

A LAYERED OPTIMIZER FOR MINING ASSOCIATION RULES OVER
RELATIONAL DATABASE MANAGEMENT SYSTEMS

By

MAHESH DUDGIKAR

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2000

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Sharma Chakravarthy for giving me an opportunity to work on this interesting topic and for providing me with excellent advice, great guidance and support throughout the course of this research work. His tireless patience and knowledge have been invaluable. Without his guidance, this thesis would not have been possible.

Next, I would like to thank my family back home for their unwavering moral, endless love and financial support.

I am grateful to Dr. Joachim Hammer and Dr. Herman Lam for graciously agreeing to serve on my committee and for taking out time from their busy schedules to read and comment on my thesis.

Many thanks go to Sharon Grant for a well administered research environment and for her help with everything. Her attention to every minute detail is really appreciated.

I would like to thank Shiby Thomas for his knowledge that he shared with me, and the long discussions we had at the start of this research.

Special thanks go to Hongen Zhang for the endless discussions and tireless work throughout the phases of design and implementation of this entire work.

Thanks also go to my friends Kiran, Sridhar, Sujith and Yadu, who made me feel at home and for being with me all the way.

Sincere thanks go to Ashwin for the long discussions and debates we had over technical matters that made both of us understand things better. Thanks go to all my friends at the database center for the chat sessions, lunch sessions, and so on. Lastly, I thank the CISE department and Matthew Belcher for providing me with all the computing facilities I needed. I gratefully acknowledge all the support.

TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGMENTS	iii
LIST OF TABLES	viii
LIST OF FIGURES	x
ABSTRACT	xii
1 INTRODUCTION	1
1.1 Data Warehouse	1
1.2 Data Mining	3
1.2.1 Association Rules.....	5
1.2.2 Sequential Patterns	6
1.2.3 Clustering and Classification	7
1.2.4 Visualization.....	10
1.3 Problem Statement.....	11
1.4 Approach.....	12
1.4.1 Mapping	15
1.4.2 Metadata.....	15
1.5 Thesis Organization.....	16
2 DATABASE CONNECTIVITY ISSUES	17
2.1 Introduction.....	17
2.2 Basic JDBC Architecture	18
2.2.1 Two Tier Architecture.....	19
2.2.2 Three Tier Architecture.....	20
2.3 Detailed Architecture	21
2.4 Stored Procedures	23
2.5 User Defined Functions	25
2.6 Stored Procedures in Oracle.....	26
2.7 User Defined Functions in DB2 (UDB™).....	28
2.8 Conclusions	32

3	ASSOCIATION RULES	33
3.1	Introduction.....	33
3.2	Apriori Algorithm.....	34
3.3	Input and Output Formats	35
3.4	Candidate Generation.....	36
3.5	Support Counting.....	38
3.5.1	K-way Join	38
3.5.2	Two Group By	39
3.5.3	Subquery Based.....	40
3.5.4	Vertical.....	42
3.5.5	Gather Join.....	46
3.5.6	Gather Count	49
3.6	Rule Generation	50
3.7	Conclusion	52
4	INPUT MAPPING AND METADATA.....	53
4.1	Introduction.....	53
4.2	Mapping	53
4.3	Reverse Mapping	58
4.4	Metadata.....	59
4.5	Conclusions	61
5	PERFORMANCE, SCALING AND COMPARISONS.....	62
5.1	Introduction.....	62
5.2	Synthetic Data Generation	62
5.3	Performance Comparisons of SQL-92 approaches.....	63
5.4	Scaling.....	66
5.5	Comparison with Other Commercial Mining Tools	67
5.6	Conclusions	69
6	SYSTEM IMPLEMENTATION.....	70
6.1	Introduction.....	70
6.2	Mapping and Rule Generation.....	70
6.3	Conclusions	81
7	CONCLUSIONS.....	82
7.1	Contributions.....	83
7.2	Proposed Extensions and Future Work.....	84

APPENDIX RULE GENERATION EXAMPLE USING VERTICAL APPROACH	86
A.1 Support Counting	88
A.1.1 First Pass	88
A.1.2 Second Pass	88
A.1.3 Third Pass	88
A.2 Rule Generation	89
LIST OF REFERENCES	92
BIOGRAPHICAL SKETCH	94

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1.1 Training Data set	9
2.1 The input and output for the saveItem stored procedure.....	28
2.2 The input and output for the saveItem UDF.....	32
4.1 Input table	56
4.2 MappedTidsTable	56
4.3 MappedItemsTable	56
4.4 Final Input.....	57
4.5 MappedTidsTable with new Tid field.....	58
4.6 Final Input table with new Tid Field	58
5.1 Description of the generated datasets	63
A-1 InputTableOne	86
A-2 InputTableTwo.....	86
A-3 MappedTidsTable	87
A-4 MappedItemsTable.....	87
A-5 FinalInput table.....	87
A-6 TID lists of each of the items	88
A-7 Table F_1	88
A-8 Table F_2	88
A-9 Table F_3	89

A-10 Table FISETS.....	89
A-11 Table SUBSETS.....	90
A-12 Table RULES	90
A-13 Table RULES with items mapped back to descriptions.....	91

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1.1 Constructed Decision Tree	10
1.2 Architectural alternatives	13
1.3 The proposed architecture for association rule mining	14
2.1 Two tier architecture	19
2.2 Three tier architecture	20
2.3 Detailed architecture	22
3.1 Candidate generation for any k	37
3.2 Support counting by K-way join approach.....	39
3.3 Tree diagram for subquery Q_i	42
3.4 Rule Generation.....	52
5.1 Performance comparison for dataset T5D10K.....	64
5.2 Performance comparison for dataset T10D10K.....	64
5.3 Performance comparison for dataset T5D100K.....	65
5.4 Pass wise comparison of Kway and Subquery approaches for T5D10K dataset with 0.20% support.	66
5.5 Number of Transactions scale-up	67
6.1 Main Window of the association rule miner	71
6.2 Sub-menus of menu item file.....	72
6.3 The login window	72

6.4 Parameter input window	74
6.5 SQL input window.....	75
6.6 Join column selection	76
6.7 User selection columns	77
6.8 Items' columns selection.....	77
6.9 The user specified WHERE clause.....	78
6.10 TID columns' selection.....	79
6.11 End of mapping and options for support counting	80
6.12 End of support counting	81

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

A LAYERED OPTIMIZER FOR MINING ASSOCIATION RULES OVER
RELATIONAL DATABASE MANAGEMENT SYSTEMS

By

Mahesh Dugdikar

August, 2000

Chairman: Dr. Sharma Chakravarthy

Major Department: Computer and Information Science and Engineering

Data mining aims at discovering important, useful and previously unknown patterns from the dataset, which is typically stored in a commercially available database. The type of underlying database can vary and should not be a constraint on the mining process. We should be able to mine the data irrespective of the database management system (DBMS) used for storing and managing the dataset. In this thesis, we use DB2 and Oracle to store our datasets, and use Java Database Connectivity (JDBC) to access the dataset managed by a relational DBMS.

In this thesis, we focus on association rule mining using a layered approach. Mining requests from a user are accepted by a mining optimizer which generates appropriate Structured Query Language (SQL) queries accepted by the underlying DBMS. We formulate SQL queries to implement association rule mining algorithms. We present three SQL-92 and three SQL-OR (object-relational extensions to SQL) approaches for the same. We also compare and contrast the approaches based on their

performances over synthetically generated datasets. Some of the commercially available data mining application tools are also compared with our mining optimizer.

We also map the input data into a format used by most mining tools. This, being a part of the optimizer, permits the user to directly identify relevant data from existing relations to the optimizer. Lastly we identify metadata, needed for determining the approaches based on the underlying database and user related constraints. Furthermore, metadata are stored in the underlying database and is accessed by the optimizer as needed.

CHAPTER 1 INTRODUCTION

Organizations generate and collect large volumes of data that they use in daily operations, e.g., billing, inventory, customer transactions in retail stores, banks, mail orders, market basket data, etc. The data necessary for each operation is captured and maintained by the corresponding department. This is a result of the tremendous growth in the data warehousing technology added to the stupendous drop in the storage prices. Yet despite this wealth of data, many companies have been unable to fully capitalize on its value because information implicit in the data is not easy to discern. However, to compete effectively in today's market, decision makers must be able to identify and utilize information hidden in the collected data and take advantage of high return opportunities in a timely fashion. For example, after identifying a group of married, two-income and high net worth customers, a bank account manager sends them information about the growth mutual funds offered by the bank, in an attempt to convince them to use the bank's services rather than those of a discounted broker.

1.1 Data Warehouse

A *data warehouse* is a collection of subject-oriented, integrated, time variant and non-volatile data in support of management's decision making process. It can also be defined as a tool for satisfying a business manager's needs for complex queries and a general facility to get quick, accurate, and insightful information. Sets of data are one of the organizations most critical and valuable assets. Data Warehousing has grown out of

the repeated attempts on the part of various researchers and organizations to provide their organization flexible, effective and efficient means of getting at these sets of data. *Data warehousing* assembles and organizes data from enterprise operations such as transaction systems (registers, online order systems, etc.) and stores that data in a format that business or technical people can analyze. The data is stored in a data warehouse. The data warehouse is then made accessible through different means to those individuals in need of detailed information, information that is optimized and implemented to make timely, accurate decisions. As relevant information becomes available from a source, or when relevant information is modified, the information is extracted from the source, translated into a common model (e.g., the relational model), and integrated with existing data at the warehouse. Queries can be answered and analysis can be performed quickly and efficiently since the integrated information is directly available at the warehouse, with differences already resolved.

A data warehouse is made up of three different functional areas, each of which must be customized to meet the needs of a business.

The first component handles acquisition of data from legacy systems and outside sources. Here the data is identified, copied, formatted and prepared for loading into the warehouse. Many vendors' products assist in data extraction and preparation.

The second component of the warehouse is the storage, which is managed by relational databases like those from Oracle Corp., IBM Corp. or Informix Software Inc., specialized hardware-symmetric multiprocessor (SMP) or massively parallel processor (MPP) machines or software. The storage component holds the data so that the many different

data mining, executive information, and decision support systems can make use of it effectively.

The third component of the warehouse is the access area. Here different end-user PCs and workstations draw data from the warehouse with the help of multidimensional analysis products, neural networks, data discovery or analysis tools. These powerful, smart software products are the real driving forces behind the viability of data warehousing. After all, what good does it do to store all this information without some way to understand it in new and different ways? Data mining applications fill that void.

The online analytical processing (OLAP) or multidimensional spreadsheet tools represent a whole new generation of high-powered, user-friendly data investigation systems. These systems, sometimes referred to as spreadsheets on steroids, enable people to look at information from dozens of different perspectives. The main strengths of OLAP products are their ability to dynamically slice and dice reports and to look at the same kinds of information at different levels at the same time.

1.2 Data Mining

Data mining is a relatively unique process. In most standard database operations, nearly all of the results presented to the user are something that they knew existed already in the database. A report showing the breakdown of sales by product line 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.

Data mining, on the other hand, extracts information unknown previously to the user from a database. Relationships between variables and customer behaviors that are non-intuitive are the jewels that data mining hopes to figure out. And 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 a solution to a business problem. Thus data mining is a process of extracting valid, previously unknown and ultimately comprehensible information from a data warehouse and using it to make crucial business decisions. The extracted information can be used to form a prediction or classification model, identify relations between database records, or provide a summary of the database(s) being mined. Data mining consists of a number of operations, each of which is supported by a variety of techniques such as rule induction, neural networks, conceptual clustering, association discovery, classification, etc. In many real world domains such as marketing analysis, financial analysis, fraud detection, etc, information extraction requires the cooperation of several data mining operations and techniques. While OLAP tools allow one to compare, say sales revenues for two quarters, data mining technology lets one perform for example, a search through all sales data and then presents with hypotheses to analyze. In fact, it is projected as the next step beyond OLAP for querying data warehouses. Data mining tools predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions. Data mining tools can answer business questions that traditionally were time consuming to resolve. Data mining tools can analyze massive databases to deliver answers to questions such as, "Which clients are most likely to respond to my next promotional mailing, and why?" With the data warehousing and decision support systems, one could answer questions like

"What were unit sales in New England last March? Drill down to Boston." But by data mining, now one can answer a question like "What's likely to happen to Boston unit sales next month? Why?" Data mining derives its name from the similarities between searching for valuable business information in a large database — for example, finding linked products in gigabytes of store scanner data — and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities.

1.2.1 Association Rules

Association models are models that examine the extent to which values of one field depend on, or are predicted by, values of another field. Association discovery finds rules about items that appear together in an event such as a purchase transaction. The rules have user-stipulated support, confidence, and length. The rules find things that "go together". These models are often referred to as Market Basket Analysis when they are applied to retail industries to study the buying patterns of their customers. Thus, given a collection of items and a set of records, each of which contain some number of items from the given collection, an association discovery function is an operation on this set of records which returns affinities that exist among the collection of items. These affinities can be expressed by rules such as "72% of all the collections of item A also contain item B" and "30% of all the collections contain items A and B". The rule is expressed symbolically as $A \Rightarrow B$ [Agr93]. A is said to be on the opposite side of the association of B. A rule can involve any number of items on either side of the association. 72% in this

rule is the confidence of the rule and 30% is the support. In the Market Basket Analysis, the retailer will run an association discovery function over the point of sales transaction log. The transaction log contains, among other information, transaction identifiers (TID) and product identifiers (ITEMS). The collection of items is of the order of 100,000 or more and transactions are comparable too. The output of the association discovery function is a list of product affinities. One of the most repeated data mining stories is the discovery that diapers and beer appear together in a shopping basket. The explanation goes that when fathers are sent out on errand to buy diapers, they often purchase a six-pack of their favorite beer as a reward. The rule would be represented as Diapers \Rightarrow Beer.

1.2.2 Sequential Patterns

Given a database of sequences, where each sequence is a list of transactions ordered by transaction-time, and each transaction is a set of items, the problem of mining sequential patterns is to discover all sequential patterns with a user-specified minimum support. Here the support of a pattern is the number of data-sequences that contain the pattern. In other words, find inter-transaction patterns within a specified time window such that the presence of a set of items is followed by another item or set of items in the time-stamp ordered transaction set.

A sequential pattern is an ordered list (sequence) of item-sets [Agr95]. The item-sets that comprise the sequence are termed the elements of the sequence. The database comprises of records. A record typically consists of the transaction date and the items bought in the transaction. Often, data records also contain customer-id, particularly when the purchase is made using a credit card or a frequent-buyer card.

An example of a sequential pattern is that customers typically rent “Star Wars”, then “Empire Strikes Back”, and then “Return of the Jedi”. Note that these rentals need not be consecutive. Customers who rent some other videos in between also support this sequential pattern. Elements of a sequential pattern need not be simple items. “Fitted sheet and flat sheet and pillow cases”, followed by “comforter”, followed by “drapes and ruffles” is an example of a sequential pattern in which the elements are sets of items (item-sets).

For Sequential pattern data mining, few constraints are added. First of which is, time constraints that specify a minimum and/or a maximum time period between adjacent elements in a pattern. They are called the *min-gap* and *max-gap*, respectively. Second, the restriction that the items in an element of a sequential pattern must come from the same transaction is relaxed, instead allowing the items to be present in a set of transactions whose transaction-times are within a user-specified time window, called the (*window-size*). Third, given a user-defined taxonomy (*is-a* hierarchy) on items, allow sequential patterns to include items across all levels of the taxonomy.

1.2.3 Clustering and Classification

Cluster analysis is a data reduction technique that groups together either variables or cases based on similar data characteristics. This technique is useful for finding customer segments based on characteristics such as demographic and financial information or purchase behavior. It is a process of dividing a data-set into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another, where distance is measured with respect to all available variables. Clustering is a descriptive task that seeks to identify homogenous groups of objects based on the values of their attributes. It can be

described as a process of maximizing inter-cluster similarities while minimizing intracluster similarities. Clustering algorithms can be classified into two categories: *partitional* and *hierarchical*. Given a set of objects and a clustering criterion, partitional clustering obtains a partition of the objects into clusters such that the objects in a cluster are more similar to each other than to objects in different clusters. The popular *K-means* and *K-medoid* methods determine K cluster representatives and assign each object to the cluster with its representative closest to the object such that the sum of the distances squared between the objects and their representatives is minimized. The only difference between the two partitional approaches is that, in *K-means*, clusters are represented by gravity center while in *K-medoid*, by a central object. On the contrary, hierarchical clustering is a nested sequence of partitions. There are again two flavors. An *agglomerative*, hierarchical clustering starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters until all objects are in a single cluster. It is a bottom up approach. A *Divisive* hierarchical clustering approach reverses the process by starting with all objects in a cluster and subdividing into smaller pieces. It is a top down approach.

Classification is the process of dividing a data-set into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another. Here that data is being classified into groups based on the distance measured with respect to specific variable(s). For example, a typical classification problem is to divide a database of companies into groups that are as homogeneous as possible with respect to a creditworthiness variable with values "Good" and "Bad". Classification, as the name implies, predicts class

membership. For example, a model predicts that Mr. ABC, a potential customer, will respond to an offer. With classification the predicted output (the class) is categorical. A categorical variable has only a few possible values, such as "Yes" or "No," or "Low," "Middle," or "High". In other words, it is a process of building a model from a training set that classifies new data, based upon the attribute values. Each record of the training set consists of several attributes that could be continuous (coming from an ordered domain), or categorical (coming from an unordered domain). Of the attributes, one of them will be the classifying attribute. Decision trees are built depending on this classifying attribute, as the decision trees suit data mining and are the fastest to build, simple and easy to understand. The training set is partitioned recursively until each node consists of a single class/category. The figure shows a sample decision tree and the training set from which it is derived.

Table 1.1 Training Data set

TID	Car	Age	Salary	Class
1	Sedan	30	65	G
2	Sports	23	15	B
3	Sports	40	75	G
4	Sedan	55	40	B
5	Limousine	55	100	G
6	Sedan	45	60	G

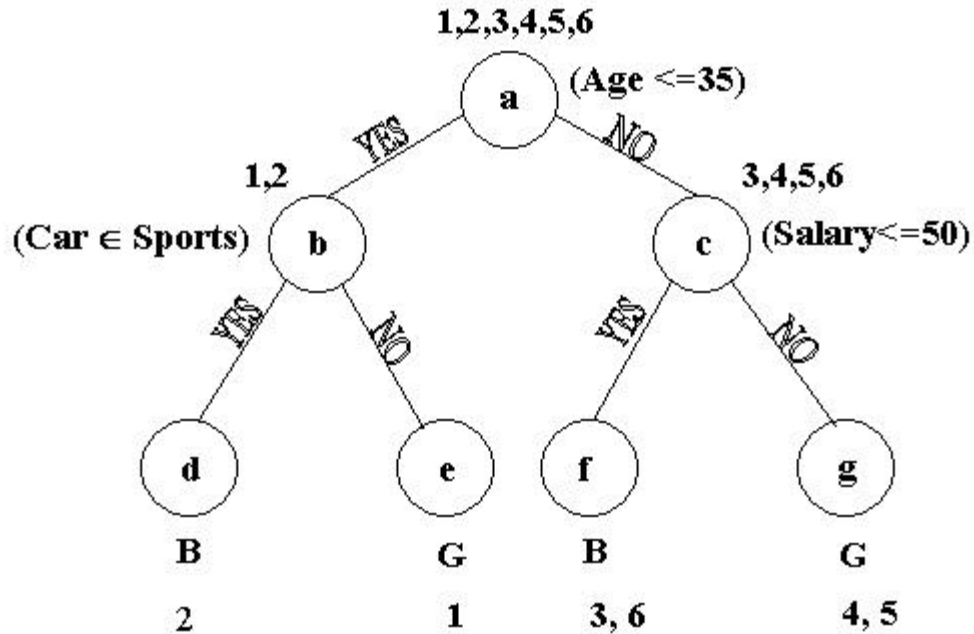


Figure 1.1 Constructed Decision Tree

1.2.4 Visualization

Visualization tools take advantage of human perception as a method for analysis. Visualization is the graphical presentation of data and information for the purposes of communicating results, verifying hypotheses, and qualitative exploration. What numbers can't show, corresponding pictures often can. For example, a linear trend in data might not be evident from a table of data. However, a scatter plot that shows a series of points lined up on a straight line provides immediate insight into data relations. With high power computer graphics, visualization tools can also be effective presentation tools. Once a discovery is made, the analyst must convey that discovery using an easily accessible language such as pictures. The human mind is not capable of comprehending large amounts of data when they are presented in tabular form. However the human mind can process visual images far better than any computer. Visualization has long been a

standard tool to assist statistical and scientific analysis and is becoming an increasingly important component in database and data mining activities, both for its ability to provide rich overviews and to permit users to rapidly detect patterns and outlines. With respect to visualization of the association rules, the mined rules are often more apprehensible by the end user if they are presented in a graphical form than in the textual form.

1.3 Problem Statement

The main aim of this thesis is to deal with the association discovery approach in data mining. Data mining has undergone a transition from file mining to the current data warehouse mining. There have been two categories in the related work. One, which proposes adding new mining operators which extend Structured Query Language (SQL) and the other which leverages the query processing capabilities of current relational database management system (DBMS). This approach exploits the capabilities of conventional relational systems and their object-relational extensions to execute mining operations. We follow the latter approach, which assumes that the data is in a relational database in the form of tables and the data when present in the tables is much more flexible and easier to work with. All data is readily available for SQL queries that manipulate the data. We pursue the SQL based approach in our thesis doing which necessarily reduces the development time of the our algorithms and also make them extremely portable across DBMSs since porting is easier when standard SQL features are used.

The other goal of our thesis is to explore the areas of mapping of input data and storing information as metadata that will aid in selection of an approach for association rule mining. The input data format for the association rule generator is (Tid, Item).

However, expecting data in this format would be unreasonable from an end user viewpoint since the real world data are not always in this format. To get around this problem, it should be made possible to map the user data into a format that is acceptable to the association rule generator. In order to achieve transparency, some metadata should be stored in the underlying database. This information would help to choose a good available approach to mine the input data. This information could be about the input table, underlying databases, pros and cons of each of the approaches, and some constraints (database and user specific). We believe that though some parts of this work may be small and some others just a beginning, this work will serve as a stepping-stone in the proper direction.

1.4 Approach

As described in the previous section, this thesis aims to explore the various approaches of association rule mining in relational databases, perform mapping of the input data, generate and use metadata. We study the various approaches and formulate SQL queries as a part of developing these approaches. These approaches have been implemented over two databases viz. DB2 and Oracle to establish the generality of our approach. The visualization is independent of the underlying DBMS or approach used for mining. It is a step to show that, regardless of the underlying database in which the users' data exists, data can be mined, in particular for association rules as done here. Below we present some of the architectures [Tho98] for mining with relational systems.

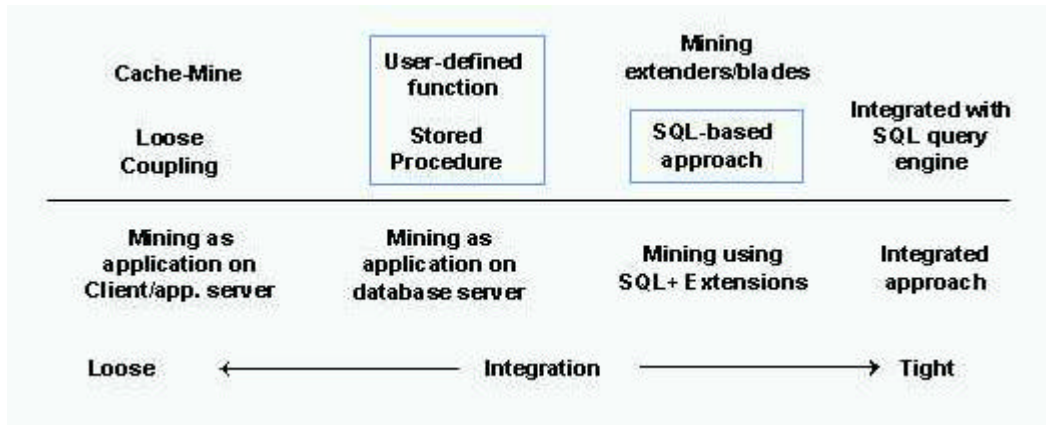


Figure 1.2 Architectural alternatives

Loose mining is an example of the client-server architecture. The server is the main mining kernel. In cache mining, the data is not read several times but is cached after the first read. The stored procedure and user-defined function approaches of mining encapsulate the whole mining logic into stored procedures and user-defined functions respectively on the database server. The flexibility of the approaches outweighs their development costs. SQL-based approach is the one that is explored in this thesis, in which the mining algorithms are formulated as SQL queries over the database in which the input data is predominantly stored as tables. We exploit the underlying database's features like character large object (CLOB), stored procedures, user-defined functions (UDFs) and the least to mention the SQL querying capabilities. There may be integration with extensions like database extenders, data cartridges or data blades that may form a new integrated architecture. The last among the architectures is the integrated approach, which is the tightest form of integration that has no boundary between querying, OLAP, and mining. This is like a black box for the user who does not care about the underlying process. As mentioned previously, we pursue the SQL-based approach and formulate

queries for the mining algorithms. We present the architecture proposed in this thesis in Figure 1.3.

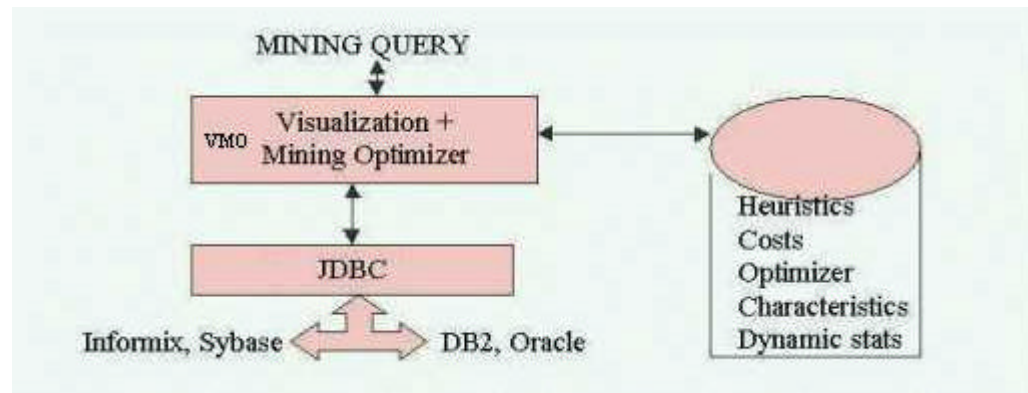


Figure 1.3 The proposed architecture for association rule mining

As seen in the figure, the mining query is input to a mining optimizer. This mining optimizer is a layered approach for optimization and visualization of the rules. It also encapsulates the mapping of input data. The JDBC/ODBC is the connection-establishing component between the optimizer and the underlying database. The type of the underlying database is immaterial to this component. It is also the router of data from the database to the optimizer. The optimizer passes on the data to the rule generator that generates the rules. The rule generator bases the rule generation process on the user specified support and confidence levels, and selected approach if any or on the best approach as specified by the optimizer. The visualization module in the optimizer visualizes the generated rules in the bar chart, 2-D, and 3-D formats [Hon00]. The different association rule mining algorithms are discussed in Chapter 3. The processes of mapping and storing metadata are described in the next sub-sections respectively.

1.4.1 Mapping

Mapping is an integral part of the input for the application. Our application, as well as most of the data mining applications/tools, for that matter, takes the input data in a two-column format. First column is the Transaction Identifier (TID) and the second column being the item. In our approach, we want to use data that is already present in the database without making the user to transform it and keep track of the transformation. Usually the data available and the data that one wants to mine is not present in the (Tid, Item) format. And it is defeats the purpose to make the end user map the data into the (Tid, Item) format. On the contrary, the user should be able to specify the data to be used in relational terms (such as selection of a subset of a relation or join and projection of 2 or more relations etc). Our system should map the relational data into the format acceptable by the application. This is precisely the objective of the mapping part of our work. The user is allowed to select a single or multiple tables for mining. If more than one table is selected, the user can either perform a join or union or difference operation on the selected tables and this table is the new input table. In this new table, the user further has a choice of selecting the Tid column and the columns that are to be treated as items among the available columns. The selected Tid column and the items' columns are then mapped. Each element in the Tid and items' columns is mapped to an integer. This will transform the user input data into the required (Tid, item) format.

1.4.2 Metadata

Metadata are another important aspect of mining. The goal is to try to store information/knowledge that will aid the mining optimizer in selecting a good approach for mining the input data. This information can be specific to the underlying database or to the input data and is stored as metadata in one of the underlying databases. This

information is accessed by the mining optimizer for making decisions and modified appropriately. A variety of characteristics of the approaches can also be stored, as part of the metadata and this would enable the selection of an approach for a particular data set in a specific database. This information in the metadata table needs to be modified dynamically so that the most recent and correct information is read by the application in the process of decision making with respect to approach selection.

1.5 Thesis Organization

The rest of the thesis is organized as follows. We present the details of using Java Database Connectivity (JDBC) for accessing DB2 and Oracle that we will be making use of, in Chapter 2. We discuss the SQL formulations for the various approaches for association rule mining in Chapter 3. Chapter 4 describes the mapping process and current implementations of the metadata with the additions to the existing metadata. Synthetic data generation, performance comparison of the various approaches and the scalability issues, and comparison of our work to the existing tools in the market are addressed in Chapter 5. We describe the system implementation in Chapter 6. Finally, in Chapter 7 we discuss our conclusions, contributions and the proposed extensions.

CHAPTER 2 DATABASE CONNECTIVITY ISSUES

2.1 Introduction

SQL is a language used to create, manipulate, examine, and manage relational databases. Because SQL is an relational database-specific language, a single statement can be very expressive and can initiate high-level actions, such as sorting and merging data. SQL was standardized in 1989 so that a program could communicate with most database systems without having to change the SQL commands. Unfortunately, one must connect to a database before sending SQL commands, and each database vendor has a different interface, as well as different extensions of SQL.

Open Database Connectivity (ODBC), a C language based interface to SQL-based database engines, provides a consistent interface for communicating with a database and for accessing database metadata. Individual vendors provide specific drivers or bridges to their particular database management system. Consequently, thanks to ODBC and SQL, we can connect to a database and manipulate it in a standard way. Though SQL is well suited for manipulating databases, it is unsuitable as a general application language and programmers use it primarily as a means of communicating with databases. Another language is needed to feed SQL statements to a database and process results for visual display or report generation. Unfortunately, one cannot easily write a program that will run on multiple platforms even though the database connectivity standardization issue has been largely resolved. For example, if a database client were written in C++, there would

be a need to totally rewrite the client for different platforms. On the contrary, a Java client can be run on any Java-enabled platform without even recompiling that program. The Java language is completely specified and, by definition, a Java-enabled platform must support a known core of libraries. One such library is JDBC, which can be thought of as a Java version of ODBC, and is itself a growing standard [Sun00].

JDBC is the Java application programming interface (API) for standardized SQL based database access. It is a database-independent API that facilitates the development of database-independent Java Applications/Applets/Beans. JavaSoft™ created the JDBC specification to meet the urgent need for a standard DBMS API for Java. JDBC provides database access via Java that is independent of both the platform and the database host system on which the application runs. The specification enables one to write Java code that establishes a connection to an SQL-capable data source, sends SQL statements to the data source, and returns status messages and data records resulting from the execution. JDBC also offers advanced functionality such as automatic conversion of different database data types to Java data types, the streaming of large data records, cursors, and multiple result data sets.

In short, the JDBC API defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata. Thus JDBC allows a Java-powered program to issue SQL statements and process the results.

2.2 Basic JDBC Architecture

There are two architectures specified for JDBC that are universally accepted. They are based on the number of layers present in the architecture, which give flexibility to the client as an end user.

2.2.1 Two Tier Architecture

In the two-tier model, a Java applet or application talks directly to the database. This requires a JDBC driver that can communicate with the particular DBMS being accessed.

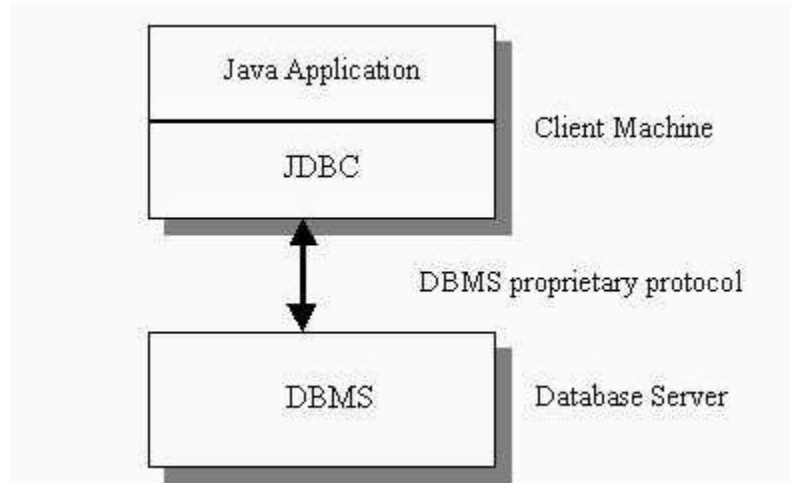


Figure 2.1 Two tier architecture

In this architecture, a user's SQL statements are directly delivered to the database. The results of those statements are sent back to the user. There is tight coupling between the client and the server as the client is directly connected to the server. The database may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the database as the server. The network can be an Intranet, which, for example, connects employees to each other within a corporation, or the Internet. The advantages are that the architecture is simple and the client-side scripting offloads work onto the client. But the disadvantages are a fat client and some degree of inflexibility.

2.2.2 Three Tier Architecture

In the three-tier model, commands are sent to a middle tier of services, which then sends SQL statements to the database. The Client is connected to an intermediate Application Server, which in turn is connected to the database. Thus the client communicates with the application server and this application server talks with the database. The SQL statements from the client are passed on to the database by the server. The database processes the SQL statements and sends the results back to the application server, which then sends them to the client. This is very much similar to the indirect mode of addressing used in the computer architecture.

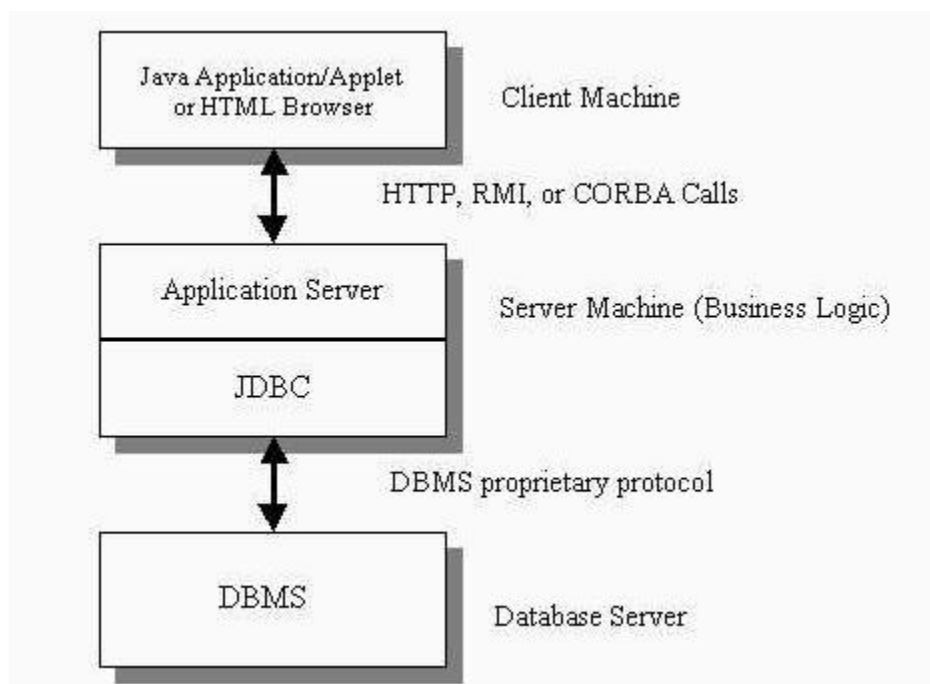


Figure 2.2 Three tier architecture

Most people find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. With the middle tier, the user can employ an easy-to-use higher-level API

which is translated by the middle tier into the appropriate low-level calls. This architecture gives flexibility in the sense that one part can be changed without affecting the others. Also one can connect to different databases without changing code. The middle tier can be used to cache queries, implement proxies, and firewalls. Finally, in many cases the three-tier architecture can provide performance advantages. But the downside is that this is a complex structure, needs higher maintenance, and more parts to configure.

There have been extensions to this architecture since Java allows for N-tier architectures. In our implementation, we will be using the two-tier architecture for its ease of use and simplicity. Our implementation does not need an application server, which would form the middle tier of services as seen in the three-tier architecture. The application in our architecture is directly connected to the database server and we need no services from the middle tier. The middle tier is needed in cases where the some parts of the application are prone to change over time and connections to different databases need to be established. No parts of our application are prone to changes and our application can connect to different databases since the JDBC driver provides us with that feature.

2.3 Detailed Architecture

Drivers for JDBC are exposed to JDBC compliant Java applications via the JDBC driver manager. The JDBC driver manager is a Java class implementation. It is an interface used by JDBC service consumers (Application/Applet/Bean developers) and service providers (JDBC driver developers).

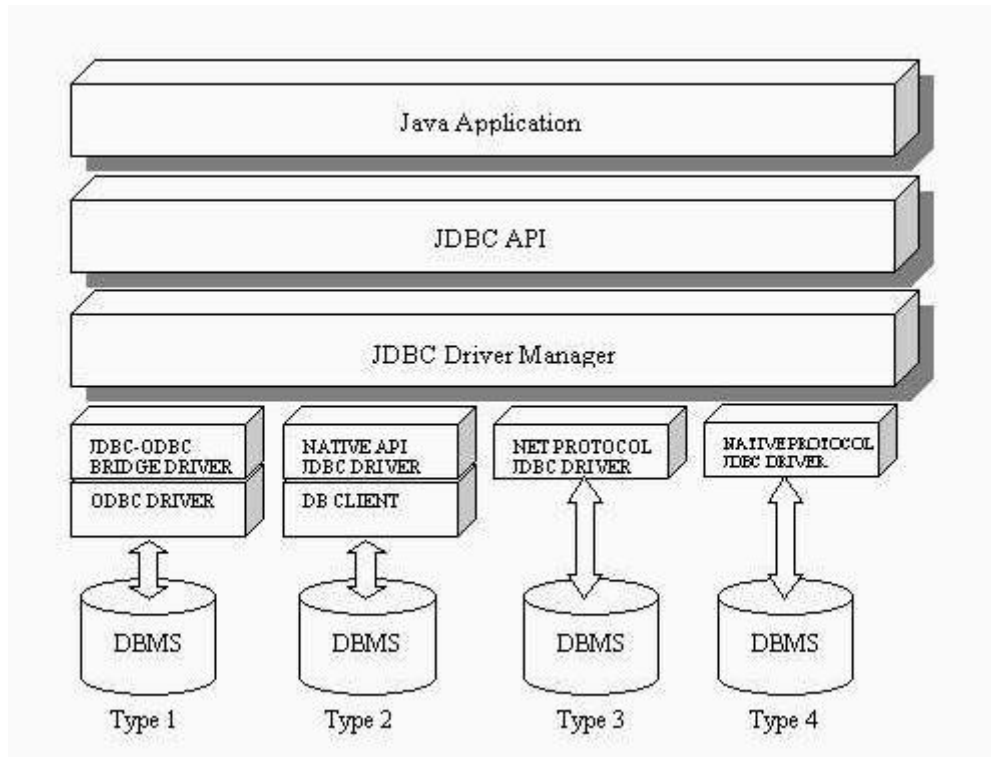


Figure 2.3 Detailed architecture

The JDBC application developers call upon the JDBC driver manager for JDBC driver association (or binding). JDBC driver developers build the JDBC classes to the specification as depicted by the JDBC driver manager class implementation. Also as seen in Figure 2.3 above, current JDBC drivers fit into one of the four categories:

1. Type 1 - The JDBC-ODBC bridge provides JDBC access via most ODBC drivers. Note that some ODBC binary code and in many cases database client code must be loaded on each client machine that uses this driver.
2. Type 2 - A native-API driver (partly Java) converts JDBC calls into calls on the client API for DB2, Oracle, Sybase, Informix, or other DBMSs. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

3. Type 3 - A net-protocol (all Java) driver translates JDBC calls into a DBMS-independent net protocol that is then translated to a DBMS protocol by a server. This net server middle-ware is able to connect all its Java clients to many different databases. In general, this is the most flexible JDBC alternative.
4. Type 4 - A native-protocol (all Java) driver converts JDBC calls into the network protocol used by DBMS directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access.

We will be using the Type 2 in our implementation. The Type 2 architecture allows faster access to databases like DB2, Oracle, etc. In our implementation, we are using DB2 and Oracle relational databases. Also that Type 1 supports Microsoft Access which we do not require and hence we settle for Type 2 architecture/driver.

2.4 Stored Procedures

Stored procedures are user written SQL programs that are stored at the server and can be invoked by the client applications. A stored procedure can contain most of the statements that an application program usually contains. Stored procedures can execute SQL statements at the server as well as application logic for a specific function. A stored procedure can be written in many different languages, such as COBOL, C, C++, PL/I, FORTRAN, etc. The client program and the stored procedure do not have to be written in the same programming language. The language in which stored procedures are written depends on the platform on which the server is installed. The client program can pass parameters to the stored procedure and receive parameters from the stored procedure.

In a regular database access without a stored procedure, the client performs all the application logic. The server is responsible only for the SQL processing on behalf of the

client. In such an environment, all database accesses must go across the network, resulting in poor performance in some cases. The model is relatively simple and this makes the application program easy to design and implement. Since all the application code resides at the client, a single application programmer can take responsibility for the entire application. But there are some disadvantages in using this orthodox approach. The application logic runs only on the client and hence there must be additional input/output operations for most SQL operations. These additional operations result in poor performance. Also there is a tight coupling between the client and the server. A slight change in the server initiates a corresponding change at the client side. Since there is one call to the server associated with each and every SQL statement, there is a lot of traffic congestion in the network. All these problems and concerns can be overcome by using stored procedures. Stored procedures enable to encapsulate many of the application's SQL statements into a program that is stored in the server. The client can invoke the stored procedure by using only one SQL statement. A typical application requires two trips across the network for each SQL statement, whereas an application using the stored procedure technique requires two trips across the network for each group of SQL statements. This group is the collection of SQL statements that are encapsulated into the stored procedure. Also, the technique of using stored procedures would best suit an application that processes large amounts of data but require only a subset of data to be returned to user. This technique would allow access to features that exist only on the database server and prohibit the client from manipulating the contents of sensitive server data.

2.5 User Defined Functions

User-defined functions (UDFs) extend and add to the support provided by built-in functions of SQL, and can be used wherever a built-in function can be used. UDFs can be created as either:

- An external function, which is written in a programming language.
- A sourced function, whose implementation is inherited from some other existing function.

There are three types of UDFs:

1. **Scalar** - Returns a single-valued answer each time it is called. For example, the built-in function SUBSTR() in DB2 is a scalar function. Scalar UDFs can be either external or sourced.
2. **Column** - Returns a single-valued answer from a set of like values (a column). It is also sometimes called an aggregating function. An example of a column function is the built-in function AVG() in DB2. An external column UDF cannot be defined but a column UDF that is sourced upon one of the built-in column functions can be defined. This is useful for distinct types. For example, if there is a distinct type SHOESIZE defined with base type INTEGER, a UDF AVG(SHOESIZE) which is sourced on the built-in function AVG(INTEGER) could be defined, and it would be a column function.
3. **Table** - Returns a table to the SQL statement that references it. Table functions may only be referenced in the FROM clause of a SELECT statement. Such a function can be used to apply SQL language processing power to data that is not actually data, or to convert such data into a virtual table. For example, table functions can take a file and convert it to a table, tabularize sample data from the World Wide Web, or access

a Lotus Notes database and return information such as the date, sender, and text of mail messages. This information can be joined with other tables in the database.

A table function can only be an external function. It cannot be a sourced function. Information about existing UDFs is recorded in the SYSCAT.FUNCTIONS and SYSCAT.FUNCPARMS catalog views in DB2. The system catalog does not contain the executable code for the UDF. (Therefore, when creating backup and recovery plans one should consider how to manage UDF executables.) A UDF cannot be dropped if a view, trigger, table check constraint, or another UDF is dependent on it. If a UDF is dropped, packages that are dependent on it are marked inoperative. An external UDF can be written by a user in C or Java. The program that implements an external table function must return one row of the result table each time it is called, and must indicate the end of the result by a special return code.

2.6 Stored Procedures in Oracle

Oracle provides a means of writing external procedures. But these procedures use Dynamic Link Libraries (DLLs) written in host language like C, FORTRAN etc. But Oracle 8.0™ does not support the procedures written in Java, which is the host language in our implementation. Thus there would be a need to use Java Native Interface (JNI) to communicate with a C external procedure if the choice of writing an external procedure in C were chosen. To get around this problem, we decided to use the Procedural Language/SQL (PL/SQL), which is a block structured programming language that supports loops, conditional statements, etc, in addition to the normal SQL operations. It also supports Data Manipulation Language (DML) statements inside the PL/SQL blocks. A stored procedure in Oracle has a name, takes parameters as input and returns values, is

a part of the data dictionary, can be invoked by many users. The input and output parameters can range from basic data types like numbers, characters, to complex types like character large objects (CLOBs) and tables [Ora97]. One of the examples in our implementation is the “saveItem.” The actual description and the declaration follow.

```
CREATE OR REPLACE PROCEDURE saveItem(rowCount IN INTEGER) AS
```

```

    Declaration Section
BEGIN
    Read the data
    Process the data
    Write to the CLOB
END
```

A **CREATE OR REPLACE PROCEDURE** clause always precedes the name of the procedure. This is to make sure that an earlier declaration (if any) is replaced by this new one. Then the name and the parameters follow. The temporary variable declarations are the next in line. The main body of the procedure is encapsulated in the **BEGIN - END** block.

The main purpose of this stored procedure is to group, for each of the transactions, all the items present in that transaction. In other words, form a CLOB of items with the same TID number. The structure CLOB is used because it can store a maximum of $2^{31}-1$ bytes (2 Gigabytes). In the stored procedure saveItem, the reading of data is the reading of a tuple at a time from the source table with the use of a cursor. Processing the data is identifying the CLOB to which this item is to be added. The writing part is adding an item to a CLOB with a TID that matches with the TID of the current item. The input and output are shown in the Table 2.1.

Table 2.1 The input and output for the saveItem stored procedure

TID	ITEM	TID	ITEMS
1	Bread	1	Bread, Milk
1	Milk	2	Bread, Cheese
2	Bread	3	Sugar
2	Cheese	4	Milk
3	Sugar		
4	Milk		

To appreciate the usefulness of the stored procedure, the number of tuples in the original and output tables should be compared. Typically in a market basket dataset, the number of tuples in input table ranges from several thousands to millions.

2.7 User Defined Functions in DB2 (UDF™)

A UDF in DB2 is created explicitly by a user using the **CREATE FUNCTION**, which declares the new function and specifies its semantics. UDFs are always created in a specific database and can be used only in that database. Within that database, UDFs can be used in the same way as built-in functions [Cha98, Pod98]. Use of a UDF does not require authorization, but the creation of one, does require certain privileges. As described in section 2.5, UDFs can return either a scalar value or a table. Table functions are very powerful because they enable the user to make almost any source of data to appear to be a DB2 table. All that is required, is to write a Java program (in our implementation) that collects the desired data, filters it according to some input parameters if so desired, and returns it to DB2 one row at a time. The table returned by the table function can participate in joins, grouping operations, set operations such as

union, and any other operation that could be applied to a read-only view. In the rest of this section, we will describe how to create and implement a table function with an example. The example is the same as the one described in section 2.6. The syntax for creating and registering a table function would be as follows:

```
CREATE FUNCTION saveItem(int, int)
RETURNS table(T_tid int, T_cnt int, T_items CLOB)
NOT FENCED SCRATCHPAD NO SQL
NO EXTERNAL ACTION LANGUAGE Java
PARAMETER STYLE DB2GENERAL FINAL CALL
DISALLOW PARALLEL DBINFO
EXTERNAL NAME fileName!saveItem
```

Note that once the UDF is created in the manner shown above, it is automatically registered with the database and can be used in the SQL queries. The **CREATE FUNCTION** clause is the beginning of the function creation. In this case, the name of the function is “saveItem”. The **RETURNS** clause follows. Since it is a table function, the clause **table** and the schema of the table are the next to follow. The schema of the return table is made of 3 columns that are all integers, viz. **T_tid**—the TID from the source table, **T_cnt**—the count of the number of items in this transaction, and **T_items**—the items in this transaction in the form of **CLOB**. The **FENCED** option specifies that the function must always be run in an address space that is separate from the database. This option causes a performance penalty due to process switching when the function is called, but integrity of the database is protected against accidental or malicious damage by the function. An unfenced function runs in the same address space as the database. Once the function is known to execute safely and correctly, one can make a fenced function unfenced. With the **SCRATCHPAD** clause, the function is given a scratchpad area in memory that it can use to

store information from one function invocation to the next. The mandatory **NO SQL** clause specifies that the UDF contains no SQL statements in the body of the function. As of now, UDFs are not allowed to access the database. The mandatory **NO EXTERNAL ACTION** clause specifies that the UDF does not perform any actions that affect the world outside the database. Hence there exist no side effects. The mandatory clause **LANGUAGE** clause specifies the programming language in which the UDF is implemented, which is Java in our implementation. The **PARAMETER STYLE** clause identifies the conventions that are used for passing parameters to the UDF and is mandatory. Since Java is our language of implementation, the keyword **DB2GENERAL** is used. The **FINAL CALL** clause specifies that the final or last call to this UDF in a SQL statement when used, be differentiated from the earlier calls. This could be used for the housekeeping and cleaning purposes, especially used with the **SCRATCHPAD** option. The **PARALLEL** clause indicates whether parallel executions of the UDFs are allowed on multiple processors. In our implementation it is **DISALLOWED** because an invocation of the UDF needs to pass along information to the next invocation using a scratchpad. It is usually disallowed, when a UDF uses **SCRATCHPAD**, **FINAL CALL** or **EXTERNAL ACTION** clauses. The **DBINFO** clause is optional and causes DB2 to pass an extra parameter to the UDF. This parameter is a pointer to a data structure containing information such as the name of the current database, the current authid, and the name of the table and columns (if any) that is being modified by the current statement. The final clause is the **EXTERNAL NAME**, which indicates that the UDF is an external function and specifies the location of the function in the Java file that serves as its implementation. Either the whole path of the binary file that is the functions implementation, can be specified followed by a “!”, followed by the name of the proper

entry point in that file or one can just specify the filename, followed by “!”, followed by the entry point. In the latter case, the system will look for the function’s binary file implementation in the `sqllib/function` directory associated with the database.

The implementation of the UDF `saveItem` is as shown below:

```
saveItem(int tid, int item, int T_tid, int T_cnt, COM IBM db2. app. Clob
T_items)
{
    Process/Write data to CLOB
    set(3, tid)
    set(4, count)
    set(5, CLOBOfItems)
    setSQLstate(“02000”)
}
```

The first two parameters in the signature of the method are the input parameters and the last three are the output parameters. In fact, they are the columns of the output table. The process/write data part of the function is the same as described in section 2.6. It determines the CLOB to which this item is to be added. Once this is done, the columns in the table are given the appropriate values by the use of the “set” method. Finally at the end of the function, we set the SQL state to “02000” to indicate the end of the table.

Once this UDF is created and registered, it can be used in the SQL statements. One example is depicted below.

```
INSERT INTO C1
    SELECT T_tid, T_cnt, T_items
    FROM (SELECT tid, item
          FROM C
          GROUP BY tid, item
        ) AS tt0, TABLE(saveItem(tid, item)) as tt2;
```

The input table C and output table C1 are shown below in Table 2.2.

Table 2.2 The input and output for the saveItem UDF.

TID	ITEM
1	Bread
1	Milk
2	Bread
2	Cheese
3	Sugar
4	Milk

T_TID	T_CNT	T_ITEMS
1	2	Bread, Milk
2	2	Bread, Cheese
3	1	Sugar
4	1	Milk

2.8 Conclusions

In this chapter, we discussed the database connectivity issues and various architectures of JDBC. We also looked into the concepts of stored procedures and user defined functions. We then gave an example for stored procedures and user defined functions in Oracle and DB2 respectively. In Chapter 3 we discuss the concepts of association rule mining and describe the various approaches that we have implemented.

CHAPTER 3 ASSOCIATION RULES

3.1 Introduction

Association rule mining is formally stated as follows: Let $I = \{i_1, i_2, i_3, \dots, i_m\}$ be a set of literals. It is the set of items. Let D be the set of transactions, where each transaction T is a set of items and $T \subseteq I$. An association rule is of the form $X \Rightarrow Y$. Here $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ is said to hold over the set of transactions D with support s if $s\%$ of the transactions in D contain $X \cup Y$. The rule $X \Rightarrow Y$ has confidence c over the set of transactions D if $c\%$ of the transactions in D that contain X also contain Y . Rule support is the relative occurrence of the detected association rules within the entire database. To determine the rule support of an association rule, say $X \Rightarrow Y$, divide the number of transactions supporting the rule by the total number of transactions.

$$\text{Rule Support}\{X \Rightarrow Y\} = \frac{\text{Count of transactions containing the item set } X \cup Y}{\text{Total number of transactions}}$$

Thus for any rule, the support for that rule is equal to the support of an itemset that contains all the items in that rule.

The confidence of an association rule is its strength or reliability. It is the percentage of transactions supporting the rule out of all the transactions supporting the rule body.

$$\text{Rule Confidence}\{X \Rightarrow Y\} = \frac{\text{Support}\{X \cap Y\}}{\text{Support}\{X\}},$$

where $X \cap Y$ is the percentage of transactions containing both items X and Y, and $\text{support}\{X\}$ is the percentage of transactions that contain item X.

In this chapter, we discuss SQL-92 and SQL-OR (SQL-Object Relational) formulations for association rule mining [Sar98]. We present three approaches each for SQL-92 and SQL-OR association rule mining. First, we present a generic apriori algorithm that serves as the basis for our formulations [Que98]. An association rule mining problem is broken down into two sub-problems.

1. Come up with all the item combinations (called itemsets) whose supports are greater than the user specified minimum support. Such sets are called the frequent itemsets.
2. Use the identified frequent itemsets to generate the rules. Suppose ABCD and AB are two frequent itemsets, then it can be determined whether the rule $AB \Rightarrow CD$ holds by computing the ratio $\text{support}(ABCD)/\text{support}(AB)$. If this ratio \geq the user specified minimum confidence, the rule $AB \Rightarrow CD$ holds.

3.2 Apriori Algorithm

The apriori algorithm is based on the above mentioned two steps of candidate itemsets and rule generation phases. We start with the frequent 1-itemsets and make k passes identifying the k^{th} frequent itemset in each of the passes. Let F_k represent the frequent k-itemsets and C_k represent the potentially frequent k-itemsets, called the candidate itemsets. The generation of the frequent k-itemsets encompasses the process of generation of the frequent (k-1) itemsets, F_{k-1} , and C_k . The candidate itemset C_k is a superset of all the F_k generated. Once the itemsets are generated, the next step is the support counting step. Here the input data (transactions) is scanned and tested for the presence of candidates in C_k . All entries that have a support greater than the user specified minimum support qualify for rule

generation and become a part of the frequent itemset F_k at the end of the pass. The process terminates when there are no more elements in C_k . The algorithm is depicted below.

```

F1 = {frequent 1-itemsets}
for (k = 2; Fk-1 ≠ 0; k++)
    Ck = generate(Fk-1)
    for all transactions t ∈ D do
        Ct = subset(Ck, t)
        for all candidates, c ∈ Ct do
            c.count++
        end for
    end for
    Fk = { c ∈ Ck | c.count ≥ minsup}
end for
Answer = ∪k{Fk}

```

3.3 Input and Output Formats

Input format. The input format is a two-column format. The first column is the transaction identifier (tid) and the second column is the item identifier (item). Hence for each transaction, if it has more than 1 item, then there will be multiple entries (rows) in the table for this transaction with the same value in the tid field and a different value in the item field. The other possible option is the normal table format with multiple columns, where one column is the tid field and the rest of the columns denote the items. For a transaction with large number of items, the first format will have as many tuples as the number of items while the latter will just have one tuple in the input table. In our implementation, the first format is assumed for two reasons. The first is that there is no prior knowledge about the number of items in each transaction. Also current databases in the market can support only up to certain number of columns for a table. Should a case arise wherein there are more number of items

in a transaction than allowed by the underlying database, there is no way we can manage the correctness of the data. Also there will be a lot of null values in rows, as all items are not used in all transactions.

Output format. The output in our case is actually a set of rules. We present the output in a tabular form. Not all rules are of the same length. Hence we take the rule with the maximum length and this would statically determine the number of columns in the output table. Should a rule have less number of items than the number of columns in the table, the extra columns for that rule are filled with NULL values. Thus the format of the table is (item₁, item₂, ... ,item_k, nullm, rulem, confidence, support). Here k is the length of the largest frequent itemset. nullm is the null marker and marks the end point of the rule. rulem is the position of the \Rightarrow in the rule. Confidence and support values for the rule are given in the confidence and the support fields. For example, a rule $AB \Rightarrow CD$ with 30% support and 90% confidence is represented in the table with $k = 8$ by a tuple (A, B, C, D, NULL, NULL, NULL, NULL, 5, 3, 90, 30).

3.4 Candidate Generation

In the k^{th} pass, we need to generate the set of candidate itemsets C_k from the frequent itemset F_{k-1} generated in the $(k-1)^{\text{th}}$ pass. We do this in the following way.

insert into C_k

select $I_1.item_1, \dots, I_1.item_{k-1}, I_2.item_{k-1}$

from $F_{k-1} I_1, F_{k-1} I_2$

where $I_1.item_1 = I_2.item_1$ and

×

$I_1.item_{k-2} = I_2.item_{k-2}$ and

$I_1.item_{k-1} < I_2.item_{k-1}$

After this step, there is some pruning done. From C_k , all itemsets $c \in C_k$ where some $(k-1)$ -subset of c is not in F_k are deleted. We use the k -way join to do this step. From the generated k -itemset from the join, we know that two of its $(k-1)$ -itemsets are already known to be frequent since it was generated from two itemsets in F_{k-1} . We validate the remaining $k-2$ subsets for memberships. These are done by additional join predicates, which skip one item at a time from the k -itemset. The tree diagram for this process is shown below.

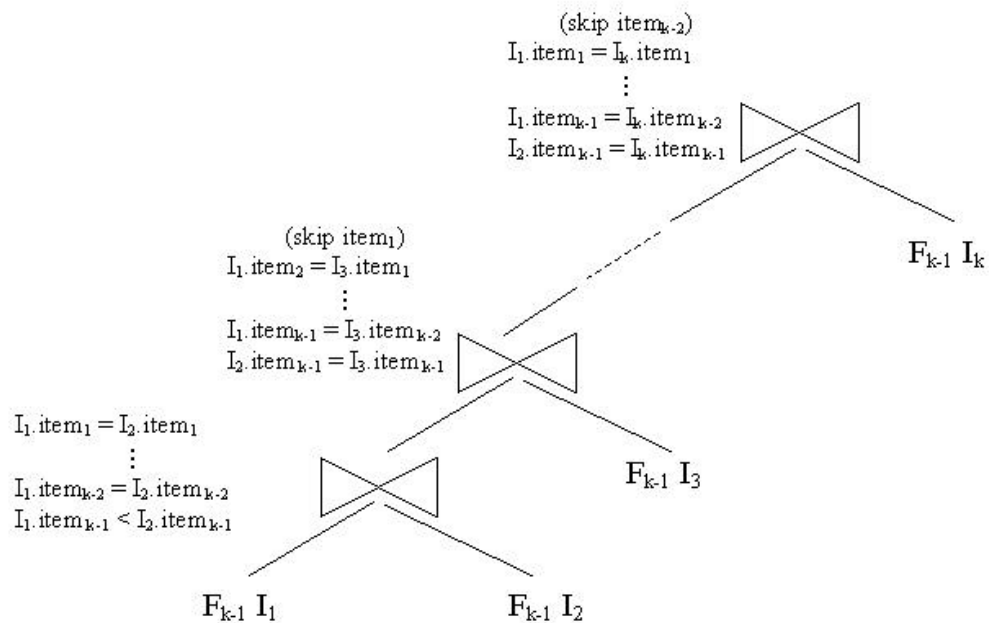


Figure 3.1 Candidate generation for any k

For example, let F_3 be $\{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\}$. After the join step, C_4 will be $\{\{1\ 2\ 3\ 4\}, \{1\ 3\ 4\ 5\}\}$. In the prune step, all itemsets $c \in C_k$, where some $(k-1)$ subset of c is not in F_{k-1} as mentioned before are deleted. Thus the prune step will delete the itemset $\{1\ 3\ 4\ 5\}$. The itemset $\{1\ 3\ 4\ 5\}$ is known to be generated from the subsets $\{1\ 3\ 4\}$ and $\{1\ 3\ 5\}$. But the subset $\{3\ 4\ 5\}$ is not in F_3 . Hence it is deleted and C_4 contains only $\{1\ 2\ 3\ 4\}$.

3.5 Support Counting

This is the important and time-consuming part of the mining process. The input table and the generated candidate itemsets are used to generate the rules. We present 3 methods of support counting in the SQL-92 category first and then the next 3 approaches belong to the SQL-OR category.

3.5.1 K-way Join

It is the simplest approach of support counting. In each pass k , we join k copies of the input table with the candidate itemsets C_k and do a group by on the itemsets.

The basic idea in using the k copies of the input tables is that, we will need to compare the k items in the candidate itemset C_k to the items in input table. We take k copies of the input tables and compare one item from each of the tables to the items in the candidate itemset C_k . Since there are k items to be compared, we use k copies of input table and hence we can compare all the k items in one shot. We then do a group by on the k items and all items whose $\text{count}(\ast)$ is $\geq \text{minsup}$ are potential items for the rule generation phase. The SQL statements and the tree diagram for support counting with k -way join approach is shown below.

```

insert into Fk
  select item1, ... , itemk, count(*)
  from Ck, T t1, ... , T tk
  where t1.item = Ck.item1 and
        ×
        tk.item = Ck.itemk and
        t1.tid = t2.tid and
        ×
        tk-1.tid = tk.tid
  group by item1, item2, ... , itemk

```

having count(*) \geq minsup

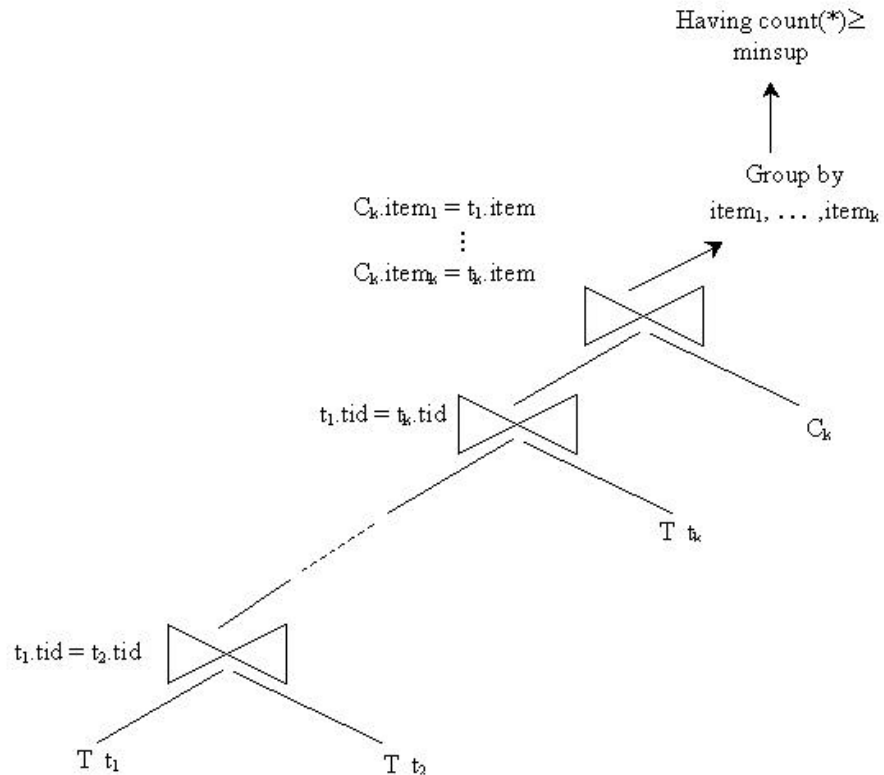


Figure 3.2 Support counting by K-way join approach

A more detailed explanation of K-way join and its flavors can be found in Thomas and Chakravarthy [Tho99]. There are certain advantages and disadvantages in using this approach. This approach is very simple and easy to use. It is very easy to understand also. But this approach would involve a lot of joins and a higher order of joins. That is to say that there are a lot of multi-way joins. In the k^{th} pass, this approach needs a $(k+1)$ way join.

3.5.2 Two Group By

This approach avoids the multi-way joins of the previous k-way join approach. Here we join the candidate itemset C_k and the input table T . The join condition checks whether the item present in the input table T is equal to any of the k items in the candidate itemset C_k . If

so, then group all such items, i.e., do a group by on the $(item_1, item_2, \dots, item_k, tid)$ with a filtering condition that the count of such items is equal to k . The result is a set of items and their tid such that this tid supports the itemset in C_k . Once all such itemsets are identified, we do a group by on $(item_1, item_2, \dots, item_k)$ on these itemsets with the having clause being the $count(*) \geq minsup$. The SQL statements involved are shown below.

```

insert into Fk
  select item1, item2, ... , itemk, count(*)
from (
  select item1, item2, ... , itemk, count(*)
from T, Ck
where item = Ck.item1 or
      ×
      item = Ck.itemk
group by item1, item2, ... , itemk, tid
having count(*) = k
) as temp
group by item1, item2, ... , itemk
having count(*) ≥ minsup

```

With this approach also, there are certain advantages and disadvantages. This approach gets around the problem of multi-way joins and the number of joins are comparatively lesser than the k -way join, but this approach suffers from the overhead involved in the comparisons and executions of group by and having clauses. In fact, the group by and the having clauses have to be executed twice. Once during the grouping of the items in the input table and the second time for the actual support counting.

3.5.3 Subquery Based

This approach makes use of the common prefixes for support counting. There are lots of intermediate subqueries generated in this approach. We will denote the subqueries by Q ,

meaning that it is the f^{th} sub-query. Note that there is no subquery Q_0 . The SQL statements and the tree diagram are shown below.

```

insert into Fk
  select item1, item2, ... , itemk, count(*)
  from (Subquery Qk) t
  group by item1, item2, ... , itemk
  having count(*) ≥ minsup

```

Subquery Q_i ($1 \leq i \leq k$)

```

select item1, item2, ... , itemk, tid
from T ti, (Subquery Qi-1) as ri-1,
  (select distinct item1, item2, ... , itemk from Ck) as di
where ri-1.item1 = di.item1 and
      ×
      ri-1.itemk-1 = di.itemk-1 and
      ri-1.tid = ti.tid and
      ti.item = di.itemk

```

Subquery Q_0 : No subquery Q_0

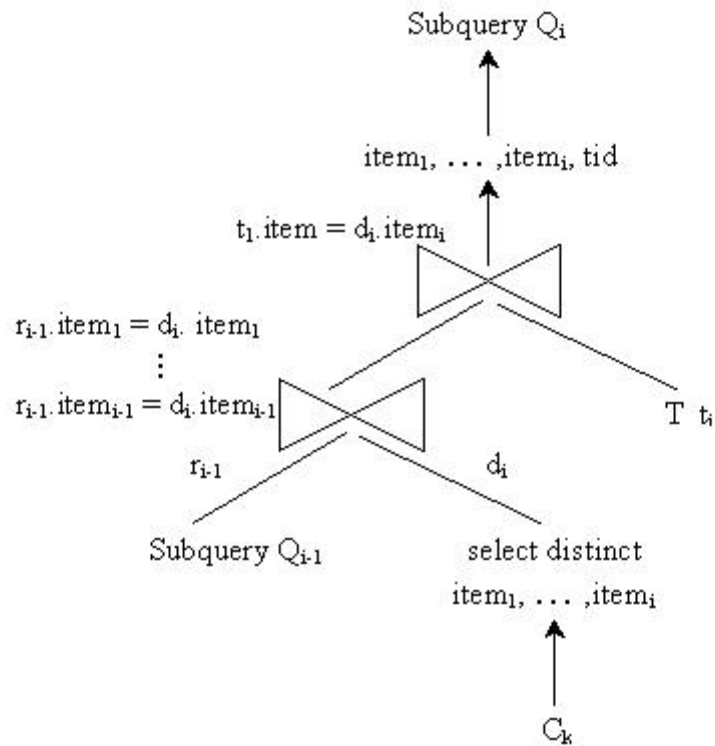


Figure 3.3 Tree diagram for subquery Q_i

Subquery Q_i will select items from tables Q_{i-1} , C_i and input table T . The condition being that the items in Q_{i-1} match with those in C_i and tid in T matches with that in Q_{i-1} . In any pass, say m , $(m-1)$ items ($item_1, item_2, \dots, item_{m-1}$) in Q_{m-1} are matched with items in C_m . Also tid in T and Q_{m-1} are matched. If all of them match, then all such items are inserted into F_k if the support of these items is $\geq \text{minsup}$.

3.5.4 Vertical

This is a SQL-OR approach. In this approach, we transform the input table into a vertical format by creating a CLOB for each of the item in the data table. The CLOB for an item contains all the TIDs that contain this item in the input table. These tid lists are then merged together as a part of the support counting phase. Basically for each of the items in F_k ,

the corresponding tid-lists are intersected for support counting. Since the tid-lists are in sorted order, the intersection is easy. We can use an optimal approach like the sort-merge algorithm for the intersection process. In this approach several DB2 UDFs are made use of viz. SaveTid, CountAnd2, CountAnd3, ..., CountAndK. The UDF saveTid takes 3 input parameters. The first is the item, second is the tid and the third is the number of rows in the input table. The tuples in the input table are input to this UDF one by one and the UDF generates the CLOB of the tids for each of the items. The UDF has 3 output parameters namely, the item, the count of the number of tids in the CLOB for each of the items and the tid list for the item. The output is actually a table and the three mentioned parameters are the columns of the output table. The pseudocode of the UDF is shown below.

```

saveTid(int  item,  int  tid,  int  rowCount,  int  T_item,  int  T_cnt,
COM ibm.db2.app.Clob T_tids)
{
    if item = prev_item
        add tid to tid_list T_tids
        increment count
    else
        create a new tid_list T_tids
        add the item to this new tid_list T_tids
        reset and increment the count
    end if
    set(4, item)
    set(5, count)
    set(6, T_tids)
    setSQLstate("02000")
}

```


The UDF compares the item that is input with the previous item. If they are the same, then this tid also supports that item and hence this tid goes into the tid_list T_tids for the previous item. Also the count is incremented to indicate the increase in the number of transactions that support this item by one. If the items differ, then the tids should be in different tid_lists. Hence a new tid_list T_tids is created and the tid is added to this list. The count is reset and is incremented by one. At the end, there is a need to set the values of the return columns of the table. The *set* API is used for this. The columns are addressed by their index numbers (column positions) and the appropriate values are set. Finally there is a need to indicate the end of the table, which is done by setting the SQL state to “02000” using the *setSQLstate* API. The other UDFs used are the CountAnd2, CountAnd3, etc. These UDFs accept a number of CLOBs (tid_lists) as the input parameters and output the count that is the number of tids that support this itemset. In other words, the tid_lists are compared and the count of the number of common items is returned. For example CountAnd2 accepts two tid_lists. Then for each of the elements in the list, it checks if the element under consideration is present in both the tid_lists. If so, the count is incremented. Thus at the end of the invocation of the UDF, we will end up with a count of the number of common elements in the two tid_lists, which is nothing but the support for the itemset. Similarly for CountAnd3, CountAnd4, etc. The only difference is in the number of tid_lists compared, which is increased by one for each of the higher named UDF. The number that is juxtaposed at the end of the name indicates the number of tid_lists compared. Following is the description of a generic CountAndK UDF where k tid_lists are compared. Note that unlike the *saveTid* UDF, the last statement of setting the SQL state is not required in this case as this is a scalar

function that just returns a scalar value while the *saveTid* UDF was a table function and returned a table.

```
CountAndK(COM i bm db2. app. Cl ob   tid s1,   COM i bm db2. app. Cl ob   tid s2,   ...   ,
COM i bm db2. app. Cl ob   tid sk, int count)
{
    for the length of a tid_list do // scanning only one of the
                                     // tid_lists suffices
        if the element in the tid_list is present
        in all the other tid_lists
            increment the count by 1
    end for
    set(k+1, count)
}
```

The steps involved in this approach can be summarized as follows.

- From the input table, prepare the *tid_list* using the *saveTid* UDF. This *tid_list* is a table, by name TIDT that is the output of the UDF *saveTid* and contains the *tid_lists* for each of the items in the input table.
- For pass 1,
 - For generation of table F_1 , all items from the input table are grouped and those with a $\text{count} \geq \text{minsup}$ are the potential items in the table F_1 .
- For all other passes,
 - For table C_k , k copies of F_{k-1} are joined with the condition that the itemset be frequent.
 - Table F_k is generated by joining k copies (I_1, I_2, \dots, I_k) of table TIDT and the table C_k with the join condition being that items in C_k be present in the copies of table TIDT. The *tid_lists* from the copies of TIDT are then passed on to the scalar function

CountAndK, which compares the tid_lists and returns the count. This count as mentioned earlier is the support of the tids for the itemset.

3.5.5 Gather Join

This is another SQL-OR approach. This is a bit different from the Vertical approach. Here we create a CLOB of items for each of the TIDs in the input data table. We have a UDF for this CLOB creation. The output of this UDF is then passed on as an input to another UDF that gives all the possible k-item combinations formed out of the items in the COLB. Each record that is output by this UDF is in the format $T_item_1, T_item_2, \dots, T_item_k$. This output is then joined with C_k and the items are compared. From the comparisons, all items whose support \geq minsup are the prospective elements of candidate itemsets. In this approach DB2 UDFs namely saveItem, Comb2, Comb3, \dots , CombK are used. The UDF saveItem takes 2 input parameters. The first is the tid and the second is the item. The tuples in the input table are input to this UDF one by one and the UDF generates the CLOB of the items for each of the tids. The UDF has 3 output parameters namely, the tid, the item_list for the tid and the count of the number of items in the CLOB for each of the tids. The output is actually a table and the three mentioned parameters are the columns of the output table. The pseudocode of the UDF is shown below.

```

saveItem(int tid, int item, int T_tid, int T_cnt, COM IBM db2. app. Clob
T_items)
{
    if tid = prev_tid
        add item to item_list T_items
        increment count
    else
        create a new items_list T_items

```

```

    add the item to this new items_list
    reset and increment the count
end if
set(4, item)
set(5, count)
set(6, T_items)
setSQLstate("02000")
}

```

The UDF compares the tid that is input with the previous tid. If they are the same, then this tid also supports that item and hence this item goes into the *items_list* for the previous tid. Also the count is incremented to indicate the increase in the number of transactions that support this item by one. If the tids differ, then the items should be in different *items_lists*. Hence a new *items_list* is created and the item is added to this list. The count is reset and is incremented by one. At the end, there is a need to set the values of the return columns of the table. The *set* API is used for this. The columns are addressed by their index numbers (column positions) and the appropriate values are set. Finally there is a need to indicate the end of the table, which is done by setting the SQL state to "02000" using the *setSQLstate* API. The other UDFs used are the Comb2, Comb3, etc. These UDFs accept two input parameters. First is the tid and the second is the CLOBs (*item_lists*) and output the 2-item, 3-item, ... , k-item combinations of the items in the COLB. For example Comb2 accepts two parameters, tid and *items_list*. It then generates the 2-item combinations of the input *items_list* and returns them. Similarly for Comb3, Comb4, etc. The only difference is in the type of item combinations returned whether they are 2-item, 3-item, or k-item combinations. The number that is juxtaposed at the end of the name indicates the type of itemset returned.

Following is the description of a generic UDF CombK, which returns all the k-item combinations.

```

CombK(int tid, COM ibm db2. app. Clob items, int T_item1, int T_item2, ... ,
int T_itemk)
{
    for the length of items list items do
        generate all the k-item combinations and
        insert them into the table
    end for
    set(3, item1 of the k-item combination)
    set(4, item2 of the k-item combination)
        ×
    set(k, itemk of the k-item combination)
    setSQLstate("02000")
}

```

The steps involved in this approach are as follows.

- From the input table, prepare the items_list using the saveItem UDF. This item_list is a table, by name TITEM that is the output of the UDF saveItem and contains the items_lists for each of the tids in the input table.
- For pass 1,
For generation of table F_1 , all items from the input table are grouped and those with a count \geq minsup are the potential items in the table F_1 .
- For all other passes,
Table F_k is generated by joining the tables TITEM and the table returned by the CombK. The join result is grouped by T_item_k, T_item₂, ... , T_item_k. The result is then passed through a having clause of condition count \geq minsup. All the resultant items are potential candidates for rules generation. Actually, the items in the table TITEM are passed to the

UDF CombK and the k-item combinations with respect to this TITEM table are returned by CombK.

3.5.6 Gather Count

This is the last SQL-OR approach. It is very much similar to the Gather Join approach. But in this approach we utilize the index on table C to probe into the table C_k , after the CLOBs and the k-item combinations are created. Also the candidate itemsets C are created here. This approach also makes use of the UDFs saveItem , Comb2, Comb3, ... , CombK.

The steps involved in this approach are as follows.

- From the input table, prepare the items_list using the saveItem UDF. This item_list is a table, by name TITEM that is the output of the UDF saveItem and contains the items_lists for each of the tids in the input table.
- For pass 1,
 - For generation of table F_1 , all items from the input table are grouped and those with a count \geq minsup are the potential items in the table F_1 .
- For all other passes,
 - Generate table C_k from k copies of F_{k-1} with the join condition that $item_1, item_2, \dots, item_k$ form a frequent itemset.
 - Generate the k-item combinations using the UDFs CombK using table TITEM. The results are then joined with C_k . There is a unique field in table C_k called oid. This oid identifies each of the itemsets uniquely and an index is created on this field for easy and fast probing into the itemsets. All the matching items are then passed through a

group by oid. This temporary table is then joined with C_k and the index on oid is used for joining. Items $item_1, item_2, \dots, item_k$ are then inserted into the table F_k .

3.6 Rule Generation

This is the second and last phase of the association rule mining. We use the frequent itemsets produced in the support counting phase to generate the rules. For each of the frequent itemsets l , we find all its non-empty proper subsets. For each of the non-empty subsets m , we find the confidence of the rule $m \Rightarrow (l-m)$ and if this confidence is at least as much as the user specified minconf, we output that rule.

In the support counting phase, we store all the resultant itemsets of size k in table F_k . The first step is to consolidate all the frequent itemsets into one table. We name this table FISETS. The schema of the table FISETS is $(item_1, item_2, \dots, item_k, nullm, count)$. Here nullm indicates the null marker, which is the end of the itemset. and count gives the support for that itemset. Now for each of the items in the FISETS, we need to generate the non-empty subsets of the form, *Rule head* **P** *Rule Body*. We generate the subsets for each of the tuples in the table FISETS and insert all the generated subsets in a table called the SUBSETS. For example, if the itemset contains {bread, milk, butter}, we generate six subsets. The first three are with one item in the rule head and two in the rule body while the last three have two items in the rule head and one item in the rule body.

bread **P** *milk, butter, milk* **P** *bread, butter, butter* \Rightarrow bread, milk; *bread, milk* **P** *butter, bread, butter* **P** *milk, butter, milk* **P** *bread*; The schema of the table SUBSETS is $(T_item_1, T_item_2, \dots, T_item_k, T_nullm, T_rulem, T_count)$. The $T_item_1, \dots, T_item_k$ are the items in the itemset. T_nullm is the same as in the FISETS. It is the null marker. T_rulem is the rule marker. It is the position of separation of the rule head and the rule body. In other words

it gives the position of the “ \Rightarrow ” symbol in the rule. The subsets are generated using the function GenSubsets. Now the tables contain itemsets whose support is at least equal to the user specified minsup. Hence the rule generation is simple. We just have to join the two tables FISETS and SUBSETS with the condition that the ratio of

$$\frac{SUBSETS_count}{FISETS_count} \geq \text{minconf.}$$

The schema for the table RULES is (item₁, item₂, ... , item_k, nullm, rulem, confidence, support)

The query below and Figure 3.4 illustrate the rule generation.

insert into Rules

```

select T_item1, ..., T_itemk, T_nullm, T_rulem,
       (float(SUBSETS.T_count)/FISETS.count)*100,
       SUBSETS.T_count
from SUBSETS, FISETS
where (SUBSETS.T_item1 = FISETS.item1 OR SUBSETS.T_rulem <= 1 )
      AND (SUBSETS.T_item2 = FISETS.item2 OR SUBSETS.T_rulem <= 2 )
      ×
      AND (SUBSETS.T_itemk = FISETS.itemk OR SUBSETS.T_rulem <= k )
      AND SUBSETS.T_rulem = FISETS.nullm
      AND SUBSETS.T_count*100/ FISETS.count >= minconf

```

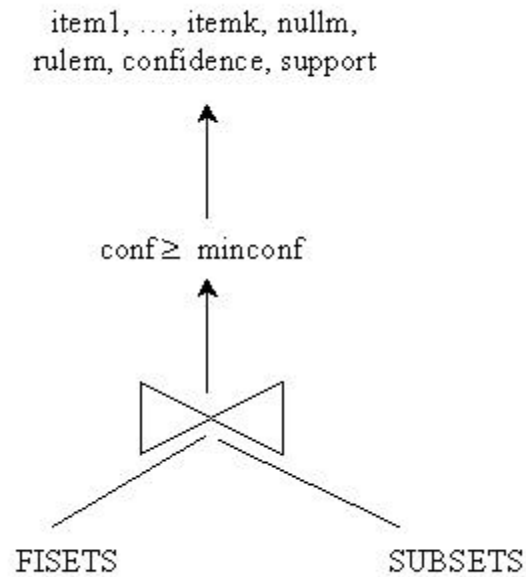



Figure 3.4 Rule Generation

The consolidation of the tables and the routines for generating the rules can be implemented as UDFs or table functions also. The fraction of total running time spent in rule generation is very small. Hence, we do not focus much on other rule generation algorithms.

3.7 Conclusion

In this chapter, we stepped through the process of association rule mining. An insight into the apriori algorithm was given first and later we discussed the candidate generation process. Various approaches for support counting of the rules were looked at in the following sub-sections. Lastly we gave a brief overview about the rule generation. In the next chapter, we look into the issues of mapping the input data into the format desired by the rule generator and also discuss the layered approach of utilizing the metadata for association rule mining.

CHAPTER 4 INPUT MAPPING AND METADATA

4.1 Introduction

This chapter gives a brief overview of the process of mapping the input data into a format that typically is used by most of the association rule mining algorithms. When there are more than one table used as input for the association rule mining, they need to be mapped to the typical input format. Lastly the topic of metadata is discussed and we elaborate on the information that can be stored as the metadata to aid in deciding the best available approach for rule mining.

4.2 Mapping

As mentioned in the earlier chapters, most of the association rule mining tools require the input to be in the (tid, item) format. But in real life applications, the data that is input to the mining algorithms are seldom in this required format. Hence there is a need to map this input before it is processed. Also since the number crunching capability of a computer outperforms the ability of processing character strings, we enforce the new constraint that the input table be in the integer format. These two constraints increase the work on the part of the user. If the user were supposed to do all the transformations, then the mining algorithms would not carry much importance. Hence, we try to incorporate the two levels of mapping into a layer of our implementation and take the burden off the end user. The whole mapping process would start once the input tables are identified. If the input table is just a single table, the mapping process can begin immediately. Otherwise, first the input table needs to be

created from the Union/Join of the user-selected tables and the mapping done on the new input table. If the operation is a join, then the user also provides the join columns in the selected tables. Once the input table is prepared, the user needs to select the columns that need to be treated as tid columns and those that need to be interpreted as items. In our implementation, we have incorporated the feature where the user can select multiple columns that would together form a tid column, similar to the concept of a composite key in a relational table. Also among the remaining columns, the user can select all or some of them to be treated as items. This feature is not available in many of the commercial data mining tools available in the market at present. The output of the mapping process will be three tables. First is the table that contains the tids mapped to unique integers, which we call the MappedTidsTable. Second is the table MappedItemsTable which has the items mapped, and the last is the FinalInput table which is the same as the original input table with all the values now mapped to integers with the information about the mapping present in the earlier mentioned two tables. We need to take care of the data types of the columns in the tables when mapping. In our implementation, the two tables generated are MappedTidsTable and MappedItemsTable. The MappedItemsTable table contains two columns. One for the item description and the other that represents its unique integer representation. Its schema is (ItemD character(20), ItemI integer). The other table MappedTidsTable will have $(k+1)$ columns, where k is the number of attributes that the user selects to be treated together as the tid column. The last column is the integer representation of the tid value. Its schema is (tidD₁ character(20), tidD₂ character(20), ..., tidD_k character(20), tidI integer). The mapping is such that a set of values for the tid columns, if different in at least one of the columns, will be treated as two different transactions. If they are the same in all the fields, they would

necessarily mean the same transaction. At the beginning of this thesis work, it was decided to implement mapping so that there would be only one column that would form the tid column. But then it was noticed that with a multiple attribute tid, the data could be more precisely interpreted than with a single attribute tid. For example, suppose an input table has a customer id as the tid column and let us assume that there is another attribute called the date, which represents the date of the transaction. If we were to have only customer id as the tid column, then the transactions on different days would be interpreted as the transactions on a single day. Now with both customer id and the date attributes representing the Tid column, the transactions of a customer on different days would be treated as different records. With this interpretation, the input data is more precise and specific. For both the tables, the description fields are characters and immaterial of the underlying data type in the input table, they are inserted into the table as characters. When generating the final input table, the data types are accordingly compared and the needed tuples generated. The generation of the final input is a join of the MappedTidsTable, MappedItemsTable and the original input table. The join condition is tids matching for tables MappedTidsTable and the original input table, and the other join condition is that any item value in the input table matching with a element in the MappedItemsTable. We try to compare the elements in the original table with those in the mapping tables and then take the corresponding elements' integer representation from the two tables. Thus considering only the integer representations from the two mapping tables, our final input table will have two columns that are both integers. Its schema is (Tid integer, Item integer). For example, let the table below represent a database table and be the input table for the mining optimizer.

Table 4.1 Input table

Date	Customer Id	Item ₁	Item ₂	Item ₃
1/1	John	Cookies	Camera	Shirt
1/2	William	Boots	Shorts	Socks
1/2	Parker	Milk	Bread	Chocolates
1/3	Meg	Shoes	Lipstick	Shampoo
1/1	John	Pen	Paper	Glue
1/4	Parker	Camera	Cookies	Gum

Suppose the customer id alone was the tid and the user selected item₁, item₂ and item₃ as the item columns. In such a case, all distinct values in the customer id column are mapped to unique integers. The distinct items in the three columns are also mapped to unique integers. The two tables are shown below.

Table 4.2 MappedTidsTable

TidD	TidI
John	1
Meg	2
Parker	3
William	4

Table 4.3 MappedItemsTable

ItemD	ItemI
Boots	1
Bread	2
Camera	3
Chocolates	4
Cookies	5
Glue	6
Gum	7
Lipstick	8

Milk	9
Paper	10
Pen	11
Shampoo	12
Shirt	13
Shoes	14
Shorts	15
Socks	16

Now we perform a join over the two mapping tables and the original table and generate the final input table, which is shown below.

Table 4.4 Final Input

Tid	Item
1	5
1	3
1	13
1	11
1	10
1	6
2	14
2	8
2	12
3	9
3	2
3	4
3	3
3	5
3	7
4	1
4	15
4	16

In the input table, customer id Parker is associated with items Milk, Bread, Chocolates, Camera, Cookies, and Gum, which are mapped to integers 9, 2, 4, 3, 5, 7 respectively as seen in the MappedItemsTable table and Parker is mapped to 3 as seen in the MappedTidsTable table. Note that here the tid is just the customer id. Hence in the final input table, Parker is associated with six items even though three items were bought in two separate transactions on different days. This would not be the case, had we used the date and customer id together as the tid columns, using which the table MappedTidsTable is shown below.

Table 4.5 MappedTidsTable with new Tid field

TidD1	TidD2	TidI
1/1	John	1
1/2	Meg	2
1/2	Parker	3
1/3	William	4
1/4	Parker	5

The table MappedItemsTable still remains the same supposing the user selected item₁, item₂ and item₃ as the itemcolumns. The resultant final input table is a shown below.

Table 4.6 Final Input table with new Tid Field

Tid	Item
1	5
1	3
1	13
1	11
1	10
1	6
2	14
2	8
2	12

3	9
3	2
3	4
4	1
4	15
4	16
5	3
5	5
5	7

We see that since the date field was also included, the two transactions with customer id Parker are treated as two separate transactions and are mapped to distinct integers 2 and 5 and each transaction is now associated with three items each.

4.3 Reverse Mapping

This process is after the rules generation phase. Once the rules are generated, they are still in the number format. Hence there is a need for mapping those integers back to the original values. This is the process of reverse mapping. For example if Beer and Diapers were mapped to 1 and 2 respectively, and we had a rule $1 \Rightarrow 2$, then this rule should be mapped to Beer \Rightarrow Diapers. Reverse mapping does this. We look up the rules

table and the MappedItemsTable table to get the desired results. We traverse through the rules table and replace each occurrence of the integer by its corresponding item description in the MappedItemsTable.

4.4 Metadata

Metadata are *data about data*. We store some information as part of metadata and utilize this information in determining the best available approach for the association rule mining. We store the database related and user input table related information as a table in the underlying database. Before the process of mining begins, we access this metadata and depending on the information available, we select an approach for rule mining. This is a layered approach of optimization on the mining process. Some user specified constraints and conditions are also a part of the metadata.

Below are listed the salient features of the approaches of support counting. This information can be stored as metadata along with some information about the input table and also some database specific information. All this can be used before selecting a approach and the best among them can be used for rule generation.

- *K-way Join*: This is the basic and simple approach. But it requires a $(k+1)$ -way join in the k^{th} pass. Suppose the underlying database has certain constraints wherein it supports only up to l -way joins, $l < k$, it becomes necessary to use some other approach.
- *Two-group by*: There are comparatively lesser number of joins in this approach and so are the multi-way joins. But in this approach some time is lost in the execution of SQL query comparison, grouping and having clauses, in fact more for the grouping clause since it is executed twice.

- *Subquery*: This approach makes use of the common prefixes between the itemsets in C_k to reduce the amount of work done during support counting. Recursively, subqueries are generated and hence the main query becomes easy to understand. For a comparatively fast processor and ample of memory, this would be a better approach.
- *Vertical*: This is one of the best SQL-OR approaches. It is particularly very much suitable for the higher passes. But if the candidate itemsets are very large, this approach suffers. In such cases, it is better to use Gather Join approach.
- *Gather Join*: If the input items are already in the horizontal format, there is no need to gather the input in the format for this approach. In such a case, the table functions are easy to code and becomes modularized. Also, if the candidate itemsets are very large, this approach is preferred over the vertical approach. But if the number of frequent items per transaction is very large, this approach suffers.
- *Gather Count*: This is very similar to the Gather Join approach except that this approach makes use of an index for faster probing into the tables.

After considerable performance testing we found that our application runs much faster on DB2 than it does on Oracle. Partly the reason for this is that in Oracle, there is a need to explicitly materialize certain temporary and intermediate tables, which in DB2 need not be materialized. This database specific information can also be stored as a part of metadata.

It has been seen that among the SQL-92 approaches, the fastest approach is the K-way join approach on DB2. If the number of tables generated in the process of rule generation is a constraint that the user needs to take care of, then the best approach would be again the K-way on either Oracle or DB2. If this is not a constraint then one would use

Subquery rather than the other two approaches. With number of joins being another constraint, the winner again is the K-way join in both DB2 and Oracle.

4.5 Conclusions

In this chapter we looked into the issues of mapping and metadata. The process of mapping was described, wherein the input data is transformed to integers so that it is easier and suits the rule generator. After the rule generation, there is a need to map back the integers to their original values. This process was discussed in the reverse mapping section. Finally, the layered approach of optimization was described in the metadata section wherein we try to store some important information that will help us in selecting an approach for mining. In the next chapter, we will discuss the performance and scalability of the approaches, synthetic data generation and comparison of our application with other commercially available data mining tools from data mining vendors.

CHAPTER 5 PERFORMANCE, SCALING AND COMPARISONS

5.1 Introduction

In this chapter, we will discuss the performance of the various support counting approaches. We also depict the scalability of our approaches by running on different sized datasets. And lastly, we compare our mining optimizer with the other commercially available tools.

5.2 Synthetic Data Generation

We use the synthetic dataset generator from IBM [Agr94]. Using this generator, one can vary the number of transactions, distinct itemsets, and number of frequent itemsets per transaction in the datasets. We generated a few datasets using this generator. The naming convention of a generated dataset is as follows. A dataset will be named $TmDn$, where m , and n are integers. Tm denotes the average number of items per transaction and Dn denotes the number of Transactions. Thus the dataset will have n number of transactions with an average m number of items per transaction. Also the number of different items utilized in generating the transactions can be specified. Suppose we specify k different items, then these k items will be distributed throughout the transactions. Typically $k \gg m$, else most of the transactions would have similar/same items, or in the worst case, each transaction would have the very same and repeated items.

5.3 Performance Comparisons of SQL-92 approaches

All the experiments were performed on Version 5 of IBM DB2 Universal Server installed on Windows NT workstation with dual Pentium II processors, 128 MB main memory and 12 GB disc.

We generated datasets using the synthetic data generator described in section 5.2 and the performance comparisons were based on these datasets. Table 5.1 summarizes the parameters associated with the datasets.

Table 5.1 Description of the generated datasets

Datasets	# of records (in thousands)	# of transactions (in thousands)	Avg. # of items
T5D1K	5.6	1	5
T5D10K	54.9	10	5
T10D10K	105.3	10	10
T5D100K	547.2	100	5

We experimented with all the datasets and we present the performance comparisons of all the three SQL-92 approaches and the IBM Intelligent Miner [IBM00] on each of the datasets. Figure 5.1 shows the performance comparison of the approaches for dataset T5D10K.

It can be seen from the figure that, Kway join is the best approach. The Subquery approach is comparable to the Kway join approach. The two group by approach performed reasonably well for small sized datasets. But for these datasets, it was running for more than 2-3 hours and had to be terminated.

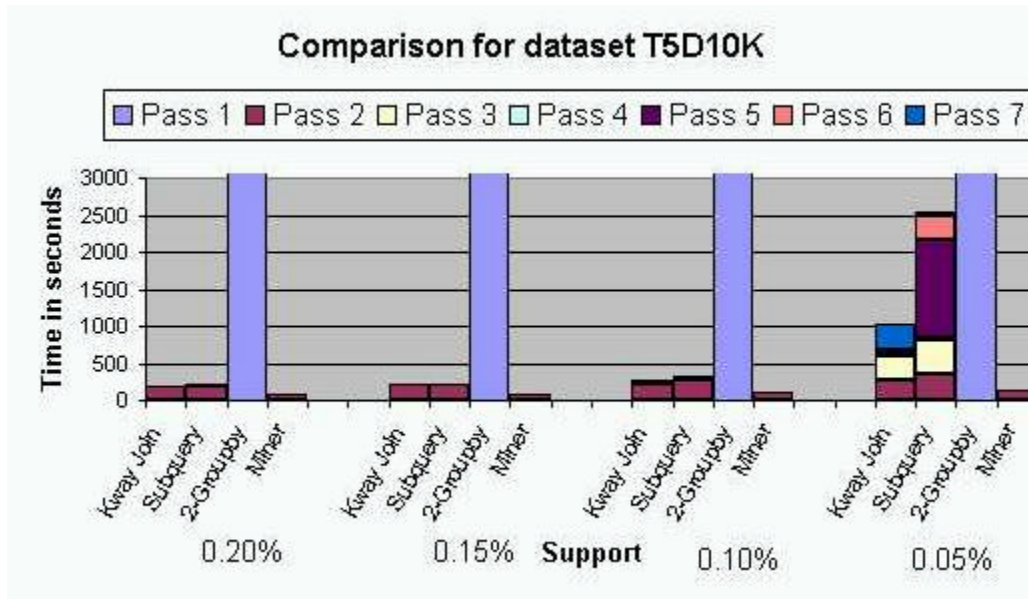


Figure 5.1 Performance comparison for dataset T5D10K.

Figure 5.2 and Figure 5.3 respectively show the performances of the approaches for datasets T10D10K and T5D100K.

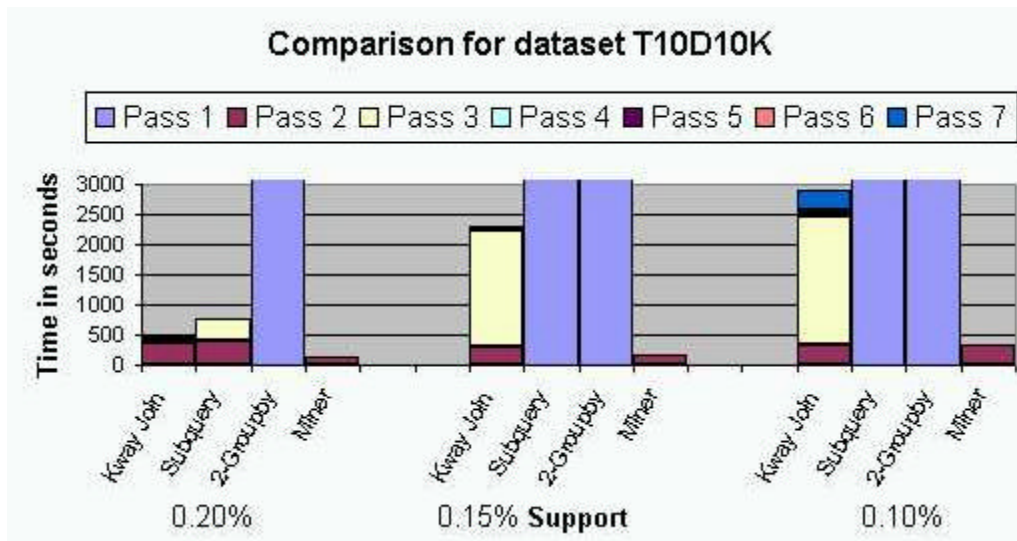


Figure 5.2 Performance comparison for dataset T10D10K.

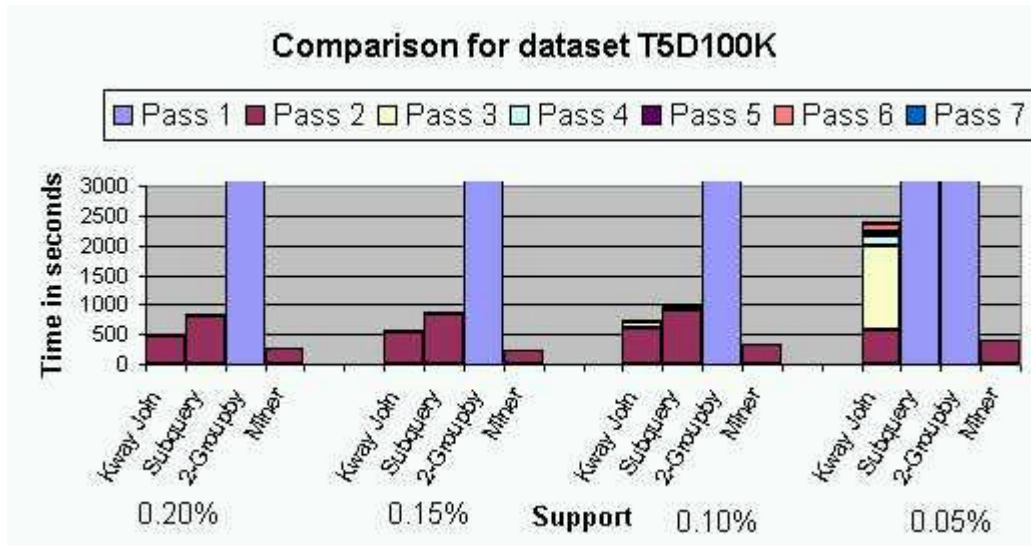


Figure 5.3 Performance comparison for dataset T5D100K.

For the datasets T10D10K and T5D100K, again Kway join was the winner whose response time with respect to rule generation was the least. Another thing to be noted for approaches running on these datasets was that as the support was decreased, all the approaches were taking more time and in a few cases some runs had to be terminated. Figure 5.4 compares the Kway and Subquery approaches pass wise for the dataset T5D10K with support 0.20%. It can be seen from the figure that the second pass is the most time consuming and the important phase. In fact the second pass can be optimized to a certain degree by pruning the non-frequent itemsets from the input transaction table and then using this pruned transaction table for further passes.

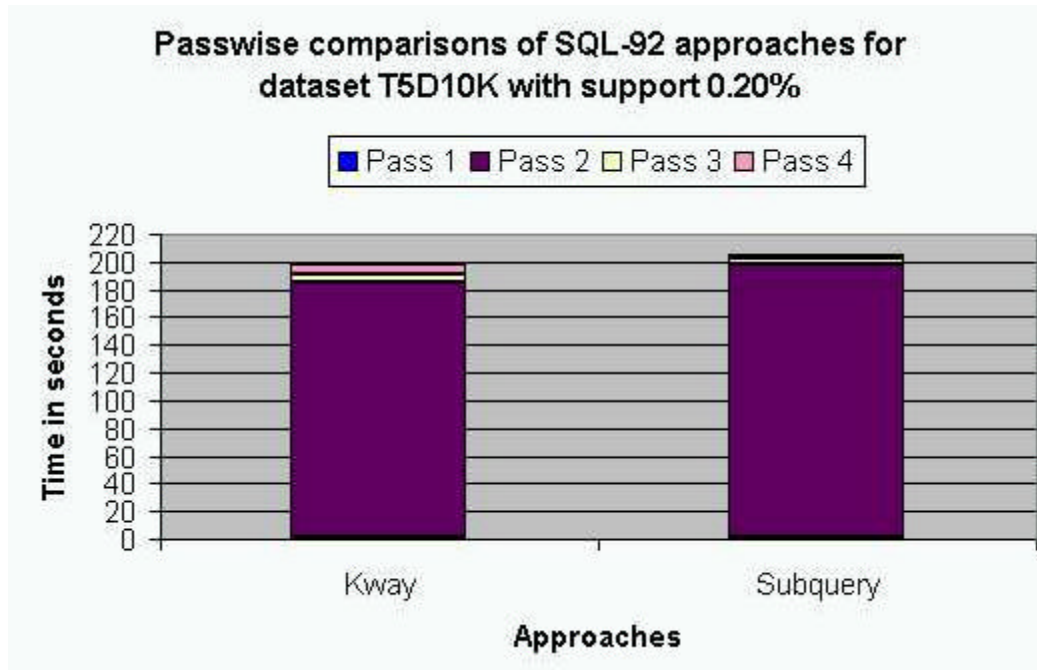


Figure 5.4 Pass wise comparison of Kway and Subquery approaches for T5D10K dataset with 0.20% support.

5.4 Scaling

We also experimented with the synthetically generated datasets to examine the scale-up behavior of the SQL-92 approaches with respect to the increase in the number of transaction/records. Figure 5.5 shows the behavior of Kway and Subquery approaches with the increase in the number transactions/records in the transaction table. Two group by approach was not taken under consideration because of its long run times. The support was kept fixed at 0.20% for all the datasets generated. As can be seen, for a smaller dataset, either of the approaches can be used, but as the number of records increased, apparently Kway emerged as the winner.

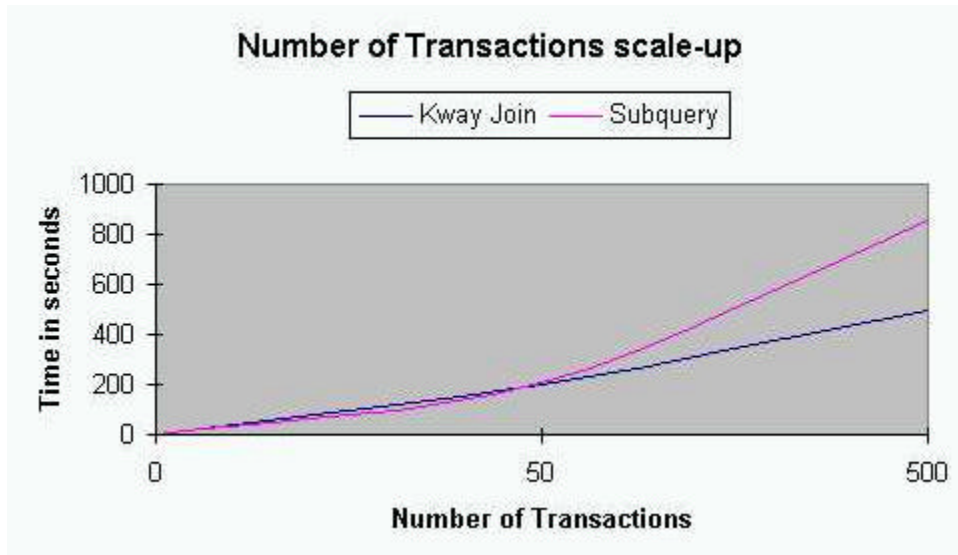


Figure 5.5 Number of Transactions scale-up

5.5 Comparison with Other Commercial Mining Tools

In this section, we try to compare our mining optimizer with the other data mining tools available in the market. Some of the tools we studied were Mineset™ from SGI Inc, and Intelligent Miner™ from IBM corp. Some of the salient differences are listed below.

- Our approaches for rule mining are the SQL based approaches. All the support counting and rule generation algorithms are SQL-based queries. In the other tools, most of them are not SQL-based. They just read the data from the table and perform computations in memory by which they appear to be faster. Since our approach uses SQL, all the intermediate and final results are stored as tables in the underlying relational DBMS. Once the results are computed, they can be easily handled and worked with because of the SQL capabilities.
- Since the others are not database-oriented, they do not have an underlying database. We have an underlying database to store our input and output.

- Our mining optimizer can connect to different databases. It is database independent. Regardless of the type of the underlying database, one can mine with our mining optimizer. The use of JDBC provides us this feature.
- We have used Java as the language of implementation. Therefore there is no platform constraint and is platform independent. Hence it is easily portable. Also the entire mining tool is modularized and hence easily comprehensible.
- In our implementation, we have provided a choice for the user to select an approach for association rule mining. Depending on the input data, database constraints, comprehensibility or personal preference, one can select one of the six approaches provided. All are SQL-based approaches. Three are SQL-92 based and the other three utilize the SQL-OR extensions. On the contrary, we have also provided a means of skipping the process of selection of an approach wherein the optimizer selects an appropriate approach and mines the input data.
- The input table need not be just a single table. The input data can span over multiple tables and the input data can be consolidated into one single chunk by means of join/union of the multiple tables.
- As opposed to other systems, the input format requirement of (Tid, Item) with both integers is simplified in our approach. We accept the horizontal format of a normal table and map the values in the tuples to integers if they are not already integers. At the end of the process, we map back the integers to original descriptions and display the rules.
- Most of the systems make it necessary to have a single field as the transaction ID field. In our implementation we have made it much more flexible. We accept multiple

fields and these together, form the transaction ID just like the composite key in a relational DBMS table.

- Like the single transaction ID field, many systems accept only one field as the items' field. Thus one can mine only two columns from a relational DBMS table. In our implementation, we have provided a means of indicating which columns need to be considered as the items. The user can select from one to all the columns in a table to be treated as the items' columns.
- We provide the user, a means of imposing a constraint on the rules generated. A means by which the user can wish to see/not see certain items in the generated rules, thus controlling the rules generated. This is incorporated by allowing the user to generate a where clause and thereby only selected portions of the input being mined.
- Lastly, as a means of easy comprehensibility for the user, there is visualization provided for the generated rules. The generated rules can be visualized in a table or 3D format.

5.6 Conclusions

In this chapter, we showed the features of our mining optimizer and rule generator, its performances and scalability on the synthetically generated data. Lastly we compared features of our system with those of the commercial tools and indicated the differences. In the next Chapter, we go over the system implementation of the mining optimizer and rule generator. With the help of some screen dumps, we try to illustrate the GUI and the logic of our association rule mining approaches.

CHAPTER 6 SYSTEM IMPLEMENTATION

6.1 Introduction

In this chapter, we present an overview of the system implementation of our mining tool. We present some screen dumps to aid in understanding and also to describe the user interface.

6.2 Mapping and Rule Generation

We start the discussion of our implementation with the main window. Figure 6.1 below shows the main window and as seen in the figure there is a menu bar, tool bar and some text area. The menu bar contains options like *file*, *import*, *result*, *help*, *approaches*, *visualization*, and *mine data*. Under the *file* menu, the user can *connect*, *disconnect*, *save*, *clear* or *print* the messages in the text area or *exit* the miner. The user can connect to different databases, here DB2 and Oracle through the *connect* sub-menu and disconnect from the same through the *disconnect* sub-menu. Under the *import* menu, the user can choose *generate rules*, which is a step-wise process that accepts more input from the user. The user can view the results through the *results* menu. The *approaches* menu contains the sub-menus that are directed towards different support counting approaches. It also has means of registering the DB2 UDFs. The generated rules can be visualized in either the

table format or 3D format by selecting the appropriate sub-menu in the *visualization*

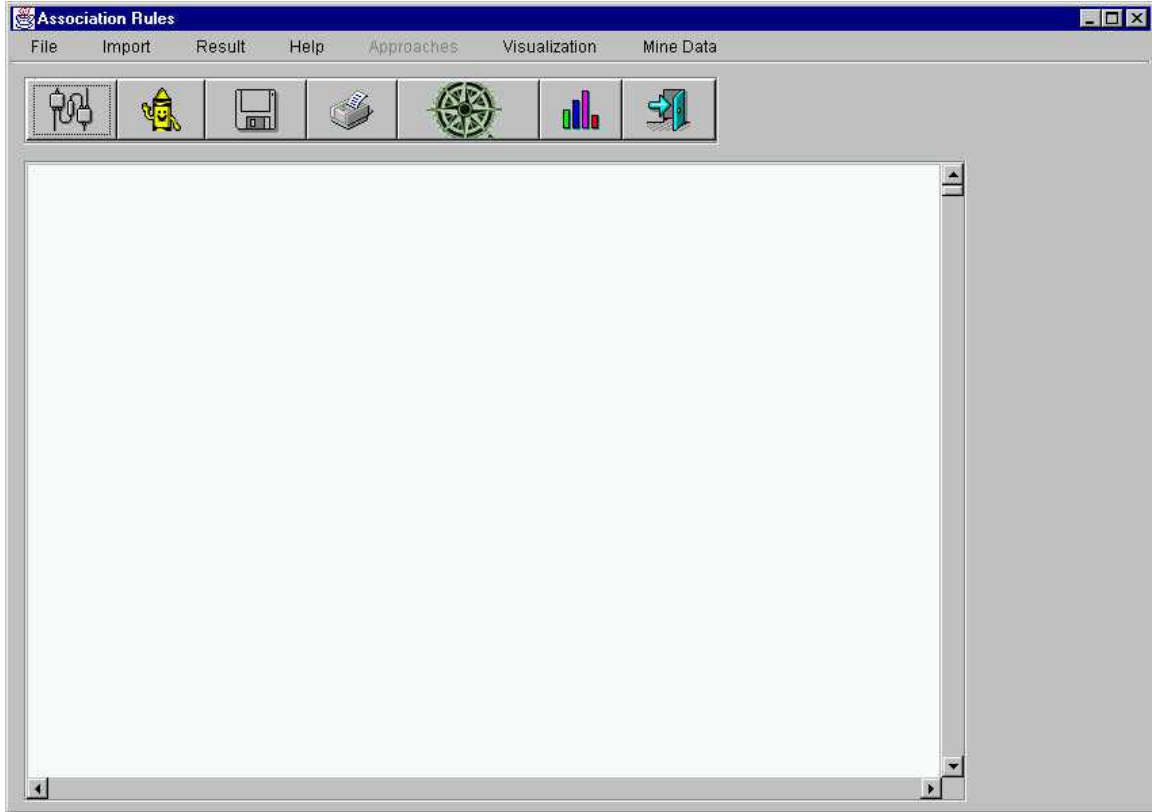


Figure 6.1 Main Window of the association rule miner

menu item. The *mine data* item provides a means of rule generation without any preference by the user for approach selection. This is the optimizer part wherein the underlying metadata decides on the approach selection. The following figure shows the sub-menus of the menu item file and depicts the genre of sub-menus for different menu items.

For the process of rule generation to begin, the user needs to connect to an underlying database. This is achieved by the *connect* sub-menu item in Figure 6.2. Another thing to be noted is that most of the sub-menu items also have a link from the tool bar, for example *connect*, *disconnect*, *generate rules*, *exit*, to name a few. Clicking

on *connect*, takes the user to the next frame wherein the user enters the login information.

This is shown in Figure 6.3.

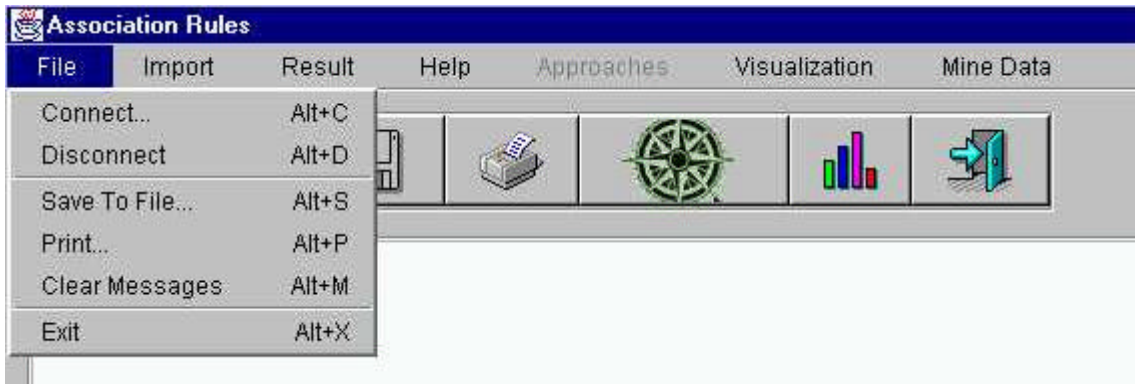


Figure 6.2 Sub-menus of menu item file

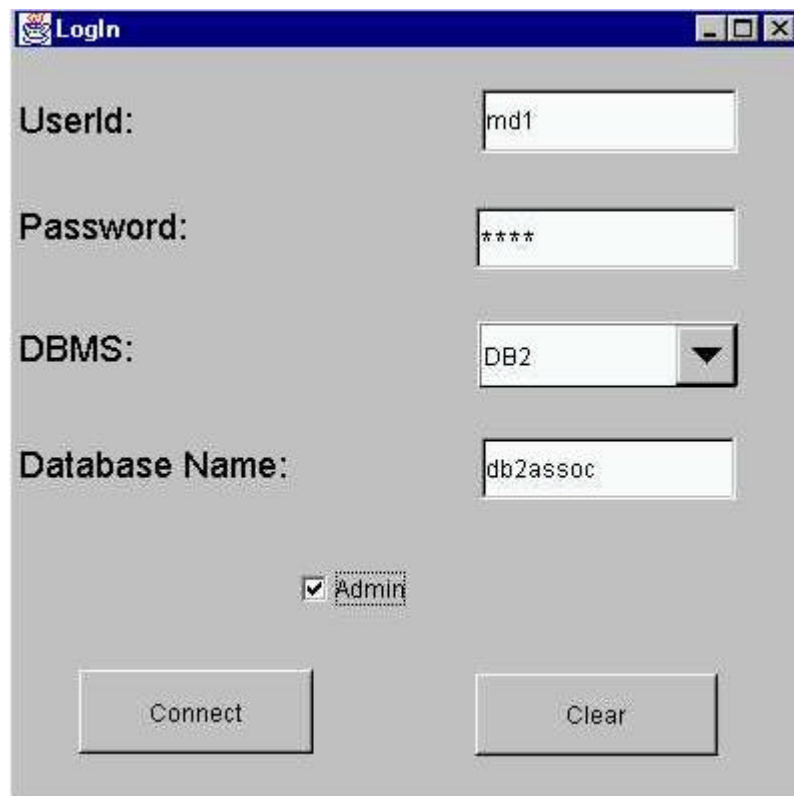
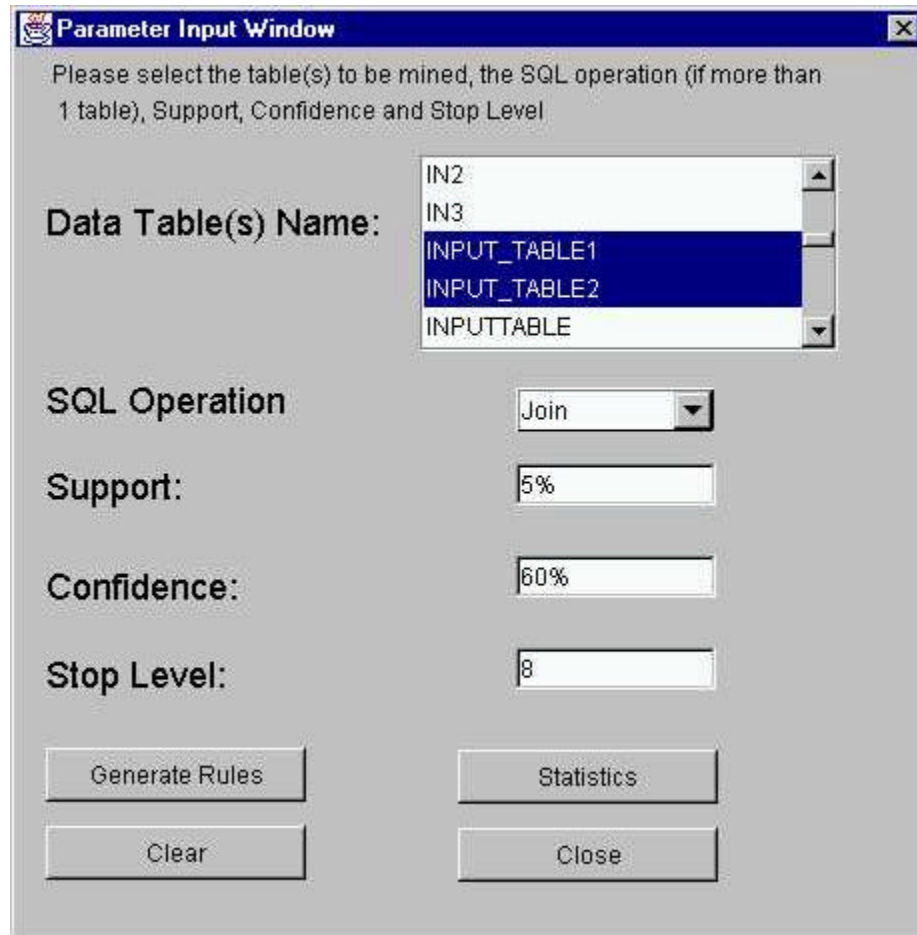


Figure 6.3 The login window

The user needs to select the type of database, here either DB2 or Oracle and also provide the database name which contains the input table. Also, the user enters the account information viz. user id and password for the selected database. There is a check box named *Developer*, which gives the user the capability to select an approach for support counting, without which the user can just mine the input data without a preference for an approach. The associated buttons are self-descriptive not only in this window but through out our implementation. The user gets connected to the selected database and is prompted further for more input using the window shown in Figure 6.4. Here the user selects the input table(s). The reason for us displaying all the existing tables is that this will ease the user not to remember the table names but can select from the displayed list. The user also selects an appropriate operation, a join or union on the selected tables if more than one table, the minimum support and confidence. Stop level is the maximum number of passes allowed by the user for the process of support counting. The *generate rules* button guides the user further into the process and *statistics* button is to give some form of statistical analysis of the input table(s).



The image shows a 'Parameter Input Window' dialog box. At the top, it contains the text: 'Please select the table(s) to be mined, the SQL operation (if more than 1 table), Support, Confidence and Stop Level'. Below this, there are several input fields: a list box for 'Data Table(s) Name:' containing 'IN2', 'IN3', 'INPUT_TABLE1' (highlighted), 'INPUT_TABLE2', and 'INPUTTABLE'; a dropdown menu for 'SQL Operation' set to 'Join'; text boxes for 'Support:' (5%), 'Confidence:' (60%), and 'Stop Level:' (8). At the bottom, there are four buttons: 'Generate Rules', 'Statistics', 'Clear', and 'Close'.

Figure 6.4 Parameter input window

The window in Figure 6.5 is displayed to the user when *generate rules* button is clicked. As can be seen, the user-selected tables are shown in the list box on the left-hand side. The user needs to click on the table names and select the join columns, which would be then displayed in the list box on the right-hand side.

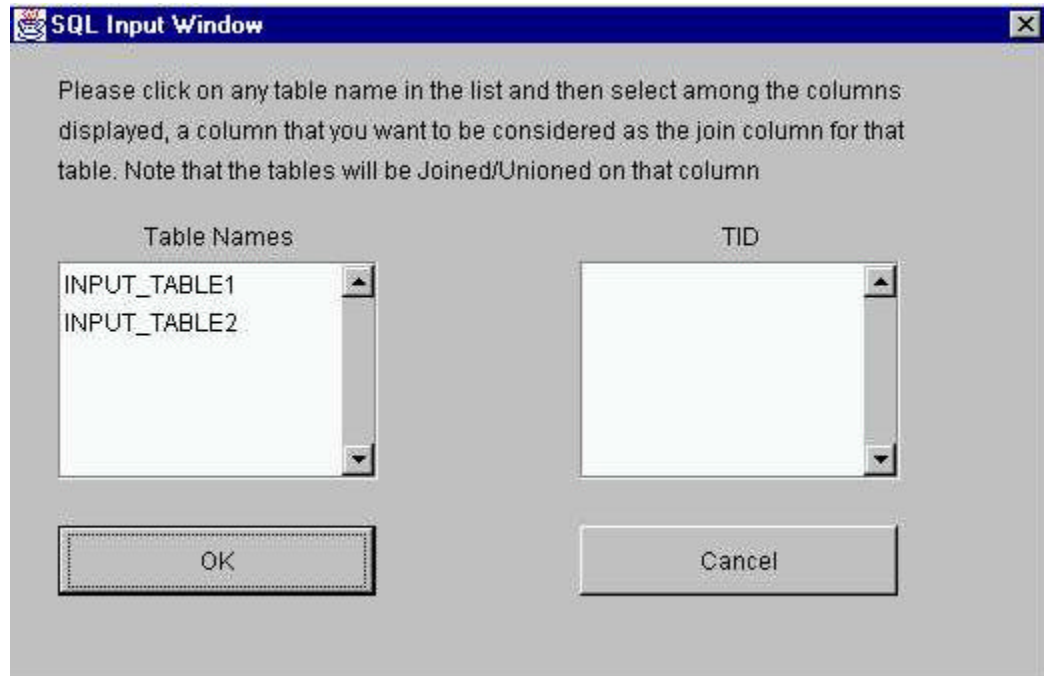


Figure 6.5 SQL input window

The user needs to select the columns that would serve as the join columns for the input tables and further select the Tid and Items' columns from the input tables. Figure 6.6 shows the join columns selection. The tables are joined on the join column. In this example, the user selects T1CustomerID and T2CustomerID as the join columns for the selected tables Input_Table1 and Input_Table2 respectively.

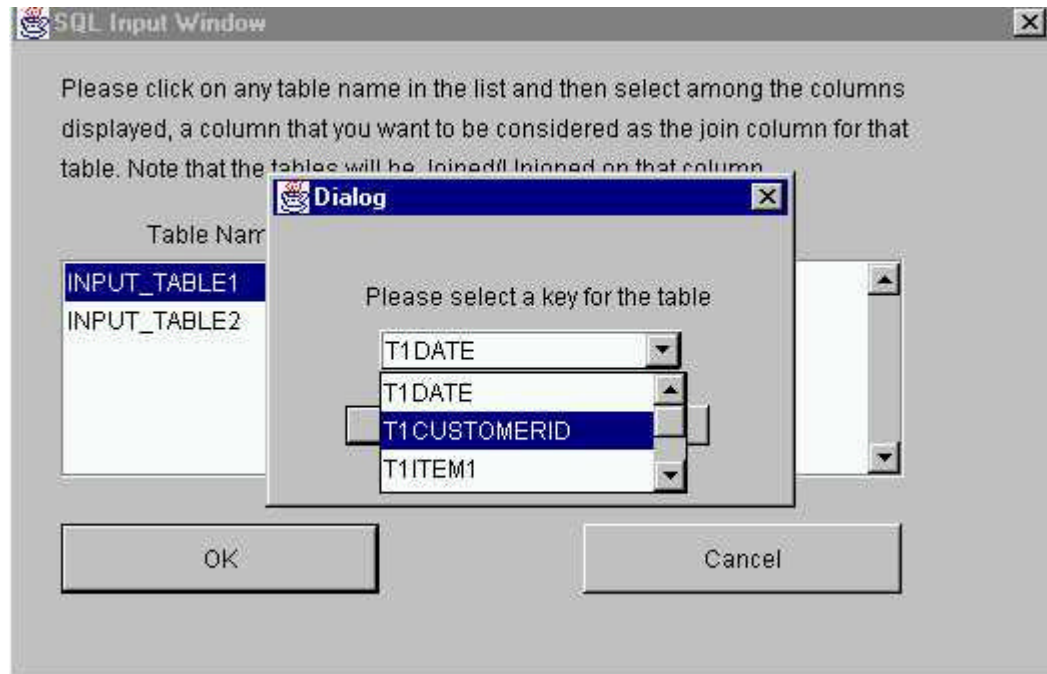


Figure 6.6 Join column selection

Once the user selects one join column for each of the selected tables, the user is then prompted by the window in Figure 6.7. Note that the system will not proceed until a join column for each of the tables is specified. The user then needs to select the columns that need to be treated as items. Figure 6.8 describes the user's selection of the items' columns.

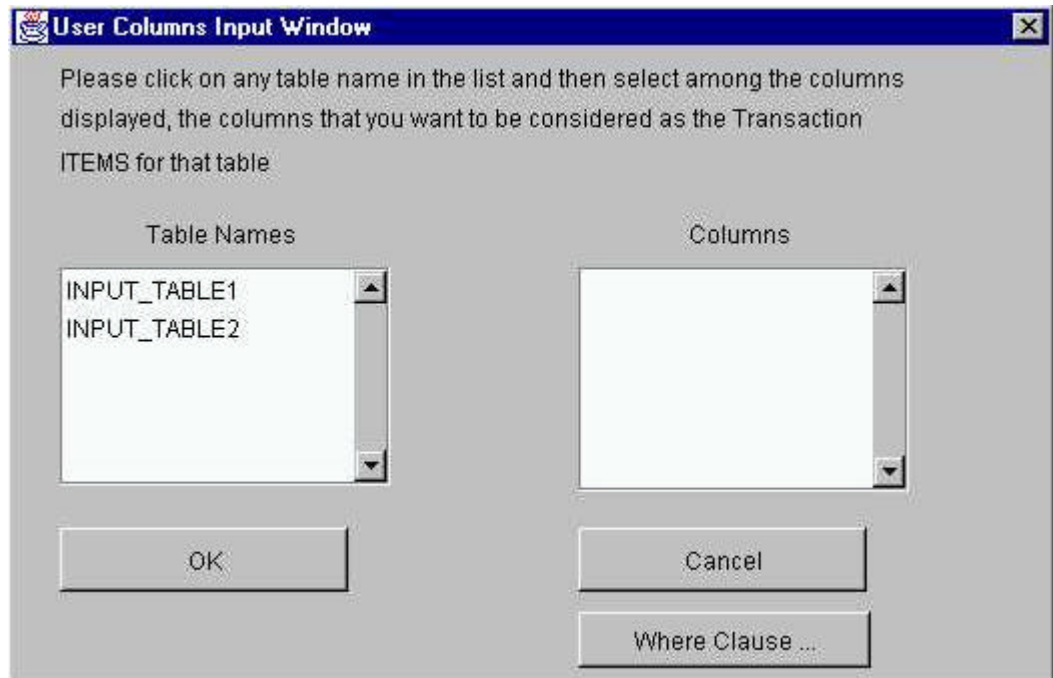


Figure 6.7 User selection columns

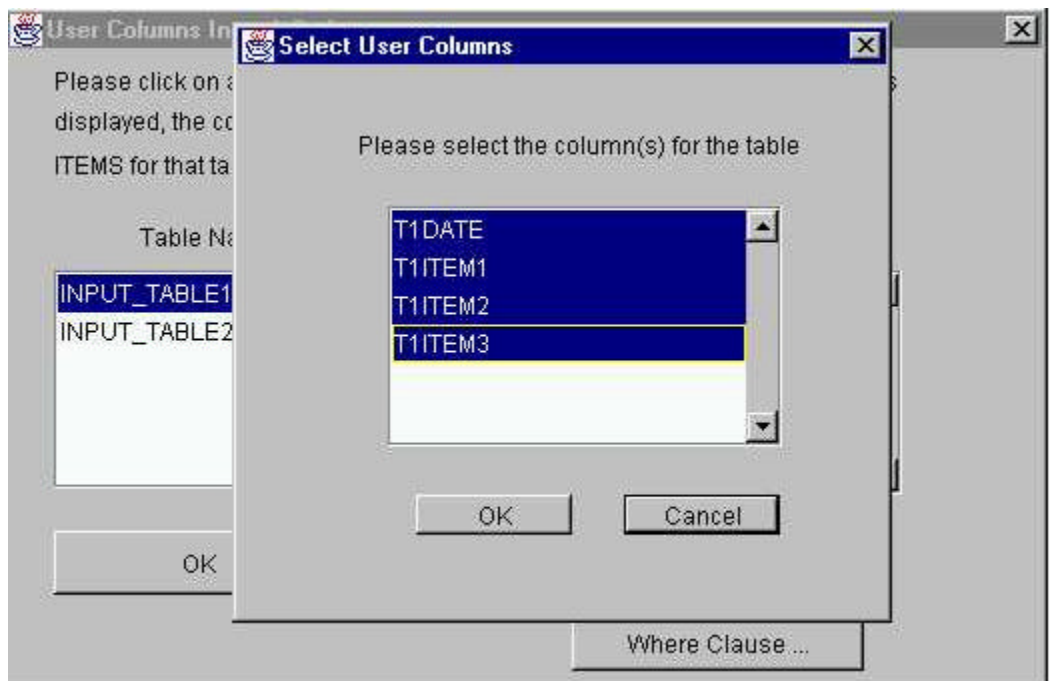


Figure 6.8 Items' columns selection

As can be seen in Figure 6.7 and also Figure 6.8, the user can specify a where clause. This will act as a constraint on input data. Instead of unnecessarily generating rules for a larger data set and then imposing a constraint on them, using the where clause, the user can constrain the input tuples so that only desired records are subject to the mining process. Figure 6.9 describes the where clause creation and the associated parameters for the current example. Effort is put in so that most of the valid conditions clauses in SQL queries can be incorporated in the where clause.

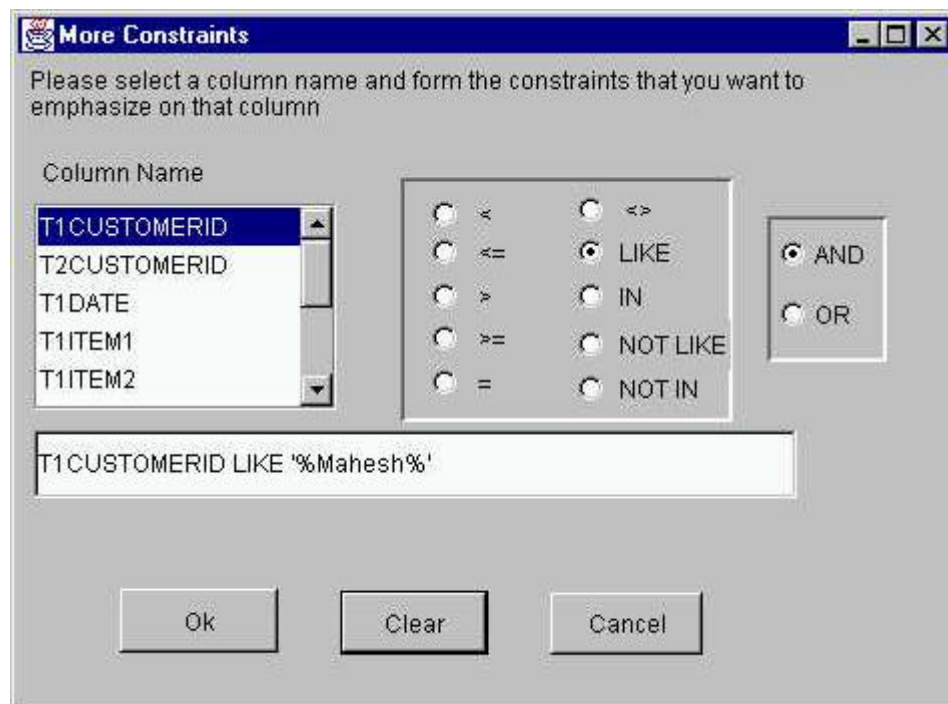


Figure 6.9 The user specified WHERE clause

The user is then given a choice for selecting the Tid columns. This is done by prompting the user with the set of columns that are potentially the Tid columns.

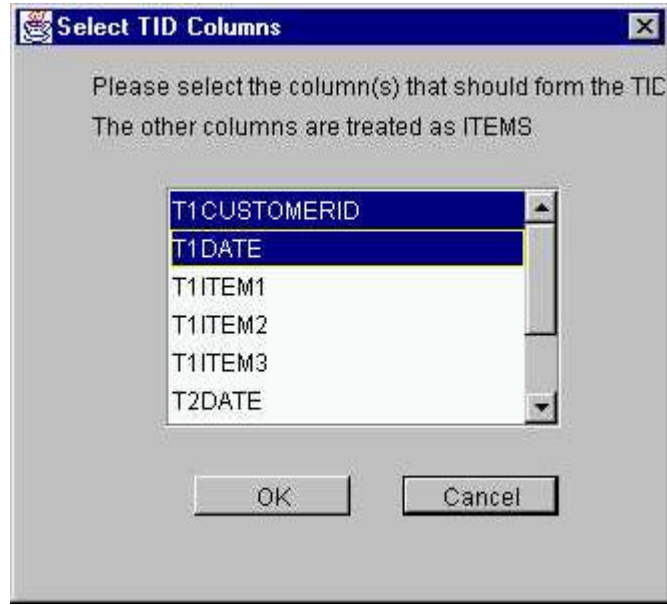


Figure 6.10 TID columns' selection

Figure 6.10 shows the window that prompts the user to select the Tid columns. Note that in this example the user has selected T1CustomerID and T1Date to indicate that they together must be treated as the Tid field. Once the user clicks OK, the corresponding Tids and Items are mapped to integers and are stored in the MappedTidsTable and MappedItemsTable respectively. The end of mapping is indicated by the text in the text area of the main frame as shown in Figure 6.11. Earlier, since the user had checked the Developer check box, the user is able to choose an approach for support counting. This can be seen in the sub-menu items of *approaches* in Figure 6.11. Had the user not checked the Admin check box, the user could just click the *mine data* menu item and the *approaches* menu item would be disabled for this session.

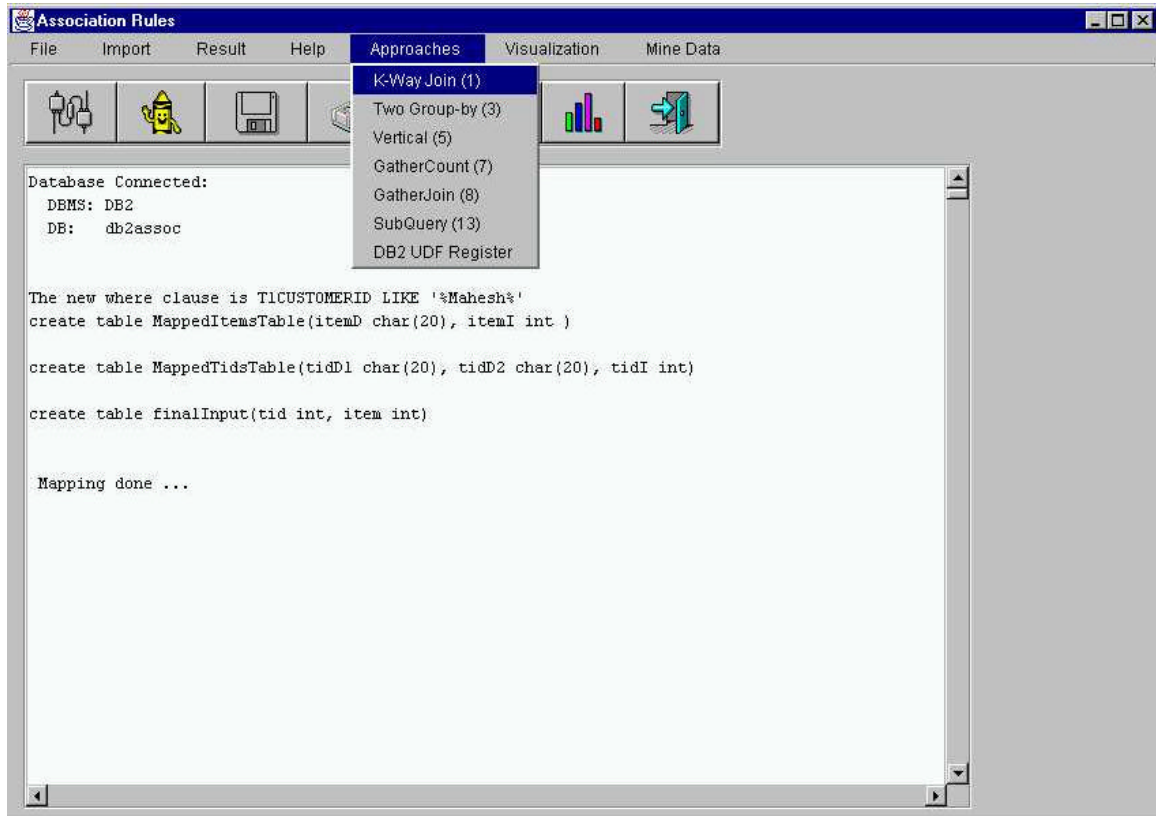


Figure 6.11 End of mapping and options for support counting

If the miner is run for the first time, then the stored procedures in Oracle and UDFs in DB2 need to be registered with the underlying database registry. If the database connected is DB2, then the user needs to register the UDFs using the sub-menu item *DB2 UDF register* in the *approaches* menu. The user then selects one of the approaches for support counting, for example here, the user has selected K-way join. The results of execution of the approach are shown in Figure 6.12. Had the user set the debug flag to true, more informative comments like the SQL statements of the tables dropped, created, their insertions, etc would be output in the text area.

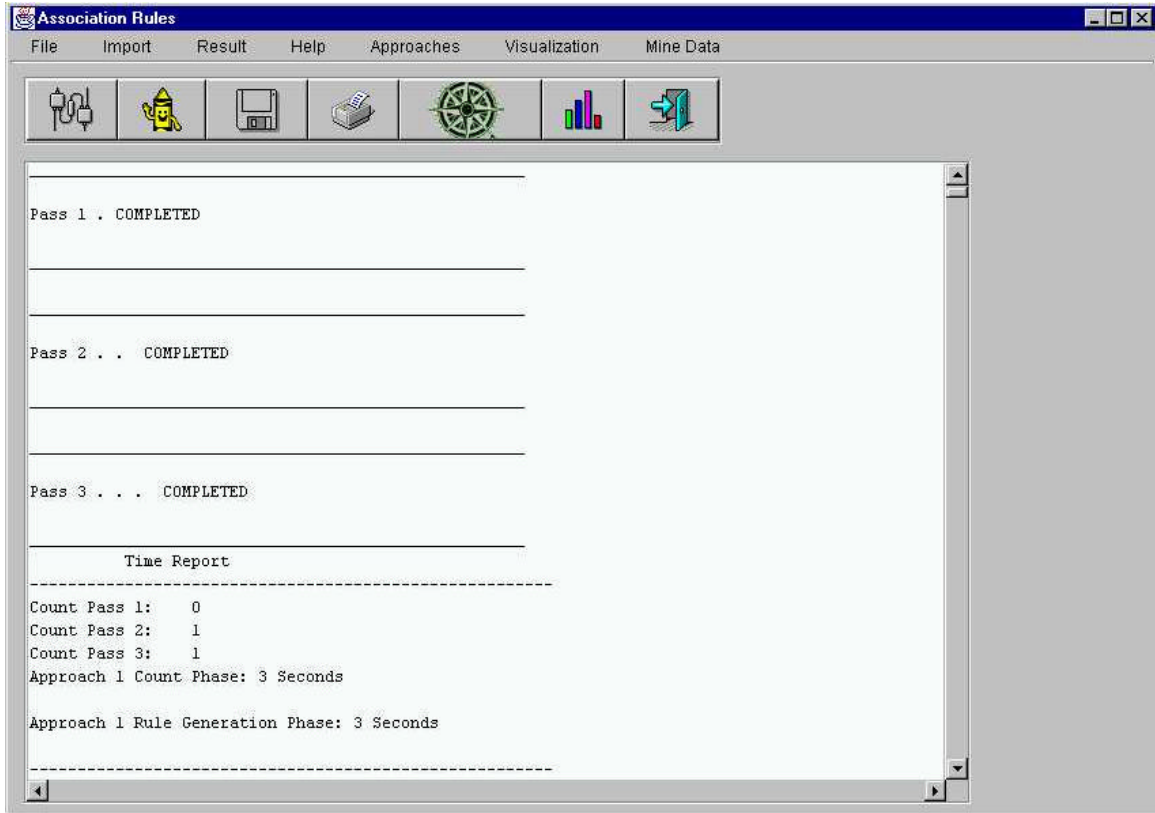


Figure 6.12 End of support counting

6.3 Conclusions

We pictured the system implementation in this chapter. This chapter was intended at providing an insight of the implementation and the associated GUI. The next chapter discusses the thesis conclusions, our contributions and further extensions.

CHAPTER 7 CONCLUSIONS

We discussed the various architectures of mining and JDBC. We explored the various approaches of support counting for association rule mining and also performed the performance tests. This was a step to discover the competitive performance of the approaches. We formulated the SQL queries for the support counting in two categories viz. SQL-92 and SQL-OR. We ran each of the approaches in the two categories on two databases namely DB2 and Oracle. In the SQL-92 category, we experimented with all the three approaches with different datasets and compared their performances. K-way was the winner on both the types of databases. It was also noticed that Oracle was slower than DB2. This is partly because of the intermediate tables that needed to be materialized in Oracle. The reason being the structure of PL/SQL against DB2, which allowed us not to materialize certain intermediate tables, which decreased the completion time of the DB2 approaches. Similar results were noticed with the SQL-OR approaches too. We then, utilized the object-relational extensions like UDFs, CLOBs, table functions, etc., as a part of implementing the SQL-OR category support counting approaches. Vertical and Gather Join performed comparably on both the types of databases. We also implemented the mapping of input data into the format that the mining optimizer needs namely the format of (Tid, Item) with both integers. This was also accompanied by the user selection of the Tid and items' columns. Some work has been done in the section of collecting useful information that forms the metadata, which is used in selection of an approach for support counting.

7.1 Contributions

The specific contributions of this thesis are as follows:

- We formulate the various SQL-based (SQL-92 and SQL-OR) approaches for association rule mining.
- We implement the mapping of input data into a format that is easily manipulable and required by the mining optimizer.
- We present an optimized approach selection wherein the user need not select an approach and the optimizer selects one from among the best approaches and mines. This is done by storing some useful information, as metadata in the underlying database and these metadata are accessed and used for approach selection.
- We provide a means of connecting to multiple databases and thereby allowing for the input data to be present in any database. When mining for the association rules, one can connect to the required database and mine for the rules.
- We allow the user to specify the Tid column(s). This selection can be either a single field or multiple fields. In case of multiple fields, all the fields together are treated as the Tid columns.
- We also allow the user to select the items' columns. Again, the number of columns can range from one to many, which is at the disposal of the user.
- We did some performance testing and scale up experiments over synthetically generated datasets.

7.2 Proposed Extensions and Future Work

Some of the work in this thesis is complete and some other parts can be continued as future research. We have identified certain directions that can be treated as proposed extensions. Some of them are as follows:

- The current work on association rule mining can be further continued for generalized association rules [Sri95, Sri96]. This will take care of the taxonomies that may exist among the items. The goal would be to find an association rule between items that may exist at any level of the taxonomy.
- The current work can also be extended for the sequential pattern mining to find the frequently occurring patterns.
- Incremental mining can be implemented over the current implementation. This will update the generated frequent itemsets from the earlier state of the database. Re-computing the frequent itemsets is infeasible. Hence they should be updated. Also a negative border [Tho98] can be maintained to decide when to scan the whole database.
- Some of the operations presently done in the mining optimizer as SQL queries or UDFs are thought of as possibilities to be developed as operators in the databases that would make data mining for association rules easier. Some of them are
 - The process where in we formulate the pruning as an SQL query can be visualized as an operator in the underlying database.
 - The process of generating the k-item combinations which is presently implemented as UDF and stored procedures viz. *CombK*.
 - The process of creation of the CLOBs of items and Tids viz. *saveItem* and *saveTid* respectively.

- The process of getting the common elements from k CLOBs. Presently implemented as UDF *CountAndK*.

APPENDIX: RULE GENERATION EXAMPLE USING VERTICAL APPROACH

This appendix lists one example input data set, the intermediate frequent itemsets F_x , and the final association rules using vertical approach with minimum support of 50%, and minimum confidence of 50%.

Let us assume that the input data is in two tables namely inputTableOne and inputTableTwo as shown in Table A-1 and Table A-2 respectively.

Table A-1 InputTableOne

Date	CustID	ITEM
1/1/00	100	Milk
1/1/00	100	Eggs
1/1/00	100	Bread
1/2/00	200	Sugar
1/2/00	200	Eggs
1/2/00	200	Cake

Table A-2 InputTableTwo

Date	CustID	ITEM
1/3/00	300	Milk
1/3/00	300	Sugar
1/3/00	300	Eggs
1/3/00	300	Cake
1/4/00	400	Sugar
1/4/00	400	Cake

The mining optimizer combines these two tables with a union operation on them. Also, the Date and CustID fields in the two input tables are selected to form the composite TID column. Tuples with unique values in both the fields are treated as separate transactions. The TIDs and ITEMS in the input table are mapped to tables

MappedTidsTable and MappedItemsTable in the Mapping process as shown in Table A-3 and Table A-4 respectively.

Table A-3 MappedTidsTable

Date (TIDD1)	CustID (TIDD2)	TIDI
1/1/00	100	1
1/2/00	200	2
1/3/00	300	3
1/4/00	400	4

Table A-4 MappedItemsTable

ITEMD	ITEMI
Bread	1
Cake	2
Eggs	3
Milk	4
Sugar	5

After the Mapping process, the input data set is transformed into the following data format into a table called FinalInput as shown in Table A-5.

Table A-5 FinalInput table

TID	ITEM
1	1
1	3
1	4
2	2
2	3
2	5
3	2
3	3
3	4
3	5
4	2
4	5

A.1 Support Counting

As a start of the support counting phase, the tid lists are created for each of the input items. Table A-6 depicts the lists.

Table A-6 TID lists of each of the items

ITEM	COUNT	TIDs
1	1	100
2	3	200,300,400
3	3	100,200,300
4	2	100,300
5	3	200,300,400

A.1.1 First Pass

The frequent itemset F_1 has the following tuples:

Table A-7 Table F_1

ITEM1	COUNT
2	3
3	3
4	2
5	3

A.1.2 Second Pass

The frequent itemset F_2 has the following tuples:

Table A-8 Table F_2

ITEM1	ITEM2	COUNT
2	3	2
3	4	2
2	5	3
3	5	2

A.1.3 Third Pass

The frequent itemset F_3 has the following tuples:

Table A-9 Table F₃

ITEM1	ITEM2	ITEM3
2	3	5

A.2 Rule Generation

Consolidating the tables generated from the three passes, the final frequent itemsets table FISETS is shown in Table A-10.

Table A-10 Table FISETS

ITEM ₁	ITEM ₂	ITEM ₃	ITEM ₄	ITEM ₅	ITEM ₆	ITEM ₇	ITEM ₈	NULLM	COUNT
2	0	0	0	0	0	0	0	2	3
3	0	0	0	0	0	0	0	2	3
4	0	0	0	0	0	0	0	2	2
5	0	0	0	0	0	0	0	2	3
2	3	0	0	0	0	0	0	3	2
3	4	0	0	0	0	0	0	3	2
2	5	0	0	0	0	0	0	3	3
3	5	0	0	0	0	0	0	3	2
2	3	5	0	0	0	0	0	4	2

The subsets of table FISETS is stored in table SUBSETS shown in Table A-11.

Table A-11 Table SUBSETS

TITEM ₁	TITEM ₂	TITEM ₃	TITEM ₄	TITEM ₅	TITEM ₆	TITEM ₇	TITEM ₈	TNULLM	TRULEM	TCOUNT
2	3	0	0	0	0	0	0	3	2	2
3	2	0	0	0	0	0	0	3	2	2
3	4	0	0	0	0	0	0	3	2	2
4	3	0	0	0	0	0	0	3	2	2
2	5	0	0	0	0	0	0	3	2	3
5	2	0	0	0	0	0	0	3	2	3
3	5	0	0	0	0	0	0	3	2	2
5	3	0	0	0	0	0	0	3	2	2
2	3	5	0	0	0	0	0	4	2	2
3	2	5	0	0	0	0	0	4	2	2
5	2	3	0	0	0	0	0	4	2	2
2	3	5	0	0	0	0	0	4	3	2
2	5	3	0	0	0	0	0	4	3	2
3	5	2	0	0	0	0	0	4	3	2

Using tables FISETS and SUBSETS, the association rules are generated and stored in the table RULES shown in Table A-12.

Table A-12 Table RULES

ITEM ₁	ITEM ₂	ITEM ₃	ITEM ₄	ITEM ₅	ITEM ₆	ITEM ₇	ITEM ₈	NULLM	RULEM	CONF	SUP
2	3	0	0	0	0	0	0	3	2	66.67	50
3	2	0	0	0	0	0	0	3	2	66.67	50
3	4	0	0	0	0	0	0	3	2	66.67	50
4	3	0	0	0	0	0	0	3	2	100	50
2	5	0	0	0	0	0	0	3	2	100	75
5	2	0	0	0	0	0	0	3	2	100	75
3	5	0	0	0	0	0	0	3	2	66.67	50
5	3	0	0	0	0	0	0	3	2	66.67	50
2	3	5	0	0	0	0	0	4	2	66.67	50
3	2	5	0	0	0	0	0	4	2	66.67	50
5	2	3	0	0	0	0	0	4	2	66.67	50
2	3	5	0	0	0	0	0	4	3	100	50
2	5	3	0	0	0	0	0	4	3	66.67	50
3	5	2	0	0	0	0	0	4	3	100	50

Combining the tables MappedItemsTable generated in the mapping process and RULES, the final association rules can be presented to the user in the following format shown in Table A-13.

Table A-13 Table RULES with items mapped back to descriptions

Rule Head	Symbol	Rule Body	Confidence(%)	Support(%)
Cake	=>	Eggs	67	50
Eggs	=>	Cake	67	50
Eggs	=>	Milk	67	50
Milk	=>	Eggs	100	50
Cake	=>	Sugar	100	75
Sugar	=>	Cake	100	75
Eggs	=>	Sugar	67	50
Sugar	=>	Eggs	67	50
Cake	=>	Eggs,	67	50
Eggs	=>	Cake,	67	50
Sugar	=>	Cake,	67	50
Cake, Eggs	=>	Sugar	100	50
Cake, Sugar	=>	Eggs	67	50
Eggs, Sugar	=>	Cake	100	50

LIST OF REFERENCES

- [Agr93] Agrawal, R. Imielinski, T. and Swami, A. Mining association rules between sets of items in large databases. Proc. of the ACM SIGMOD Conference on Management of Data, Washington, DC, May 1993.
- [Agr94] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. Proc for the 20th Int'l Conference on Very Large Databases, Santiago, Chile, September 1994.
- [Agr95] Agrawal, R. and Srikant, R. Mining sequential patterns. Proc. of Int'l Conference on Data Engineering, Taipei, Taiwan, 1995.
- [Cha98] Chamberlin, D. A complete guide to DB2 Universal Database, Morgan Kaufmann Publishers, Inc, San Mateo, California, 1998.
- [Hon00] Hongen, Z. Mining and visualization of association rules over relational DBMS, UF MS thesis, August 2000.
- [IBM00] Web documentation on the mining tool Intelligent Miner from IBM. <http://www.software.ibm.com/data/iminer>, January, 2000.
- [Ora97] Oracle 8. Application developer's guide, Chapter 10: Using procedures and packages, Oracle, Belmont, California, 1997.
- [Pod98] Podcameni, S. Leung, M. Fischer, M. and Letham, G. Getting started with DB2 stored procedures, ITSO Redbooks, San Jose, California, March 1998.
- [Que98] QUEST: Collection of technical and journal papers about data mining. <http://www.almaden.ibm.com/cs/quest/publications.html>, December, 1998.
- [Sar98] Sarawagi, S. Thomas, S. and Agrawal, R. Integrating associating rule mining with relational database systems: Alternatives and implications. Proc. of the ACM SIGMOD Int'l Conference on Management of Data, Seattle, Washington, June 1998.

- [Sri95] Srikant, R. and Agrawal, R. Mining generalized association rules, Proc. of the 21st Int'l Conference on Very Large Databases, Zurich, Switzerland, September 1995.
- [Sri96] Srikant, R. and Agrawal, R. Mining quantitative association rules in large relational tables. Proc. of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, June 1996.
- [Sun00] Sun documentation about JDBC.
<http://java.sun.com/products/jdk/1.2/docs/guide/jdbc>, July, 2000.
- [Tho98] Thomas, S. Architectures and optimizations for integrating data mining algorithms with database systems, UF Ph.D. dissertation, August 1998.
- [Tho99] Thomas, S. and Chakravarthy, S. Performance evaluation and optimization of join queries for association rule mining. Proc. of the First International Conference on Data Warehousing and Knowledge Discovery, DaWaK '99, Florence, Italy, August 1999.

BIOGRAPHICAL SKETCH

Mahesh Dudgikar was born on November 30, 1975 in Belgaum, India. He received his Bachelor of Engineering degree in computer science and engineering from Sri Jayachamarajendra College of Engineering, Mysore, India, in September 1997. After his graduation, he worked for a short stint at Siemens Public Communication Network Limited, Bangalore, India. He joined the Department of Computer and Information Science and Engineering at the University of Florida in Fall, 1998. He worked as a research assistant in the Database Systems Research and Development Center of the department. He will receive his Master of Science degree in August 2000. His research interests include database systems, data mining, active databases and main memory databases.