 Motivation

The point of data visualization is to let the user understand what is going on. Since DM usually involves extracting "*hidden*" information from a database, this

understanding process can get somewhat complicated. In most standard database operations, nearly everything the user sees is something that they knew existed in the database already. A report showing the breakdown of sales by product and region is straightforward for the user to understand because they intuitively know that this kind of information already exists in the database. If the company sells different products in different regions of the county, there is no problem translating a display of this information into a relevant understanding of the business process.

But if we want to visualize DM, then we have to visualize information from a database that the user did not already know about. Useful relationships between variables that are non-intuitive are the jewels that data mining hopes to locate. Since the user does not know beforehand what the data mining process has discovered, it is a much bigger leap to take the output of the system and translate it into an actionable solution to a business problem. Since there are usually many ways to graphically represent a model, the visualizations that are used should be chosen to maximize the value to the viewer. This requires that we understand the viewer's needs and design the visualization with that end-user in mind.

1.5.2 History of data visualization in DM

There is thousands of visualization systems used on computers today. Every new piece of software produced today usually has a nice *Graphical User Interface* (GUI) where the "*visual*" aspect of the user interaction and information displayed is very important. The trend is still toward "*higher resolution*" with more colors. *High Definition Television (HDT)* will allow better information visualizations. However, most visualizations systems are for a specific domain, such as weather, scientific displays, information retrieval, software analysis, network analysis, etc.. General purpose multidimensional information visualizations are needed for today's DM problems. Programs, systems, or algorithms which provide these general tools are not so abundant. It does seem that the newer DM packages are realizing the importance of these new visualization techniques and are starting to incorporate them in their products.

1.5.3 Importance of data visualization in DM

The purpose of data visualization is to give the user an understanding of what is going on. Since data mining usually involves extracting *"hidden"* information from a database, this understanding process can get somewhat complicated. Because the user does not know beforehand what the DM process has discovered, it is a much bigger leap to take the output of the system and translate it into an actionable solution to a business problem. There are several methods to visualize DM models [12]. In the other hand side data visualization is defined as the presentation of processed information in a coherent and easily accessible way. Information can be presented in different forms using traditional devices such as pie charts, scatter graphs, line charts etc.

There are many methods for data visualization (examples include bar graphs, pie charts, scatter graphs, and linear plots), usually held within a two-dimensional

data rectangle. In addition to 'traditional' methods, computer visualizations allow more complex and dynamic approaches to presenting information. Different methods are suitable for different purposes. For example a scatter graph allows clustering of large sets of data. A line graph on the other hand can allow comparison of two or more sets of data as they change in value.

The advantages of using data visualization are that, the user can absorb large sets of data, so data can be easily accessed and patterns perceived. Pointing devices can also more easily access it. Visual pattern perception is a '*natural*' function of the human brain (see Pattern perception and Gestalt psychology below). Navigation through complex and disparate sets of data is easier. Communication with other people is made more straightforward.

1.5.4 Appearances of data visualization in DM

Visualization and DM are committed to deliver data visualization solutions to business customers who use multiple, large databases. The mission is to:

- Empower users to make better decisions, faster, through visual comparisons;
- Enable transformation of complex data into useful graphic information;
- Facilitate real-time decision making ;
- Facilitate the continuous flow of information from corporate data stores to information consumers;
- Enable easy integration with applications and data sources.

These approaches are developed additionally, to a strong commitment for the customers with a primary reason companies such as Oracle, Sybase, Sun Microsystems, IBM and hundreds of others select our charting tools. We can discover soon the value found in using the supported data visualization products available [14].

1.5.5 Tools of data visualization in DM

There are several tools which can serve high performance data visualization visions to the end user to help him create, program and display an online charts and applications with the most robust charting components available. I will point out some of theses tools: *Visual Mining, Data Mining Gallery, Mineit*, and *IBM Visualization Data Explorer (DX)*.

1.6 XML as a visualization approach

XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Using XML to retrieve databases can facilitate the discovery process by enabling the researcher to integrate and annotate complex query results within a highly visual and interactive environment. The result can be represented in different databases mode together in visual interfaces. The value

of converting both the queries and the results into XML can be also shown for remarkable purpose. Meanwhile, the XML description of the GUI can be handling by using various scripting languages. With XML we could configure at run time to have 2D or 3D graphics depending on the needs of the end-user at that moment [14].

Chapter 2
Research main aspect

2.1 Introduction

Knowledge discovery in databases (KDD) is the non-trivial process of extraction of implicit, previously unknown, and potentially useful information from data. In the KDD process, humans play a key role, because ``knowledge discovery involves a knowledge-intensive task consisting of complex interactions between a human and a large database, possibly supported by a heterogeneous variety of tools" (Brachman 1996, [16]). In general, three basic steps compose any KDD process: preliminary data analysis, model selection and refinement, and output generation. Ideally, the user and data analyst should be closely involved in each of these steps. Typically, the user provides a set of requirements to the data analyst, who extracts data relevant to the goal by querying a database, performs data analysis using analytical, algorithmic and/or visual techniques, and finally uses presentation tools to report the insight to the user. In real data mining projects, the role of visualization is key to performing an adequate analysis of the data, developing understandable models, and generating useful output. The insight that can be obtained from appropriate visualizations of the data, for example, is hardly reachable by looking at tables or simple summary statistics. Although data visualization is usually not considered a formal kind of analysis, for some tasks it may be the best tool to solve a problem or confirm a hypothesis.

In addition, in real KDD applications, the analyst needs to be supported by iterative use of algorithm-based tools (e.g., statistical packages, machine learning, neural networks, case-based reasoning, and classification tools) and visualization-based analysis tools (e.g., geometric, icon-based, pixel-oriented, hierarchical, and graph-based techniques). However, in most KDD systems the integration of these approaches is lacking, causing frustration in data analysts, who are unable to use the output from one tool to refine the input to another tool.

Besides the three basic steps of the KDD process, other complex tasks are performed in real KDD problems. Mainly, they include task discovery, data discovery, data cleansing, and the use of background knowledge. Task discovery involves clarifying and refining the user requirements for the task. Commonly, task discovery is a very time consuming and difficult process, but it is essential to guarantee that we are not wasting time answering the wrong questions. Data discovery, on the other hand, involves a preliminary analysis of the raw data, so that the analyst can understand the structure, coverage, and quality of the data. Data discovery and task discovery are intimately related, because the goals that arise from the task discovery phase can not be reached without a good understanding of the data.

Data cleansing involves all work needed to prepare the data for the analysis phase. It involves handling missing and empty values, handling non-numerical variables, normalization and redistribution of variables, etc.

This process requires user assistance and knowledge, because what looks like an anomaly for the data analyst for an exhaustive description and comparison of visual techniques for exploring databases, see (Keim 1997, [17]) may be an interesting domain

phenomenon for the user; in other words, outliers may be key data points where the knowledge discovery process should be focused, according to the user goals and knowledge.

Also, it is really important to incorporate domain and background knowledge in the KDD process. For example, simple forms of background knowledge in databases include the data dictionary, integrity constraints, and various forms of meta-data from database management systems. However, more complex or richer forms of background knowledge are difficult to capture, mainly because they are resident only in the mind of the expert.
Some tools, however, give users the ability to provide richer inputs to the data analysis methods. For example, IMACS (Brachman 1993, [24]) can represent meta-data or abstract concept descriptions, and ReMind (REMIND 1992, [25]) can represent abstract data elements using formula fields.

The most accepted model of the KDD process was formulated by Fayyad, Piatetsky-Shapiro, and Smyth (1996, [10]). An outline of this process is shown in Figure 1.1. In this model of the KDD process, data mining is considered a step in the KDD process that applies an algorithm to data and generates an enumeration of patterns over the data. The KDD process described in Figure 1.1 is interactive and iterative, and it involves several steps, summarized as (Fayyad et al. 1996, [5]):

1- Understanding the application domain. It includes learning relevant prior knowledge and the goals of the application.

2- Data selection. It includes selecting a dataset from an operational data repository, and creating a target dataset on which the KDD process will be performed.

3- Data Preprocessing. It includes basic operations for handling noisy data (e.g., de-duplication of records, correction of domain inconsistencies, and disambiguation of attribute values), deciding on and applying of strategies for dealing with missing attribute values, as well as accounting for time sequence information and known changes.

4- Data transformation. It includes applying strategies to identify relevant and irrelevant features, reduce the number of attributes under consideration (dimensionality reduction), and code some attributes (e.g., flattening an attribute 2).

5- Defining the purpose of the data mining task. It includes deciding which will be the main goal of the data mining task (e.g., classification, regression, clustering, summarization, deviation detection, dependency modeling, link analysis, sequence analysis).

6- Choosing the DM algorithm(s). It involves deciding which set of data mining techniques may be best suited for the problem under consideration, and what parameter settings seem to be more appropriate for each data mining algorithm selected.

7- DM includes applying the selected data mining method(s) to the transformed dataset, in order to induce a model and then generate an enumeration of patterns.

8- Pattern interpretation. It includes the evaluation/interpretation of the enumerated patterns, by assessing their quality in terms of a measure of goodness. Appropriate visualizations of discovered patterns can help users to understand the useful ones. This evaluation/interpretation of patterns may also result into iterations of the KDD process from any of the previous steps.

9- Using discovered knowledge. It includes the incorporation and effective use of the knowledge mined from data. The actions based on the discovered knowledge can range from simple documenting and reporting activities to more elaborate uses, such as decision support and decision making based on the mined knowledge.

The DM step has been the main focus of most previous work on KDD. Other steps in the KDD process have received less attention, although some of them impact significantly the quality of the results. For example, algorithm selection, parameter selection, and model understanding, may significantly improve the quality of the patterns generated; however, they are ignored in most of the current data mining projects, because of the lack of guidelines and tools to support them. Other steps involve a significant amount of work and effort, like data cleaning and pre-processing, and they are starting to be seriously studied only recently (Pyle 1999, [18]).

2.2 Current role of visualization in the KDD process

Visualization is one of the most promising areas in the computer science discipline, because it exploits the human visual system capabilities, which have shown to be more developed than other human perception systems (Pickett & Grinstein 1988, [19]). However, computer visualization has been only partially utilized in the KDD process described in the previous section, and it has been rather limited to data visualization and output generation (Grinstein 1996, [20]).

Although there is a wide variety of techniques currently used to visualize low dimensional (1, 2 or 3 dimensions) and multidimensional (more than 3 dimensions) data sets, and to present results of the knowledge discovery process, they have not been integrated with analytic and algorithmic approaches used to extract knowledge and identify patterns from large data sets.

However, visualization should not be only limited to datasets and output generation, but it might be used experimentally to support some other stages of the DM process. For example, understanding the application domain and defining the problem. Although it may seem obvious that before attempting any DM tasks we do need to understand well the application domain and define the right problem to be solved, in real KDD applications this step is key and essential, because without a clear understanding of the problem, the results may be worthless. The history of projects that involve DM tasks includes a good number of cases where little time was spent in understanding the application domain and defining the problem, and as consequence, the application of DM algorithms to the database generated useless results for the goals that the end-user pursued. In this sense, DM is different from other analytical processes: it may become very easy to apply DM algorithms to the data set, but without spending a significant amount of time in understanding the application domain and defining the problem to be

solved, the results gotten may be deceptive and worthless for the end-user expectations and goals.

Sometimes the problem, as formulated by the domain expert, seems clear to him or her, but it may be unsolvable from a DM perspective, or it may be solved in many different forms, some more appropriate than others. Thus, task discovery and data discovery should be applied in order to understand the application domain and define the problem to be solved. However, currently there is a lack of tools that support these early stages in the DM process, which makes these phases complex and expensive. In this sense, developing appropriate visualizations can help both domain expert and DM analyst to perform these tasks more efficiently and therefore to reduce the costs usually incurred in DM projects.

In relation to other tasks in the KDD process, besides data visualization and output generation, the use of visualization has been rather primitive and uninformative. For example, regarding the model used to predict or describe patterns, the approach used in some current DM tools, both commercial and public domain, is mainly focused to present the topology of the model visually, and/or to show some parameters of the model. Instead, we argue that visualization can be used effectively and efficiently for a better understanding of the model, supporting its evaluation and tuning more consistently, reducing the space of models (which in some cases becomes a very huge space) to be searched, and comparing several of them in order to choose the best one.

Other examples are the activities performed before reaching the DM step: data selection, data pre-processing, and data transformation. Usually these activities are expensive in terms of time and budget, although they are essential to guarantee that the right problem is being solved and to reduce the probability that the results produced by the DM algorithm may contain invalid or spurious patterns.

Other uses of visualization for knowledge discovery tasks, besides data visualization and output generation, have not been formally investigated. Typically, visualization for knowledge discovery tasks has been used to get an overview of the data set before applying a DM algorithm to it. However, visualization has not been used or has been used poorly for helping both user and analyst to define and ``discover'' the real goal of the knowledge discovery process, to support a good and full understanding of the syntax and semantics of the data set, and the possible errors or outliers that it may contain, or any abnormality that may indicate interesting patterns in the data set being explored.

Also, visualization may give valuable insight for understanding in what aspects of the hidden information the data the mining activity should be focused. For example, appropriate visualizations of the raw and transformed data, parameter space, induced model, and generated patterns, may provide valuable knowledge. This knowledge can be used as additional input for selecting the most appropriate DM technique, tuning the parameters associated with the algorithm, understanding the induced model and its complexity, and interpreting and/or evaluating the patterns generated.

2.3 Research questions and problems

Before formulating the goals that we pursue in this research, we list several research questions and problems, which form the core motivation to the development of this dissertation.

1-What exactly do we mean by the integration of visualization within the DM process? Traditionally, integration of visualization and DM has been viewed simply as the incorporation of data visualization techniques in software tools already equipped with inductive learning algorithms and this strictly at the output level (back-end). However, in this thesis, the integration of visualization with DM has a broader meaning, and it should be understood as the effective and efficient use of visualization to support some of the phases in the knowledge discovery process; for example, early stages such as definition of the problem, concepts, and goals, and cleansing and pre-processing of the data set, and later phases such as selection and optimization of the induced model, and interpretation and evaluation of the patterns generated.

2-How can visualization is used effectively and efficiently in the knowledge discovery process, beyond visualizing data sets? The KDD process is basically an exploratory process, and therefore many hypotheses and models need to be generated, optimized, and evaluated before the hidden knowledge may become visible and understandable. In this sense, visualization should be used to support the expression of end-user hypotheses, the analysis and ``debugging'' of the data set, the understanding, tuning, and selection of one or more candidate models, and the interpretation and evaluation of patterns produced by DM algorithms.

3-If some visualization techniques seem to be more appropriate to help certain DM tasks, then which are the features of these techniques that make them perform better than others? An ongoing research activity at University of Massachusetts Lowell (Hoffman 1998, [21]) pursues the formulation of a formal model of array visualizations that permits assessing the usefulness of several multidimensional visualization techniques to perform certain DM tasks, among other goals. Instead of looking for new visualizations created by combining others, an alternative research direction is to focus on discovering the features of some visualization techniques that make them more appropriate to use in certain DM tasks, and use these criteria to select visualization techniques that are complementary to other visualization techniques. Moreover, the integration of visualization techniques with analytical and algorithmic approaches should take advantage of the fact that these visualization techniques perform better than others for the type of task(s) being sought. For example, parallel coordinates (Inselberg 1990, [22]), a multidimensional visualization technique well suited for datasets with large number of dimensions (attributes) but with small number of observations (records), can be integrated with an algorithmic technique or other visual technique that works well for data sets with large number of records.

4-Can the models induced from a data set be better understood and tuned if they are appropriately visualized? Some predictive and descriptive models are poor with respect to supporting understanding how the model is making decisions to classify data or describe patterns. This commonly produces frustration in the analyst, who is

unable to tune and improve the model in a systematic manner, but the analyst explores the model and parameter spaces without clues. Therefore, in order to enhance the tuning of the DM algorithm and understandability of the induced model, these tasks should be supported with adequate techniques and tools. In this sense, if those kinds of models are appropriately visualized then, we hypothesize, the user or analyst should be able to understand and tune them more effectively.

5-How can the parameters of an inductive learning algorithm be tuned to maximize its predictive or descriptive capability? If this can be viewed as an optimization problem, then how can visualization are used to help the user/analyst in the exploration and selection of the best parameter setting for a particular problem or data set, through the space of all possible parameter settings? (John 1997, [23]) describes a method, based on a wrapper approach, for estimating a measure of goodness of a set of parameters for a DM algorithm, and heuristically searching in the entire parameter space. Then this method was applied to tune the parameters of the C4.5 algorithm.

6-Even though the tuned version of C4.5 outperforms the default version of C4.5 in most of the data sets, it is not clear why some parameter values improve the performance of the algorithm for certain data sets, and how they are related to the characteristics of the data set. Thus, visualization may provide a useful input and feedback to algorithm engineering tasks, such as parameter tuning, when integrated with these algorithmic methods.

7-Some DM techniques seem to be best suited for certain DM tasks. In this context, are there any application domain independent criteria that may be used to select the best method or subset of methods that would solve a specific problem better than other techniques? Can these domain independent criteria, if any, be combined with domain dependent features of the specific problem, in order to improve the selection of the most appropriate DM techniques to solve a specific problem? Although the algorithm space is rather sparse, the selection of the algorithm to be used to perform a DM task impacts the results significantly. Even though some authors have formulated guides or developed experimental work in order to assist the user in the selection of an appropriate method, there is a lack of guides or heuristics that combine application domain independent information with domain dependent features in order to guide the user in the selection of the set of best suited techniques for solving a particular problem.

2.4 Research goals

The KDD process is complex, and many obstacles, research questions and problems still need to be investigated and clarified. In the previous Section, we pointed out some of the research questions and problems that motivate this dissertation work. In this Section, we describe our research goals.

Once the database has been pre-processed (cleaned and appropriately transformed), two basic problems arise: what DM algorithm(s) should be used, and how to apply the selected algorithm(s). Many of the problems reported in DM projects are related to wrong decisions about what DM algorithm should be used and how to apply it, in order to get useful and understandable patterns. Our basic premise is that the visualization

paradigm has been applied in a very limited form in the DM process, and mainly focused toward data visualization. Therefore, we argue that visualization can be applied and integrated in other DM tasks and phases in the KDD process, for which currently only analytical and algorithmic approaches have been explored, or visualization has been applied poorly. Thus, we pursue three basic goals in this dissertation work:

- To investigate the relationship between a problem and the subset of DM algorithms that are best suited for solving it. This research goal pursues to clarify the matching between the problem and DM algorithms. Given a well defined problem P, typically there are several algorithms from which to choose. Let $A = \{a_1,a_2,...,a_N\}$, be the set of DM algorithms feasible to use for solving P. The choice of the algorithm a_i to be used would have a significant impact on the results, in other words, in the quality of the mined patterns. Currently, how to match a problem P with a DM algorithm that performs well in solving P, is still an open research problem. In some cases, a mismatch between the problem and algorithm may be caused by either having an inadequate set of DM tools or by trying to apply DM to problems for which adequate algorithms have not been developed. Also, a mismatch between problem and algorithm may be an indication that the problem has not been well defined, or that the algorithm's application domain is diffuse, or not well understood by the analyst. In this dissertation work, we will investigate the problem of matching a DM problem with the set of DM algorithms that are suitable for solving it.

  In this context, we propose to formulate heuristics that can be used to clarify the matching between the problem and the algorithm (algorithm selection problem), by considering simultaneously domain independent and domain dependent features of both the problem and the DM algorithm. These heuristics would help and guide the user in the selection of the best algorithm (or the best subset of algorithms) to solve a specific problem that involves DM tasks.

- To investigate whether the use of visualization and its integration with analytical and algorithmic approaches can improve algorithm engineering tasks. The problem of how to apply a specific DM algorithm involves selecting a set of parameters, with which the algorithm would be executed. The important thing to highlight here is that for many algorithms, the patterns discovered by them depend highly on the values set for their parameters. The usual approach is to run the algorithms with parameters set at their default values, hoping to discover some ``*interesting*'' patterns, or to run several experiments with different parameter settings, compare their results, and choose the one which performs best. This approach is equivalent to exploring the huge parameter space without clues and intuition about what parameter settings would give the best results, and it ignores the fact the parameters can be tuned. In fact, this parameter setting problem can be viewed as an optimization problem: given a training set, a DM algorithm and a vector of parameters for it, the goal is to find an instance of the vector of parameters, such that the patterns discovered by the DM algorithm optimizes some objective function that assigns some real-valued measure of "interestingness" of the patterns. Unfortunately, for this optimization problem, no analytical method is known. However, some algorithmic methods have been created that heuristically search within the parameter space for an appropriate parameter setting.

The parameter settings used are correlated not only with the results of the DM activity, but also with the complexity of the induced model, and the data sample and features selected to induce, test, and validate the model. Therefore, making visible and understandable these correlations may help the analyst to perform the parameter selection task more consistently and systematically.

In this sense, we will experiment with the integration of visualizations with algorithmic approaches to tune the parameters of DM algorithms, in order to support the parameter selection process, currently only explored by algorithmic approaches, in a more systematic form than using default values or setting parameter values without clues.

- To visualize inductive learning models generated by the application of DM algorithms to a training data set. As mentioned in the previous Sections, most of the applications of visualization have been focused on visualizing the database (or dataset) to which some DM algorithms would be applied, and to visualize patterns (e.g., clusters) generated by those algorithms. However, little work has been done in visualizing the models generated by inductive learning algorithms, and statistic entities (e.g. confusion matrix, cost matrix) in a richer way than just presenting the topology of the model or listing the statistical results of the application of the DM algorithm to a testing data set. A key issue in improving the predictive power (accuracy) or descriptive capability of the model is to understand how the inductive model is really working in making decisions, how some parameter settings affect its predictive or descriptive capability, what parameter changes have more impact on the model, and how the attribute space is split up by a model.

In this sense, we will introduce visualizations in order to provide richer information about induced models and statistics entities, and to support the interactive and dynamic exploration of induced models for DM. Therefore, we will address the research issue of how visualization can support the evaluation of individual models, the comparative evaluation of several candidate models, the exploration of the model, and the sensitivity analysis.

Part II

Knowledge Discovery in Remote Access Databases (KDD) Models

Chapter 3

Remote access KDD models

3.1 Introduction

With the explosive growth of information sources available on the remote sides, it has become increasingly necessary for users to utilize automated tool in finding the desired information resources, and to track and analyze their usage patterns. Theses factors give rise to the necessity of creating server side and client side a reliable system that can effectively discover knowledge in databases via a remote access. A remote access knowledge discovery in databases (KDD) model can be broadly defined as the discovery and analysis of useful information from a remote database. To discover new information from the server side databases we will use DM or commonly known as KDD approaches.

The connection between ODBC and KDD is to use KDD process to the retrieved database table from the ODBC connection. KDD is a process for extracting useful patterns and hidden information from raw data stores and making discovered patterns more understandable. KDD process will investigate useful patterns in the retrieved database table. This table is obtained from remote access databases by ODBC drivers [11].

In part II, we will take these aspects (ODBC, KDD) into account. The interaction between the KDD process and the ODBC driver is simply shown in figure 3.1. In this direction, we will describe all the related components on the *client* and *server* sides. In this connection, we will capture a new connection between the raw database and the KDD process by getting data from ODBC driver. We will investigate all the related components of this bidirectional connection between the two approaches ODBC, KDD. We will also propose a conceptual classical model for this bidirectional connection. This model will be called ODBC_KDD (1). On the other hand we suggested a new conceptual model for ODBC_KDD (1) with some new components and more capability known as ODBC_KDD (2) [26]. We will investigate the architecture of both models.

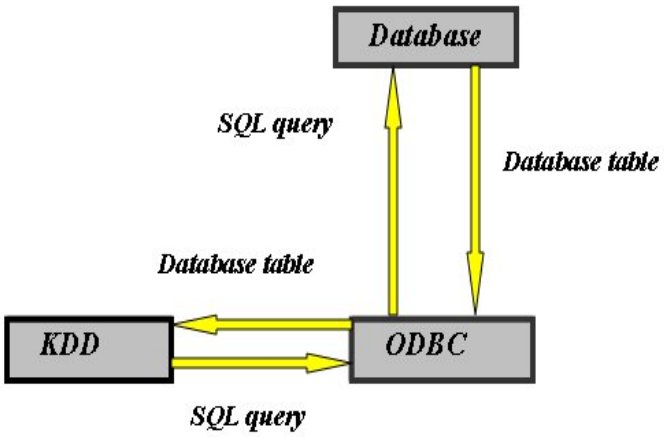

Figure 3.1 shows the connection between DM and ODBC

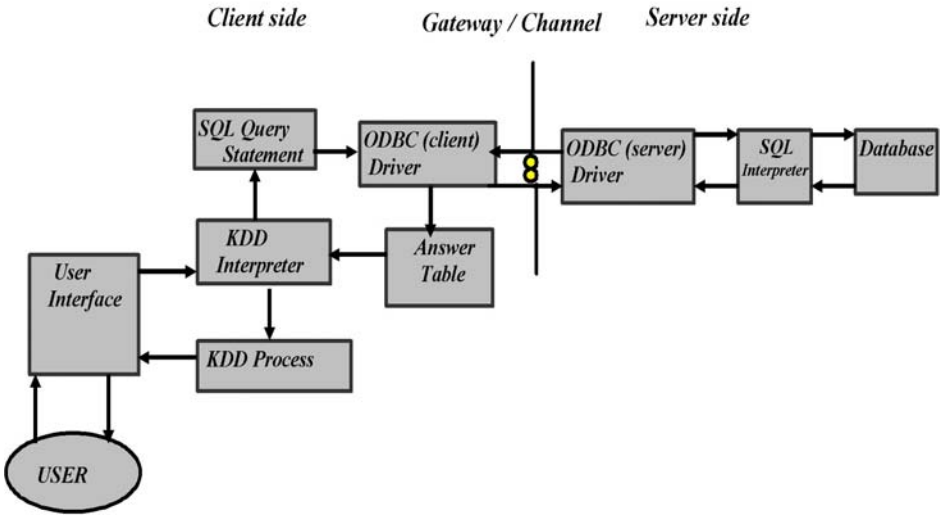

Figure 3.2 ODBC_KDD (1) model architecture

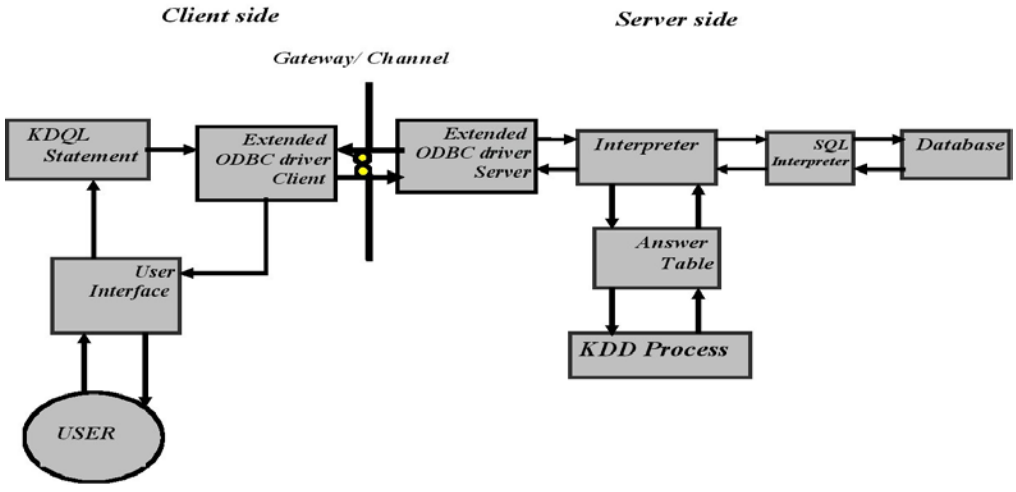

Figure 3.3 ODBC_KDD (2) model architecture

3.2 Main characteristics of ODBC_KDD models

In ODBC_KDD models the requests are formulated in SQL from the client side by the user. These SQL statements are submitted via ODBC connection through ODBC driver manager. ODBC driver links the application to specific databases and processes ODBC function calls, submits SQL requests to specific data sources, and returns results to the application. The server responds by receiving and processing client requests, and sending the results back to the client. The received database table representing the result of retrieving the databases is submitted to the KDD process. KDD process extracts all the necessary rules and patterns in the table and returns back the result to the user interface.

3.2.1 ODBC_KDD (1) model

A user sends a query to the user interface. User interface submits the request to the KDD interpreter. KDD interpreter translates the query into SQL statements and passes the request to the ODBC driver on the client side. ODBC driver forwards the request to the server side using the gateway border. ODBC driver forwards the request to the SQL application. SQL interpreter looks up the databases to get the necessary information. The result table is submitted then, to the client side in a database table form to the KDD interpreter. KDD interpreter gets the result table, and then calls the KDD process to extract the interesting pattern from the table. The final answer is sent to the user through the user interface.

3.2.2 ODBC_KDD (2) model

User formulates a *knowledge discovery query language* (*KDQL*) statement and put it to the user interface. KDQL query passes both Extended ODBC (EODBC) drivers located in two sides (client / server) over the gateway border. The EODBC driver on the server side sends the query to the interpreter. The interpreter transforms the KDQL statement to be represented in SQL mode as an SQL application. The SQL application seeks the raw database located in the server side for an answer to the retrieved statement. The result is sent back to the interpreter which is located on the same side. The interpreter initiates a KDD process in order to discover the interesting patterns in the retrieved database table. The interpreter retransforms the filtered table and sends it back to the EODBC drivers on the server side and then to client sides through the gateway border. The EODBC driver located on the client side submits back the result to the user interface and then to the user.

3.3 Architecture of ODBC_KDD models

3.3.1 Description of the first classical model

The first classical model consists of several components. Theses components can be counted and described from left to right according to figure 3.2, as follows.

1- User interface: It is the program that controls the display mode for the user (usually on a computer monitor). The program allows the user to interact with the model by writing different queries in high level languages.

2- KDD interpreter: This program translates queries from the high level form to Structured Query Language (SQL) statements in order to retrieve databases via remote access mode. The KDD interpreter passes the translated SQL application to the Open Database Connectivity (ODBC) driver. KDD interpreter encodes the result table and submits it to the KDD process to discover all necessary patterns.

3- ODBC drivers: ODBC driver on the client side receives the SQL query and send it to the ODBC driver on the server side through the getaway border. The ODBC driver on the server side passes the SQL applications to the databases.

4- SQL interpreter: The SQL interpreter processes SQL statements and retrieves information from the database. The SQL application captures all the necessary information and sends it back to the ODBC driver located on the server side.

5- DBMS: DBMS is a database management system which is a collection of programs strongly structured to manipulate data, and offering query facilities to different users.

6- Answer table: After the SQL application submits the request and retrieves the results, these results will be presented in a database table back to the client side. The table contains all the related information the user needs. The table is send to the KDD interpreter.

7- KDD process: The KDD process extracts all the interesting patterns from the result table and then sent it to the user interface.

3.3.2 Description of the second conceptual model

This model is suggested as a new conceptual model to the ODBC_KDD (1) with some new components. These new components add more capability to this model. This model is named as ODBC_KDD (2)**.** This model consist of several components, these components are counted and described from left to right according to figure 3.3, as follows:

1- User interface: It is a screen which represents the output of the KDD query in visual mode.

2- KDQL**:** KDQL is a proposed query that can handle SQL request with some visualization aids to the hidden patterns which is discovered in the retrieved database table. This KDD query will be the main point of my work. We will talk about it in more details in the Chapters 7 and 9.

3- Extended ODBC driver (EODBC): It is a conceptual remote access (client/server) databases driver. Theses drivers were suggested to the model to

handle KDD queries and keep it more attractive. In the other hand, this driver was conceptually maintained without a practical use until now. I won't focus on it in this dissertation, but nevertheless it could be a good topic for my future work. The purpose of the EODBC drivers (client/server) sides, is to (send / receive), (receive / send) KDD queries from both sides or from the *Interpreter*.

4- Interpreter: It is a server side program that interacts between the EODBC and the SQL interpreter. The goal of this program is to encode the requested KDD queries to classical SQL statements and then pass them to the SQL interpreter. In the other direction, the *Interpreter* receives the answer table. The *Interpreter* translates the answer table to a form known by EODBC drivers and KDD process. The Interpreter was proposed to deal with the KDD query and with the undefined structure until now.

5- SQL interpreter: The SQL interpreter acts just the same way as the SQL interpreter in the ODBC _ KDD (1) model. It executes SQL statements and gets tables as result**.**

6- DBMS: DBMS is a database management system which is a collection of strongly structured programs to manipulate data, offering query facilities to different users.

7- Answer table: The Interpreter receives the result table from the SQL interpreter and then calls the KDD process. The table is a simple database table with all possible patterns.

8- KDD process: The process starts to discover and extract all the interesting patterns available in the table and then sends them back the user interface. The result will be returned to the user interface on the client side.

These abstract models are motivated by extensive analyses of accessing huge databases by remote access applications.

3.4 Retrieving in ODBC_KDD models

A query is a user request for information submitted to the raw database scheme. In other description, the query is a formal request to a raw *database scheme*. A database scheme is a collection of logical structures of data, or objects. In the practice we usually use database schemes commonly known as relational database scheme which is a list of attributes and their corresponding domains. A database is a large amount of data that can be accessible for any user who use the system [27]. Database scheme can be created and manipulated with any database management system (DBMS). We can access a database by the use of SQL commands or any other query languages.

Let us consider a small *banking* relational database scheme as an example. This example could be a part of our model. This example also could demonstrate how queries can be translated from the natural language to one of the existing query languages that correspond to our ODBC_KDD models.

Brach-scheme = (bname, assets, bcity)
Customer-scheme = (cname, street, ccity)
Account-scheme= (account_number ,bname, balance)
Loan-scheme = (loan_number, bname ,amount)
Depositor-scheme = (cname, account_number)
Borrow-scheme = (cname, loan_number)

3.4.1 Comparison of SQL and Knowledge Discovery Query Language (KDQL)

As we know a query is a question or a request formulated in a natural language. Before processing it has to be converted into one of our query languages: classical SQL query or KDD query. See Chapter 1 for more information about SQL. As an example consider the previous relational scheme. We can submit natural questions in SQL forms such as: *"Find the names of the branches whose assets are greater than the assets of some branch in Debrecen"*. In SQL forms it will be suited as:

```
SELECT DISTINCT T.bname, T.assets
FROM branch T, branch S
WHERE T.assets > S.assets AND S.bcity = 'Debrecen'
```

In natural level language, we can also ask: *"Find all customers who purchase pasta and purchase cheese"*. It is a frequently occurring request written in a most common implemented query language SQL. The corresponding recursive query expression in SQL is:

```
SELECT cname, I1.iname, I2.iname
FROM customer , item I1, item I2
WHER I1.iname='pasta' AND I2.iname='cheese'
```

Both of these queries are *combinatorial* in the sense that both of them could be answered by single or multiple look up of the underlying database tables. These tables are considered as finite sets and the look up results an another table (finite set) as an answer. Consequently the result is always deterministic (it is based on deterministic finite set operation) although the result table could be very large.

There are queries which couldn't be formulated and answered by using a single SQL statement. For example, if we want to formulate an assertion such as *"Customers who purchase pasta are three times more likely to purchase cheese than customers who don't buy pasta"*. In this assertion we are going to investigate all the possibilities which "*pasta*" and "*cheese*" appears. Our goal is to prove whether this assertion is true or not. If the assertion is true then this assertion could be set as an association rule. Moreover, formulating this assertion in SQL would require some sophisticated multiple SQL statements. This answer can be the unification of multiple answers queried by the sub-queries in SQL. The alternative to this problem could be the use of KDQL instead of SQL.

KDQL has been proposed by us as we mentioned before in Section 3.3. This query language is able to express some more sophisticated user requests. We use SQL as a basic syntax structure of KDQL, but we never use KDQL syntax as an underlying language or technique in SQL. In KDQL we may apply some aggregation methods while KDQL is locating all the related patterns in the database. We may also visualize the results in different charts as well. Consequently KDQL consists of SQL syntax plus a showing statement [29]. KDQL is a related framework to DMQL in [12] in discovering patterns manner and SQL+D in [29] in visualizing the result tables. The KDQL will be presented as the main part of this thesis. Especially in Chapters 7 and 9 we are going to describe the KDQL in more detail.

For instance, we can solve the previous example "*Customers who purchase pasta are three times more likely to purchase cheese than customers who don't buy pasta"* in KDQL by a convenient way descried in Chapter 9 by using some association rules.

The rule that could be proved here is "*buying cheese* ***motivates*** *people to buy pasta*". The rule that might be proved here is "*buying cheese* "***motivates***" *people to buy pasta*". However, if we have a book called "*You can lose five pounds a week by eating pasta with cheese*" the book will be bought together with them and this may cause an *explosion*.

Retrieving databases to discover hidden information requires an attractive query language to locate all inferences rules possible in the databases. This query language could be an attractive SQL, visualization query, or, any attractive query language.

Chapter 4

Logical Foundations in Data Mining

4.1 Introduction

DM is the process of discovering particular patterns over database [10]. Typically hidden patters are stored in databases. Approaches using First Order Logic (FOL) languages for the description of such patterns offer data mining the opportunity of discovering more complex regularities which may be out of reach for attribute-value languages and classical statistical algorithms. Logical Foundations in Data Mining (LFDM) still has other advantages. Complex background knowledge provided by experts can be encoded as first order formulae and be used in the discovery task. The expressiveness of FOL enables discovered patterns to be described in a concise way, which in most cases increases readability of the output. Multiple relations can be naturally handled without explicit (and expensive) joins.

The obvious drawback of LFDM is efficiency. A LFDM algorithm must consider a much larger set of possible hypothesis (language complexity), and may have to access the database repeatedly. However, recent works in Inductive Logic Programming (ILP) show that logical approaches to data mining are feasible and can scale up to handle large databases. Before that happens, many practical and theoretical problems have to be solved, which offers many research opportunities.

LFDM fits naturally into the field of Inductive Logic Programming (ILP) [30, 31], and is inherently related to Logic Programming [53] and Deductive Databases [47].

4.2 Advantages of LFDM

- **Expressiveness**. First order logic can represent more complex concepts than traditional attribute-value languages. Typically, structural concepts are hard to represent using a zero-order language.
- **Readability**. It is arguable whether logic formulae are easier to read than decision trees or a set of linear equations. However, they are potentially readable. If the knowledge is structural, a first order representation is probably easier to read than a zero-order one.
- **Background knowledge**. Domain knowledge can be encoded and given as background knowledge. The source of the background knowledge can be an expert or a discovery system. In some cases, background knowledge can be grown during discovery time for example, in time series.
- **Multiple tables**. Handling multiple relations is natural in first order logic. Therefore, multiple database tables can be handled without explicit and expensive joins.

- **Deductive databases**. Logical discovery engines can be transparently linked to relational databases via deductive databases. Logical approaches benefit from the work done in this area of logic programming.

4.3 Disadvantages

- **Language complexity**. First order hypothesis are usually constructed through heavy search. Appropriate biases must be set a priori to make the discovery task feasible.
- **Database access times**. Candidate hypothesis must be regularly confronted with the data to guide the search. Checking one single candidate might involve heavy querying.
- **Number handling**. Logical approaches to discovery usually suffer from poor number handling capabilities. However, some proposals have been made to overcome this limitation.

4.4 DM tasks

The DM task that could be employed will follow the categorizes below [10]:

- **Classification and prediction**. The aim is to predict the value of some database field based on the values of other fields. The field to predict is sometimes called class. If the class takes discrete values, then it is a *classification* problem. If the class takes continuous numerical values, it is a *regression* problem.
- **Clustering**. The aim is to partition the set of data items into smaller subsets. The elements of one subset are similar to each other and significantly different from elements in other subsets.
- **Data summarization**. The aim is to discover patterns that describe subsets of the data. *Association rules* relate different fields we will focus on it later.
- **Dependency modeling**. The aim is to derive some causal structure within the data. One example is *functional dependency* between predicates.
- **Change and deviation detection**. Data has a sequential structure, temporal, physical or other. The aim is to find patterns assuming an ordering of the observations.

4.5 Logical approaches

Logical approaches to DM state the discovery tasks in a logical setting. Hypothesis, evidence, and background knowledge are represented in some first order logic framework, such as Horn clauses[1], first order temporal logic (FOTL)[2], logical decision trees[3], etc [79].

(1) Horn clauses: came from the logician Alfred Horn in 1951 and it is a set of atomic literals with at most one positive literal. Usually written L <- L1, ..., Ln , where n≥0. If L is false the clause is regarded as a goal and it could be expressed as a subset statements of first order logic..
(2) FOLT: first-order temporal logic, obtained by augmenting classical first-order logic with temporal operators (such as since and until) operating over various kinds of flows of time. FOLT are known by Scott and Lindström in the 1960s.
(3) Logical decision trees: A logical decision trees is binary decision tree in which each node contains a conjunction of literals. This conjunction may share variables with nodes above it in the tree.

4.5.1 The general logical setting

The general logical approaches can be sets by given predicate formulas to find some statistical inferences.

Given:

B, background knowledge. Potentially any set of logical formulae. Typically a logic program. Sometimes a definite logic program.

E, evidence/examples/data. Potentially any set of logical formulae. Typically ground atoms. Sometimes unconstrained atoms.

A language of hypothesis L_H.

A notion of satisfaction.

Find:

H in L_H, set of hypothesis/model, such that *H* satisfies *E* relatively to *B*. Potentially *H* is any set of logical formulae. Typically a set of clauses, either definite or indefinite. Currently, other first order formulae are being used.

4.6 Classification and prediction

This is by far the most studied problem in relational learning. It can be stated as follows.

Given evidence *E* and background knowledge *B*, find hypothesis *H* such that $H\&B \models E$, where *H*, *B* and *E* are logical formulae.

In other words, the discovered hypothesis should *explain* the evidence.

System FOCL discovers rules for early diagnosis of Alzheimer's disease [32]. Rules are Horn clauses. One particular form of language bias, monotonicity constraints, prevents learned rules to be inconsistent with existing knowledge. Expressing one relation R in terms of a set of relations $\{R_1,R_2,...,R_n\}$ [33] can be useful to detect redundancies in a database. System P-Progol[4] constructs *Horn clauses* to predict the carcinogenicity of chemicals used by people [34]. Background knowledge contains general information about structure and composition of chemicals. This is one good example of the suitability of logical approaches to handle complex, structured knowledge.

---

(4) System P-Progol : is a system implemented to contain aspects of the Progol algorithms by Aleph. It was commenced in 1993 as a part of a project undertaken by Ashwin Srinivasan and Rui Camacho at Oxford University. Progol algorithm is described in detail in S.H. Muggleton (1995), Inverse Entailment and Progol, New Gen. Comput..

4.7 Clustering

The approach to first order clustering described in [35] discovers a set of Horn clauses that partition a set of ground examples into homogeneous classes. A typical ILP refinement operator is adapted for clustering. System KM[5] of [36] is applied to a real chemical research database about compounds, their ingredients, and properties. There is one table for the ingredients of the compounds and more than 50 tables for the properties. The system uses meta-queries to restrict the search.

In one experiment, given the meta-query

$$Ingredient(C, X, Y) \wedge Property(C, Z) \Rightarrow Cluster\ (Z)$$

The system produced the rule

$$Ingredient(C, 3060, X) \wedge densityBM(C, Y) \Rightarrow$$
$$Clusters\ (Y) = \{(1.243, 0.46, 0.99), (-999.9, 0.2, 0.01)\}$$

The rule says that 99% of the compounds with ingredient 3060 have densityBM[6] around 1.243 (with a variance of 0.46). The other cluster possibly indicates the presence of noisy data.

The rule indicates that for this type of compounds, the majority (99%) of values of property densityBM are near 1.243 (with a variance 0.46). Thus, the ingredient 3060 has a very specific effect on the compound's densityBM property, irrespective of the other ingredient X. This information is extremely valuable to the chemists. This rule also indicates (with a very low probability cluster) that there may be some noisy data in the densityBM table.

Logical decision trees for clustering [37]. System RIBL [38] finds groups of patients with unusual cost or success structure. This is a real application on relatively large hospital database.

4.8 Data summarization

The most common logical statement of this problem is given evidence E and background knowledge B, find hypothesis H such that $B\&E \models H$, where H, B and E are logical formulae.

In other words, the hypothesis should *describe* the evidence. This statement is related to the non-monotonic setting of [39], also known as descriptive ILP.

---

(5) KM: is a Knowledge Miner system extracted automatically from the schema by the meta query generator the important information and returns a set of patterns describing the relationships between specific ingredients and properties.

(6) densityBM: is a majority of the variance in the density function.

4.8.1 Integrity constraints

The hypothesis H can be a set of integrity constraints. An *integrity constraint* is a formula of the form

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$$

Where the A_i and the B_i are first order atoms like $p(a,X,b(t))$, *and* $a \subseteq A_i$, $b \subseteq B_i$, t is a term and X is a variable.

System *CLAUDIEN* finds integrity constraints in a database appeared in [40].

4.8.2 Association rules

In [41] they define first order *association rules* as expressions of the form

$$\{A_1 ,\ldots ,A_n \} \Rightarrow \{B_1 ,\ldots, B_m\}$$

Where the A_i and the B_i are first order atomic forms. Each data item may be covered or not covered by a given association rule. The proportion of data items covered by one rule is its *support.* The proportion between the number of data items that are covered by the rule and the data items that are covered by its antecedent is the *confidence* of the rule. We are going to consider them in more details in Chapter 5.

Example 4.1 The rule

$$\{ \text{Likes (KID, A), has (KID, B)} \} \Rightarrow \{ \text{Prefers (KID, A, B) (C: 98, s: 70)} \}$$

Says that 98% of the kids prefer a thing they like over a thing they have, and 70% of the kids like and have something and prefer the first over the second.

The implemented system WARMR extends APRIORI [42] and learns association rules. It was applied to part-of-speech tagging (natural language processing).

In another work, [38] association rules are defined as first order clauses. The experiments shown are with the KRK data (chess).

4.9 Other tasks

4.9.1 Dependency modeling

System INDEX induces attribute dependencies in a relational database [44]. The two types of attribute dependencies considered are *functional dependencies* and *multi-valued dependencies*.

4.9.2 Change and deviation detection

In [40] the non-monotonic setting and system CLAUDIEN are used to discover regularities between temporal states in clinical databases. In [45] the *first order temporal logic* (FOTL) is used to describe discovered patterns. One example of one temporal pattern is X_1 *Until* X_2, where X_1 and X_2 are second order variables. The method is applied to capture the behavior of financial analysts in the stock market.

4.10 Links to relational databases

A LFDM engine can be linked to a relational database at two alternative levels.

- First Order predicates can be mapped to database tables/views.
- First Order queries can be translated to SQL or other similar query languages.

The first option is a more transparent solution for the designer of the LFDM engine (if the data mining system is implemented in a first order language like Prolog). However, this solution may increase database access times [43,46].

The second option of translating first order formulae into SQL queries may take advantage of the relational database query strategies [27]. For example, the coverage of a first order rule can be computed via SQL counts. Hypothesis may also be directly constructed in the query language using (*Inductive Logic Programming* ILP/ *Machine Learning* ML), ILP/ML techniques.

4.10.1 Mapping first order predicates into views

We consider the mapping of the first order predicates into database tables or other type of views by several ordinarily steps. These steps are more likely mentioned in [118] and it is pointed below:-

- *Step1.* For each table R with fields $A_1,...,A_n$, construct a predicate $R(A_1,...,A_n)$.
- *Step2.* For each table R with fields $A_1,...,A_n$, where the attributes $A_j,...,A_k$ are the primary key, for each A_x not in the primary key, construct a predicate $R\_Ax(A_j,...,A_k,A_x)$**.**
- *Step.3* For each attribute A_x which is not a primary key and has values $a_1,...,a_m$, construct a set of predicates $R\_A_x\_a_i(A_j,...,A_k)$, where $A_j,...,A_k$ are the primary key.
- *Step4.*The same as the previous one, but considering a set of value intervals instead of a set of values.

Other possibilities exist, obviously. The choice of the most appropriate mapping depends on how hypothesis are constructed. The first mapping introduces less predicates. The other mappings have more predicates with smaller arity. From the second to the third mapping we lose the possibility of manipulating the value of A_x.

4.10.2 Translating first order queries into SQL

Example 4.2 If we ask in our natural language a question such as *"find all employers who are mangers and getting salary or expenses more than 75000 USD a year"*. This question could be reconstructed by the first order rule in [36] as follows.

expensive_employee(Name) ← employee(Name, Salary1, Manager), Salary1 > 75000, employee(Manager, Salary2, _),Salary1 > Salary2

This first order rule can be converted into the following rule

expensive_employee(Name)← sql_node(Name).

The SQL query corresponds to the predicate *sql_node* and formulates the following query.

SELECT *employee_0.NAME*
FROM *employee employee_0, employee employee_1*
WHERE *employee_0.SALARY > 7500 0* ***AND***
employee_1.NAME = employee_0.MANAGER ***AND***
employee_0.SALARY > employee_1.SALARY

4.10.3 Deductive databases

A third way of linking the discovery engine to a relational database is via deductive databases [47]. This solution provides a natural link between the two worlds. Moreover, LFDM inherits a good deal of research work and results from the DDB field.

4.11 Managing complexity

To reduce complexity we are going to focus of two main aspects: the *database size* and the *language* that could retrieve that database.

4.11.1 Database size

One of the majorities of reducing complexity is to reduce the size of the database. The main methods for reducing the size of database are pointed as follows:

- Partitioning the database into bits that fit in the main memory [41].
- Sampling [32].

4.11.2 Languages

The problem of language complexity can be managed by restricting the language of hypothesis. In other words, by defining an appropriate language bias.

One line of work in ILP exploits ways of defining the language in a declarative manner. Similar approaches can also be found in the areas of KDD and DM.

- *Rule Models* [48] also known as *Clause Schemata* [49] are second order rules like

$$P(X,Z) \leftarrow Q(X,Z),P(Y,Z)$$

Where P and Q are variables that range over predicate names. The search for clauses consists mainly in instantiating the predicate variables to all possible values.

Example 4.3 Rule models, clause schemata:

$P(X,Y) \leftarrow Q(X,Z),P(Z,Y).$ $\quad$ $(P \text{ in } \{p_1, p_2\}, Q \text{ in } \{q_1, q_2\})$

- *Meta-patterns* [36], also known as *meta-queries* [36], are second order rules very similar to rule models. They can be automatically discovered or provided by the user. Example to meat-pattern will be mentioned in the next Section in M-SQL [51].

- *DLAB [*41], *Clause Sets* [50], MILES-CTL [28].

Example 4.4 to DLAB is as follows:

$p_1(X,Y) \leftarrow [q_1(X,Z),q_2(X,Z)],p_1(Z,Y)$
$p_2(X,Y) \leftarrow [q_1(X,Z),q_2(X,Z)],p_2(Z,Y)$

- *Antecedent Description Grammars* [54], *Clause Structure Grammars* [55].

Example 4.5 to *Antecedent Description Grammars* is as follows:

$body(p_1(X,Y)) \rightarrow q\_lit, p\_lit(p_1)$
$body(p_2(X,Y)) \rightarrow q\_lit, p\_lit(p_2)$
$q\_lit \rightarrow [q_1(X,Z)];[q_2(X,Z)]$
$p\_lit(p_1) \rightarrow [p_1(Z,Y)]$
$p\_lit(p_2) \rightarrow [p_2(Z,Y)]$

Example 4.6 to *Clause Structure Grammars* is :

$body \rightarrow q\_lit(+1),rec\_lit(+1)$
$q\_lit \rightarrow [q_1/2],[q_2/2]$
$rec\_lit \rightarrow [P]$

4.12 SQL queries with extensions

In the KDD field two proposed extensions to SQL enable the user to write queries whose answer is a set of rules instead of a set of records. These query languages have the spirit of declarative bias. We take M-SQL in [51] and DMQL in [12] as an example as an extension SQL query.

- In M-SQL [51].

We can answer such a question*" find all rules in table T involving the attributes disease, age and claimant with confidence of at least 50%*"

SELECT *
FROM Mine(T) R
WHERE
R.Body **<** *{Disease=*, Age=*, ClaimAnt=*}*
AND { } **<** *R.Body*
AND *R.Consequent* ***IN***
{Disease=, Age=*, ClaimAnt=*}*
AND *R.Confidence > 0.5*

- In DMQL [12].

We can maintain a query such as " find all rules involving the attributes *major*, *gpa*, *status*, *birth_place*, *address* in relation *student* for those born in *Canada* in a *university* database".

Find association rules in the form of
major(s:student,X)/\Q(S,Y) -> R(S,Z)
related to *major,gpa,status,birth_place,address*
From *student*
Where *birth_place = "Canada"*

Result rules will be represented instead of database table

major(S,'Science') /\ gpa(S,'Excellent') $\rightarrow$ *status(S,'Graduate') (60 %)*

We can conclude to the following, the LFDM is a part in DM that is concerned with the following points:

- Study and optimize database access patterns of ILP algorithms [52].
- Use of ILP algorithms for feature extraction as preprocessing [52].
- Use of ILP algorithms to build more complex hypothesis from patterns derived by other approaches [52].
- Extend query languages to allow vagueness adapting declarative bias languages.

In DMQL [12] and M-SQL [51] we use the following.

- Explore forms of declarative bias Meta-patterns [36].
- Improve number handling capabilities.

Finally, LFDM is feasible and some fielded applications exist. LFDM is worthwhile since it is necessary (or at least convenient) for problems with multiple relations or complex knowledge. LFDM offers many research opportunities.

Chapter 5
Mining the Discovered Association Rules

5.1 General foundation of association rules

Association rules identify collections of data attributes that are statistically related in *i-extended database*. An association rule is of the form $X \Rightarrow Y$ where X and Y are disjoint conjunctions of attribute-value pairs. The *confidence* of the rule is the conditional probability of Y given X, $Pr(Y|X)$, and the *support* of the rule is the prior probability of X and Y, $Pr(X$ and $Y)$ [74]. Here probability is taken to be the observed frequency in the data set. The traditional association rule mining problem can be described as follows. Given a database of transactions, a minimal confidence threshold and a minimal support threshold, find all association rules whose confidence and support are above the corresponding thresholds. In our research, we have extended this traditional framework to better fit into this application domain.

5.2 Mining the association rules

Association rule mining has been most widely studied of all data mining functions. Association rule mining though easy to use, but in its current form lacks for exploration. Minimal interaction with the user during the mining process, slow response, voluminous output etc. prevents end-user to exploit full potential of association rule mining [56]. One of the reasons behind maintaining any database is to enable the user to find interesting patterns and trends in the data. For example, in a supermarket, the user can figure out which items are being sold most frequently. Nevertheless, this is not the only type of `trend', which one can possibly think of. The goal of database mining is to automate this process of finding interesting patterns and trends. Once this information is available, we can perhaps get rid of the original database. The output of the DM process should be a "*summary*" of the database. This goal is difficult to achieve due to the vagueness associated with the term `interesting'. The solution is to define various types of trends and to look for only those trends in the database [57]. One such type constitutes the association rule.

In the some Sections, we shall assume the supermarket example, where each record or tuple consists of the items of a single purchase [58]. However, the concepts are applicable in a large number of situations.

In the present context, an association rule tells us about the association between two or more items. For example: *"In 80% of the cases when people buy bread, they also buy milk"*. This tells us of the association between *bread* and *milk*. We represent it as following.

$Bread \Rightarrow milk\ /80\%$

This should be read as *"Bread means or implies milk, 80% of the time."* Here *80%* is the *"confidence factor"* of the rule.

Association rules can be between more than 2 items.

Example 5.1

Bread, milk $\Rightarrow$ *jam / 60%*

Bread $\Rightarrow$ *milk, jam /40%*

Given any rule, we can easily find its confidence. For example, for the rule

Bread, milk $\Rightarrow$ *jam*

We count the number say n_1, of records that contain bread and milk. Of these, how many contain jam as well? Let this be n_2. Then required confidence is n_2/n_1.

This means that the user has to guess which rule is interesting and ask for its confidence. However, our goal was to *'automatically'* find all interesting rules. This is going to be difficult because the database is bound to be very large [59]. We might have to go through the entire database many times to find all interesting rules.

According to this description, we will motivate mining association rules as the rules that are concerned in finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories [60]. Our concern is to find frequent association rules among set of items in relational database. On the other side, there are existing applications for mining association rules. Theses rules could be one of the following *Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering and classification* [69,70,74].

Rules formalization will be as follow.

Rule form: *"Body → Head [support, confidence]"*

Example 5.2
buys(x, "bread") → *buys(x, "milk") [0.6%, 65%]*
major(x, "CS") $\wedge$ *takes(x, "DB")* → *grade(x, "A") [1%, 75%]*

The basic concepts of mining the association rules will be the following.

A transaction is a set of items*:* $T=\{i_a,i_b,....,i_t\}$, $T \subset I$, where I is the set of all possible items $\{i_a,i_b,....,i_n\}$ and D, the task relevant data, is a set of transactions [60]. The association rule is of the form $P \to Q$, where $P \subset I, Q \subset I, and P \bigcap Q = \phi$.

Some examples for some existing applications for mining association rules.

Basket Data: *Tea* $\wedge$ *Milk* $\Rightarrow$ *Sugar [0.3, 0.9]*

In this association rule the *Tea* and *Milk* was implied by *Sugar* with *0.3 supports* and *0.9 confidences*. It says" Over 0.3 of the customers who purchase tea and milk at least 0.9 of them purchase sugar".

Relational Data: X.Diagnosis = "heart disease" ∧ X.Sex ="male" ⇒ X.Age > 50 [0.4, 0.7].

Suppose we have a relational database of *Patinas* with these attributes *(Age, Sex, Diagnosis*). This rule say's "Over 0.4 of the patinas are men who had heart disease and at least 0.8 of them are over the age of 50".

Object-Oriented Data: S.Hobbies = "sport"∧ "art" ⇒ S.Aeg(30)= Young [0.5 , 0.8]

If we are working with Object-Oriented data and we had a database with fields such as hobbies and age then we may stress a rule that say's *" 0.5 of people who have hobbies like sport and art at least 0.8 of them are young in the age of 30".*

5.3 Support itemset

The common-sense approach to solving this problem is following.

Let $I = \{ i_1, i_2, ..., i_n \}$ be a set of items. We will referre to it as itemset. An itemset containing *k* items are called *k-itemset***,** and the itemset could also be seen as a conjunction of items (or a predicate) [9]. The number of times, this itemset appears in the database is called its "*support*". Note that we can speak about support of an itemset and confidence of a rule. The other combinations, *support* of a rule and *confidence* of an itemset are not defined.

Now, if we know the support of '*I*' and all its subsets, we can calculate the confidence of all rules which involve these items. For example, the confidence of the rule $i_1, i_2, i_3 \Rightarrow i_4, i_5$**,** which *support of* $\{i_1, i_2, i_3, i_4, i_5\}$**,** will only *support of* $\{ i_1, i_2, i_3 \}$.

So, the easiest approach would be to let *`I'* contain all items in the supermarket. Then setup a counter for every subset of *`I'* to count all its occurrences in the database [61]. At the end of one pass of the database, we would have all those counts and we can find the confidence of all rules. Then select the most *"interesting"* rules based on their *confidence* factors.

The problem with this approach is that, normally *`I'* will contain at least about 100 items. This means that it can have 2^{100} subsets. We will need to maintain that many counters. If each counter is a single byte, then about 10^{20} *GB* will be required. Clearly this can't be done.

Referring to *P, Q* we can say that. If $P \rightarrow Q$ holds in *D* with *support s* and $P \rightarrow Q$ has a *confidence c* in the transaction set *D.* The support or the confidence of P, Q will have the following probabilities.

$Support(P \rightarrow Q) = \Pr obability(P \cup Q)$
$Confidence\ (P \rightarrow Q) = \Pr obability\ (Q / P)$

The support of $P = P_1 \wedge P_2 \wedge ... \wedge P_n$ in D help us to know the $\sigma(P/Q)$ which are the percentage of the transaction T in D *satisfying in P (number of T by cardinality of D) [62].*

The confidence of the rule $P \rightarrow Q$ where $\varphi(P \rightarrow Q / D)$ ratio $\sigma((P \wedge Q) / D) by \sigma(P / D)$, and the thresholds will consider the *minimum support* σ' and the *minimum confidence* φ' [63].

5.4 Minimum support with confidence

To make the problem tractable, we introduce the concept of minimum support. The user has to specify this parameter - let us call it *minsupport*. Then any rule

$i_1, i_2, ..., i_n \Rightarrow j_1, j_2, ... , j_n$

Needs to be considered, only if the set of all items in this rule which is $\{ i_1, i_2, ... , i_n, j_1, j_2, ... , j_n \}$ has support greater than minsupport.

The idea is that in the rule

Bread, milk $\Rightarrow$ *jam*

If the number of people buying bread, milk and jam together is very small, then this rule is hardly worth consideration (even if it has high confidence).

Our problem now becomes - Find all rules that have a given minimum confidence and involves itemsets whose support is more than minsupport. Clearly, once we know the supports of all these itemsets, we can easily determine the rules and their confidences. Hence, we need to concentrate on the problem of finding all itemsets, which have minimum support [64]. We call such itemsets as frequent itemsets.

In fact, one of the strong rules could be the frequent itemset which represent in the frequent (or large) predicate P in a set D that support of P large minimum support. Other rule $P \rightarrow Q\ (c\%)$ is strong where $(P \wedge Q)$ is frequent (or large) and c is larger than minimum confidence.

5.5 Properties of the frequent itemsets

The methods used to find frequent itemsets are based on, the following properties.

1. *Every subset of a frequent itemset is also frequent***.** Algorithms make use of this property in the following way: we need not find the count of an itemset, if all its subsets are not frequent [65]. So, we can first find the counts of some short

itemsets in one pass of the database. Then consider longer and longer itemsets in subsequent passes. When we consider a long itemset, we can make sure that all its subsets are frequent. This can be done because we already have the counts of all those subsets in previous passes.

2. Let us divide the tuples of the database into partitions, not necessarily of equal size. Then *an itemset can be frequent only if it is frequent in atleast one partition*. This property enables us to apply divide and conquer type algorithms. We can divide the database into partitions and find the frequent itemsets in each partition [66]. An itemset can be frequent only if it is frequent in atleast one of these partitions. To see that this is true, consider k partitions of sizes $n_1, n_2,..., n_k$.Let minimum support be s.
3. Consider an itemset which does not have minimum support in any partition. Then its count in each partition must be less than $sn_1, sn_2,..., sn_k$ respectively. Therefore its total count must be less than the sum of all these counts, which is $s(n_1 + n_2 +...+ n_k)$. This is equal to *s*(size of database)*.Hence, the itemset is not frequent in the entire database [67].

5.6 Different kinds of association rules

We will point out some of the known association rules and it is as follow [123].

- ***Boolean vs. quantitative associations***: Based on the types of values handled and the association on discrete vs. continuous data.

  Ex. $Age(X,30-45) \wedge Income(X,50K-75K) \rightarrow Buys(X,SUVcar)$

- ***Single dimension vs. multiple dimensional associations***: Based on the dimensions in data involved [68]. One predicate then single dimension and more predicates then multi-dimensions.

  Ex. $Buys(X,bread,2) \rightarrow Buys(X,milk,1)$
  $Age(X,30-45) \wedge Income(X,50K-75K) \rightarrow Buys(X,SUVcar)$

- ***Single occurrence vs. multiple occurrences*:** In this type one item may occur more than once in the transaction and not only the presence of the item is important but its frequency [69].

  Ex. $Buys(X,bread,2) \rightarrow Buys(X,milk,1)$

- ***Association vs. correlation analysis*:** Association does not necessarily imply correlation.

  Ex.: $\frac{P(A \wedge B)}{P(A)P(B)}$ it could be equal to 1 or it is greater than 1 or less than 1.

5.7 How to mine the association rules?

To answer this question we should consider two main ides that are the *input* and *finding rules*.

- The input will be a database of a transactions and each transaction is a list of items (Ex. Purchased by a customer in a visit).
- Finding *all* rules that associate the pressure of one set of the items.

Example 5.3 98% of people who purchase tires and auto accessories also get automotive services done.

There are no restrictions on the number of items in the head or body of rule.

5.8 Rule measures for support and confidence

To measure the support and confidence we will consider the following example.

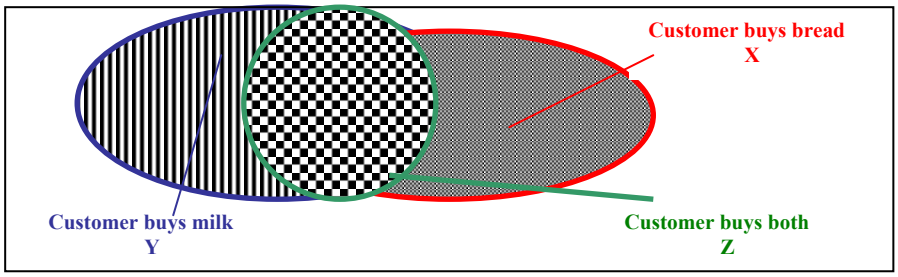

Figure 5.1 Support and confidence example

If our interest is to find all the rules $X \wedge Y \rightarrow Z$ with minimum confidence and support. Support s, probability that a transaction contains $\{X,Y,Z\}$ and confidence c, conditional probability that a transaction having $\{X,Y\}$ also contains Z.

Transcation ID	Items bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Table 5.1 Support and confidence example

In table 5.1 we will give the minimum *support* 50%, and the minimum *confidence* also 50% then we will have.

$$A \rightarrow C(50\%,66.6\%), C \rightarrow A(50\%,100\%).$$

After defining the support and confidence percentage mining the association rules will be as follows.

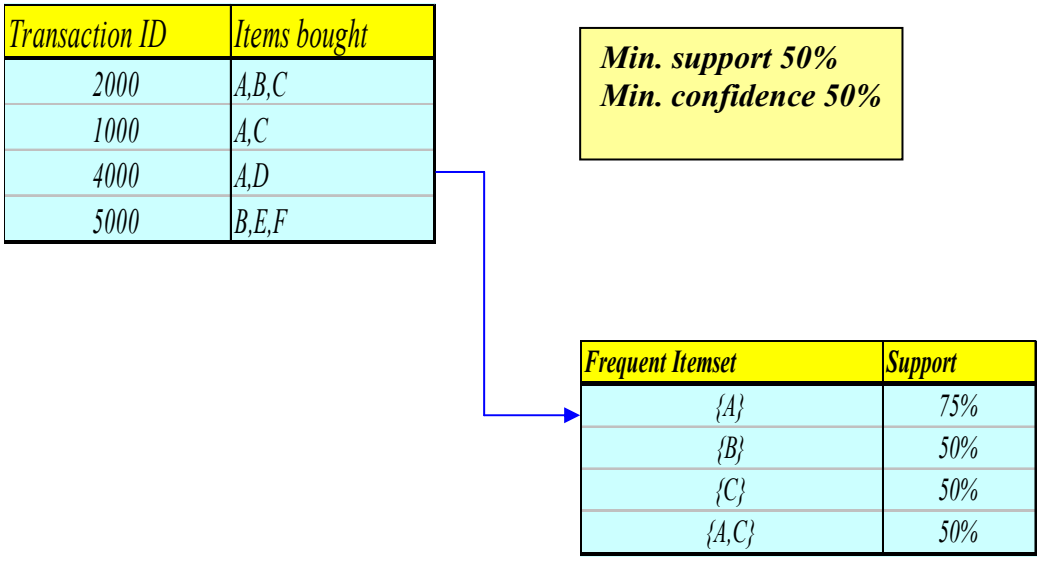

Figure 5.2 Mining the support and confidence percentage

Referring to the previous figure and from rule $A \rightarrow C$ we can define that.

Support =support ({A, C}) =50%.
Confidence= support ({A, C})/support ({A}) =66.6%.

The Apriori *principle algorithm* defined the frequent itemset as *'any subset of a frequent itemset must be frequent'* [42].

- ***Apriori algorithm :*** The Apriori algorithm is as follow.

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) ***do begin***
C_{k+1} = candidates generated from L_k;
for each transaction t in database ***do***
increment the count of all candidates in C_{k+1} that are contained in t
L_{k+1} = candidates in C_{k+1} with min_support
end
return $\cup_k L_k$;

- ***Apriori algorithm example***

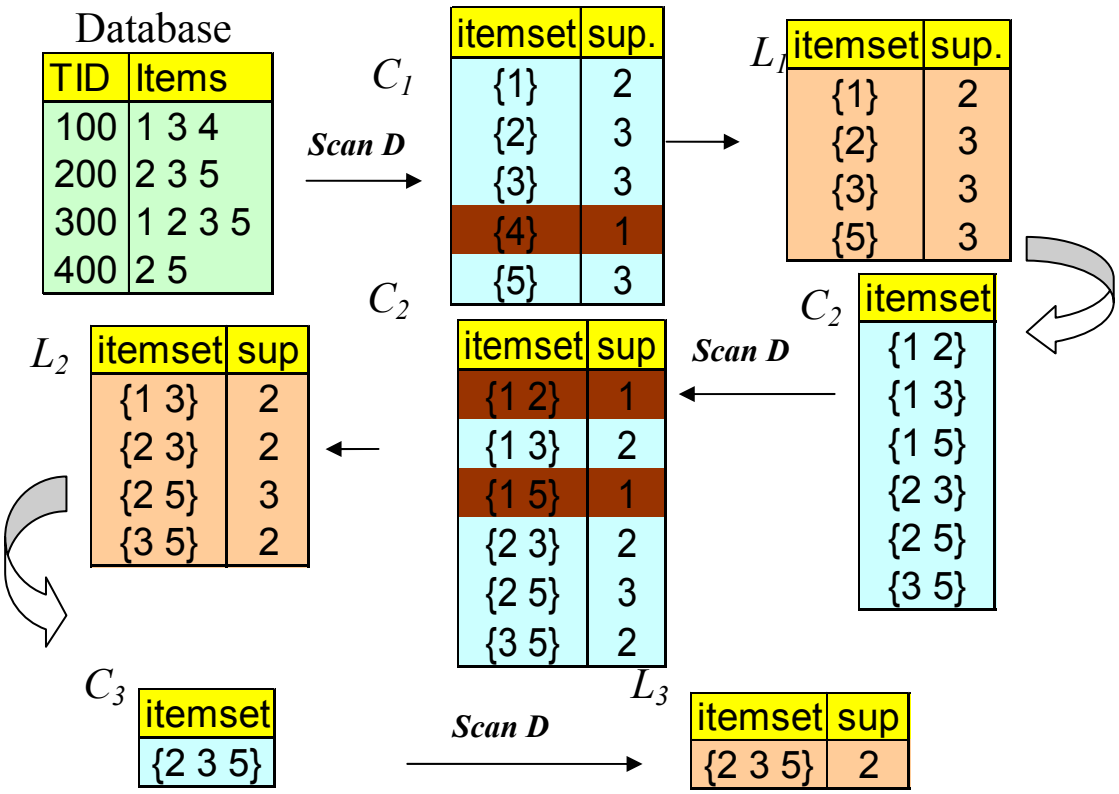

Figure 5.3 Apriori algorithm example

Note that the sets *{1,2,3},{1,2,5}* and *{1,3,5}* are not in *C*.

5.9 Mining frequent itemsets

Mining frequent itemsets is the key step of finding supported items and generating the association rules. Mining frequent itemsets was defined from two main aspects.

- Find the *frequent itemsets*: the sets of items that have minimum support: A subset of a frequent itemset must also be a frequent itemset; i.e., if *{AB}* is a frequent itemset, both *{A}* and *{B}* should be a frequent itemset; iteratively find frequent itemsets with cardinality from *1* to *k (k-itemset)*.

- Using the frequent itemsets to generate association rules, to generates the association rules we are going to use the following properties: Only strong association rules are generated; frequent itemset are satisfying minimum support threshold; strong *AR* satisfy minimum confidence threshold; *Confidence* $(A \rightarrow B) = (\Pr ob(B/A) = \frac{Support(A \cup B)}{Support(A)}$.

To apply these properties we will consider this algorithm.

For each frequent itemset, f, generate all non-empty subsets of f.
For every non-empty subset s of f
do
Output rule $s \rightarrow (f - s)$ ***if*** *support* ***(f)*** */ support(****s****)* $\geq$ *min_confidence*
end.

5.10 From association mining to correlation analysis

To reach the correlation analysis from the association rule we should consider the following.

5.10.1 Interestingness measurements

The interesting measure we will focus on is the *object and subject* measure

■*Objective measures:* With two popular measurements ∂ *support,* and confidence.

■*Subjective measures* was introduced by (Silberschatz & Tuzhilin, KDD 95) in [70]: A rule (pattern) is interesting if ∂ it is *unexpected* (surprising to the user), and/or *actionable* (the user can do something with it).

5.11 Criticism to support and confidence

For a deep understanding to the *criticism to support and confidence* we consider two examples produced by (Aggarwal & Yu, PODS98) in [71].

Example 5.4 Among 5000 students there are 3000 play basketball, 3750 eat cereal and 2000 both play basket ball and eat cereal.

1.*Play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%.
2.*Play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is far more accurate, although with lower support and confidence.

	basketball	***not basketba***	***sum(row)***
cereal	*2000*	*1750*	*3750*
not cereal	*1000*	*250*	*1250*
sum(col.)	*3000*	*2000*	*5000*

Table 5.2 Criticism to support and confidence

Example 5.5 In table 5.3 X and Y *positively* correlated, X and Z, *negatively* related, and for *support* and *confidence* of $X \Rightarrow Z$ dominates.

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Table 5.3 X, Y, Z positively or negatively correlated

We need a measure of dependent or correlated events $corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$

Rule	Support	Confidence
X=>Y	25%	50%
X=>Z	37.50%	75%

Table 5.4 X, Y, Z support and confidence

5.12 Other interestingness measures for interest

The *P(B|A)/P (B)* are also called the *lift* of rule *A => B*. Interest (*correlation, lift*) can be defined by the following.

- Taking both *P(A)* and *P(B)* in consideration.
- *P(A^B)=P(B)*P(A)*, if *A* and *B* are independent events.
- *A* and *B negatively correlated*, if the value is less than *1*, otherwise *A* and *B positively correlated.*

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Itemset	Support	Interest
X,Y	25%	2
X,Z	37.50%	0.9
Y,Z	12.50%	0.57

Table 5.5 Interest (correlation, lift)

5.13 Rule constraints in association mining

- There are two kinds of rule constraints such as following.

1- Rule form constraints: meta-rule guided mining.

Ex.: $P(x, y) \wedge Q(x, w) \rightarrow takes(x, \text{“database systems”})$.

2- Rule (content) constraint: constraint-based query optimization (Ng, et al., SIGMOD’98) [72].

Ex.: *Sum (LHS) < 100 ∧ Min (LHS) > 20 ∧ Count (LHS) > 3 ∧Sum (RHS) > 1000.*

- There are 1-variable vs. 2-variable constraints which was presented by (Lakshmanan, et al. SIGMOD'99) in [73] and it is as following:

1- Var: A constraint confining only one side *(L/R)* of the rule, e.g., as shown above.

2- Var: A constraint confining both sides *(L and R)*.

Ex.: *Sum (LHS) < Min (RHS) ∧Max (RHS) < 5* Sum (LHS).*

5.14 Constrain-Based association query

Constraint data by using *SQL-like* queries can be defined by given some existing algorithms below.

- Given *CAQ = { (S1, S2) | C }*, the algorithm should be:
  - Sound: It only finds frequent sets that satisfy the given constraints *C*.
  - Complete: All frequent sets satisfy the given constraints *C* are found.
  - Ex. Find product pairs sold together in *Debrecen in Dec.'2001'*.

- A naïve solution:
  - Apply Apriori for finding all frequent sets, and then to test them for constraint satisfaction one by one.

- Other approach:
  - Comprehensive analysis of the properties of constraints and try to *push them as deeply as possible inside* the frequent set computation.

Finally we can say that association rule mining is probably the most significant contribution from the database community in KDD. Many interesting issues have been explored to association analysis in other types of data such as spatial data, multimedia data, time series data, etc.

Chapter 6

Data Mining Query Languages (DMQL)

6.1 Introduction

With the explosive growth of information stored in databases, it has become increasingly necessary for users to utilize automated tools in order to find, extract, filter, and evaluate the desired information and resources [4]. In addition, with the primary tool for electronic commerce, it is imperative for organizations and companies, who have invested millions of records in large databases on remote (client/server) side databases technologies, to track and analyze user access patterns.

These factors give rise to the necessity of creating server-side and client-side intelligent systems that can effectively discover the knowledge and mine it in both across the Internet and in particular query languages, these query languages are called *Data minin*g or *KDD query languages* [5]. The question that will appear is that what kind of databases *Data mining query* could interact.

6.2 DM is a part of KDD process

For understanding the *data mining query languages* descriptions, we should point out the difference between DM and KDD. As it has been said "*data mining and knowledge discovery in databases, refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases*" [5]. While DM and KDD are frequently treated as synonyms, DM actually is a part of the knowledge discovery process. In figure 1.1 DM was shown as a step in an iterative knowledge discovery process.

The *knowledge discovery in databases* process comprises of a few steps leading from raw data collections to some form of new knowledge. The iterative process consists of the following steps [6]:

- *Data Cleaning*: also known as data cleansing, it is a phase in which noise data and irrelevant data are removed from the collection.

- *Data Integration*: at this stage, multiple data sources, often heterogeneous, may be combined in a common source.

- *Data Selection*: at this step, the data relevant to the analysis is decided on and retrieved from the data collection.

- *Data Transformation*: also known as data consolidation, it is a phase in which the selected data is transformed into forms appropriate for the mining procedure.

- *Data Mining*: it is the crucial step in which clever techniques are applied to extract patterns potentially useful.

- *Pattern Evaluation*: in this step, strictly interesting patterns representing knowledge are identified based on given measures.

- *Knowledge Representation*: is the final phase in which the discovered knowledge is visually represented to the user. This essential step uses visualization techniques to help users understand and interpret the data mining results.

It is common to combine some of these steps together. For instance, *data cleaning* and *data integration* can be performed together as a *pre-processing* phase to generate a *data warehouse*. *Data selection* and *data transformation* can also be combined where the consolidation of the data is the result of the selection, or, as for the case of data warehouses, the selection is done on transformed data.

Data mining query (DMQ) is an iterative process to the KDD process. Once the DMQ discovered knowledge and presented the knowledge to the user, the evaluation measures can be enhanced, the mining can be further refined, new data can be selected or further transformed, or new data sources can be integrated, in order to get different, more appropriate results [122].

6.3 Types of discovered data by data mining queries

In principle, *data mining queries* are not specific to one type of media or database. DMQ should be applicable to any kind of information repository. However, algorithms and approaches may differ when applied to different types of data. Indeed, the challenges presented by different types of data vary significantly. *Data mining query* is used and studied for databases, including *relational databases, object-relational databases* and *object-oriented databases*, *data warehouses*, *transactional databases*, *unstructured and semi-structured repositories* such as the *World Wide Web*, advanced databases such as s*patial databases, multimedia databases, time-series databases and textual databases*, and even *flat files [8]*. Here are some examples in more detail:

- *Flat files*: Flat files are actually the most common data source for data mining algorithms, which can be interacting commonly by *data mining query* especially at the research level. Flat files are simple data files in text or binary format with a structure known by the data-mining algorithm to be applied and most of data mining query supports flat files in the source database. The data in these files can be transactions, time-series data, scientific measurements, etc.

- *Relational Databases*: Briefly, a relational database consists of a set of tables containing either values of entity attributes, or values of attributes from entity relationships. Tables have columns and rows, where columns represent attributes and rows represent tuples. A tuple in a relational table corresponds to either an object or a relationship between objects and is identified by a set of attribute values representing a unique key. In figure 6.1, we present some relations Customer, Items, and Borrow representing business activity in a fictitious video store OurVideoStore. These relations are just a subset of what could be a database for the video store and is given as an example.

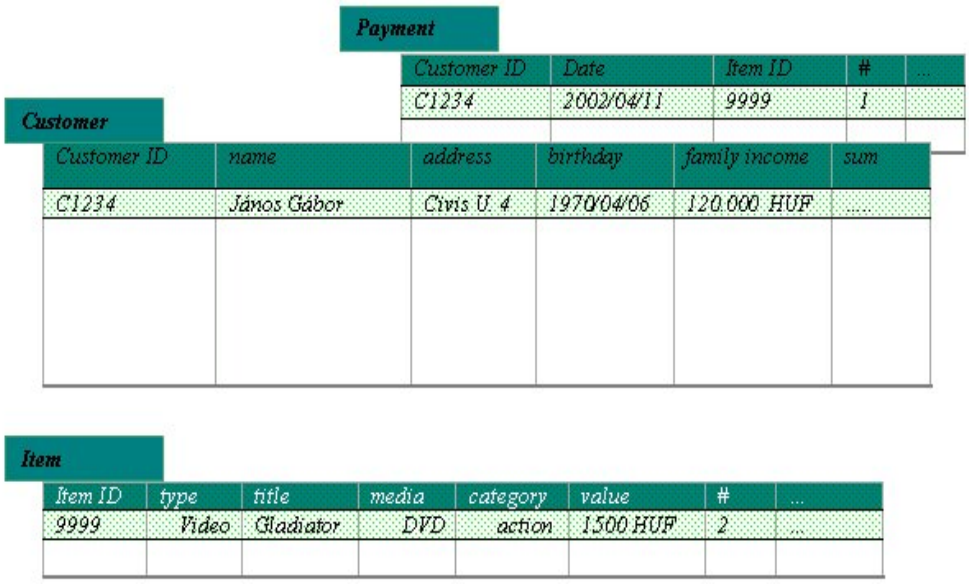

Figure 6.1 Fragments of some relations from relational databases for OurVideoStore (in DBMiner system)

The most commonly used query language for relational database is SQL, which allows retrieval and manipulation of the data stored in the tables, as well as the calculation of aggregate functions such as average, sum, min, max and count. For instance, an SQL query to select the videos grouped by category would be:

SELECT count(*) FROM Items WHERE type=video GROUP BY category

Data mining query algorithms using relational databases can be more versatile than data mining algorithms specifically written for flat files, since they can take advantage of the structure inherent to relational databases. While *data mining query* can benefit from SQL for data selection, transformation and consolidation, it goes beyond what SQL could provide, such as predicting, comparing, detecting deviations, etc [9]. Data mining query such as DMQL presented in [12] used relational databases as database sources in DBMiner system.

- *Data Warehouses*: A data warehouse as a storehouse is a repository of data collected from multiple data sources (often heterogeneous) and is intended to be used as a whole under the same unified schema. A data warehouse gives the option

to the DMQ to analyze data from different sources under the same roof. Let us suppose that OurVideoStore becomes a franchise in Hungary. Many video stores belonging to OurVideoStore Company may have different databases and different structures. If the executive of the company wants to access the data from all stores for strategic decision-making, future direction, marketing, etc., it would be more appropriate to store all the data in one site with a homogeneous structure that allows interactive analysis. In other words, data from the different stores would be loaded, cleaned, transformed, and integrated together [10]. To facilitate decision-making and multi-dimensional views, data warehouses are usually modeled by a multi-dimensional data structure to make data accessible for *data mining query*. Figure 6.2, shows an example of a three dimensional subset of a data cube structure used for OurVideoStore data warehouse.

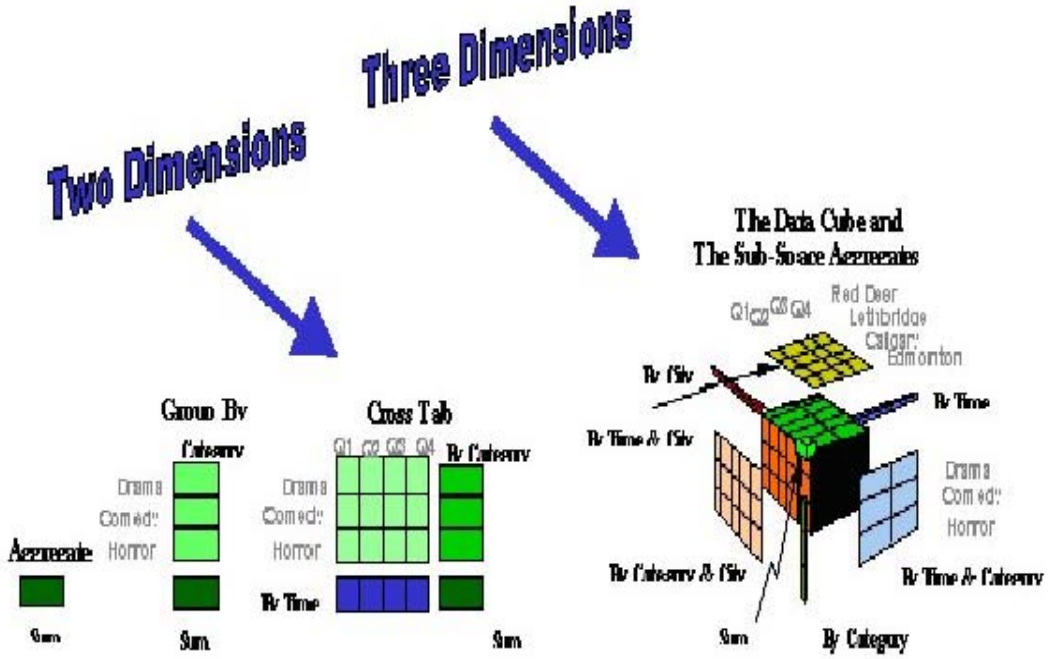

Figure 6.2. A multi dimensional data cube structure commonly used in data for data wherehousing (in DBMiner system)

The figure shows summarized rentals grouped by film categories, then a cross table of summarized rentals by film categories and time (in quarters). The data cube gives the summarized rentals along three dimensions: category, time, and city. A cube contains cells that store values of some aggregate measures (in this case rental counts), and special cells that store summations along dimensions. Each dimension of the data cube contains a hierarchy of values for one attribute. Because of their structure, the pre-computed summarized data they contain and the hierarchical attribute values of their dimensions, data cubes are well suited for fast interactive querying and analysis of data at different conceptual levels, known as *On-Line Analytical Processing (OLAP)*. OLAP operations allow DMQ to retrieve data at different levels of abstraction, such as drill-down, roll-up, slice, dice, etc. DMQL in [12] can also interact multi-dimensional databases in different OLAP stages. Figure

6.3, illustrates the drill-down (on the time dimension) and roll-up (on the location dimension) operations.

- *Transaction Databases*: A transaction database is a set of records representing transactions, each with a time stamp, an identifier, and a set of items. Associated with the transaction files could also be descriptive data for the items. For example, in the case of the video store, the rentals table such as shown in figure 6.4, represents the transaction database. Each record is a rental contract with a customer identifier, a date, and the list of items rented (i.e. video tapes, games, VCR, etc.). Since relational databases do not allow nested tables (i.e. a set as attribute value), transactions are usually stored in flat files or stored in two normalized transaction tables, one for the transactions, and one for the transaction items. One typical data mining analysis on such data is the so-called market basket analysis or association rules in which associations between items occurring together or in sequence are studied.

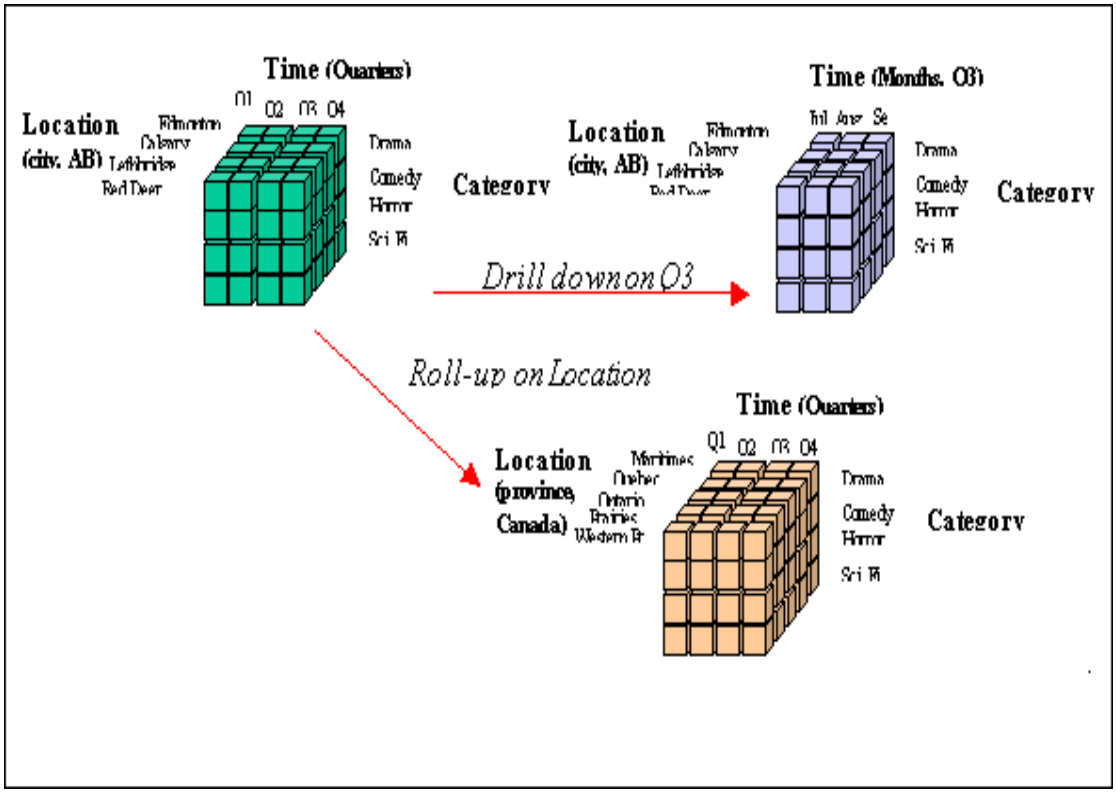

Figure 6.3. Summarized data from OurVideoStore before and after drill-down and roll-up operations (in DBMiner system)

Transaction ID	Date	Time	Customer ID	ItemList
T12345	2002/04/06	13:30	C1234	{T2,T6,T10,T45,..}

Figure 6.4. Fragment of a transaction database for the rentals at OurVideoStore (in DBMiner system)

- *Multimedia Databases*: Multimedia databases include video, images, and audio and text media. They can be stored in extended object-relational or object-oriented databases, or simply on a file system. Multimedia is characterized by its high dimensionality, which makes *data mining query* even more challenging. *Data mining query* from multimedia repositories may require computer vision, computer graphics, image interpretation, and natural language processing methodologies which can be interacted parallel with DMQ such as DMQL in [12] by using MultiMediaMiner system described in [13]. We also may use *Visual Query Languages VQL* to retrieve multimedia databases.

- *Spatial Databases*: Spatial databases are databases that, in addition to usual data, store geographical information like maps, and global or regional positioning. Such spatial databases present new challenges to *data mining query* based on data mining algorithms. *Geo-mining query language (GMQL)* is one of the DMQL's that is used to retrieve data from a spatial database and extract all the new information [7].

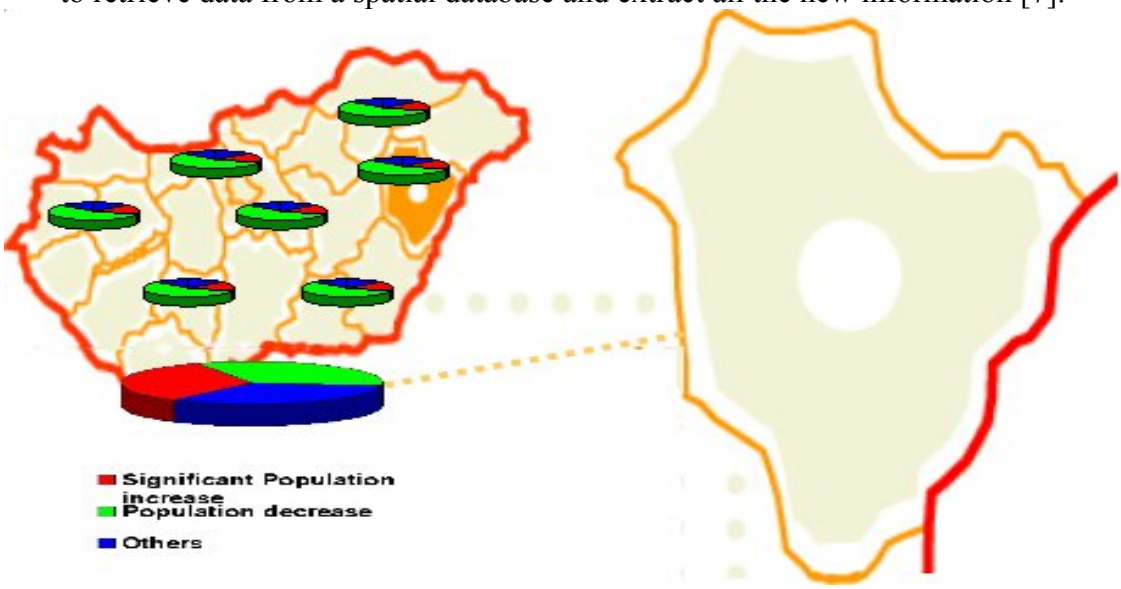

Figure 6.5 Visualization of Spatial OLAP (from a Geo- mining system)

- *Time-Series Databases*: Time-series databases contain time related data such as stock market data or logged activities. These databases usually have a continuous

flow of new data coming in, which sometimes causes the need for a challenging real time analysis. DMQ in such databases commonly includes the study of trends and correlations between evolutions of different variables, as well as the prediction of trends and movements of the variables in time. Figure 6.6, shows some examples of time-series data. In this type of databases the *data mining queries* will deal with dynamic databases to discover and present the expected behavior of that databases in a given time interval.

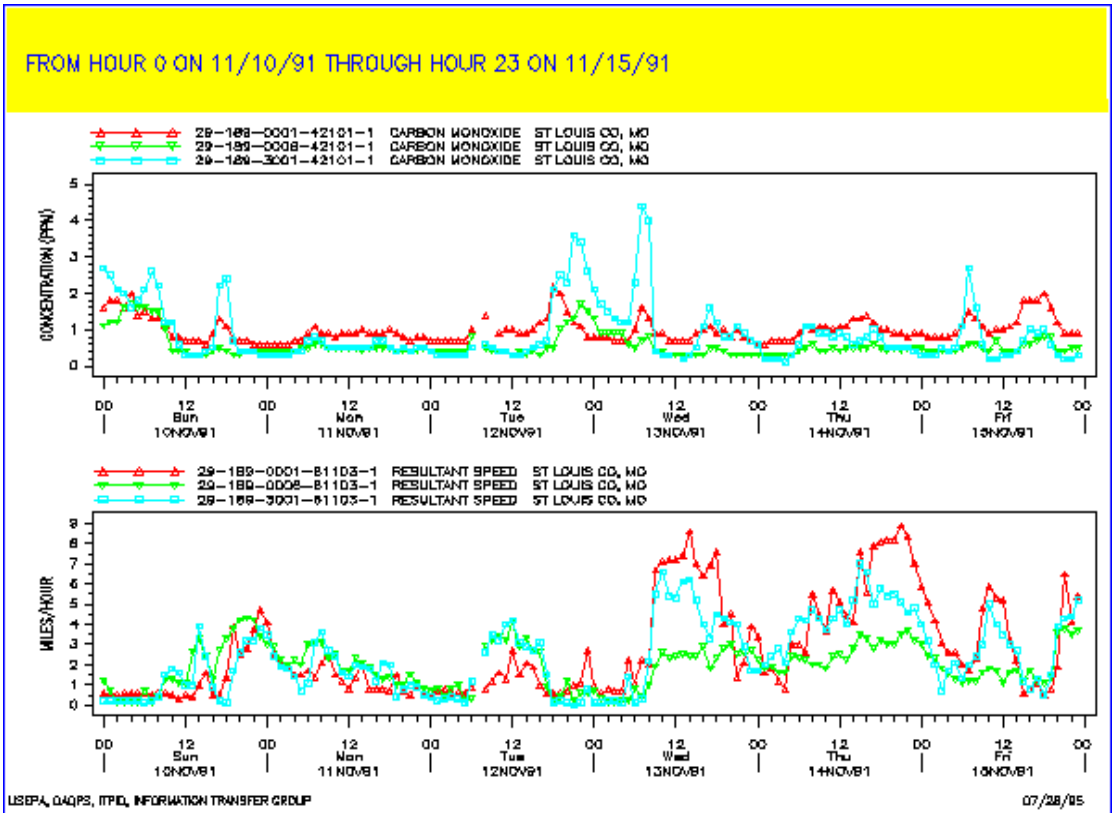

Figure 6.6 Example of Time-Series Data retrieved by DM Quires

- *World Wide Web*: The World Wide Web is the most heterogeneous and dynamic repository available. A very large number of authors and publishers are continuously contributing to its growth and metamorphosis, and a massive number of users are accessing its resources daily. Data in the World Wide Web is organized in inter-connected documents. These documents can be text, audio, video, raw data, and even applications. Conceptually, the World Wide Web is comprised of three major components: The content of the Web, which encompasses documents available; the structure of the Web, which covers the hyperlinks and the relationships between documents; and the usage of the web, describing how and then the resources are accessed. A fourth dimension can be added relating the dynamic nature or evolution of the documents. Data mining in the World Wide Web, or Web mining, tries to address all these issues and is often divided into web content mining, web structure mining, and web usage mining. An example to a web-mining query language could be WEBMINER [12].

6.4 The discovered patterns by data mining query

The kinds of patterns that can be discovered depend upon the data mining tasks employed. By and large, there are two types of data mining tasks: descriptive data mining tasks that describe the general properties of the existing data, and predictive data mining tasks that attempt to do predictions based on inference on available data.

The data mining functionalities and the variety of knowledge they discover are briefly presented in the following list:

- *Characterization*: Data characterization is a summarization of general features of objects in a target class, and produces what is called characteristic rules. The data relevant to a user-specified class are normally retrieved by a database query and run through a summarization module to extract the essence of the data at different levels of abstractions. For example, one may want to characterize the OurVideoStore customers who regularly rent more than 30 movies a year. With concept hierarchies on the attributes describing the target class, the attribute-oriented induction method can be used, for example, to carry out data summarization. Note that with a data cube containing summarization of data, simple OLAP operations fit the purpose of data characterization.

- *Discrimination*: Data discrimination produces what are called discriminant rules and is basically the comparison of the general features of objects between two classes referred to as the target class and the contrasting class. For example, one may want to compare the general characteristics of the customers who rented more than 30 movies in the last year with those whose rental account is lower than 5. The techniques used for data discrimination are very similar to the techniques used for data characterization with the exception that data discrimination results include comparative measures.

- *Association analysis*: Association analysis is the discovery of what are commonly called association rules. It studies the frequency of items occurring together in transactional databases, and based on a threshold called support, identifies the frequent item sets. Another threshold, confidence, which is the conditional probability than an item appears in a transaction when another item appears, is used to pinpoint association rules. Association analysis is commonly used for market basket analysis. For example, it could be useful for the OurVideoStore manager to know what movies are often rented together or if there is a relationship between renting a certain type of movies and buying popcorn or pop. The discovered association rules are of the form: $P \rightarrow Q$ described in Chapter 5, where P and Q are conjunctions of attribute value-pairs, and s (*for support*) is the probability that P and Q appear together in a transaction and c (for confidence) is the conditional probability that Q appears in a transaction when P is present. For example, the hypothetic association rule: *RentType(X, "game") AND Age(X, "13-19")* $\rightarrow$ *Buys(X,"pop") [s=2% ,c=55%]* would indicate that *2%* of the transactions considered are of customers aged between *13* and *19* who are renting a game and buying a pop, and that there is a certainty of *55%* that teenage customers who rent a game also buy pop.

- *Classification*: Classification analysis is the organization of data in given classes. Also known as supervised classification, the classification uses given class labels to order the objects in the data collection. Classification approaches normally use a training set where all objects are already associated with known class labels. The classification algorithm learns from the training set and builds a model. The model is used to classify new objects. For example, after starting a credit policy, the OurVideoStore managers could analyze the customers' behaviors vis-à-vis their credit, and label accordingly the customers who received credits with three possible labels *"safe", "risky" and "very risky"*. The classification analysis would generate a model that could be used to either accept or reject credit requests in the future.

- *Prediction*: Prediction has attracted considerable attention given the potential implications of successful forecasting in a business context. There are two major types of predictions: one can either try to predict some unavailable data values or pending trends, or predict a class label for some data. The latter is tied to classification. Once a classification model is built based on a training set, the class label of an object can be foreseen based on the attribute values of the object and the attribute values of the classes. Prediction is however more often referred to the forecast of missing numerical values, or increase / decrease trends in time related data. The major idea is to use a large number of past values to consider probable future values.

- *Clustering*: Similar to classification, clustering is the organization of data in classes. However, unlike classification, in clustering, class labels are unknown and it is up to the clustering algorithm to discover acceptable classes. Clustering is also called unsupervised classification, because the classification is not dictated by given class labels. There are many clustering approaches all based on the principle of maximizing the similarity between objects in a same class (intra-class similarity) and minimizing the similarity between objects of different classes (inter-class similarity).

- *Outlier analysis*: Outliers are data elements that cannot be grouped in a given class or cluster. Also known as exceptions or surprises, they are often very important to identify. While outliers can be considered noise and discarded in some applications, they can reveal important knowledge in other domains, and thus can be very significant and their analysis valuable.

- *Evolution and deviation analysis*: Evolution and deviation analysis pertain to the study of time related data that changes in time. Evolution analysis models evolutionary trends in data, which consent to characterizing, comparing, classifying or clustering of time related data. Deviation analysis, on the other hand, considers differences between measured values and expected values, and attempts to find the cause of the deviations from the anticipated values.

6.4.1 Examples of different data mining query languages

We will consider an university database schema as a universal example for some important DMQL functions as following:

Student (name, sno, status, major, gpa, birth date, birth place, address)
Course (cno, title, department)
Grading (sno, cno, instructor, semester, grade)

DMQL examples are presented below in different data mining languages in each function as follows. Notice that the statements like "*use database university database"* is omitted in the example queries.

(Q1) characteristic rules: - By the use of DMQL described in [12]. The query will be requested to find the general characteristics of the graduate students in computing science in relevance to attributes *gpa, birth_place* and *address*, for the students born in *"Hungary"*.

(A_Q1): find characteristic rule
related to *gpa, birth_place, address, count(*)%*
from *student*
where *status = "graduate" and major = "cs"*
and *birth place = "Hungary"*
with *noise threshold = 0.05*

This DMQ will first retrieve data from the database using a transformed SQL query, where the high level constants "Hungary" and "graduate" are transformed into low level primitive concepts in the database according to the provided (default) concept hierarchy for each attribute. The algorithm for finding characteristic rules [12] is then executed with the data generalized to high level for manipulation and presentation. The set of generalized data grouped according to the high level concept values of the attributes *gpa, birth_place* and *address* are presented, associated with the corresponding *count(*)%* (i.e., the count of tuples in the corresponding group in proportion to the total number of tuples). The noise threshold *0.05* means that a generalized tuple taking less than 5% of the total count will not be included in the final result.

(Q2) Discriminant rules: - Generating queries using DBMiner data mining query system issued in [14]. The query will try to find the discriminant features to compare graduate students versus undergraduate students in computing science in relevance to attributes *gpa, birth_ place* and *address*, for the students born in *"Hungary"*.

(A_Q2) : find discriminant rule
for *cs_grads* ***with status*** *= "graduate"*
in contrast to *cs_undergrads*
with status *= "undergraduate"*
related to *gpa, birth_place, address, count(*)%*
from *student*
where *major = "cs" and birth_place = "Hungary"'*

This DMQ will first retrieve data into two classes, *"cs_grads"* and *"cs_undergrads"*, using a transformed SQL query which maps the high level constants in (A_Q2) into low level ones. The algorithm for finding discriminant rules [14] is then executed for data mining and result manipulation.

(Q3) Classification rules: DMQL use to classify students according to their gpa's and find their classification rules for those majoring in computing science and born in *"Hungary"*, with the attributes *birth_place* and *address* in consideration.

(A_Q3) : find classification rules for *cs_students*
according to *gpa*
related to *birth_place, address*
from *student*
where *major = "cs"* ***and*** *birth_place = " Hungary"*

This query will first collect the relevant set of data, and then execute some data classification algorithm, such as [12] to classify students according to their gpa's and present each class and its associated characteristics.

(Q4) Association rules: Querying the DBMiner data mining query system to find strong association relationships for those students majoring in computing science and born in *"Hungary"*,in relevance to the attributes *gpa, birth_place* and *address*.

(A_Q4): find association rules
related to *gpa, birth_place, address*
from *student*
where *major = ``cs"* ***and*** *birth_place = "Hungary"*
with *support threshold = 0.05*
with *confidence threshold = 0.7*

This query will first collect the relevant set of data and then execute an association mining algorithm, such as [12] or [14], to find a set of interesting association rules. The support and confidence thresholds are specified (otherwise using default values) for mining strong rules.

It is common that users do not have a clear idea of the kind of patterns they can discover or need to discover from the data at hand. It is therefore important to have a versatile and inclusive *data mining query systems* that allows the discovery of different kinds of knowledge and at different levels of abstraction. This also makes interactivity an important attribute of a *data mining query systems*.

6.5 Usefulness of the discovered patterns

DMQ allows the discovery of knowledge potentially useful and unknown. Whether the knowledge discovered is new, useful or interesting, is very subjective and depends upon the application and the user. It is certain that *data mining query* can generate, or discover, a very large number of patterns or rules. In some cases, the number of rules can reach the millions. One can even think of a meta-mining phase to mine the oversized data mining results. To reduce the number of patterns or rules discovered that have a high probability to be non-interesting, one has to put a measurement on the patterns. However, this raises the problem of completeness. The user would want to

discover all rules or patterns, but only those that are interesting. The measurement of how interesting a discovery is, often called interestingness, can be based on quantifiable objective elements such as validity of the patterns when tested on new data with some degree of certainty, or on some subjective depictions such as understandability of the patterns, novelty of the patterns, or usefulness. Discovered patterns can also be found interesting if they confirm or validate a hypothesis sought to be confirmed or unexpectedly contradict a common belief. This brings the issue of describing what is interesting to discover, such as meta-rule guided discovery that describes forms of rules before the discovery process, and interestingness refinement languages that interactively query the results for interesting patterns after the discovery phase. Typically, measurements for interestingness are based on thresholds set by the user. These thresholds define the completeness of the patterns discovered.

Identifying and measuring the interestingness of patterns and rules discovered, or to be discovered is essential for the evaluation of the mined knowledge and the KDD process as a whole. While some concrete measurements exist, assessing the interestingness of discovered knowledge's by *data mining query* is still an important research issue [10].

6.6 Categorization of data mining queries

There are many DMQL available or being developed. Some are specialized systems dedicated to a given data source or are confined to limited data mining functionalities, other are more versatile and comprehensive. DMQ can be categorized according to various criteria. Among other classification are the following [12]:

- *Classification according to the type of data source mined*: this classification categorizes DMQ according to the type of data handled such as spatial data, multimedia data, time-series data, text data, World Wide Web, etc.

- *Classification according to the data model drawn on*: this classification categorizes DMQ based on the data model involved such as relational database, object-oriented database, data warehouse, transactional, etc.

- *Classification according to the kind of knowledge discovered:* this classification categorizes DMQ based on the kind of knowledge discovered or data mining functionalities, such as characterization, discrimination, association, classification, clustering, etc. Some systems tend to be comprehensive systems offering several data mining functionalities together.

- *Classification according to mining techniques used:* DMQ employ and provide different techniques. This classification categorizes DMQ according to the data analysis approach used such as machine learning, neural networks, genetic algorithms, statistics, visualization, database-oriented or data warehouse-oriented, etc. The classification can also take into account the degree of user interaction involved in the data mining process such as query-driven systems, interactive exploratory systems, or autonomous systems. A comprehensive system would

provide a wide variety of data mining techniques to fit different situations and options, and offer different degrees of user interaction.

6.7 Issues in data mining query

DMQ algorithms embody techniques that have sometimes existed for many years, but have only lately been applied as reliable and scalable tools that repeatedly outperform older classical statistical methods. While DMQ are still in its infancy, it is becoming a trend and ubiquitous. Before DMQ system develops into a conventional, mature and trusted discipline, many still pending issues have to be addressed. Some of these issues are addressed below. Note that these issues are not exclusive and are not ordered in any way.

- *Security and social issues:* Security is an important issue with any data collection that is shared and/or is intended to be used for strategic decision-making. In addition, when data is collected for customer profiling, user behaviour understanding, correlating personal data with other information, etc., large amounts of sensitive and private information about individuals or companies is gathered and stored. This becomes controversial given the confidential nature of some of this data and the potential illegal access to the information. Moreover, DMQ could disclose new implicit knowledge about individuals or groups that could be against privacy policies, especially if there is potential dissemination of discovered information. Another issue that arises from this concern is the appropriate use of data mining. Due to the value of data, databases of all sorts of content are regularly sold, and because of the competitive advantage that can be attained from implicit knowledge discovered, some important information could be withheld, while other information could be widely distributed and used without control.

- *User interface issues:* The knowledge discovered by DMQ is useful as long as it is interesting, and above all understandable by the user. Good data visualization eases the interpretation of DMQ results, as well as helps users had better understand their needs. Many data exploratory analysis tasks are significantly facilitated by the ability to see data in an appropriate visual presentation. There are many visualization ideas and proposals for effective data graphical presentation. However, there is still much research to accomplish in order to obtain good visualization tools for large datasets that could be used to display and manipulate mined knowledge. The major issues related to user interfaces and visualization is "*screen real-estate*", information rendering, and interaction. Interactivity with the data and data mining results is crucial since it provides means for the user to focus and refine the mining tasks, as well as to picture the discovered knowledge from different angles and at different conceptual levels.

- *Mining methodology issues:* These issues pertain to the DMQ approaches applied and their limitations. Topics such as versatility of the mining approaches, the diversity of data available, the dimensionality of the domain, the broad analysis needs (when known), the assessment of the knowledge discovered, the exploitation of background knowledge and metadata, the control and handling of noise in data, etc. are all examples that can dictate mining methodology choices. For instance, it is often desirable to have different data mining methods available since different

approaches may perform differently depending upon the data at hand. Moreover, different approaches may suit and solve user's needs differently.

Most algorithms assume the data noise-free. This is of course a strong assumption. Most datasets contain exceptions, invalid or incomplete information, etc., which may complicate, if not obscure, the analysis process and in many cases compromise the accuracy of the results. Consequently, data preprocessing (data cleaning and transformation) becomes vital. It is often seen as lost time, but data cleaning, as time-consuming and frustrating as it may be, is one of the most important phases in the knowledge discovery process. Data mining techniques should be able to handle noise in data or incomplete information.

More than the size of data, the size of the search space is even more decisive for data mining techniques. The size of the search space is often depending upon the number of dimensions in the domain space. The search space usually grows exponentially when the number of dimensions increases. This is known as the curse of dimensionality. This "*curse*" affects so badly the performance of some DMQ approaches that it is becoming one of the most urgent issues to solve.

- *Performance issues*: Many artificial intelligence and statistical methods exist for data analysis and interpretation. However, these methods were often not designed for the very large data sets DM is dealing with today. Terabyte sizes are common. This raises the issues of scalability and efficiency of the data mining methods when processing considerably large data. Algorithms with exponential and even medium-order polynomial complexity cannot be of practical use for DMQ. Linear algorithms are usually the norm. In same theme, sampling can be used for mining instead of the whole dataset. However, concerns such as completeness and choice of samples may arise. Other topics in the issue of performance are incremental updating, and parallel programming. There is no doubt that parallelism can help solve the size problem if the dataset can be subdivided and the results can be merged later. Incremental updating is important for merging results from parallel mining, or updating DMQ results when new data becomes available without having to re-analyze the complete dataset.

- *Data source issues*: There are many issues related to the data sources, some are practical such as the diversity of data types, while others are philosophical like the data glut problem. We certainly have an excess of data since we already have more data than we can handle and we are still collecting data at an even higher rate. If the spread of database management systems has helped increase the gathering of information, the advent of DMQ are certainly encouraging more data harvesting. The current practice is to collect as much data as possible now and process it, or try to process it, later. The concern is whether we are collecting the right data at the appropriate amount, whether we know what we want to do with it, and whether we distinguish between what data is important and what data is insignificant. Regarding the practical issues related to data sources, there is the subject of heterogeneous databases and the focus on diverse complex data types. We are storing different types of data in a variety of repositories. It is difficult to expect a DMQ to effectively and efficiently achieve good mining results on all kinds of data and sources. Different kinds of data and sources may require distinct algorithms and methodologies. Currently, there is a focus on relational databases and data

warehouses, but other approaches need to be pioneered for other specific complex data types. A versatile DMQ, for all sorts of data, may not be realistic. Moreover, the proliferation of heterogeneous data sources, at structural and semantic levels, poses important challenges not only to the database community but also to the data mining and KDD communities.

We cannot specify a concrete DMQL for all types of databases, which support all common DM techniques. Although there are many approaches to DMQ, but only six common and essential elements qualify each as a knowledge discovery technique to each DMQ. The following are basic features that all DMQ share (adapted from [6] and [7]):

- All approaches deal with large amounts of data.
- Efficiency is required due to volume of data.
- Accuracy is an essential element.
- All require the use of a high-level language.
- All approaches use some form of automated learning.
- All produce some interesting results.

Large amounts of data are required to provide sufficient information to derive additional knowledge. Since large amounts of data are required, processing efficiency is essential. Accuracy is required to assure that discovered knowledge is valid. The results should be presented in a manner that is understandable by humans. One of the major premises of DMQ is that the knowledge is discovered using intelligent learning techniques that sift through the data in an automated process. For this technique to be considered useful in terms of knowledge discovery the discovered knowledge must be interesting; that is, it must have potential value to the user. DMQ provides the capability to discover new and meaningful information by using existing data. KDD quickly exceeds the human capacity to analyze large data sets. The amount of data that requires processing and analysis in a large database exceeds human capabilities, and the difficulty of accurately transforming raw data into knowledge surpasses the limits of traditional databases. Therefore, the full utilization of stored data depends on the use of knowledge discovery techniques. The usefulness of future applications of DMQ is far-reaching. DMQ may be used as a means of information retrieval, in the same manner that intelligent agents perform information retrieval on the web. New patterns or trends in data may be discovered using these techniques. DMQ may also be used as a basis for the intelligent interfaces of tomorrow, by adding a knowledge discovery component to a database engine or by integrating DMQ with spreadsheets and visualizations.

6.8 Data mining query environments

DMQ environment with good performance discovered tasks such as *classification; summarization; dependency analysis; visualization; prediction; class comparison* has the following distinct features [8]:

- It incorporates several interesting DMQ techniques, including attribute-oriented induction; progressive deepening for mining multiple-level rules and meta-rule

guided knowledge mining, etc., and implements a wide spectrum of *data mining functions including generalization, characterization, association, classification, and prediction.*

- It performs interactive DMQ at multiple concept levels on any user-specified set of data in a database using an SQL-like DMQL, or a graphical user interface. Users may interactively set and adjust various thresholds, control a DM process, perform roll-up or drill-down at multiple concept levels, and generate different forms of outputs, including generalized relations, generalized feature tables, multiple forms of generalized rules, visual presentation of rules, charts, curves, etc**.**

- Efficient implementation techniques have been explored using different data structures, including generalized relations and multiple-dimensional data cubes, and being integrated with relational database techniques. The data mining process may utilize user- or expert-defined set-grouping or schema-level concept hierarchies which can be specified flexibly, adjusted dynamically based on data distribution, and generated automatically for numerical attributes.

- Both UNIX and PC (Windows/NT) versions of the system adopt client/server architecture. The latter communicates with various commercial database systems for DMQ using the ODBC technology.

Part III

Knowledge Discovery Query Language Techniques

Chapter 7

Knowledge Discovery Query Language (KDQL)

7.1 Abstract

KDD and DM represent very important tools for processing and analyzing data in large databases such as relational databases. The combination of powerful DM techniques and sophisticated resources of different database systems brings results that are even more considerable. These emerging tools and techniques require a powerful DMQL serving as an interface between the applications and the DM tools [12]. These requirements motivate us to design a KDD query language for mining association rules in the databases (i.e. relational database). This makes it possible to visualize the discovered results in different charts (i.e. 2D and 3D). We give an overview of this language in this Chapter, and its syntax and semantics will be present in Chapter 9 and some examples of practical use will be shown as well. We will call this query language Knowledge Discovery Query Language (KDQL) such a query was not implemented namely yet.

7.2 Introduction

KDD is a general process of useful knowledge discovery from databases. This process involves data pre-processing, DM itself and then the interpretation of minded patterns. DM is only one part of KDD process. It analyses preprocessed data and produces information patterns which are then interpreted, and convenient ones are considered as knowledge. Development of modern database management systems (DBMS) has advanced considerably. It is naturally then to investigate knowledge discovered in RDB [15] offering richer structure and semantics, which can be employed in the DM process.

In the present time there is many practical applications employing or based on the KDD technology. These applications require introducing certain standards. This standard is called DMQL it was described in details in Chapter 7. DMQL would offer standard interface between application and the KDD system. Design of such language for DM to interact databases such as relational databases is described in [12]. The language will use data visualization to represent the result in a simple form for the user.

Joining KDD technology and data visualization with conjunction of the request of creating query language for DM leads us to develop a language tool that can handle two approaches in one session. Figure 7.1 shows the relation between database and visualization. Designing a query language that could deal with both approaches is a good challenge these days. This language will be called Knowledge Discovery Query

Language (KDQL); such a language has not been designed namely yet. KDQL would enable an application to get knowledge from databases, in certain standard simple way similar to getting stored data by the help of DBMS. KDQL is the result of a long searching and investigations in this area. It is described in this dissertation exactly in Chapters 7, 8, 9.

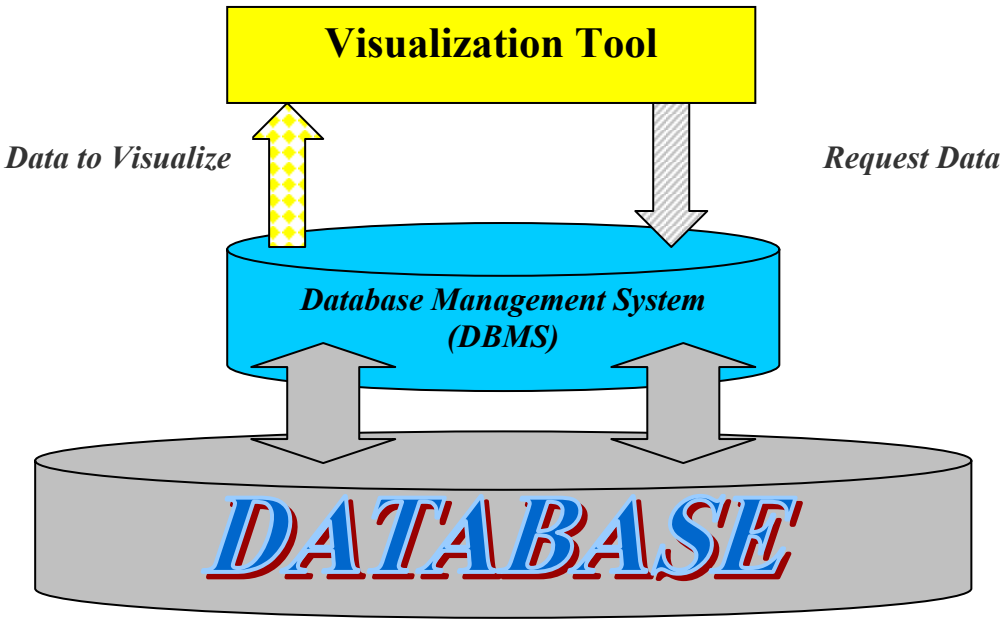

Figure 7.1 Database and visualization

7.3 Background of the KDQL

To consider the KDQL we should ask ourselves some questions such like: what is the difference between DM and a normal query environment? What can DM tool do that SQL cannot?

Firstly, it is important to realize that query languages and DMQL are complementary. A *data mining language (DML)* does not replace a query language, but it does give the use many additional possibilities. Suppose that we have a large file containing millions of records that describe your customers' purchases over the last ten years. This is a wealth of potentially useful knowledge in such a file, most of which can be found by firing normal queries at the database, such as following.

'Who bought which product on what date?'
'What is the average turnover in a certain sales region in July?'

There is, however, knowledge hidden in your database that is much harder to find using SQL. Examples would be the answer to questions such as *"What is an optimal segmentation of my clients?"* (That is, *"how do I find the most important different customer profiles?"*), or *"what are the most important trends in customer behavior?"* Of course, these questions could be answered using SQL. You could try to guess for yourself some defining criteria for customer profiles and search the database to see

whether they work or not. In a process of trial and error, one could gradually develop intuitions about what important distinguishing attributes are proceeding in such away, it could take days or months to find an optimal segmentation for a large database [75]. while a machine learning algorithm like a neural network or genetic algorithm could find the answer automatically in a much shorter time, sometimes even in minutes or a couple of hours. Once the data mining query DMQ has found segmentation, we can use the classical query environment again to query and analyze the profiles found.

One could say that if we know exactly what we are looking for, use SQL; but if we know only vaguely, what we are looking for, then we may turn to DMQ. Generally, there are far more occasions when the initial approach is vague than times when you know precisely what you are looking for. It is that has motivated the recent surge of interest in DM.

It is clear that KDD is not an activity that stands on its own: good foundation in terms of a data warehouse is necessary condition of its effective implementation. Noisy and incomplete data, and legal and privacy issues, constitute important problems [76]. One must pay attention to the process of data cleaning – remove duplication records, correct typographical errors in strings, add missing information, and so on. In KDD in an organization is to start a process of permanent refinement and detailing of data. The real aim should be ultimately to create a self-learning query.

7.4 Analyzing the pre discovered data by using traditional query tools

The first step in KDQL should always be a rough analysis of the data set using traditional query tools. Just by applying simple SQL request to a data set, we can obtain a wealth of information. However, before we can apply more advanced pattern analysis algorithms, we need to know some basic aspects and structures of the data set. With SQL we can uncover only shallow data (known data), which information that is easily accessible from the data set. Although yet we cannot find hidden data, for the most part 80% of the interesting information can be abstracted from a database using SQL.

	Average
Age	*46.9*
Income	*20.8*
Credit	*34.9*
Car owner	*0.59*
House owner	*0.59*
Car magazine	*0.329*
House magazine	*0.702*
Sports magazine	*0.447*
Music magazine	*0.146*
Comic magazine	*0.081*

Table 7.1 Average

The remaining 20% of hidden information requires more advanced techniques, and for large marketing driven organizations, this 20% can prove of vital importance. A good way to start is to extract some simple, statistical information from the data set, and averages are important example in this respect [76].

In our data set in table 7.1 we see that the average age is 46 years old, the average income 20, the average credit 34, and so on. It is interesting to look at the averages of the input fields: we see that 329 clients out of every 1000 subscribe to a car magazine, whereas only 81 out of 1000 subscribe to comic. These numbers are very important, because they give us a norm by which top judge the performance of pattern recognition and learning algorithms. Suppose that we want to predict how many clients will buy a car magazine. Now an algorithm that always predicts 'no car magazine' would be correct in 671 out of 1000 cases, which is about 70%. Any learning algorithm that clams to give some insight into the data set and do some real predicating has to improve on this. A trivial result that is obtained by an extremely simple method is called *naïve prediction*, and an algorithm that claims to learn anything must always do better than the naïve prediction (table 7.2) [75]. Here we can see also that it is more difficult to make predictions for the small group in our sample set. Since only 81 out of 1000 clients subscribe to comics, a learning algorithm that claims to predict which clients will subscribe to comics has to give prediction accuracy better than the 92% achieved by using the naïve predication. This will be difficult in most cases. Table 3 illustrates the averages per magazine.

Magazine	*A priori probability that client buys magazine*	*Naïve prediction Accuracy*
Car	*32.9%*	*67.1%*
House	*70.2%*	*70.2%*
Sports	*44.7%*	*55.3%*
Music	*14.6%*	*85.4%*
Comic	*8.1%*	*91.9%*

Table 7.2 Naïve predictions

It is interesting to see how these averages change when we focus on different magazines. For example, we see that the average age of a reader of a car magazine is 29, which is considerably lower than the average age of the clients – about 47. As was too expected, the average age of a comic's reader is the lowest. Other interesting piece of information is the number of multiple buyers in sample, and this is illustrated in Figure 7.2.

Magazine	Average				
	Age	Income	Credit	Car	House
Car	29.3	17.1	27.3	0.48	0.53
House	48.1	21.1	35.50	0.58	0.76
Sports	42.2	24.3	31.4	0.70	0.60
Music	24.6	12.8	24.6	0.30	0.45
Comic	21.4	25.5	26.3	0.62	0.60

Table 7.3 Result of applying a naïve prediction

Here we see that almost 40% of clients subscribe to only one magazine. However, it is interesting to note that 31% subscribe to two magazines, which indicates that there might be interesting patterns to discover between groups of multiple and single buyers and then to quantifying what is interesting and what is not?. Quite alarming, however, is the fact that almost 9% of clients in the sample subscribe to no magazine at all, which can only be the result of pollution in the database, and it is essential to investigate how this pollution has occurred and what can be done to prevent it in the future. This illustrates the developmental nature of DM: an ongoing process by which knowledge and understanding of data improves and depends all the time.

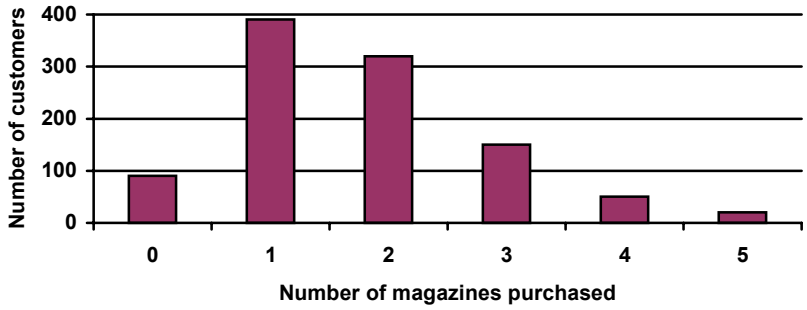

Figure 7.2 Overview of multiple subscriptions

We have seen some interesting patterns in the age attributes and we would like to concentrate on this to extract more information. In order to demonstrate this process, we will investigate the general age structure of our sample. We see that the ages are apart from the very young and very old people, almost equally spread over the sample (Figure 7.3).

Interesting differences occur when we analyze certain sub-groups. Readers of the car magazine cluster around the age class of 30 (Figure 7.3) while readers of the sports magazine are spread much more evenly over the population (Figure 7.5). SQL can yield detailed information on the structure of a data set and this information can be very useful for marketing or other purposes. We have to go through this phase before we can

turn our attention to more advanced learning algorithms. Remember, however, that we can never judge the performance of an advanced learning algorithm properly if we have no information concerning the *naïve* probabilities of what it is supposed to predict.

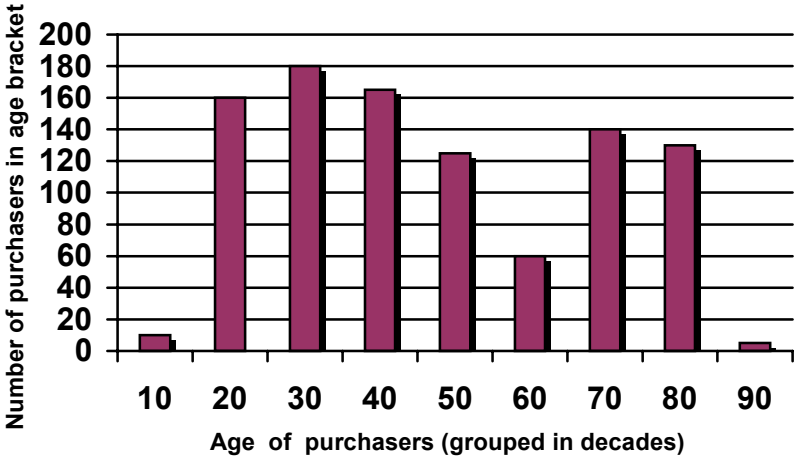

Figure 7.3 Age distribution of readers

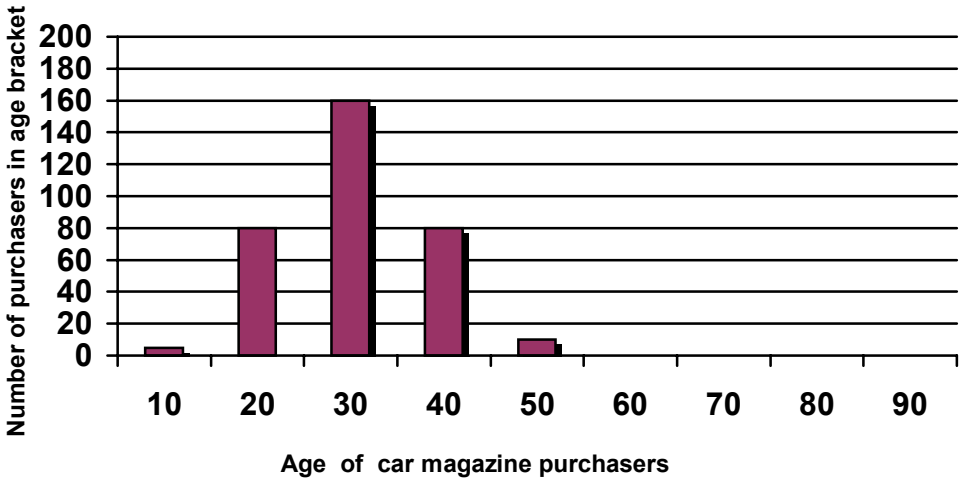

Figure 7.4 Age distribution of readers of the car magazine

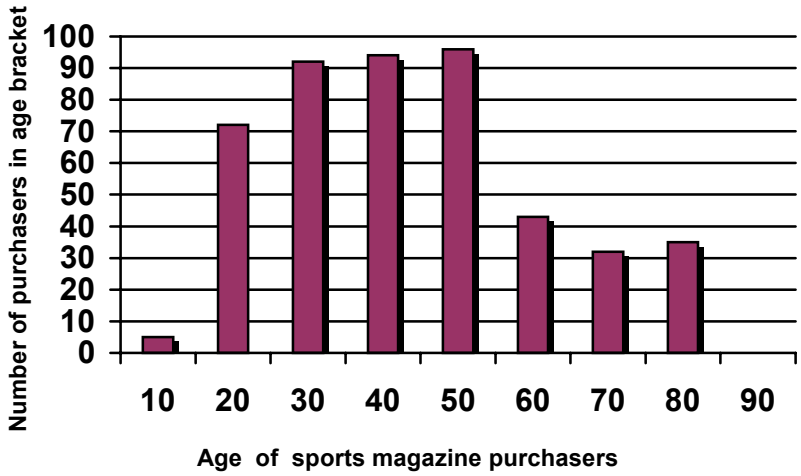

Figure 7.5 Age of sports magazine purchasers

7.5 Visualization techniques for DMQ

Visualization techniques are a very useful method of discovering patterns in data sets, and may be used at the beginning of a DM process to get rough felling of the quality of the data set and where patterns are to be found. Interesting possibilities are offered by the two and three dimensional tool kits, such as charts, which enable the user to explore charts such as pies, points and bar structures interactively. Such techniques are developing rapidly: advanced graphical techniques in virtual reality enable people to wander through artificial data spaces, while historic development of data seta can be displayed as kind of animated movie. For most users, however, these advanced features are not accessible, and they have to rely on simple, graphical display techniques that are contained in the query tool or DM tools they are using. These simple methods can provides us with a wealth of information. An elementary technique that can be of great value is so-called *scatter diagram*; in this technique, information on two attributes is displayed in *Cartesian space*. Scatter diagrams can be used to identify interesting sub-set of the data sets so that we can focus on the rest of the DM process it could be possible as a future work. There is a whole field of research dedicated to the search for interesting projections of data sets – this is called *projection pursuit*. Now we can compare how simple visualization techniques can help give a felling for the structure of a data set. A much better way to explore a data set is through an interactive three dimensional environment. In simple word visualization techniques, helps visualize the data so that DM is facilitated. As DM techniques mature, it will be important to integrate them with visualization techniques. Figure 7.6 illustrates interactive data mining. Here, the database management system, visualization tool, and machine-learning tool all interact with each other for DM.

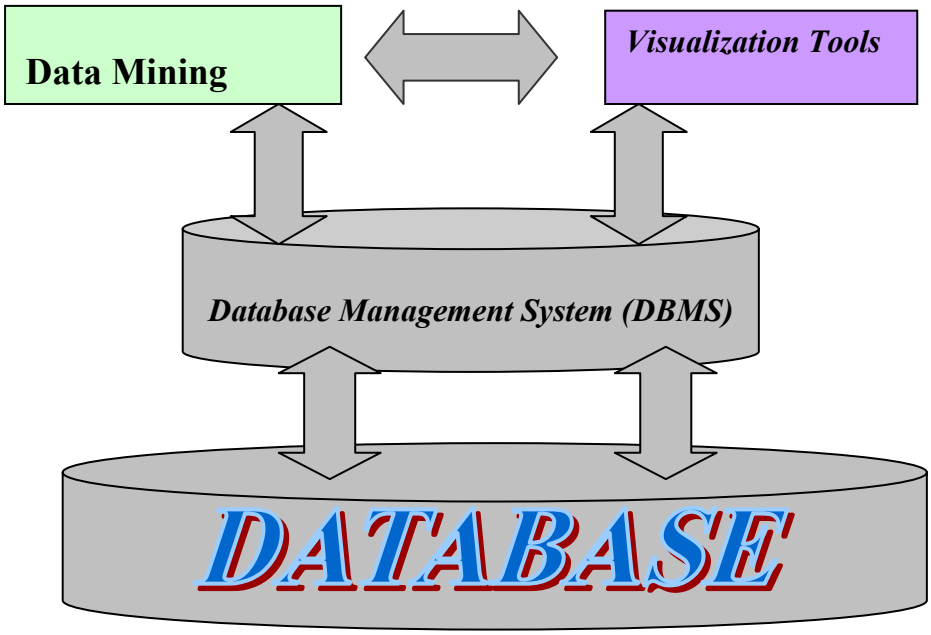

Figure 7.6 Interactive DM

7.6 Integrating DMQ with visualization

We re-exam some issues for integrating DMQ with visualization. We point out four possible approaches that were used in visualizing DMQ. *One* is to use visualization techniques to present the results that were obtained from mining the data in databases. These results may be in the form of clusters or they could specify correlation between the data in the database. The *second approach* applies DM techniques to visualization; this approach leads the assumption to apply DMQ to data in the visual form. Therefore, rather than applying the DMQ to large and complex databases, one captures some of the essential semantics visually, and then applies the DMQ. The *third approach* is to use visualization techniques to complements the DM techniques. For example, one may use DM techniques to obtain correlations between data or detect patterns. However, visualization techniques may still be needed to obtain a better understanding of the data in database. The *fourth approach* uses visualization techniques to steer the mining process. We can conclude to that visualization tools can be used to display visually the responses from the database system directly so that the visual displays can be used by the DMQ. On the other hand, the visualization tools can be used to visualize the results of the DMQ directly.

We will consider the third type which will detect all the patterns in the databases. Detecting patterns from databases require investigating from the databases to find out some existing association rules. Theses rules will be represented in visual form to help the user to make some decision.

7.7 User interface aspect

As in any kind of system, having a good user interface is critical to mining. Note that some of the early database management systems had very primitive user interfaces. Therefore, users had to spend a great deal of time writing SQL statements and application programs. After much work, current database systems have excellent user interface tools. The include tools for generating queries, application programs, as well as reports. Various multi-modal interfaces are also being provided for database management.

User interface support for current *data mining systems* (DMS) is fairly primitive. Visualization tools are being developed to help with DM, but tools for generating queries, application programs to carry out DM, and reports are not sophisticated. To make DM a success we need better user interface tools [76]. Computer scientists and technologists are not the only one who should be involved in developing such tools. Interactions between technologists, scientists, psychologists, and human computer specialists are necessary to develop better tools. Figure 7.7 illustrate an example user interface for DMQ. The interface has buttons not only for generating DMQ, applications for DM, and reports, but also for selecting the outcomes desired, approaches to be followed, and the techniques to utilize.

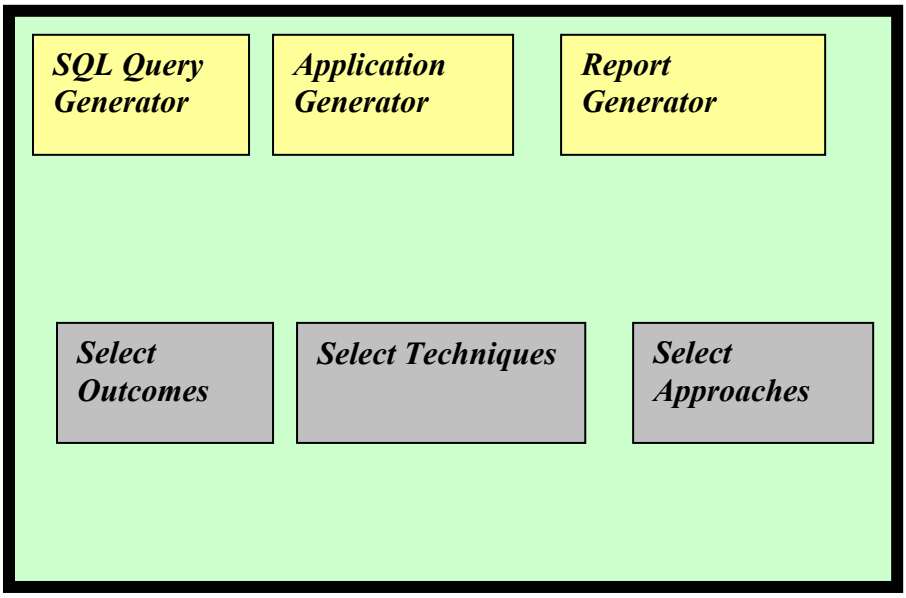

Figure 7.7 Example user interface for DM

Chapter 8

I-extended Databases

8.1 Motivation

One of the most challenging problems in data manipulation in the future is to be able to efficiently handle very large databases but also multiple induced properties or generalizations in that data. Popular examples of useful properties are association rules, and inclusion functional dependencies. Our view of a possible approach for this task is to specify and query *i-extended databases*, which are databases that in addition to data also contain exceedingly defined generalizations about the data. *I-extended database* is a database that has similar properties then an *inductive database* [43] in certain scene and it could be also an alternative to inductive database. We formalize this concept and show how it can be used throughout the whole process of DM due to the closure property of the framework. We show that simple query languages can be defined using normal database terminology. We demonstrate the use of this framework to model typical DM processes. It is then possible to perform various tasks on these descriptions like, e.g., optimizing the selection of interesting properties or comparing two processes.

8.2 Introduction

DM or KDD sets new challenges to database technology: new concepts and methods are needed for general purpose query languages [9]. A possible approach is to formulate a DM 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* described in [80].

Discovering knowledge from data, the so called KDD process, contains several steps: understanding the domain, preparing the data set, discovering patterns (i.e., computing a theory), post-processing of discovered patterns, and putting the results into use. This is a complex interactive and iterative process for which many related theories have to be computed: different selection predicates but also different classes of patterns must be used [76].

For KDD, we need a query language that not only enables the user to select subsets of the data, but also to specify DM tasks and select patterns from the corresponding theories. Our special interest is in the combined pattern discovery and post-processing steps via a querying approach. For this purpose, a closure property of the query language is desirable: the result of a KDD query should be an object of a similar type than its arguments. Furthermore, the user must also be able to cross the boundary between data and patterns, e.g., when exceptions to a pattern are to be analyzed. This gives rise to the concept of i-extended database, i.e., databases that contain extended generalizations about the data, in addition to the usual data. The KDD process can then

be described as a sequence of queries on i-extended database. A similar database concept has been suggested in [9].

In i-extended database, we use the simple formalization this form will be described in details in next Chapter (Chapter 9). However, the difference is that we considered the KDQL rules operator described in Chapter 9 as a possible querying language on mining association rules for i-extended database. Here we emphasize the generality of the framework and its use for complex KDD process modeling. It leads us to propose a research agenda to design general purpose query languages for KDD applications. The basic message in this Chapter is very simple: (1) I-extended 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. (2) I-extended database can be queried (in principle) just by using normal relational algebra or SQL, with the added property of being able to refer to the values of the evaluation function on the patterns. (3) Modeling KDD processes as a sequence of queries on i-extended database gives rise to chances for reasoning and optimizing these processes.

This Chapter is organized as follows. In Section 8.3 we define i-extended database framework and introduce KDD queries by means of examples. Section 8.4 considers the description of KDD processes and the add value of the framework for their understanding and their optimization. Section 8.5 is a short remark with open problems concerning i-extended database.

8.3 I-extended databases

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 pattern semantics. This function maps each pair (r, θ_i) to an element of V, where r is a database over R and $\theta_i \in P_R$ is a pattern. An instance of the schema, i-extended database (r, s) over the schema R consists of a database r over the schema R and a subset $s \subseteq P_R$.

Example 8.1 If *the patterns are Boolean formulae about the database, V is {true, false}, and the evaluation function e(r, θ) has value true iff the formula θ is true about r. In practice, a user might be interested in selecting from the intentionally defined collection of all Boolean formulas, the formulas which are true or the formulas which are false.*

At each stage of manipulating the i-extended database (r, s), the user can think that the value of $e(r, \theta)$ is available for each pattern θ which is present in the set s. Obviously, if the pattern class is large (as it is the case for Boolean formulas), an implementation can 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.

A typical KDD process operates on both of the components of i-extended database. The user can select a subset of the rows or more generally select data from the database or the data warehouse. In that case, the pattern component remains the same. The user can also select subsets of the patterns, and in the answer the data component is the same as before.

The situation can be compared with deductive databases where some form of deduction is used to augment fact databases with a potentially infinite set of derived facts. However, within i-extended database framework, the intentional facts denote generalizations that have to be learned from the data. So far, the discovery of the patterns we are interested in can not be described using available deductive database mechanisms [78].

Using the above definition for i-extended database it is easy to formulate query languages 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 semantics of the patterns.

In the following, we first illustrate the framework on association (Section 8.3.1), and then we generalize the approach and point out key issues for query evaluation in general (Section 8.3.2).

8.3.1 Association Rules

The association rule mining problem has received much attention since its introduction in [81]. 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 $B \in R \setminus X$. 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. This semantics is captured by frequency and confidence values. Given $W \subseteq R$, *support (W, r)* denotes the fraction of rows of r that have a *1* in each column of W. The frequency of $X \Rightarrow B$ in r is defined to be $support(X \cup \{B\}, r)$ while its confidence is $support(X \cup \{B\}, r)/support(X , r)$. Typically, we are interested in association rules for which the frequency and the confidence are greater than given thresholds. Though an exponential search space is concerned, association rules can be computed thanks to these thresholds on one hand and a safe pruning criterion that drastically reduce the search space on the other hand (*the so-called apriori trick* in [42]).

However, the corresponding i-extended database schema defines intentionally all the potential association rules. 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 θ in the database r. Notice that many other objective interestingness measures have been introduced for that kind of patterns (e.g., the J-measure in [82], the conviction [83] or the intensity of implication [84]). All these measures could be taken into account by a new evaluation function.

s_0	$e(r_0).f$	$e(r_0).c$
$A \Rightarrow B$	0.25	0.33
$A \Rightarrow C$	0.50	0.66
$B \Rightarrow A$	0.25	0.50
$B \Rightarrow C$	0.50	1.00
$C \Rightarrow A$	0.50	0.66
$C \Rightarrow B$	0.50	0.66
$AB \Rightarrow C$	0.25	1.00
$AC \Rightarrow B$	0.25	0.50
$BC \Rightarrow A$	0.25	0.50

s_1	$e(r_1).f$	$e(r_1).c$
$A \Rightarrow B$	0.33	0.33
$A \Rightarrow C$	0.66	0.66
$B \Rightarrow A$	0.33	1.00
$B \Rightarrow C$	0.33	1.00
$C \Rightarrow A$	0.66	1.00
$C \Rightarrow B$	0.33	0.50
$AB \Rightarrow C$	0.33	1.00
$AC \Rightarrow B$	0.33	0.50
$BC \Rightarrow A$	0.33	1.00

S_2	$e(r_2).f$	$e(r_2).c$
$B \Rightarrow C$	0.50	1.00

Instance r_0

A	B	C
1	0	0
1	1	1
1	0	1
0	1	1

Table 8.1 Patterns in three instances of i-extended database

We now describe the querying approach by using self explanatory notations for the simple extension of the relational algebra that fits to our need. Selection of tuples and patterns are respectively denoted by σ and τ. As it is always clear from the context, the operation can also be applied on i-extended database instances while formally, we should introduce new notations for them. This could be a future work.

Example 8.2 Mining association rules is now considered as querying i-extended database instances of schema $(R, (P_R, e, [0, 1]^2))$. Let us consider the data set is the instance r_0 in table 1 of the relational schema $R = \{A, B, C\}$.

The i-extended database $idb = (r_0 , s_0)$ associates to r_0 the association rules on the left most table of table 8.1 Indeed, in such an example, the intentionally defined collection of all the association rules can be presented. We illustrate (1) the selection on tuples, and (2) the selection on patterns in the typical situation where the user defines some thresholds for frequency and confidence.

1. $\sigma_{A\neq 0}(idb) = (r_1 , s_1)$ where $r_1 = \sigma_{A\neq 0}(r_0)$ and s_1 contains the association rules in the middle table of table 1.

2. $\tau_{e(r_0).f \geq 0.5 \wedge e(r_0).c \geq 0.7}(idb) = (r_2 , s_2)$ where $r_2 = r_0$ and s_2 contains the association rules from the rightmost table (on the top) of table 8.1.

To simplify the presentation, we have denoted by e(r).f and e(r).c the values for frequency and confidence.

An important feature is that operations can be composed due to the closure property.

Example 8.3 Consider that the two operations given in example 8.2 are composed and applied to the instance idb = (r_0 , s_0). Now, τ *e(r_0).f* $\geq$ *0.5* $\wedge$ *e(r_0).c* $\geq$ *0.7* ($\sigma_{A\neq 0}$ *(idb) = (r_3 , s_3) where* $r_3 = \sigma_{A\neq 0}$ *(r_0) and* s_3 *is reduced to the association rule* $C \Rightarrow A$ *with frequency 0.66 and confidence 1.*

The selection of association rules given in that example is rather classical. Of course, a language to express selection criteria has to be defined. It is out of the scope of this thesis to provide such a definition. However, let us just emphasize that less conventional association rule mining can also be easily specified.

Example 8.4 Consider an instance *idb = (r_0 , s_0)*. It can be interesting to look for rules that have a high confidence and whose right-hand side does not belong to a set of very frequent attributes *F :* τ *e(r_0).c* $\geq$ *0.9* $\wedge$ *e(r_0).rhs* $\notin$ *F (idb)) = (r_0 , s_1)*.

The intuition is that *rhs* denotes the right-hand side of an association rule. The rules in s_1 are not all frequent (no frequency constraint) but have a rather high confidence while their right-hand sides are not very frequent. Indeed, computing unfrequent rules will be in practice untractable except if other constraints can help to reduce the search space (and are used for that during the mining process).

The concept of exceptional data w.r.t. a pattern or a set of patterns is interesting in practice. So, in addition to the normal algebraic operations, let us introduce the so-called apply operation, denoted by α , that enables to cross the boundary between data and patterns by removing the tuples in the data set such that all the patterns are true in the new collection of tuples.

In the case of association rules, assume the following definition: a pattern θ is false in the tuple *t* if its left-hand side holds while its right-hand side does not hold; in the other cases a pattern is true. In other terms, an association rule θ is true in a tuple $t \in r$ if and only if *e ({t},*θ*).f = e ({t},*θ *).c = 1*. Let us define α *((r; s)) = (r_0, s)* where r_0 is the greatest subset of *r* such that $\forall \theta \in s$, *e(r_0,*θ *).c = 1*. Note that $r \setminus r_0$ is the collection of tuples that are exceptions *w.r.t.* the patterns in *s*.

Example 8.5 Continuing example 8.2, assume the instance *(r_0 ,s_4)* where s_4 only contains the rule $AC \Rightarrow B$ with frequency *0.25* and confidence *0.5*. Let σ ((r_0 , s_4)) = *(r_4 , s_4)*. Only the tuple $\langle 1,0,1 \rangle$ is removed from r_0 given s_4 since the rule $AC \Rightarrow B$ is true in the other ones. The pattern $AC \Rightarrow B$ remains the unique pattern (s_4 is unchanged) though its frequency and confidence in r_4 are now *0.33* and *1*, respectively.

Finally, it is clear that a selected collection of association rules can be materialized in a rule database and be queried by available query languages.

8.3.2 Generalization to other pattern types

The formal definition we gave at the beginning of Section 8.3 is very general. In this section, we first consider the other example of DM task where i-extended database concepts can be illustrated. We also point out crucial issues for query evaluation.

One typical KDD process we studied is the discovery of approximate inclusion and functional dependencies in a relational database. It can be useful either for debugging purposes, semantic query optimization or even reverse engineering [85]. We suppose that the reader is familiar with data dependencies in relational databases.

Example 8.6 Assume $R = \{A, B, C, D\}$ *and* $S = \{E, F, G\}$ with the two following instances in which, among others, $S[<G>] \subseteq R[<A>]$ is an inclusion dependency and $AB \rightarrow C$ a functional dependency (*see table 8.2(a-b)*).

Dependencies that almost hold are interesting: it is possible to define natural error measures for inclusion dependencies and functional dependencies. For instance, let us consider an error measure for an inclusion dependency $R[X] \subseteq S[Y]$ in r that gives the proportion of tuples that must be removed from r, the instance of R, to get a true dependency.

A	*B*	*C*	*D*
1	2	4	5
2	2	2	3
3	1	1	2
4	2	2	3

(a)

E	*F*	*G*
1	2	3
2	3	4
3	2	2

(b}

Inclusion dependencies	*Error*	*Functional dependencies*	*Error*
$R[<B>] \subseteq S[<E>]$	0	$B \rightarrow A$	0.5
$R[<D>] \subseteq S[<E>]$	0.25	$C \rightarrow A$	0.25
$S[<E>] \subseteq R[<B>]$	0.33	$BC \rightarrow A$	0.25
$R[<C,D>] \subseteq S[<E,F>]$	0.25	$BCD \rightarrow A$	0.25

(c)

Table 8.2 Tables for examples 8.6 and 8.7

With the same idea, let us consider an error measure for functional dependencies that gives the minimum number of rows that need to be removed from the instance r of R for a dependency $R: X \rightarrow B$ to hold.

Example 8.7 Continuing example 8.6, a few approximate inclusion and functional dependencies are given (see table 8.2(c)).

It is now possible to consider the two i-extended databases that associate to a database all the inclusion dependencies and functional dependencies that can be built from its schema. Evaluation functions return the respective error measures. When the error is null, it means that the dependency holds. Indeed, here again it is not realistic to consider that querying can be carried out by means of queries over some materializations of all the dependencies that almost hold.

Example 8.8 Continuing again example 8.6, a user might be interested in "selecting" only inclusion dependencies between instances r and s that do not involve attribute R.A in their left-hand side and have an error measure lower than 0.3. One expects that a sentence like R [<C, D>] $\subseteq$ S [<E, F>] belongs to the answer. The "apply" operation can be used to get the tuples that are involved in the dependency violation. One can now search for functional dependencies in s whose left-hand sides are a right-hand side of a previously discovered inclusion dependency. For instance, we expect that a sentence like EF $\rightarrow$ G belongs to the answer. Evaluating this kind of query provides information about potential foreign keys between R = {A, B, C, D} and S = {E, F, G}.

Query evaluation: We already noticed that object-relational query languages can be used as a basis for i-extended database query languages such as KDQL. However, non-classical optimization schemes are needed since selections of properties lead to complex DM phases. Indeed, implementing such query languages is difficult because selections of properties are not performed over previously materialized collections. First one must know efficient algorithms to compute collection of patterns and evaluate the evaluation function on very large data sets. But the most challenging issue is the formal study of selection language properties for general classes of patterns: given a data set and a potentially infinite collection of patterns, how can we exploit an active use of selection criteria to optimize the generation/evaluation of the relevant patterns.

Example 8.9 When mining association rules that do not involve a given attribute, instead of computing all the association rules and then eliminate those which contain that attribute, one can directly eliminate that attribute during the candidate generation phase for frequent sets discovery.

Furthermore, three important questions arise:

1. *How to evaluate a class of similar patterns faster than by looking at each of them individually*? An explicit evaluation of all the patterns of the schema against the database (and all databases resulting from it by queries) is not feasible for large data sets.

2. *How to evaluate patterns without looking at the whole data set?* This is an important issue to reduce dimensionality of the DM task, e.g., via sampling. In some cases, it might be also possible not to use the data set and perform a simple selection over a previously materialized collection of patterns.

3. *How to evaluate operation sequences, e.g., in replays, more efficiently*? Compiling schemes can be defined for this purpose.

The complexity of mining frequent association rules mainly consist of finding frequent sets. The frequency threshold is used to safely prune the search in the exponential space of attribute sets and it leads to algorithms that scale nicely to relations containing thousands of columns and tens of millions of rows. The confidence threshold or other interestingness measures can then be used to prune search in the exponential space of association rules. Provided Boolean constraints over attributes, [86] show how to optimize the generation of frequent sets using this kind of constraints during the generation/evaluation process. This approach has been considerably extended in [72].

Other interesting ideas come from the generalization of the *Apriori tricks*, and it can be found in different approach like [41] or [87]. Propose an algorithm that generalizes the *Apriori trick* to the context of frequent atom sets [41]. This typical inductive logic programming tool enable to mine association rules from multiple relations but can also be used for mining frequent *Datalog queries* [88]. Consider *query flocks* that are parameterized Datalog queries for which a selection criterion on the result of the queries must hold [87]. When the filter condition is related to the frequency of answers and queries are conjunctive queries augmented with arithmetic and union, they can propose an optimizing scheme.

The formal study of selection criteria and *query optimizing* schemes for more pattern classes that are more complex than frequent sets and association rules is to be done. A framework for object-oriented query optimization when using expensive methods [89] can also serve as a basis for optimization strategies.

8.4 I-extended databases and KDD processes

Already in the case of a unique class of patterns, real life mining processes are complex. This is due to the dynamic nature of knowledge acquisition, where gathered knowledge often affects the search process, giving rise to new goals in addition to the original ones.

In the following, we introduce a scenario about telecommunication networks fault analysis using association rules. It is a simplified problem of knowledge discovery to support offline network surveillance, where a network manager tries to identify and correct faults based on sent alarms. A comprehensive discussion on this application is available in [90].

Alarm type	*Alarming element*	*Element Type*	*Date*	*Time*	*Week*	*Alarm severity*	*Alarm Text*
1111	*E1.1*	*ABC*	*980119*	*233605*	*4*	*1*	*LINK FAILURE*
2222	*E2*	*CDE*	*980119*	*233611*	*4*	*3*	*HIGH ERROR RATE*
3333	*A*	*EFG*	*980119*	*233627*	*4*	*4*	*CONNECTIONNOT ESTABLISHED*
4444	*B2.1*	*GHI*	*980119*	*233628*	*4*	*2*	*LINK FAILURE*

s_0	$e(r_0).f$	$e(r_0).c$
alarm_type=1111 $\Rightarrow$ *element_type=ABC*	0.25	1.00
alarm_type=222 $\Rightarrow$ *alarming_element=E2, element_type=CDE*	0.25	1.00
alarm_ type=1111, element_ type=ABC $\Rightarrow$ *alarm_ text=LINK FAILURE*	0.25	1.00
alarm_type=5555 $\Rightarrow$ *alarm_severity=1*	*0.00*	*0.00*

Table 8.3 Part of i-extended database consisting of data part r0 (upper table) and rule part s0 (lower table)

Assume that the schema for the data part of the relevant i-extended database is *R = (alarm type, alarming element, element type, date, time, week, alarm severity, alarm text)*. By *(r, s)* we denote i-extended database for association rules where *r* refers to the data part and ***s*** refers to the pattern part. With association rules, we consider items as equalities between attributes and values, while rule left-hand and right-hand sides are sets of items. Notice also that we use in the selection conditions expressions that concern subcomponents of the rules. Typically, one wants to select rules with a given attribute on the left-hand side *(LHS)* or on its right-hand side *(RHS*), or give bounds to the number of occurring items. Self-explanatory notations are used for this purpose. A sample of an instance of this schema is given in table 8.3.

In table 8.3, the network manager decides to look at association rules derived from r_0, the data set for the current month. Therefore, he/she "tunes" parameters for the search by pruning out all rules that have confidence under 5% or frequency under 0.05% or more than 10 items (phase 1 in table 8.4). The network manager then considers that attributes *"alarm text"* and *"time"* are not interesting, and projects them away (phase 2). The number of rules in the resulting rule set, s_2, is still quite large. The user decides to focus on the rules from week 30 and to restrict to 5 the maximum amount of items in the rule (phase 3). While browsing the collection of rules s_3, the network manager sees that a lot of rules concern the network element *E*. That reminds him/her of maintenance operation and he/she decides to remove all rules that contain *"alarming element = E or its subcomponent"* (phase 4). We omit the explanation of dealing with the taxonomy of components. The resulting set of rules seems not to show anything special. So, the network manager decides to compare the behavior of the network to the preceding similar period (week 29) and find out possible differences (phases 5-6). The network manager then picks up one rule, $\boldsymbol{s_8}$, that looks interesting and is very strong (confidence is close to 1), and he wants to find all exceptions to this rule; i.e. rows, where the rule does not hold (phases 7-8). Except for the last phases, the operations are quite straightforward. In the comparison operation, however, we must first replay the phases 3-4.

This is because we have to remove the field *"week"* from the schema we used in creating rules for *week 30*, so that we can compare these rules with the rules from *week 29*. Then we create for *week 29* the same query (except for the week information), take the intersection from these two rule sets, and calculate the frequencies and confidences of the rules in the intersection. The search for exceptions is performed using the apply operation introduced in Section 8.3.

Phase	*Operation*	*Query and conditions*
1	***Selection***	τ_{F1} $((r_0,s_0)) = (r_0,s_1)$ $F1 = e(r0).f \geq 0.005 \wedge e(r0).c \geq 0.05 \wedge \lvert LHS \rvert \leq 10$
2	***Projection***	π_T $((r_0,s_1)) = (r_1, s_2)$ $T = R \setminus \{$ *alarm text, time* $\}$
3	***Selection***	$\tau_{F2}(\sigma_{C1}((r_1, s_2))) = (r_2, s_3)$ $C_1 =$ *(week = 30) and* $F_2 = \lvert LHS \cup RHS \rvert \leq 5$
4	***Selection***	$\tau_{F3}((r_2, s_3)) = (r_2, s_4)$ $F3 =$ *(alarming element = E *)* $\notin \{LHS \cup RHS\}$
5	***Replay 3-4 (week 30)***	$\tau_{F3}(\tau_{F2}(\pi_U(\sigma_{C1}((r_1, s_2)))))=(r_3, s_5)$ $U = T \setminus \{$*week*$\}$, *other conditions as in 3-4*
6	***Replay 3-4 (week 29)***	$\tau_{F3}(\tau_{F2}(\pi_U(\sigma_{C2}((r_1, s_2)))))=(r_4, s_6)$ $C_2 = T \setminus \{$*week = 29*$\}$, *other conditions as in 5*
7	***Intersection***	$\cap((r_3,s_5),(0,s_6))=(r_3,s_7)$
8	***Apply***	$\alpha((r_3,s_8)) = (r_5, s_9)$

Table 8.4 Summary of the phases of the experiment

The network manger system in table 8.3, illustrates a typical real-life DM task. Due to the closure property, KDD processes can be described by sequences of operations, i.e., queries over relevant i-extended database. In fact, such sequences of queries are abstract and concise descriptions of DM processes. An interesting point here is that these descriptions can even be annotated by statistical information about the size of selected dataset, the size of intermediate collection of patterns etc., providing knowledge for further use of these sequences.

8.5 Remarks

We presented a framework for i-extended database considering that the whole process of DM can be viewed as a querying activity. Our simple formalization of operations enables the definition of mining processes as sequence of queries, by using the closure property. The description of a non-trivial mining process using these operations has been given and even if no concrete query language or query evaluation strategy is available yet, it is a mandatory step towards general purpose query languages for KDD applications.

KDD query language KDQL is a good candidate for i-extended database querying though because it is dedicated to Boolean and association rule mining, respectively. Simple *pattern discovery algebra* has been proposed in [91]. It supports pattern generation, pattern filtering and pattern combining operations. This algebra allows the user to specify discovery strategies, e.g., using different criteria of interestingness but at a macroscopic level; implementation issues or add-value for supporting the mining step are not considered.

Recent contributions to the logic programming framework make it suitable for DM purposes. Descriptive logic programming (or learning from interpretations [37]) is a framework for the discovery of first-order regularities in relational databases.

We also presented, as an example, i-extended database for association rules, and gave a realistic network system in table 8.3 using simple operations. It appears that without introducing any additional concepts, standard database terminology enable to carry out i-extended database querying and those recent contributions to query optimization techniques can be used for i-extended database implementation. A significant question is whether i-extended database framework is interesting for a reasonable collection of DM problems. We currently study KDD processes that need different classes of patterns.

Chapter 9

Implementation of Knowledge Discovery Query Language (KDQL)

9.1 Motivation 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 Chapter 6. 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) is based on the *association rules* to interact *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, and the graphically represented in a 2D and 3D charts. The KDQL syntax will be present also in the appendix A, and the KDQL visual scripts will be achieved in appendix B but these scripts didn't use the KDQL syntax in their retrieve statements. Moreover, we hope that we will write queries in KDQL syntax in the near future.

9.2 Principles of DMQL rules to interact relational databases

Interacting relational databases is often necessary to specify the interesting set of data to be studied, 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 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 *"Hungary"*, and by using the DMQL described in [12] the structure will be as follows.

discover rules *in the form*
major*(s : student, x)* $\wedge$ *Q(s, y)* $\rightarrow$ *R(s, z)*
from *student*
where *birth_place = "Hungary"*
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* [92].

9.3 Using KDQL to interact i-extended databases

As we know from previous Chapters 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. 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 Chapter 5. 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]. 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 θ 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] for a survey and [93] 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 [94]. 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 [95]. 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].

The contribution of this dissertation 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 [72].

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 [80, 96].

9.4 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 Chapter 3. Thus the database has to contain both data and generalizations about that data. I-extended database was described in the previous Chapter. This motivates the following definition (simplified from the one in [96]).

***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,θ_i) to an element of V, where r is a database over R and θ_i is a pattern from P_R.

***Instance**:* An *instance (r; s)* of a 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 [81]. 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.

s_0	$e(r_0).f$	$e(r_0).c$
$B \Rightarrow A$	0.25	0.33
$C \Rightarrow A$	0.50	0.66
$A \Rightarrow B$	0.25	0.50
$C \Rightarrow B$	0.50	1.00
$A \Rightarrow C$	0.50	0.66
$B \Rightarrow C$	0.50	0.66
$C \Rightarrow AB$	0.25	1.00
$B \Rightarrow AC$	0.25	0.50
$A \Rightarrow BC$	0.25	0.50

s_1	$e(r_1).f$	$e(r_1).c$
$B \Rightarrow A$	0.33	0.33
$C \Rightarrow A$	0.66	0.66
$A \Rightarrow B$	0.33	1.00
$C \Rightarrow B$	0.33	1.00
$A \Rightarrow C$	0.66	1.00
$B \Rightarrow C$	0.33	0.50
$C \Rightarrow AB$	0.33	1.00
$B \Rightarrow AC$	0.33	0.50
$A \Rightarrow BC$	0.33	1.00

s_2	$e(r_2).f$	$e(r_2).c$
$C \Rightarrow B$	0.50	1.00

Instance r_0

A	B	C
1	0	0
1	1	1
1	0	1
0	1	1

9.1 Patterns in three instances of i-extended database

Example 9.1 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 $B \in R \setminus X$ [81]. 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 $W \subseteq R$, *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 *freq(X∪ {B}, r)* while its confidence is *freq(X ∪ {B}, r)/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, θ) are the frequency and the confidence of the *rule* θ 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 θ 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 9.1 is now considered as querying a i-extended database instances of schema $(R, (P_R, e, [0, 1]^2))$.

Example 9.2 Assume the dataset is the instance r_0 in table 9.1 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 9.1. 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_1, s_1) where the data part r_1 does not contain the tuple *(0, 1, 1)*, and the pattern part s_1 contains the rules in the second table of table 9.1, 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 9.1.

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 $Q_2 \, o \, Q_1$ 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 9.1, 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 σ and τ it is clear from the context, the operation is also applied on a i-extended database instances, e.g., we write σ_C ((*r, s*)) to denote (σ_C (*r), s*)).

Example 9.3 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 9.2(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 in a nested relation, e.g., an SQL3 table [93].

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 9.2(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 [94].

Tid	Item	Price	Date
1	A	7	1
2	A	7	1
2	B	5	1
2	C	9	1
3	A	7	1
3	C	9	1
4	B	5	1
4	C	9	1

body	head	lbody	$e(r_0).f$	$e(r_0).c$
{A}	{B}	1	0.25	0.33
{A}	{C}	1	0.50	0.66
{A,B}	{C}	2	0.25	1
{A,C}	{B}	2	0.25	0.5

(a)

Phase	Query and conditions
1-	$\tau_{F_1}((r_0, s_0)) = (r_0, s_1)$ $F_1 = e(r_0).f \geq 0.05 \wedge e(r_0).c \geq 0.3 \wedge lbody \leq 6.$
2-	$\tau_{F2}(\sigma_{C_1}((r_0, s_0))) = (r_1, s_2)$ $C_1 = (Date=13)$ $F_2 = e(r_1).f \geq 0.05 \wedge e(r_1).c \geq 0.3 \wedge lbody \leq 4.$
3-	$\tau_{F3}((r_1, s_2)) = (r_1, s_3)$ $F_3 = D \notin body.$
4-	$\tau_{F4}(\sigma_{C2}((r_1, s_0))) = (r_2, s_4)$ $C_2 = (Price \geq 7)$ $F_4 = e(r_2).f \geq 0.01 \wedge e(r_2).c \geq 0.3 \wedge lbody \leq 4 \wedge D \notin body$

(b)

Table 9.2 Basket data as i-extended data (a) and a few queries (b)

9.5 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 [95] for other examples and a complete definition of its syntax and operational semantics. Given the dataset r_1 as defined in table 9.2, phase 4 is defined by the *KDQL rules* statement in table 9.3. The *KDQL rules* operator takes a relational database and produces an SQL3 table [93] in which each tuple denotes a mined rule. Several possibilities exist to precisely define the input data. Basically, the whole potential of 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)*.

```
KDQL RULE s1 AS
SELECT DISTINCT
  1..6 Item AS BODY,
  1..1 Item AS HEAD,
  SUPPORT, CONFIDENCE
FROM  r0
GROUP BY Tid
EXTRACTING RULES WITH
        SUPPORT: 0.05,
        CONFIDENCE: 0.03
```

(Phase 1)

```
SELECT * AS
FROM   s2
WHERE D NOT IN BODY
```

(Phase 3)

```
KDQL RULE s2 AS
SELECT DISTINCT
  1..4 Item AS BODY,
  1..1 Item AS HEAD,
  SUPPORT, CONFIDENCE
FROM (SELECT * AS r1
        FROM   r0
        WHERE Date=13
GTOUP BY Tid
EXTRACTING RULES WITH
        SUPPORT: 0.05,
        CONFIDENCE: 0.03
```

(Phase 2)

```
KDQL RULE s1 AS
SELECT DISTINCT
  1..4 Item AS BODY,
  1..1 Item AS HEAD,
  SUPPORT, CONFIDENCE
WHERE BODY.Item<> D
FROM (SELECT * AS r2
        FROM   r1
        WHERE Price >=7)
GTOUP BY Tid
EXTRACTING RULES WITH
        SUPPORT: 0.01,
        CONFIDENCE: 0.3
```

(Phase 4)

Table 9.3 Phases 1 to 4 of table 2 using KDQL rules

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 9.2(b) are easily translated into KDQL rules queries as given in table 9.3. 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.

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 [97] are:

- *Preprocessor:* After the interpretation of a KDQL rules statement, preprocessor retrieves source data, evaluates the mining, grouping and 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 unfrequent 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 [42]. 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 defied in the future work.

9.6 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 a lot of 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 [98, 99]. 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.

9.7 KDQL algorithms and architecture

9.7.1 Architecture

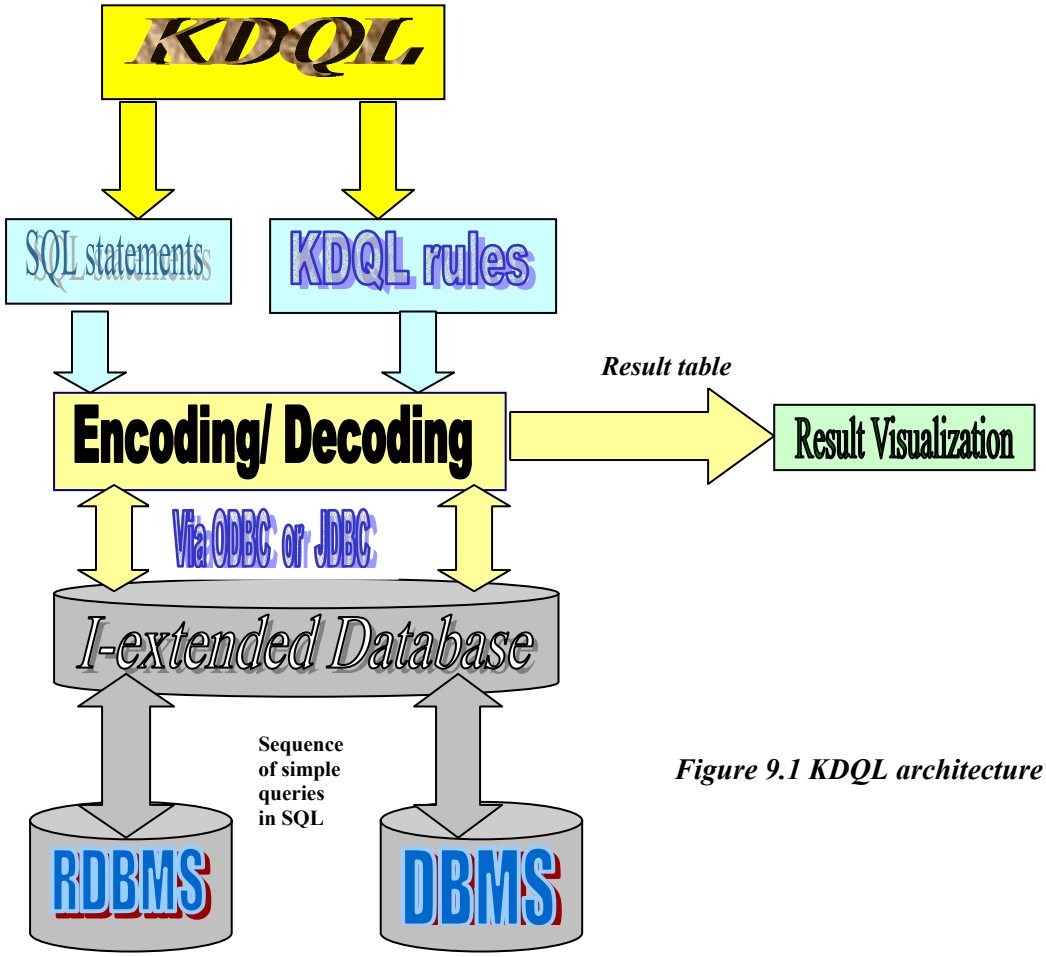

Figure 9.1 KDQL architecture

In figure 9.1, 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 rules in i-extended database or from traditional databases.

9.7.2 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:

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

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 [55,98,100, 101,102] some theoretical results can be found in [103], and a suggested logical formalism in [104].

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

1. $\boldsymbol{C} := \{ p \in P \mid$ *for no* $q \in P$ *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. $\boldsymbol{F} := F \cup \{p \in C \mid$ *p is sufficiently frequent in d};*
6. $\boldsymbol{C} := \{p \in P \mid$ all q $\in P$ with q $<$ p have been considered already and it is possible that p 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 [105, 98] 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., [106]. 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.

9.8 Association rules algorithms

In this Section we discuss using *KDQL* algorithm 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 σ and the confidence of the rule is at least another threshold θ. 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 σ typically is around 10^{-2}—10^{-4}. The confidence threshold or 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, 77, 81, 107, 108, 109]. 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 9.2 Finding frequent sets for association rule.

```
1. C := {{A} | A ∈ R};
2. F := 0;
3. i := 1;
4. while C≠0; do
5.      F':= the sets X ∈ C that are frequent;
6.      add F' to F ;
7.        C := sets Y of size i+1 such that
8.                each subset W of Y of size i is frequent;
9.       i:=i+1;
10. 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 $r_X \subseteq r$ 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 $X'' = X / \{B\}$ 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 [92, 97, 110].

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 [104]. 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 [111].

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: only candidate sets that contain "Data Management" need to be considered.

9.9 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, 109] or functional dependencies [113] that hold in a relation. See [112] 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 [109, 113]; also the generic algorithm can be modified to correspond to this approach. We give the sample-and-correct algorithm for finding functional dependencies.

Algorithm 9.3 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 := a sample of r;
2. K := keys(s);
3. **while** *there is a set X ∈ K such that X is not a key of r do*
4. add some rows u,v ∈ r with u[X]= v[X] to s;
5. K := keys(s);
6. **od**;
7. **output** *K.*

9.10 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 i-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. Each transaction contains the set of bought items with the purchased quantity and the price. The simplest way to organize this data is the table Purchase, depicted in table 9.4. The transaction column (*tr.*) contains the identifier of the customer transaction; the other columns correspond to the customer identifier, the type of the purchased item, the date of the purchase, the unitary price and the purchased quantity (*q.ty*).

tr.	*Customer*	*item*	*date*	*price*	*q.ty*
1	*cust1*	*milk*	*12/17/95*	*140*	*1*
1	*cust1*	*corn_flaks*	*12/17/95*	*180*	*1*
2	*cust2*	*bread*	*12/18/95*	*25*	*2*
2	*cust2*	*cheese*	*12/18/95*	*150*	*1*
2	*cust2*	*coke*	*12/18/95*	*300*	*1*
3	*cust1*	*coke*	*12/18/95*	*300*	*1*
4	*cust2*	*bread*	*12/19/95*	*25*	*3*
4	*cust2*	*coke*	*12/19/95*	*300*	*2*

Table 9.4 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***,
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 9.5 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 9.6 shows the resulting *associations* table; observe that if we change the minimum support to *0.3,* we then loose almost all rules of table 9.6 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.

tr.	Customer	Item	date	Price	q.ty
1	customer1	milk	12/17/95	140	1
	customer1	corn_flaks	12/17/95	180	1
2	customer2	bread	12/18/95	25	2
	customer2	cheese	12/18/95	150	1
	customer2	coke	12/18/95	300	1
3	customer1	coke	12/18/95	300	1
4	customer2	bread	12/19/95	25	3
	customer2	coke	12/19/95	300	2

Table 9.5 The Purchase table grouped by transaction

BODY	HEAD	S.	C.
{milk}	{corn_flaks}	0.25	1
{corn_flaks}	{milk}	0.25	1
{bread }	{cheese}	0.25	0.5
{bread }	{coke}	0.5	1
{cheese}	{bread }	0.25	0.5
{cheese}	{coke}	0.25	1
{coke}	{bread }	0.5	0.66
{coke}	{cheese}	0.25	0.33
{bread ,cheese}	{coke}	0.25	1
{bread ,coke}	{cheese}	0.25	0.5
{cheese ,coke}	{bread }	0.25	1

Table 9.6 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*

In [110] 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*

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 with out 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 thesis 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*
EXTRACTING RULES WITH SUPPORT: *0.1,*
CONFIDENCE: *0.2*

9.11 Condensed KDQL representations

We remarked in Section 9.6 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 [111]: 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 [114] can also be viewed as a condensed representation for a class of queries. Similarly, in computational geometry the notion of an *"approximation"* [115] 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.

9.12 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 has 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 multidimensional curves [116]. 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 [117], 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.

9.12.1 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 [110] 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 [110]. 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.

9.12.2 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. [26]. 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.

9.13 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.

9.14 I-extended databases and KDQL scripts

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 [29], 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. Executing the KDQL program we will get some interfaces these interfaces are describing in the following scripts.

- Interacting aliases in i-extended database will be issued in this script in figure 9.2.

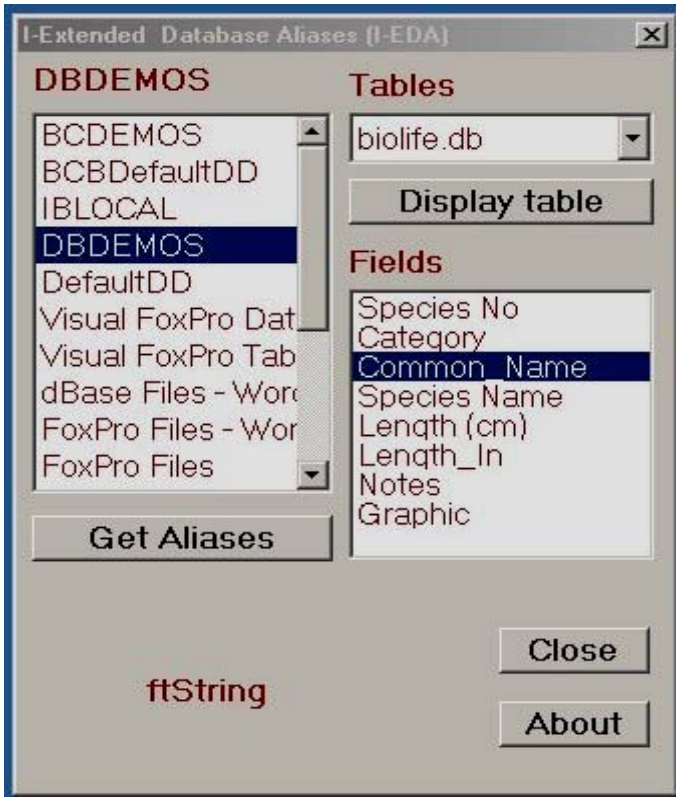

Figure 9.2 Integrating i-extended database aliases

- In i-extended database we can also review the images or comments for each particular database table see figure 9.3.

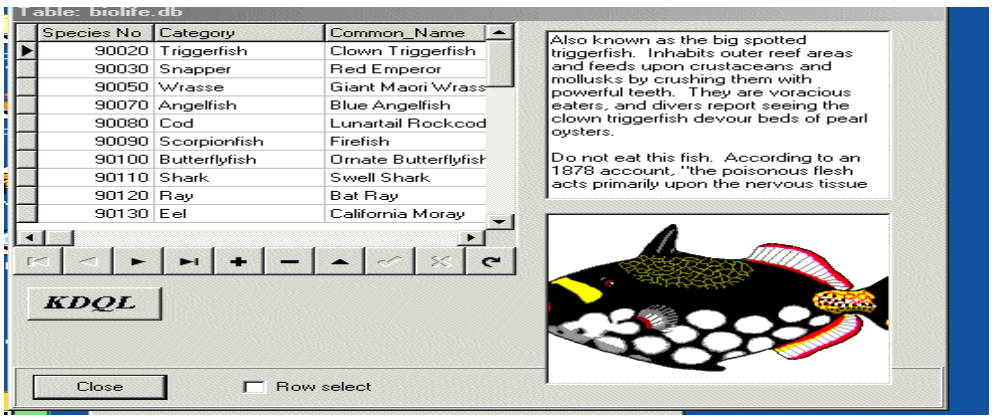

Figure 9.3. Reviewing the selected table to be retrieved

• Applying the KDQL after selecting the target databases table to be retrieve will be specify by the KDQL program, figure 9.4 contains a view of the KDQL program.

Figure 9.4 Retrieving a selected database table in KDQL mode

• Visualizing the retrieved results in 2D charts. See figure 9.5 for more details concerning the results.

Figure 9.5 2D charts in KDQL result mode

• KDQL have the same visualization results in 3D charts mode, it can be seen in figure 9.6.

Figure 9.6. 3D charts in KDQL result mode

• Getting the 2D and 3D results helps the user to demonstrate the differences between values but knowing the average could helps him more to make his own decision. Figure 9.7 shows the average of the values.

Figure 9.7. The average decision result in KDQL mode

9.15 Conclusion

In this Chapter 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 and some remarkable algorithms such as association rules were shown also. Moreover, examples were given as well. In this Chapter 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 program instead of the classical SQL statement .

Chapter 10

Conclusion

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 so called ODBC_KDD(2).One of the ODBC_KDD(2) model requirements is the implementation of a query language that could handle DM rules. This query language is called *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 could be retrieved by the use of KDQL. KDQL RULES operation was also theoretically proposed and also some examples were given. Using KDQL RULES is used only to find out the association rules in i-extended database we have.

The development and results of this thesis 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.

In addition, in this dissertation work we attempted to show how two different approaches (i.e., association rules and visual techniques) can be integrated to discover hidden rules. In particular, we pursue the visualization of the application of a DM rules to a particular problem, through the creation of new ways of visualizing parameter spaces and induced models for DM, and their integration with algorithmic methods. This integration will provide support for user navigation and exploration of huge parameter spaces and complex induced models.

Also, in this thesis, we have investigated the role of the visualization paradigm in knowledge discovery tasks, and extend of its applicability to parameter selection and model exploration problems, where currently only algorithmic approaches have been applied, or visualization has been poorly or inadequately applied. Here, we are not saying that visualization can replace other methods for knowledge discovery tasks or work better than methods from other disciplines, but we are arguing in favor of a useful integration of visualization with other approaches, in order to visualize specific DM tasks that have a significant impact in the quality of the results of the data mining process.

The conclusion of this dissertation is organized as follow. Section 10.1 presents a summary of my thesis work. In Section 10.2 discussion and research direction. Finally, in Section 10.3 future research work.

10.1 Summary of the thesis work

In this thesis, we introduced a remote access knowledge discovery in databases (KDD) model called ODBC_KDD (2). In ODBC_KDD (2) model we proposed a query language called Knowledge Discovery Query Language (KDQL) as well. KDQL was suggested to retrieve proposed databases called *i-extended database*. The KDQL and i-extended database were theoretically implemented for mining the discovered association rules. The discovered association rules can be defined by a set of expressions of a First Order Logical (FOL) language. A similar approach to KDQL could be Data Mining Query Language (DMQL) which is presented in [12]. Syntax for KDQL RULES operation was also proposed in appendix A and some examples were shown as well. Visualizing the KDQL results will be presented in appendix B and some remarks regarding the program will be given also. Moreover, by using visualization mode in KDQL for reviewing the result data chats in 2D and 3D forms will use the same technique as in SQL+D query which was described in [29]. The major contributions of my thesis work are summarized as follows:

1. The idea of proposing a new remote access KDD model called ODBC_KDD (2) has been introduced in [26]. The goal of proposing this model is to build an attractive model that could get results with more detailed description such as visualization, scripts, statistical inferences and more. One of the architecture components in this model provides a query language the so called Knowledge Discovery Query Language (KDQL) in their server side. This query tool plays a deterministic role in this thesis work.

2. The use of a conceptual KDD remote access model leads to access to a database. Accessing a database by a query language such as KDQL in the ODBC_KDD (2) model will be issued as well. We proposed a database concept called i-extended database (I-ED) to be addressed by the use of KDQL query language. Frameworks for i-extended database consider that the whole process of DM can be viewed as a querying activity. Our simple formalization of operations enables the definition of mining processes as sequence of queries, by using the closure property. The description of a non-trivial mining process using these operations has been given and even if no concrete query language or query evaluation strategy is available yet, it is a mandatory step towards general purpose query languages for KDD applications.

   KDD query language such as KDQL is a good candidate for i-extended database querying though because it is dedicated to Boolean and association rule mining, respectively. A simple Pattern Discovery Algebra has been proposed in [91]. It supports pattern generation, pattern filtering and pattern combining operations. This algebra allows the user to specify discovery strategies, e.g., using different criteria of interestingness but at a macroscopic level; implementation issues or add-value for supporting the mining step are not considered.

   Recent contributions to the logic programming framework make it suitable for DM purposes. Descriptive logic programming (or learning from interpretations [37]) is a framework for the discovery of first-order regularities in relational databases. We also presented, as an example, i-extended database for association rules, and gave a

realistic network system in table 8.3 using simple operations. It appears that without introducing any additional concepts, standard database terminology enables to carry out i-extended database querying and those recent contributions to query optimization techniques can be used for i-extended database implementation. A significant question is whether i-extended database framework is interesting for a reasonable collection of DM problems. We currently study KDD processes that need different classes of patterns.

3. Earlier a DMQL was described in [12]. It was designed to explore and discover relations in relational databases. These relations hold interesting patterns or rules. These rules or patters could be investigated by the association rules. Mining the association rules requires logical foundations in DM tasks. KDQL is similar KDD query language to the DMQL, however KDQL discovers only the association rules.

4. Applying logical foundation approaches using First Order Logic (FOL) languages for the description of such patterns offers DM the opportunity of discovering more complex regularities which may be out of reach for attribute-value languages and classical statistical algorithms. Without logical foundation approach we are unable to locate the entire supported data item (patters) and then confidence will be concluded. Without support and confidence of data item we can not capture any associated rule from the database that that could be mined by KDQL.

5. For mining the discovered association rules from i-extended database we have described a general foundation of association rules. By theses rules we can mine association regarding to the frequent rate of support and confidence in the itemset (table 9.6). The main majority of association rules are the measures of the confidence and support. With support and confidence of the itemset we could compute the interestingness. The interesting itemset in the database led us to mine or discover the data that have got the highest priority of support and then the confidence.

6. In ODBC_KDD (2) model we proposed a query language called Knowledge Discovery Query Language (KDQL). This KDQL is an attractive part of the ODBC_KDD (2) model and it was theoretically investigated by us. In figure 10.1 we show the KDQL connections in their environments.

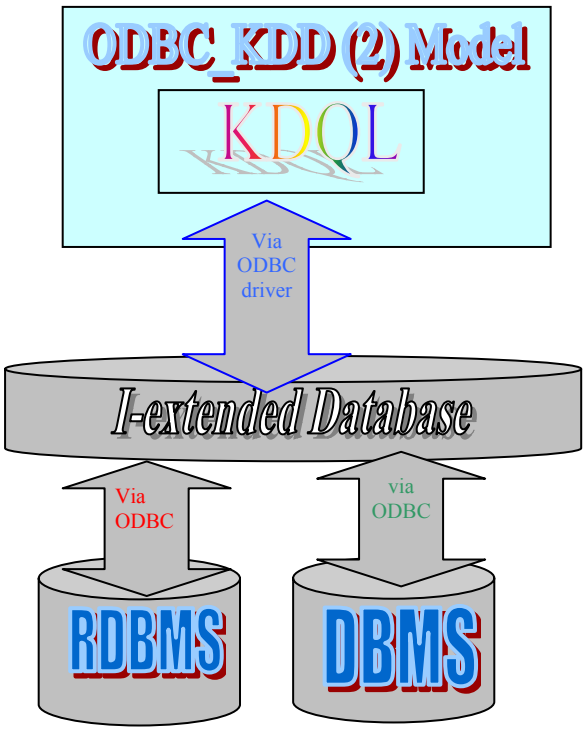

Figure 10.1 The envaroumnets of KDQL

In this figure we can see that the KDQL is a part of the ODBC_KDD (2) model. KDQL will call i-extended database via ODBC connection. I-extended database will call all the requested information from traditional databases via the specific ODBC diver for each database. KDQL was implemented to handle DM task. One of the DM common tasks is visualization. DM visualization techniques can be maintained to visualize interesting association rules discovered from the databases. The visual result of the KDQL query will be represented in 2D and 3D charts (i.e. Pie, Bar, Points, Lines). In KDQL we use two different techniques of mining or discovering the association rules. We use a technique similar to the DMQL in DBMiner system in [12]. The second is the visualization techniques. It was also similar to SQL+D visual query in [29], which can retrieves and then represents charts, graphs, and images stored in the database. The KDQL program was written in Delphi programming language. Syntax to the KDQL has been given and some examples were also expressed in appendix A.

10.2 Discussion and research direction

The growths of the data in the databases are increased day by day and extracting all the interesting data from that database is impossible some times. This challenge motivated us to think of a database concept (called i-extended database) that could interact to different types of databases and capture all the required information and store it temporary to make it accessible for the KDQL. KDQL and the i-extended database

are proposed and then located in ODBC_KDD (2) model. This fact led us to divide this framework in this thesis into two main parts i-*extended database* and *KDQL*.

1. In the introduction of i-extended database we considered a simple query language that could be defined by using normal database terminology. We generally demonstrate the use of this framework to model typical DM processes. For modeling DM process we use association rules to discover or mine all the frequent and confidence values that appear in the itemset. Using only association rules is not the only hope but it could be the starting points to several kinds of rules such as clustering, discriminant, etc in the future. The idea behind that is to help the user gather all the requested information in one place and then learn the behavior of the information he found. A significant question is whether i-extended database framework is interesting for a reasonable collection of DM problems. We currently study KDD processes which need different classes of patterns.

2. KDQL has two main aspects. The first is to mine or discover association rules from the I-extended database. The second is to use this discovered association rules to make the information "*visible"*.

Firstly, mining or discovering association rules from I-extended database was proposed. It requires mining frequent association rules that mainly consist of finding frequent itemsets. Discovering these frequent sets led us to compute the support and confidence itemset and then interestingness of the itemsets that appears in the i-extended database. We also proposed syntax for the KDQL. The syntax was not practically implemented yet but we gave examples for a general aspect. The results also are summarized in Appendix A.

Second, developing a visualization result of mining the association rules in i-extended database helps the user to understand the behavior of the information that was laying in i-extended database. The visualization mode of the KDQL was designed to view 2D and 3D charts in four different types such as pie, line, points and bars, also an average mode is included as well. Results also were presented in Appendix B.

10.3 Future work

Some inserting future research problems will be presented respectively in this Section. The main focus of my future work will be to improve all the related components of the ODBC_KDD (2) model which was proposed by *Fazekas G.* and *me* in [26]. Moreover, my future task will be listed as follows.

1. In the ODBC_KDD (2) model we suggested to develop a special ODBC driver in both sides (client / server). This driver is called Extended ODBC driver (EODBC). The aim of this driver is to handle the KDQL syntax instead of the classical SQL query statements.

2. In the server side in ODBC_KDD (2) model we proposed an interpreter to be the interface between the Extended ODBC (EODBC) driver and the SQL driver. The main goal is to translate the KDQL syntax into common SQL syntax and inversely, from SQL to KDQL syntax. This interpreter helps the requested KDQL syntax to

reach the data safely. SQL is a standard query language that could access any type of traditional databases.

3. Regarding to the databases in the server side in ODBC_KDD (2) model the SQL application can access only one type of databases which is DBMS. This problem led us to propose a new database concept for the model. It is called i-extended database. This i-extended database could be placed between DBMS and the SQL application to give reliability to the model. It is clear from the context in Chapter 8, that the operation can also be applied on i-extended database instances while formally; we should introduce new notations for them. This requirement could be identify if we add more practical capability to i-extended database by giving i-extended database the right to apply one part of the whole KDD process which is extracting all the interesting association rules from many other database sources. These rules could be temporary stored in i-extended database. The other part of the KDD process will be maintained by the KDQL which collects the most interesting rules from i-extended database.

4. I want to implement a new *core operator* that provides an encoding of association rules [42].

5. If I'll succeed in implementing a new core operator, then I should design a new post-processor to decode the rules and put the relations containing the desired rules in the table which is stored in i-extended database.

6. I want also to add more capability to the visualization mode in the KDQL program. KDQL program require new implementations such as *simplified display specifications syntax and instantiation of display elements, implementing of directed and undirected graphs as a visualization aid for extracted data, implementation and new syntax of charts with categorization, directed and undirected graphs for advanced visualization of data and implementation and expanded features for temporal presentations with time constraints using simple and intuitive in-the-query specifications.*

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***>:=

[< ***Cardinaly_Sheap*** >] < ***AttrName*** > < ***AttrList*** > ***AS BODY***
/* default cardinality sheap for the Body: 1..n */

< ***Head_Description_KDQL***>:=

[< ***Cardinaly_Sheap*** >] < ***AttrName*** > < ***AttrList*** > ***AS BODY***
/* default cardinality shaep for the Head: 1..1 */

< ***Cardinaly_Sheap*** >:=< ***Number***> .. (< ***Number***> | n)
<***AttributeList***>:={,<***AttributeName***>}

KDQL adopts a DMQL association rules syntax in [12] joining it with SQL+D for a graphical representation aspect [29]. KDQL RULES are comparisons of two kinds of query syntaxes. One of them is the data mining query language (DMQL) and the other is a visual query. KDQL will facilitate both syntaxes for a high level data mining and natural integration with relational query language SQL. We will introduce both syntaxes.

DMQL syntax:-

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

Mining association rules:
<rule_spec> ::=

find association rules [*as <rule_name>*]

SQL+D syntax:-

<***query***>➔ <***SQL Query***>| <SQL Query> ***DISPLAY*** <disp specs>

<***disp*** specs>➔ panel <***id***>, <container panels>
WITH<disp element>
[***SHOW*** <presentation>]
[***AUTOSKIP*** <time length>]

<***container panels***> ➔
<panel list> ON <id>.<loc>[(<layout>)] |
<panel list> ON <id>.<loc >[(<layout>)],
<container panels>

<***panel list***> ➔ panel <id> |
panel <id>, <panel list>

<***id***> ➔ <alphabetic> | <alphabetic> <alphanumeric>

<***loc***>➔ North | South | East | West | Center.

<***layout***>➔ Horizontal | Vertical | Overlay.

< ***chart specs***>➔ (AttrX, AttrY) ***AS*** <chart elem> ***ON*** <id> [<trigger>] | (AttrX, AttrY, AttrZ) ***AS*** <chart elem> ***ON*** <id> [<trigger>]

< ***chart elem***> ➔ linechart | barchart | piechart | xyscatter.

A.2 Syntax examples

A.2.1 DMQL example for discovering association rules

Querying the data mining query system to find strong association relationships for those students majoring in computing science and born in *"Hungary"*, in relevance to the attributes *gpa, birth_place* and *address*. We are only concerned with the structure of the query.

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

A.2.2 SQL+D example for visual representation

For charts representation we have to consider a database to keep statistical information about the recreational activity preferred by people in different states. The schema looks as follows:

ACTIVITY (state, activity, pop)

where all the attributes are character strings. To see how the people in *Hjdu Bihar*, spend their free time, we might submit this query, we included both a bar chart and a pie chart , to see the different displays that might be obtained with the same data. The resulting will be display in a dialog in [29] if we use this tool.

SELECT
ACTIVITY.activity
ACTIVITY.population
FROM
ACTIVITY
WHERE
ACTIVITY.state = " Hjdu Bihar"
DISPLAY
PANEL main
WITH
(activity, pop) AS barchart ON main.West,
(activity, pop) AS piechart ON main.East,

"Recreational Activities Preferences" AS boxedtext ON main.North, the query will display a bar chart of preferred recreational activities. The two-attribute line chart is similar to the *bar chart*. A categorized chart will group values on *AttrZ* (categories) and plot all the resulting *AttrX, AttrY* series in the same *xy* plane, using a different color for each category.

A.2.3 KDQL example for discovering association rules

Referring to table no 9.3 in Chapter 9, we are interesting in a rule *describes the regularities of purchased items in customer transactions, such as {cheese, coke} ⇒ bread.* This rule *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)⇒ 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.

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

Appendix B

I-extended database & KDQL views

I-extended database and also the KDQL program was implemented by using *Delphi programming language version 5.0,* the underlying programming language source code of Delphi is *Pascal Source Code*.

Starting with i-extended database and KDQL to select and retrieve traditional databases via ODBC or JDBC we can get the first interface follows in i-extended database mode.

- ***Interface (1):***

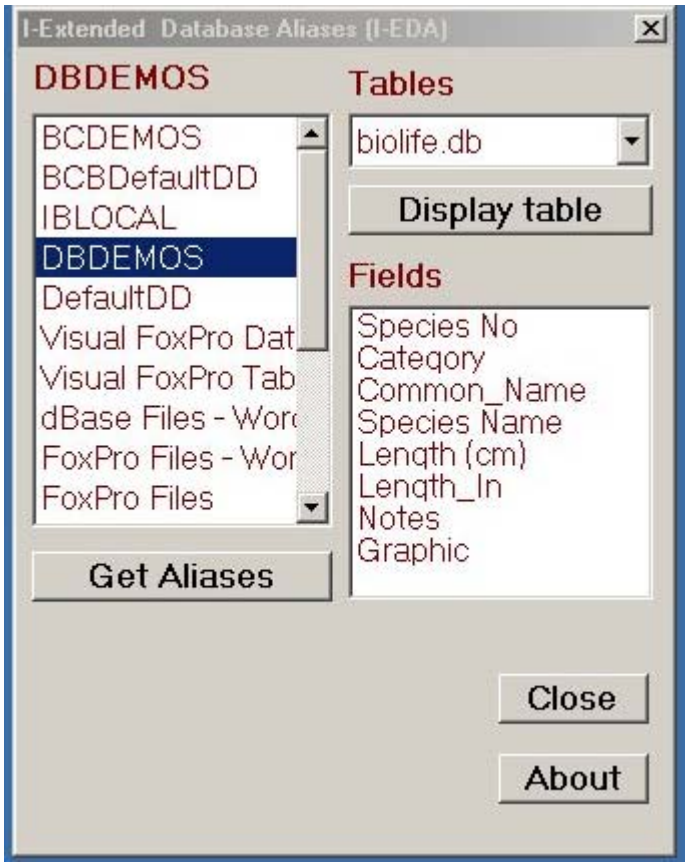

Figure B.1 Interface (1)

In this Interface we get all the database aliases which are located on you local PC and by pressing on **Get Aliases** button the program will seek for all aliases in the local hard disk and listed it in the Aliases list. The user is asked to choose on of the database aliases. After choosing the database the name of the chosen database aliases

will appear on the top of the list **DBDEMOS**. In the table section we can chose any table from the list that holds all the tables' name **Tables**. See interface (2).

- ***Interface (2):***

Figure B.2 Interface (2)

In interface(2) the user already selected the database table which is **biolife.db** in this database table we can also see the fields and we also may know the types of each field by pointing on it and the type will appear below like **Common_Name** field the type of the field is "ftString". We can also close completely the program by clicking on the key **Close**. But if we click on the button **About** we will get interface (3).

- ***Interface (3):***

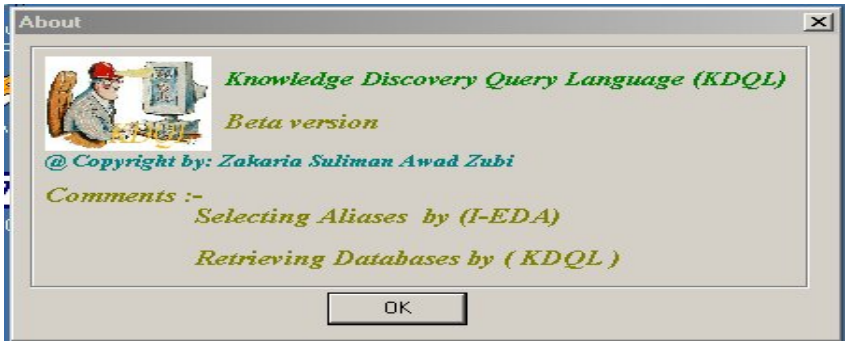

Figure B.3 Interface (3)

In this interface it shows only the **About** dialog and if we click on the **OK** bottom it will go back to the interface (2) and in interface (2) again you can to the **biolif.db** database table that you have selected. See interface (4).

- ***Interface (4):***

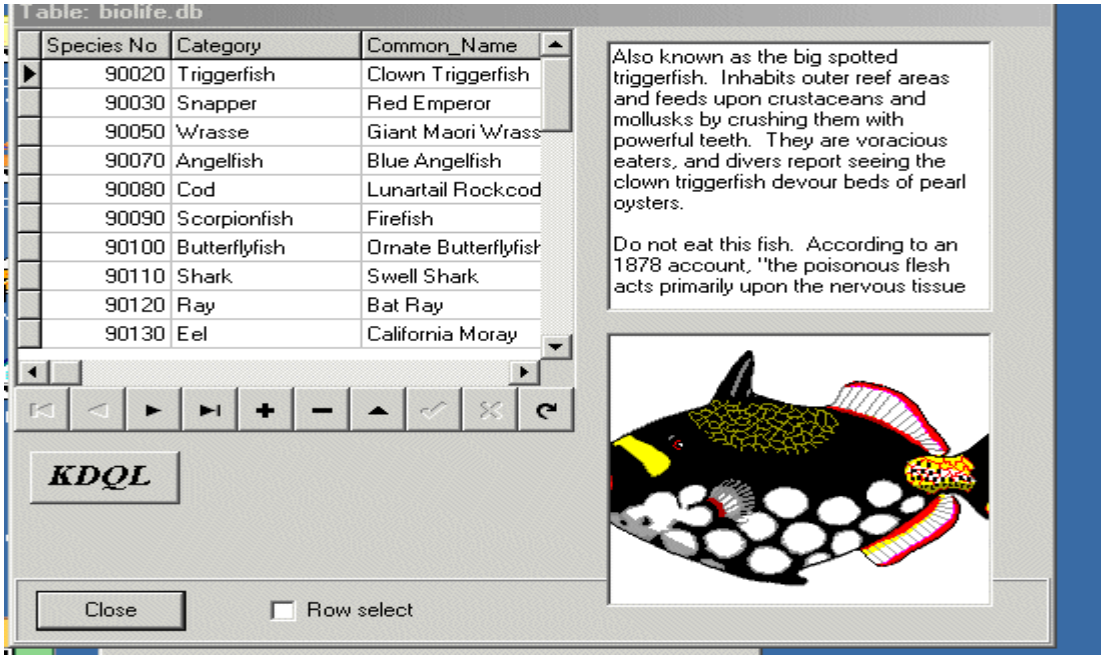

Figure B.4 Interface (4)

In this interface we will get the database table that we have chosen from the list. In this database we view all the storage that is related to each value in the database table. In interface (4) we view the values comments and also images for each particular value. This will give i-extended database reliability. After viewing the contents of the table we can apply the KDQL to retrieve the database table by clicking on the bottom **KDQL** . This will take us to KDQL program which will be achieved in interface (5).

- ***Interface(5):***

Figure B.5 Interface (5)

Interface (5) was called from i-extended database program to retrieve selected database and to work with this mode we should see interface (6).

- ***Interface (6):***

Figure B.6 Interface (6)

By clicking on the bottom **KDQL Text Capture** the program will grape the current SQL syntax and past it on the KDQL syntax console and then the user can change or modify the query the way he/she wants. To execute the query we should see the next interface.

- ***Interface (7):***

Figure B.7 Interface (7)

To execute a query written on the console all we have to do is to click on the key labeled **Execute KDQL**. After clicking the user can be able to see the results in different forms on interface (8).

- ***Interface (8):***

Figure B.8 Interface (8)

The first result that we can see is the retrieved database table and we can view it only if we click on the tabbed sheet labeled **KDQL Phase2 (Retrieved Table)** on the top of the notebook tabbed main form of the program. To see visualization data we should see the other interfaces.

- ***Interface (9):***

Figure B.9 Interface (9)

The results in the retrieved database table will be presented in four different 2D charts forms such as **Line, Bar, Pie and Points.** We can see this 2D charts form by clicking on the tabbed sheet labeled **KDQL 2D Charts** on the top of the notebook

tabbed main form of the program. We can see the 3D charts which will be viewed in Interface 10.

- ***Interface (10):***

Figure B.10 Interface (10)

The same charts as the 2D can be visualized in the 3D tabbed sheet. These results visualize the same data as the 2D did and the only different is that it is in the 3D mode. We can see this 3D charts form by clicking on the tabbed sheet labeled **KDQL 3D Charts** on the top of the notebook tabbed main form of the program. For a conclusion to the user we can see the average of the data in the final vision of the program in interface (11).

- ***Interface (11) :***

Figure B.11 Interface (11)

Knowing the average of the data values in the table helps the user to make his own decisions regarding the requested data that he is looking for. In interface (11) such an average could be calculated and then visualized in 3D mode.

Bibliography

[1] Usama M. Fayyad, *Data Mining and Knowledge Discovery*, 1, 5–10 (1997), 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[2] Kurt Thearling, *Keys to the Commercial Success of Data Mining*, A workshop held in conjunction with The Fourth International Conference on Knowledge Discovery and Data Mining (KDD '98), P-11,New York City, August 31, 1998.

[3] Awad Zakaria, Fazekas Gabor **,** *On Some Software Tools For Data Mining* **,**4th International Conference on Applied Informatics**,** Eger-Noszvaj, Hungary, pages 331-336 30 August-3 September 1999.

[4] M. S. Chen, J. Han, and P. S. Yu. *Data mining: An overview from a database perspective*. IEEE Trans.Knowledge and Data Engineering, 8:866-883, 1996.

[5] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. P-6, AAAI/MIT Press, 1996.

[6] W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus, *Knowledge Discovery in Databases: An Overview*,P-11, AAAI/MIT Press, 1996.

[7] In G. Piatetsky-Shapiro et al. (eds.), *Knowledge Discovery in Databases*. AAAI/MIT Press, P-8,1991.

[8] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann,P-13, 2000.

[9] T. Imielinski and H. Mannila. *A database perspective on knowledge discovery*. Communications of ACM, 39:58-64, 1996.

[10] G. Piatetsky-Shapiro, U. M. Fayyad, and P. Smyth. *From data mining to knowledge discovery: An overview*. In U.M. Fayyad, et al. (eds.), Advances in Knowledge Discovery and Data Mining, 1-35. AAAI/MIT Press, 1996.

[11] Zakaria Awad, *The Standard Open Database Connectivity (ODBC)*, Technical Report No. 2000/02, Institute of Mathematics and Informatics, University of Debrecen, P-12, Debrecen, Hungary, 2000.

[12] Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: *DMQL: A Data Mining Query Language for Relational Databases*. In Proc. 1996 SiGMOD' 96 Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD'96), pages 27-34, Montreal, Canada, June 1996.

[13] O. R. Zaiane, J. Han, Z. N. Li, J. Y. Chiang, and S. Chee, ",: *MultiMediaMiner: A System Prototype for MultiMedia Data Mining* , *Proc. 1998 ACM-SIGMOD Conf. on Management of Data*, (system demo), Seattle, Washington, June 1998, pp. 581-583.1998.

[14] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong,K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaiane. *DBMiner: A system for mining knowledge in large relational databases*. In Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96), pages 250--255, Portland, Oregon, August 1996.

[15] B. Mobasher, N. Jain, E. Han, and J. Srivastava. *Web mining: Pattern discovery from world wide web transactions*. Technical Report TR 96-050, University of Minnesota, Dept. of Computer Science, Minneapolis, 1996.

[16] R.J. Brachman, P.G.. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D.L. McGuinnes, and L.A. Resnick. *Integrated support for data archeology*. International Journal of Intelligent and Cooperative Information Systems, 2(2):159--185, 1993.

[17] Keim, D.A., (1997). *Visual Techniques for Exploring Databases*. Tutorial Notes in the Third International Conference on Knowledge Discovery and Data Mining, KDD-97. Newport Beach, CA, August, 1997.

[18] Pyle, D., (1999). *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, March, 1999.

[19] Pickett, R.M., Grinstein, G.G., (1988). *Iconographic Displays for Visualizing Multi-dimensional Data*. Proceedings of the 1988 IEEE Conference on Systems, Man, and Cybernetics, Beijing and Shenyang, China, August 1988, pp. 514-519.

[20] Grinstein, G.G., (1996). *Harnessing the Human in Knowledge Discovery*. Invited paper in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD96), Portland, Oregon,(edited by Simoudis, E., Han, J., Fayyad, U.), August, 1996, pp. 384-386.

[21] Hoffman, P.E., (1998). *Array Visualizations: A Formal Model and its Applications*. Doctoral Thesis Proposal, Department of Computer Science, University of Massachusetts Lowell.

[22] Inselberg, A., Dimsdale, B., (1990) . Parallel Coordinates: *A Tool for Visualizing Multi-dimensional Geometry*. In Proceedings of the IEEE Visualization 90 Conference, San Francisco, CA, pp. 361-370.

[23] John, G.H., (1997). *Enhancements to the Data Mining Process*. Doctoral Dissertation, Department of Computer Science, Stanford University.

[24] Brachman, R., Selfridge, P., Terveen, L., Altman, B., Halper, F., Kirk, T., Lazar, A., McGuinness, D., Resnick, L., Borgida, A., (1993). *Integrated Support for Data Archaeology*. International Journal of Intelligent and Cooperative Information Systems, 2(2), pp. 159-185.

[25] ReMind, (1992). *ReMind Developer's Reference Manual*. Boston, MA, Cognitive Systems, Inc.

[26] Awad Zakaria, Fazekas Gábor, *On ODBC_KDD models*, paper,5th International Conference on Applied Informatics, dedicated to the 70th birthday of Prof. Mátyás Arató´ and Prof. László Varga, 28 January-3 February 2001, P-13, Eger, Hungary,2001.

[27] Sunita Sarawagi, Shoilby Thomas, Rakesh Agrawal, *Integration Association Rule Mining with relational database system : Alternatives and Implications*, IBM Almaden Research Center, research, pages 95-120,650 Harry Road, San Jose, CA, 1995.

[28] Jeffery D. Ullman, Principle of Database Knowledge-Base System, book, volume I and II, Chapter 2 in I, pages 33-95, Stanford University, Computer Science Press, 1803 Research Boulevard, Rockville, Maryland ,USA,1976.

[29] Graciela Gonzalez, Chitta Baral and Amarendra Nandigam, *SQL+D: Extended Display Capabilities for Multimedia Database Queries*, paper, ACM Multimedia 98 - Electronic Proceedings,USA, 1998.

[30] Hienhuys-Cheng, S. *Data Mining: From Statistics to ILP*. IJCAI97 Workshop "Frontiers of Inductive Logic Programming". Ed. De Raedt, L. 1997.

[31] Liu, C., Zhong, N., Oshuga S. *Constraint Inductive Logic Programming and its Application to Knowledge Discovery in Databases*. IJCAI97 Workshop "Frontiers of Inductive Logic Programming". Ed. De Raedt, L. 1997.

[32] Shen, W-M. Leng, B. *Metapattern Generation for Integrated Data Mining*. Procs of the 2nd International Conference on Knowledge Discovery and Data Mining, KDD-96. Ed. Simoudis, E., Han, J., Fayyad, U. AAAI Press. 1996.

[33] Blockeel H., De Raedt L. *Relational Knowledge Discovery in Databases*. ICML-96 Workshop on ILP & KDD. Ed. Fuernkranz, J., Pfahringer, B. 1996.

[34] Tausend, B. *Representing Biases for Inductive Logic Programming*. Ed. Bergadano, F., De Raedt, L. Proc. of European Conference on Machine Learning. Springer-Verlag. 1997.

[35] Padmanabhan, B., Tuzhilin, A. *Pattern Discovery in Temporal Databases: A Temporal Logic Approach*. Proceedings of the Third Internatinoal Conference On Knowledge Discovery and Data Mining, KDD-97. Ed. Heckerman, D., Mannila, H., Pregibon, D., Uthurusamy, R. AAAI Press. 1997.

[36] Srinivasam, A., King, R., Muggleton, S., Sternberg, M. *Carcinogenesis Prediction Using ILP*. Proceedings of ILP-97. Ed. Lavrac, N, Dzeroski, S. Springer-Verlag. 1997.

[37] De Raedt, L., Dehaspe, L. *Clausal Discovery*. Machine Learning, 26, 99-146. 1997.

[38] Wrobel, S. *ILP for KDD. ILP & KDD Summer School on Inductive Logic Programming and Knowledge Discovery in Databases*, Lecture Notes. Org. Dzeroski, S., Lavrac, N. 1997

[39] Imielinski, T., Virmani, A., Abdulghani, A. *DataMine: Application Programming Interface and Query Language for Database Mining*. Procs of the 2nd International

Conference on Knowledge Discovery and Data Mining, KDD-96. Ed. Simoudis, E., Han, J., Fayyad, U. AAAI Press. 1996.

[40] Muggleton, S., De Raedt, L. *Inductive Logic Programming*. The Journal of Logic Programming., special issue Ten Years of Logic Programming. Vol. 19, 20. 1994.

[41] Dehaspe L. De Raedt L. *Mining Association Rules in Multiple Relations*. Proceedings of ILP-97. Ed. Lavrac, N, Dzeroski, S. Springer-Verlag. 1997.

[42] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, I. *Fast Discovery of Association Rules*. Advances in Knowledge Discovery and Data Mining. Ed. U. Fayyad, G. Piatetski-Shapiro, P. Smyth and R. Uthurusamy. MIT Press. 1996.

[43] Heikki Mannila, *Inductive databases*, ILP'99 invited talk, www.research.microsoft.com/~mannila/, 1999.

[44] Flach, P. *Conjectures: an inquiry concerning the logic of induction*. PhD thesis. ITK-dissertation series 1. 1995

[45] Pazzani M., Mani S., Shankle W.R. *Beyond Concise and Colorful: Learning Intelligible Rules*. Proceedings of the Third Internatinoal Conference On Knowledge Discovery and Data Mining, KDD-97. Ed. Heckerman, D., Mannila, H., Pregibon, D., Uthurusamy, R. AAAI Press. 1997.

[46] Blockeel H., De Raedt L. *Relational Knowledge Discovery in Databases*. ICML-96 Workshop on ILP & KDD. Ed. Fuernkranz, J., Pfahringer, B. 1996.

[47] Weber, I. *Discovery of First-Order Regularities in a Relational Database Using Offline Candidate Determination*. Proceedings of ILP-97. Ed. Lavrac, N, Dzeroski, S. Springer-Verlag. 1997.

[48] Lavrac, N., Weber, I. Zupanic, D., Kazakov, D., Stepankova, O. and Dzeroski, S. *ILPNET repositories on WWW: Inductive Logic Programming systems, datasets and bibliography*. AI Communications 9 (1996). IOS Press.

[49] Feng, C., Muggleton, S. *Towards Inductive Generalization in Higher Order Logic*. Proceedings of ML92. Ed. Derek Sleeman and P. Edwards. Morgan Kauffmann. 1992.

[50] Bergadano, F., Gunetti, D. *Inductive Logic Programming*: *From Machine Learning to Software Engineering*. MIT Press. 1996.

[51] Jorge, A. *Iterative Induction of Logic Programs: an approach to logic program synthesis from incomplete specifications*. PhD thesis. 1997.

[52] Fayyad, U. *Knowledge Discovery in Databases: An Overview*. Proceedings of ILP-97. Ed. Lavrac, N, Dzeroski, S. Springer-Verlag. 1997.

[53] Lorenzo, D. *Application of Clausal Discovery to Temporal Databases*. ICML-96 Workshop on ILP & KDD. Ed. Fuernkranz, J., Pfahringer, B. 1996.

[54] Cohen, W. *Gramatically Biased Learning: Learning Logic Programs Using an Explicit Antecedent Description Grammar*. Artificial Intelligence, 68 (2):303-366, 1994.

[55] Kietz, J-U., Wrobel, S. *Controlling the Complexity of Learning in Logic.* Inductive Logic Programming. Ed. Stephen Muggleton. Academic Press Ltd. 1992.

[56] T. Fakuda, Y. Morimoto, S. Morishita: *Constructing Enceinte Decision Trees by Using Optimized Numeric Association Rules*. In Proceedings of Int'l Conf on Very Large Databases, (1996).

[57] V. Ganti, J. Gehrke, R. Ramakrishnan: *DEMON: Data Evolution and Monitoring*. In Proceedings of ICDE, (2000)

[58] S. K. Gupta, V. Bhatnagar, S. K. Wasan, D. Somayajulu: Intension Mining: A New Paradigm in Knowledge Discovery. Technincal Report No. IITD/CSE/TR2000/001, Indian Institute of Technology, Delhi, INDIA, (2000).

[59] A. Hafez, V. V. Raghvan, J. Deogun: *The Item-Set Tree: A Data Structure for Data Mining.* In Proceedings of Ist Int'l Conf. on Data Warehousing and Knowledge Discovery, pages 183{192, Aug (1999).

[60] J. Han, J. Pei, Y. Yin: *Mining Frequent Patterns without Candidate Generation*. In Proceedings of Int'l Conf SIGMOD 2000, (2000).

[61] W. A. Kosters, E. Marchiori, A. A. J. Oerlemans: *Mining Clusters with Association Rules*. In Proceedings of III Int'l Symposium on Intellegent Data Analysis, pages 39{50, (Aug 1999).

[62] B. Liu, W. Hsu, et al: *Discovering Interesting Knowledge using DM-II*. In Proceedings of Int'l Conf on Knowledge Discovery and Data Mining, (Aug 1999).

[63] B. Liu, W. Hsu, Y. Ma.: *Integrating Classification and Association Rule Mining* . In Proceedings of Int'l Conf on Knowledge Discovery and Data Mining , (1998).

[64] B. Liu, W. Hsu, Y. Ma.: Pruning , *Summarizing the Discovered Associations* . In Proceedings of Int'l Conf on Knowledge Discovery and Data Mining (KDD-99), (Aug 1999).
[65] R. Meo: *A New Approach for Discovery of Frequent Itemsets*. In Proceedings of 1'st Int'l Conference on Data Warehousing and Knowledge Discovery, (Aug 1999).

[66] B. Uma: *Incremental Association Rule Alogorithm for Intension Mining*. Master's thesis, Indian Institute of Technology, New Delhi, India. (1999).

[67] A. Virmani: *Second Generation Data Mining: Concepts and Implementation*. PhD thesis, Rutgers University, NJ, USA, (1998).

[68] K. Wang, X. Chu, B. Liu: *Clustering Transactions Using Large Items*. In Proceedings of ACM CIKM-99, (1999).

[69] M. J. Zaki, S. Parthasarthy, et al.: *New Algorithms for Fast Discovery of Association Rules*. In Proceedings of 3rd Int'l Conf. on Knowledge Discovery and Data Mining, (Aug 1997).

[70] B. Ozden, S. Ramaswamy, and A. Silberschatz. *Cyclic association rules*. ICDE'98, 412-421, Orlando, FL.

[71] R. Agarwal, C. Aggarwal, and V. V. V. Prasad. *A tree projection algorithm for generation of frequent itemsets*. In Journal of Parallel and Distributed Computing Special Issue on High Performance Data Mining), 2000.

[72] 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.

[73] G. Grahne, L. Lakshmanan, and X. Wang. *Efficient mining of constrained correlated sets*. ICDE'00, 512-521, San Diego, CA, Feb. 2000.

[74] R. Agrawal , R. Srikant: *Fast Algorithm for Mining Association Rules*. In Proceedings of 20th VLDB Conference, (1994).

[75] Bhavani Thuraisingham, *Data Mining: Technologies, Techniques, Tools and Trends*, Book, CRC Press, USA, ISBN: 0-8493-1815-7, 1999.

[76] Pieter Adriaans, Dolf Zantinge, *Data Mining*, Book, Addison Wesley Longman, ISBN 0-201-40380-3, Harlow, England, 1996.

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

[78] Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, book second edition, Chapter 7, pages 158-222, Redwood City, CA 94065,USA,1994.

[79] Antony Galton, *Logic for Information Technology*, book, John Wiley and Sons, 1990.

[80] H. Mannila. *Methods and problems in data mining*. In ICDT'97, volume 1186 of LNCS, pages 41--55. SpringerVerlag, 1997.

[81] R. Agrawal, T. Imielinski, and A. Swami. *Mining association rules between sets of items in large databases*. In SIGMOD'93, pages 207 -216, May 1993. ACM.

[82] P. Smyth and R. M. Goodman. *An information theoretic approach to rule in duction from databases*. IEEE Transactions on Knowledge and Data Engineering, 4(4):301 -- 316, August 1992.

[83] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. *Dynamic itemset counting and implication rules for market basket data*. In SIGMOD'97, pages 255 -- 264, 1997. ACM Press.

[84] S. Guillaume, F. Guillet, and J. Philipp'e. *Improving the discovery of association rules with intensity of implication*. In PKDD'98, volume 1510 of LNAI, pages 254 -- 262, September 1998. Springer- Verlag.

[85] J.F. Boulicaut. *A KDD framework to support database audit*. In WITS'98, volume TR 19, pages 257 - 266, December 1998. University of Jyväskylä.

[86] R. Srikant, Q. Vu, and R. Agrawal. *Mining association rules with item constraints*. In KDD'97, pages 67 -- 73, 1997. AAAI Press.

[87] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. *Query flocks: A generalization of association rule mining.* In SIG MOD'98, pages 1 -- 12, 1998. ACM Press.

[88] L. Dehaspe and H. Toivonen. *Frequent query discovery: A unifying ILP ap proach to association rule mining*. Technical Report CW258, Department of Com puter Science, Katholieke Universiteit Leuven, Belgium, March 1998. Available at http://www.cs.kuleuven.ac.be/ ,publicaties/rapporten/ CW1998.html.

[89] J. M. Hellerstein. *Optimization techniques for queries with expensive methods*. ACM Transaction on Database Systems, 1998. Available at http://www.cs.berkeley.edu/ ,jmh/miscpapers/todsxfunc.ps.

[90] M. Klemettinen, H. Mannila, and H. Toivonen. *Rule discovery in telecommunica tion alarm data*. Journal of Network and Systems Management, 1999. To appear.

[91] A. Tuzhilin. *A pattern discovery algebra*. In SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Technical Report 9707 University of British Columbia, pages 71 -- 76, 1997.

[92] Randall Davis. *Meta-rules: Reasoning about control*. Artificial Intelligence, 15(3):179--222, 1980

[93] G. Lausen and G. Vossen. *Models and Languages of Object-Oriented Databases*. Addison-Wesley Publishing Company, 1997.

[94] J.-F. Boulicaut, M. Klemettinen, and H. Mannila. *Modeling KDD processes within the inductive database framework*. Technical Report C-1998-29, Department of Computer Science, P.O. Box 26, FIN-00014 University of Helsinki, Finland, June 1998. Submitted.

[95] R. Meo, G. Psaila, and S. Ceri. *A new SQL-like operator for mining association rules*. In VLDB'96, pages 122--133. Morgan Kaufmann, 1996.

[96] H. Mannila. *Inductive databases and condensed representations for data mining*. In ILPS'97, pages 21--30. MIT Press, 1997.

[97] R. Meo, G. Psaila, and S. Ceri. *A tightly-coupled architecture for data mining*. In ICDE'98, pages 316--322. IEEE Computer Society Press, 1998.

[98] W. Kloesgen. *Efficient discovery of interesting statements in databases*. Journal of Intelligent Information Systems, 4(1):53 -- 69, 1995.

[99] C. J. Matheus, G. PiatetskyShapiro, and D. McNeill. *Selecting and reporting what is interesting*. In U. M. Fayyad, G. PiatetskyShapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 495 -515. AAAI Press, Menlo Park, CA, 1996.

[100] L. De Raedt and M. Bruynooghe. *A theory of clausal discovery*. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), pages 1058 -- 1053, Chamb'ery, France, 1993. Morgan Kaufmann.

[101] L. De Raedt and S. DŸzeroski. *First order jk-clausal theories are PAC-learnable*. Artificial Intelligence, 70:375 -- 392, 1994.

[102] H. Mannila and K.J. R¨aih¨a. *Design by example: An application of Armstrong relations.* Journal of Computer and System Sciences, 33(2):126 -- 141, 1986.

[103] M. Jaeger, H. Mannila, and E. Weydert. *Data mining as selective theory extraction in probabilistic logic*. In R. Ng, editor, SIGMOD'96 Data Mining Workshop, The University of British Columbia, Department of Computer Science, TR 9608, pages 41--46, 1996.

[104] H. Mannila and H. Toivonen. *On an algorithm for finding all interesting sentences.* In Cybernetics and Systems, Volume II, The Thirteenth European Meeting on Cybernetics and Systems Research, pages 973 -- 978, Vienna, Austria, Apr. 1996.

[105] M. Holsheimer, M. Kersten, and A. Siebes. *Data surveyor: Searching the nuggets in parallel*. In U. M. Fayyad, G. PiatetskyShapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 447-467. AAAI Press, Menlo Park, CA, 1996.

[106] Z. Galil and E. Ukkonen, editors, *On a combinatorial pattern*, 6th Annual Symposium on Combinatorial Pattern Matching (CPM 95), volume 937 of Lecture Notes in Computer Science, Berlin, 1995. Springer.

[107] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. *A perspective on databases and data mining*. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95), pages 150-155, Montreal, Canada, Aug. 1995.

[108] A. Savasere, E. Omiecinski, and S. Navathe. *An efficient algorithm for mining association rules in large databases*. In Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95), pages 432-444, Zurich, Swizerland, 1995.

[109] H. Toivonen. *Sampling large databases for association rules*. In Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96), pages 134-145, Mumbay, India, Sept. 1996. Morgan Kaufmann.

[110] R. Agrawal and R. Srikant, *Mining generalized association rules*. In Process 1995 Int. Conf. Very Large Data Bases, Zrich, Switzerland, Sept., 1995.

[111]H. Mannila and H. Toivonen. *Multiple uses of frequent sets and condensed representations*. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96), pages 189-194, Portland, Oregon, Aug. 1996. AAAI Press.

[112]J. Kivinen and H. Mannila. *The power of sampling in knowledge discovery*. In Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94), pages 77 -- 85, Minneapolis, MN, May 1994.

[113]J. Kivinen and H. Mannila. *Approximate dependency inference from relations*. Theoretical Computer Science, 149(1):129 -- 149, 1995.

[114] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: *A relational aggregation operator generalizing group-by, cross-tab, and sub-totals*. In 12th International Conference on Data Engineering (ICDE'96), pages 152 -- 159, New Orleans, Louisiana, Feb. 1996.

[115] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, New York, 1993.

[116] J. Han and Y. Fu. *Exploration of power of attribute- oriented induction in data mining*. In U. M. Fayyad, G. Piatestsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Advanced in Knowledge Discovery and Data Mining. AAAI / MIT Press, 1995.

[117] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. *Finding interesting rules from large sets of discovered association rules*. In Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), pages 401 -- 407, Gaithersburg, MD, Nov. 1994. ACM.

[118] Brockhausen,P., Morik, K. *Direct Access of an ILP Algorithm to a Database Management System*. ICML-96 Workshop on ILP & KDD. Ed. Fuernkranz, J., Pfahringer,B. 1996.

[119] *MERANT ,Inc. Develops and markets tools*, www.merant.com.

[120] SYWARE, Inc. *Developer of intuitive database tools for business and personal productivity,* www.syware.com.

[121] Zakaria Awad, *Using Analysis Tools to Represent Data Mining Solutions,* Technical Report No. 99/81, Institute of Mathematics and Informatics, University of Debrecen, P-12,Debrecen, Hungary, 1999.

[122] Zakaria Awad, Gábor Fazekas, *Data Mining Query Languages*, Technical Report No: 2001/15, Preprints No. 273, Institute of Mathematics and Informatics, University of Debrecen, P-13,Debrecen, Hungary, 2001.

[123] Mohamed Omar, Ali Miloud and Zakaria Awad, *Smart Query Answering by KDD Techniques*, paper, 3rd International Conference on Computer Science, In print, Tripoli, Libya, 2001.

Ismeretfeltárás Távoli Adatbázisokból

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében

Matematika és Számítástudományok (Informatika) tudományágban

Írta: ***Zakaria Suliman Awad Zubi*** okleveles programtervező matematikus

Készült

A Debreceni Egyetem Matematika és Számítástudományok doktori iskola Informatika programja keretében.

Témavezető: ***Dr. Fazekas Gábor***

A doktori szigorlati bizottság:

elnök*:* ***Dr. Kormos János .....***
tagok *:* ***Dr. Kuki Attila.....***
Dr. Hajas Csilla.....

A doktori szigorlat időpontja: 2001. március 8.

Az értekezés bírálói:

Dr……
Dr…….....

A bírálóbizottság:

elnök: ***Dr……...***
tagok: ***Dr……...***
Dr…….
Dr…….
Dr…….

Az értekezés védésének idôpontja: 2002 /...../.....