Data Mining Query Language Design and Implementation

Xiaolei YUAN

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Systems Engineering and Engineering Management

December 2003

Abstract

The need for an integrated query language that supports ad hoc data mining queries has been long recognized. Several query languages with enhanced expressive power to incorporate data mining queries have been proposed. These query languages differ considerably in the ways they deal with the internal expressions of a data mining query and the syntax they define. Because of these differences, it has been difficult to compare the proposed query languages and to combine them as a unified one. However, state-of-the-art query languages are not powerful and general enough for many advanced data mining applications, and especially for multi-step data mining tasks.

In this thesis, a new query language focused on multi-step association rule mining *M2MQL*(Mining-to-Mining-Query-Language) is presented. Nested relation is adopted in this work. A relational algebra for data mining is proposed on top of the nested relational algebra. On this algebraic basis, different association rule mining queries are interpreted into a sequence of internal algebraic expressions.

Due to its flexibility, SQL-compatibility and ease of use, our work is particulary beneficial if the database system in use handles both database queries and data mining queries in the same manner.

摘 要

隨著數據挖掘技術的日益成熟及廣泛使用,數據挖掘查詢語言的語言規範已經成爲一個亟待解決的瓶頸問題. 該規範的匱缺導致了數據挖掘語言設計中的不確定性及現有數據挖掘語言間的不可比性.

本論文論述了用以表達多步關聯規則挖掘查詢的查詢語言-M2MQL. 這種查詢語言以嵌套型關係數據庫爲操做對象, 突破了傳統的 1NF 關係數據庫在數據挖掘應用過程中的模式限制; 並通過定義一組支持關聯規則挖掘的特殊算子擴展了傳統的關係代數運算, 該組特定算子與嵌套型關係代數的整合使用實現了關聯規則挖掘查詢的過程化求解.

本文通過對不同種類關聯規則挖掘例證的分析, 闡述了 M2MQL 查詢語言的靈活性, 兼容性及可行性. 隨著現有數據庫技術對數據挖掘功能的逐步支持, 這種查詢語言的使用靈活性, 與 SQL 語言的兼容性及內部操作的可行性將在實踐中得到充分的證實.

Acknowledgement

My deepest thanks go to my supervisor Professor Jeffrey YU who has taught me not only the knowledge but also the real connotation of research in these two and a half years. Without his continuous advice, guidance, inimitable patience and support, this thesis could hardly be completed. His profound knowledge and kindness will always be an inspiration.

I would like to give my sincere thanks to Professor Kam Fai WONG, for his conscientious review of my thesis, for his cogent comments and direction. Without his kind help, the thesis could not gain its current ends.

Also I would like to thank Professor Christopher YANG for his kind support. Also my faithful thanks goes to Professor Hongjun LU of HKUST for his valuable comments on this thesis. I also wish to thank Professor Youyi FENG, for his encouragements and concerns.

I am grateful to friends of our database group, Fiona CHOI, Yabo XU, Gang GOU, Gabriel FUNG, Zheng LIU and Zhiheng LI, for taking the time to discuss with me, and for their warm support. Especially I wish to thank Fiona, for her unaffected encouragement and help. Also I will never forget Gabriel's kind help for checking my thesis time and again.

The most appreciation from me should be given to my dearest parents, with their endless love and trust, I get over times of difficulty.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Data Mining: A New Wave of Database Applications

The progress of data-collection technology, such as bar-code scanners in commercial domains and a variety of sensors in scientific and industrial domains, generates huge amounts of data. This explosive growth in data and databases generates the need for new techniques and tools that can intelligently and automatically transform data into useful information and knowledge. Data mining is such a kind of techniques.

DATA MINING, also referred to knowledge discovery in database (KDD), is a process of nontrivial extraction of implicit, previously unknown and potentially useful information (such as knowledge rules, constraints, regularities, etc.) from one or more databases.

Now, let us consider a transactional database [AIS93b] (market baskets) that is

obtained from a supermarket. A transaction is a set of goods or items.
In the past, the most experienced decision-makers of the supermarket may have summarized patterns such as "when a customer buys milk, he/she also buys bread " and "customers like to buy Nestle products". These patterns or rules are used to predict customer consuming behaviors or to estimate the sales of a new product. These decision-makers could draw upon years of general knowledge and knowledge about specific associations to form effective selections on the data.
With the increasing use of databases, the need for digesting the large volumes of data now being generated is critical. Recent estimations show that only 5-10% of commercial databases have been analyzed. Database technology is successful in managing data but fails in discovering knowledge as a key strategic weapon for enhancing business competitiveness. The large volume and high dimensionality of databases lead to a breakdown in traditional human analysis.
Data mining incorporates technologies for analyzing data in very large databases and identifies potential useful patterns in the data. Due to the wide availability of huge amounts of data in electronic forms and the ubiquitous World Wide Web, data mining has become very important in information industry. The imminent need for extracting useful information and knowledge out of such bulk of data becomes much more desirable. In general, the process of knowledge discovery in databases consists of an iterative sequence of steps as follows shown in Figure 1.1:

1. *Defining the problem.* The goals of the knowledge discovery project must be identified. The goals must be verified as actionable. For example, if the goals are met, a business can then put newly discovered knowledge to use. The data to be used must also be identified.

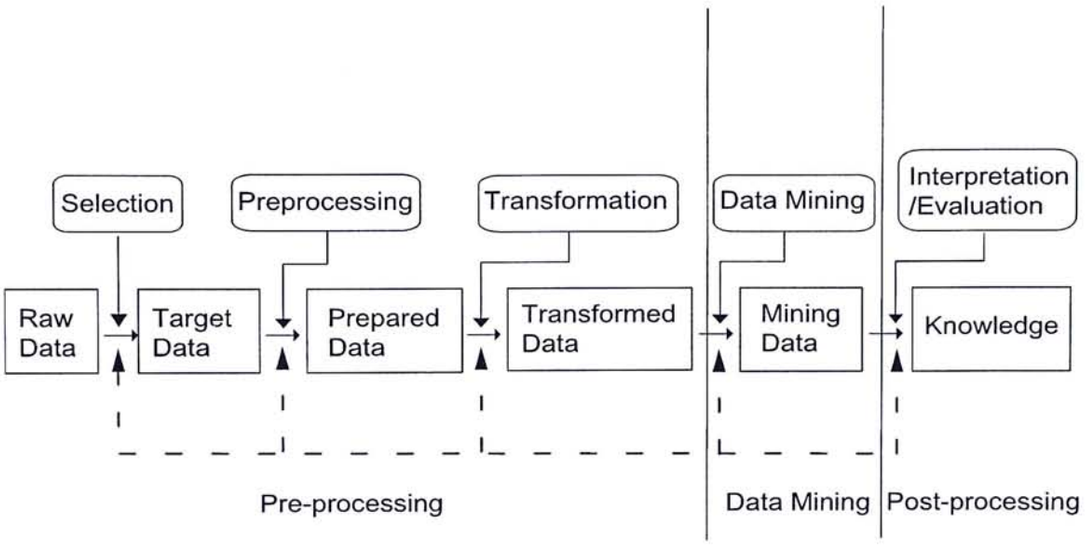

Figure 1.1: Data Mining Knowledge Procedure

2. *Data preprocessing.* This includes data collecting, data cleaning, data selection and data transformation.

   (a) Data collecting. Obtaining necessary data from various internal and external sources; resolving representation and encoding differences; joining data from various tables to create a homogeneous source.

   (b) Data cleaning. Checking and resolving data conflicts, outliers, noisy or erroneous, missing data, and ambiguity; using conversions and combinations to generate new data fields such as ratios or rolled-up summaries. These steps require considerable effort often as much as 60% or more of the total data mining cost [Han96].

   (c) Data selection. Data is transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations.

   (d) Data mining. An essential process, where intelligent methods are applied in order to extract data patterns. Patterns of interests in a

particular form of representational, or a set of such representations are searched for, including classification rules or trees, regression, clustering, sequence modelling, dependency, and so forth. The user can significantly aid the data mining method by correctly performing the preceding steps.

(e) Post data mining. This includs pattern evaluation, deploying the model, maintenance, and knowledge presentation.

1.1.2 Association Rule Mining

The task of association rule mining is to find the association relationships among a set of objects (called items) in a database. These relationships are described in terms of rules. Each rule has two measurements, support and confidence. Confidence is a measure of the rule's strength, while support corresponds to statistical significance. The task of discovering association rules was first introduced in 1993 [AIS93b]. Originally, association rule mining focused on market "basket data" which stores items purchased on a per-transaction basis.

Finding association rules is valuable for cross-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. Apart from business applications, association rule mining can also be applied to other areas, such as medical diagnosis, remotely sensed imagery, credit card transactions, telecommunication service purchases, banking services, insurance claims, world wide web page visiting record analysis.

A typical example of association rule mining is market basket analysis. This process analyzes customer buying habits by finding associations between different

items that customers place in their shopping baskets.

Another illustrative example is the US Congress voting records. Voting record can be simplified into a $m \times n$ matrix in which m denotes votes and n denotes congress members. The matrix elements have already simplified into two cases: one for cases of votes which contains "voted for", "paired for", "announced for" and zero for all other cases. Data mining aims at discovering patterns in this matrix. In particular, we will find columns or items which display similar voting patterns and we aim at discovering rules relating to the items which hold for a large proportion of members of congress.

Problem Definitions The formal definition of association rules given in [AIS93c] is as follows. Let I $= \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that T $\subseteq$ I. Associated with each transaction is a unique identifier, called its *TID*. An association rule is an implication of the form X $\rightarrow$ Y, where X $\subseteq$ I, Y $\subseteq$ I, and X $\bigcap$ Y $= \phi$. X is the antecedent (the *body* of the rule) while Y is the consequence (the *head* of the rule) of the rule. Association rules used by us also expressed in form of : $body \rightarrow head \quad [s, c]$.

Two measurements are associated with each rule, support and confidence. The rule X $\rightarrow$ Y has support s in the transaction set D if s% of transactions in D contains X $\bigcup$ Y. The rule has confidence c if c% of transactions in D that contain X also contain Y. Support indicates how frequently the pattern occurs, and confidence indicates the strength of the rule. The equations for these two thresholds are given as: $support(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{\text{num. of groups supporting X}\rightarrow\text{Y}}{\text{num. of all record groups}}$, $confidence(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{\text{num. of groups supporting X}\rightarrow\text{Y}}{\text{num. of all record groups supporting X}}$

Category of Association Rule Association rules can be classified in various ways, based on different criteria:

(1) Types of values handled in the rule.

If a rule concerns associations between the presence or absence of items, it is a `Boolean association rule`. For example, in basket data analysis, a boolean association rule is :

computer $\Rightarrow$ *financial_software [support = 20%, confidence = 60%]*

If a rule describes associations between quantitative items or attributes, then it is a `quantitative association rule`. In these rules, quantitative values for items or attributes are partitioned into intervals. Follows is an example of this kind:

age(X, "30..39") $\wedge$ *income(X, "42K..48K")* $\Rightarrow$ *buys(X, TV)*

(2) Dimensions of data involved in the rule.

Based on the definition of data warehouse, dimension corresponds to an attribute or a set of attributes in the schema; following the terminology used in multidimensional databases, each dimension refers to a distinct predicate. If the items or attributes in an association rule reference only one dimension, then it is a `single dimensional association rule`; if they reference two or more dimensions, it is a `multi_dimensional association rule`. Following is an example for single dimensional association rule which only references the dimension/predicate *buy*:

buy(X, "computer" $\Rightarrow$ *buy(X, "statistic_software"))*

Main Steps in Association Rule Mining

Given a user specified minimum support and minimum confidence, the problem of mining association rules is to find all the association rules whose support and

confidence are larger than the respective thresholds. Thus, it can be decomposed into two sub-problems:

1. Finding the frequent itemsets which have support above the predetermined minimum support.

2. Deriving all rules, based on each frequent itemset, which have more than predetermined minimum confidence.

Since the second step is not computationally intensive, the first step is the main concern in data mining research.

1.2 Motivation

Progress in database technology has provided the foundation that made massive database applications ubiquitous. One growing area in data-intensive applications is decision support, which includes data warehousing and data mining. Data mining is the process of discovering hidden patterns and relationships in data by using advanced statistical analysis and modelling techniques. Increasingly, data warehouses are built using relational database technology. But the inadequate power and extensibility of SQL has limited the generality and ability of DBMSs to support these applications. Substantial extensions have been added to SQL over the years, including support for recursive queries, active rules, ROLAP, etc.; yet data mining remain an unsolved challenge in DBMSs, especially for mining of association rules. The lack of standards in the field results in the following open problems: How to establish a "*uniformed*" data mining model? How to integrate data mining applications with database "*seamlessly*"? And how to define an "*expressive*" query language?

1.3 Main Contribution

The main contributions of this thesis are as following:

1 We focus on nested relational algebra for integrating multi-step association rule mining. The introduction of nested relational algebraic operations releases Codd's 1NF restriction. This allows both database and data mining queries be processed in a similar manner.

2 We propose a sequence of algebraic operations for association rule mining. These sequences simplify the internal expressions of the mining tasks compared with existing works.

3 We study a new data mining query language *M2MQL (Mining to Mining Query Language)* based on the formal semantics sustained by the nested relational algebraic operations. We show that *M2MQL (Mining to Mining Query Language)* can play a major role in supporting association rule mining applications by suitable examples of typical association rule mining queries.

4 In *M2MQL*, we introduce *Association Rule Mining Primitives* which gives flexibility for users to reuse the mining results and conduct mining tasks based on them. These primitives are expressed by a series of complex queries in *M2MQL*.

5 We show that *M2MQL* is built on SQL by adding minimal extensions. It inherits many traditional benefits of SQL is easy-to-use.

1.4 Thesis Organization

The thesis is organized as follows. Literature review is given in Chapter 2, the reviews mainly focuses on two approaches related to query language designed for Association Rule Mining. Chapter 3 proposes a new query language for database mining specifically for association rule mining. The proposed data mining query language *M2MQL* (Mining to Mining Query Language) is based on the formal semantics of nested relational algebraic operations. With various application examples, we show that queries based on *M2MQL* is effective. Usage of the nested relational algebra and set of specific algebra for association rule mining are described in Chapter 4. Chapter 5 illustrates the way to interpret an association rule query as a sequence of internal algebraic expressions. Conclusions are drawn and future works are proposed in Chapter 6.

Chapter 2

Literature Review

In this chapter, literature contributing to this thesis is summarized in five categories. The first section reviews literature for data mining, and association rule mining. The second section is about main branches of integration of Data Mining and DBMSs. The third section discusses works on query language design especially for association rule mining. The fourth section lists existing unified data mining models and frameworks. Other topics including nested relations, query execution, optimization and system implementation are summarized in the fifth section.

2.1 Data mining and association rule mining

Researchers and developers in many fields have contributed to state-of-the-art in data mining [AIS93a, CHY96]. Agrawal, Imielinski and Swami referred data mining as a branch of machine learning and the emphasis on the databases [AIS93a]. They present three main classes of database mining problems, which involves classification, association rule mining, and sequence mining. The lack of functionality

offered by database systems to support "mining" applications is described in this work. Chen, Han and Yu give a good survey, from a database researcher's point of view, on data mining techniques [CHY96]. A taxonomy of existing data mining techniques is provided in this paper, and a comparative study of such techniques is also included. The emphasis of this work is on applicative data mining systems. It provides an easy overview of data minining. A desire for a flexible and convenient data mining language is recognized and brought forward by the authors.
Since it was introduced in 1993 [AIS93c], association rule mining has been an important branch in the data mining research. A classical definition of the association rule is given in [AIS93c], as well as the way to decompose the rule mining problem into two sub-problems - a frequent itemset searching phase and a rule generating phase - which takes dominant positions in the whole rule mining procedure. [HGN00] presents a survey on the existing wide acknowledged algorithms for association rule mining. Systematized by its strategy to traverse the search space or to determine the support values of the itemsets, comparison between these algorithms is given based on a series of experiments.
[ZZ02] is a systemic professional book about association rule mining. It presents the recent achievements in mining quantitative association rules, causal rules, exceptional rules, negative association rules, association rules in multi-databases, and association rules in small databases.

2.2 Integration data mining with DBMS

The target of early research on data mining based on file systems and specially designed data structures. It focused on data mining algorithms and the new mining operators. Exploiting the maturity of DBMSs, researchers proposed to

integrate data mining and databases. In [STA98], architectures for coupling association rule mining with database systems are studied and their pros and cons analyzed. This paper points to several directions for future research including the topic of this thesis. For example: How much we can leverage on existing relational engines? What data model and language extensions are needed? Etc. In another article related to this trend, Rajamani, Cox et al present a new relational operator for efficient SQL implementation of Apriori in database [RCIC99]. This new operator enhances operators' expressive power in data mining.

2.3 Query language design for association rule mining

[IM96] introduced the concept of *kdd query language*. They claimed that specification, compilation and execution of query languages for database mining was a challenge, whick required long-term effort extending to the next decade and beyond. This paper was the first work of its kind and aroused researchers' concerns about a new branch of research, ie. a general query language for data mining. Current research on data mining query langauge design can be mainly organised into two classes, one specifically extends SQL to support mining operators while another particularly emphasises developing extended functionality for commercial DBMSs. We present an overview of works related to the first class. Note also that most of the works focused on query languages for association rule mining, which is the theme of the thesis.

1. DMQL (Data Mining Query Language) defines operators for mining characteristic rules, discriminant rules, classification rules, and association rules

[HFW+96]. DMQL is a query language adopted in a commercial data mining system - DBMiner developed by the same research group.

2. MSQL defines *Mine*, a unified operator, to generate and query propositional rules. The mine rule operator is a generalized version of the association rule discovery problem [IV99]. Its corresponding application programming interface is introduced in [IVA99].

3. MINE RULE was proposed in [MPC96, MC98]. This operator extracts a set of association rules from the database and stores them back in the database in a separate relation. This language is an extension of SQL. The MINE RULE operator is supported by well defined semantics. This facilitates its effective implementation. But, the expression for association rule mining is complicated and difficult to execute. In Appendix C, we use a running example to illustrate its complexity. Technical reports for details about the MINE RULE can be found in [ME03, MS03, BMS03].

4. By defining a set of extended relational algebras, Gopalan presents how to translate association rule mining into a set of algebraic expressions [GNS02]. This approach utilizes nested relations as its relation schema and focuses on formulating algebraic expressions. On top of their algebraic basis, they continue to use the syntax proposed in MINE RULE operators [MPC96, MC98]. However, they do not discuss how to set up a more compatible query language, nor how to reuse existing mining results for further mining applications. We differ from their approach, in this thesis, we develop a new query language M2MQL which aims at supporting multi step association rule mining tasks from both semantic and syntactic aspects.

To explore a wider use of current relational database techniques, there have been several attempts have been made to develop methods for data mining using SQL, e.g., [Jam01a, Jam01b, HS95, MWZ00, TS98, AN01].

1. Ad hoc rule mining in SQL3[Jam01a, Jam01b] are presented by Hasan M. Jamil. This work proposes a new approach for association rule mining in an object relational database by using only basic SQL3 constructs and functions. It demonstrates that several SQL3 expressions are applicable to mining of rules. But, as mentioned by the author, some of the functions used are not well supported. It is still unclear whether this approach is effective for computing association rule. Thomas and Sarawagi present a similar work [TS98].

2. OLE DB DM [AN01] is an Application Programming Interface (API) proposed by Microsoft Corporation. Two new concepts are supported: Data Mining Models and Cases. The objective of this work is to expedite the establishment of a standard as well as to enrich the adaptivity of the SQL-server series products.

3. Set-oriented mining of association rules in the form of SQL queries using the SETM algorithm was proposed by [HS95]. DB2's object relational extensions for mining association rules uses SQL to express association rule extraction algorithms.

Other query language for data mining include Data Retrieval Language [AU79], Query Flocks [TUA+98] and sequential pattern query language [SZZA01]. Tsur et al use a generate-and-test model for association rule mining recorded in [TUA+98]. Sadri et al propos a sequential pattern query language for supporting instant data mining for e-services [SZZA01].

2.4 Unified data mining models

Johnson, Lakshmanan and Ng in [JLN00] propos an algebra based method on a 3W model. It describes and unifies intensional dimension, extensional dimension, and data dimension in mining processing. These three dimensions are called three *worlds* (3W). It employs a horizontal unfolding schema. A paper related to this work is [HLN99].

A similar work to the 3W model is [Gei02]. Differing from the decomposition of the whole data mining model in the 3 dimensions in 3W model, [Gei02] proposes a framework consisting of a model view, a data view and a process view. Its algebraic design and implementation issues are introduced in [GS02].

ATLaS is an acronym for "Aggregates and Table Language and System" [HXW00, WZ00]. It is developed by the Computer Science Department,University of California in Los Angeles, USA. It adds to SQL the ability to define new user defined aggregates and table functions based on the SQL-3 standard. It also partially adopt the design style of LDL++.

2.5 Other topics

[Rot88, Sch, AFS89, SS87] are early works on nested relations. Basic concepts, main advances, data model specification and algebraic operations are provided in these literatures. [IBM01] is a recent white paper for nested relational databases. It explains the differences between traditional relational databases and the IBM nested relational databases. The publication of IBM U2 in some sense shows the practicality of nested relational database technology as adopted in this thesis.

[RPN00, Ran02, RSMW03] are published materials concerning set operations in

nested relational databases.

[IM96, Jam01b] discuss issues of query evaluation and optimization. In [IM96], authors presented two approaches to query evaluation: top-down and bottom-up. [Jam01b] proposes a proof on the equivalence of top-down and bottom-up evaluation methods. [Bus01] is a detailed work on evaluation powerset algebra expressions. It also provides a direct proof of the equivalent expressive power between the flat relational algebra and the nested relational algebra.

[GS02] is concerned with implementation and query execution engine design issues. This work discusses how to implement a set of DBMS-coupled operators for decision tree mining.

Information about parser implementation tools and developing an execution engine for data mining query language refer to [Les75, Joh79, ASU86, RG03].

Chapter 3

A New Data Mining Query Language M2MQL

In this chapter, we introduce our rule mining query language *Mining to Mining Query Language (M2MQL)*. It focuses on two aspects: a series of practical queries and a formal language syntax. The organization of this chapter is as follows: In the first section, we show how *M2MQL* represents queries for mining `simple item-based association rule`. Example queries in the second section are cases related to mining `generalized item-based association rule`. Queries written by M2MQL for mining `quantitative association rule` are given in the third section. In the fourth section, we present the full language syntax of the M2MQL.

M2MQL should be *declarative* and be *compatible with SQL*. Based on these principles of M2MQL, we give three kinds of example queries in the first three sections. In our terminology *rule set* indicates a relation storing result sets of rules derived from rule mining operations; and *source data set* indicates relations storing normal data. The first kind of queries are only related to one *rule set*. The second

tid	*item*
1	a c d
2	b c e
3	a b c e
4	b e

Table 3.1: (left:)A `purchase` table for a simple transaction record

tid	*tidcontent*
	item
1	a, c, d
2	b, c, e
3	a, b, c, e
4	b, e

Table 3.2: (right:)`purchase` table in nested relation

kind of queries refer to both *rule set* and *source data set*. The third kind of queries are those concerning manipulations between two or more *rule sets*.

3.1 Simple item-based association rule

The source transaction data we use in this section is the classical basket data set. When a customer buys a set of products (also called *items*), the whole purchase is referred to as a *transaction*. Each transaction is given a unique identifier. Let item universe be I={a, b, c, d, e}. Table 3.1 and Table3.2 depict the transactional database `purchase`.

3.1.1 One rule set

Query 1 *Given two thresholds:* `minsupport=0.5` *and* `minconfidence=0.6`*, get all valid rules from the* `purchase` *table.*

M2MQuery 1

```
CREATE RULE R1 AS
SELECT tidcontent.item AS BODY
FROM purchase
WITH support THRESHOLD = 0.5
WITH confidence THRESHOLD = 0.6
```

This M2MQuery generates a new table, `R1`, where each tuple corresponds to a discovered rule. The `SELECT` clause defines the structure of the rules and the `BODY` is defined as a set of items with random cardinality. Note that here is no specification for the structure of the `HEAD`. When this statement is omitted, rules' `HEAD` should have the same structure as their `BODY`. The clause `WITH .. THRESHOLD` indicates that the query produces only those rules whose support and confidence are greater than or equal to the minimum support and confidence. In this case, we have a minimum support threshold of 0.5 and a minimum confidence threshold of 0.6. Table 3.3 and Table 3.4 show the resulting `R1` table.

Query 2 *From the rule set* `R1`*, get all those rules whose* `BODY` *or* `HEAD` *contain 2 items.*

This is a database query with post processing on the existing association rule set. In this case, the resulting rule table `R1` is assumed to be preciously a produced rule set. Constraints for the rule's length are added into the new query. The query as represented in M2MQL is given as below:

BODY	HEAD	sup	conf
bc	e	0.5	1
be	c	0.5	0.667
ce	b	0.5	1
b	ce	0.5	0.667
c	be	0.5	0.667
e	bc	0.5	0.667
a	c	0.5	1
c	a	0.5	0.667
b	c	0.5	0.667
c	b	0.5	0.667
b	e	0.75	1
e	b	0.75	1
c	e	0.5	0.667
e	c	0.5	0.667

Table 3.3: (left):

Resulting rule set `R1` derived from `purchase`

BODY	HEAD	sup	conf
item	item		
b c	e	0.5	1
b e	c	0.5	0.667
⋮	⋮	⋮	⋮
c	b e	0.5	0.667
⋮	⋮	⋮	⋮
e	c	0.5	0.667

Table 3.4: (right):

Resulting rule set `R1` as a nested relation

M2MQuery 2

```
SELECT RULE R1_2items
FROM R1
WHERE length(BODY)=2
   OR length(HEAD)=2
```

`SELECT RULE` statement produces a new resulting relation, `R1_2items`, which contains only rules with a 2-item length body or head. Constraints for rules' length are denoted as a set of conditional expressions in the `WHERE` clause. We define several rule functions including `length(BODY)` and `length(HEAD)` to help managing rules. Note that inequalities can also be used as conditional expressions in the `WHERE` clause neatly.

Query 3 *Get rules with a specified item, say item* `a`*, out from the rule table* `purchase`.

These kinds of queries are proposed quite often in practice. It is well recognized that not every rule in a rule table will be used. Every decision maker has their own focus. They do not wish to use a mass of rules, only those they actually need. Queries for this target could be written in M2MQL as below:

M2MQuery 3

```
SELECT RULE rule_for_a
FROM purchase
WHERE (a) IN purchase.BODY
   OR (a) IN purchase.HEAD
```

Operator `IN` represents the subset or superset conditions. The embraced item or itemset on the left hand side of the `IN` operator should be a subset of the

element given on its right hand side. This implies the right side is the superset or equivalent to the left hand side.

Query 4 *One manager of a TV company is only interested in the sale performance of a "TV set + antenna" package. Item 'a' is an alias of TV set while 'b' is one for the antenna. Get rules fulfilling this manager's requirement from the rule set* `purchase`.

Two kinds of rules could be taken as result candidates for the above query. The first kind is set of rules of pure association between the two items in the "TV + antenna" package. The other kind reveals associations between this package, or itemsets containing this package, and other itemsets. The query in M2MQL is given below:

M2MQuery 4

```
SELECT RULE rule_for_pack_2nd
FROM purchase
WHERE (a) IN purchase.BODY, (b) IN purchase.HEAD
   OR length(BODY)>=2, (a,b) IN purchase.BODY
   OR length(HEAD)>=2, (a,b) IN purchase.HEAD
```

3.1.2 Rule set and Source data set

Generating rules from transaction records or original data sets has been well studied, but how to reuse generated rule sets on the source data set, or how to filter raw data set with the rule set is still left to be done. In this section, examples focusing on this aspect are given, illustrating that our new query language could play a role as a bridge connecting raw data sets and the rule sets.

Query 5 `R1` *is a rule set retrieved from the raw sale records -* `purchase_january`. *Rule* a $\Rightarrow$ *b is one valid rule. Now we are looking for tuples in the* `purchase_january` *which hold up this rule.*

M2MQuery 5

```
SELECT *
FROM purchase_january
WHERE EXIST {(a,b) IN purchase_january.tidcontent}
```

In this example, note that `purchase_january` is a nested relational database. This query retrieves tuples "supporting" one fixed rule. Here we use `EXIST` which is based on set comparison. Note that, normal association rule mining lacks the ability to connect between the rule set and the source data set. Normal methods based on the `group by` operation cannot keep the grouping information between itemsets and transaction records, which is valuable "mapping" information. The use of nested relations and the set comparison operator `IN` overcome this. In this way, not only associations between items or itemsets but also concealed information can be retrieved from the source database again. Differing from the classical knowledge discovery process, two kinds of knowledge can be "mined" from one source data set. One is the usual association rule set and the other is a set of specific tuples, which satisfies one or more given rules. This is the reason why *Mining to Mining* is emphasized in this work. Similarly, someone may be interested in those records which disobey a given rules, and this is equally easy to mine with our system.

Query 6 `R1` *is a rule set retrieved from the raw sale records -* `purchase_january`. *Rule* a $\Rightarrow$ *b is one valid rule. Now we are looking for tuples in the* `purchase_january` *which have no relation to this rule.*

M2MQuery 6

```
SELECT *
FROM purchase_january
WHERE NOT EXIST {(a,b) IN purchae_january.tidcontent}
```

Data mining research has always focused on retrieving rules from the raw data set. This is only a kind of "mono-direction" data mining procedure. Consider that by proposing a series of queries, a connection or mapping from the rule set back to the raw data is set up. This is a positive step towards establishing interactive data mining procedures.

3.1.3 New rule sets from existing ones

Now consider how to retrieve new rules from one or more existing rule sets. The value of this observation is to explore hidden information which might exist in the normal rule sets. For example, we have 12 purchase records, one for each month. From each record, we get one corresponding rule set and so more than one rule set is available for our analysis. We may be interested to know whether there is any connection between items or item sets in a time gap longer than 1 month. Here are examples to illustrate thus. Note that, in our examples, we assume the source data sets - sale records for each month - have similar transaction sizes.

Query 7 *Given two rule sets:* `R1` *and* `R2`*, containing all valid association rules derived from two transaction records gathered in January and February respectively. Retrieve rules with the same HEAD, a longer BODY and an increasing support.*

M2MQuery 7

```
SELECT R2.HEAD AS HEAD
FROM   R1, R2
WHERE  R1.HEAD = R2.HEAD
AND    length(R1.BODY) > length(R2.BODY)
AND    R1.support < R2.support
```

In this example, the statement `SELECT R2.HEAD AS HEAD` indicates target rules are selected from rule set `R2`. `FROM R1, R2` indicates `R1` is also used in this query (it is used as a contrastive rule set). Note here, `length(`R_i`.BODY)` is one of the assistant functions supported by M2MQL. These functions are mainly used to return some statistical values of set valued attributes.

The reason for proposing this kind of query here is as follows. For instance, there are two independent association rules in `R1` and `R2`, $a \rightarrow b$, $supp = 60\%$ in `R1` and $a \rightarrow (b, c)$, $supp = 65\%$ in `R2`. If we assume that the two transaction records are of comparable sizes, then a 5% rise of $supp(R2)$ in February implies that there is at least a 5% sales enhancement on item `a`. What stimulated product `a`'s sales increase? Given the two rules above, conclusions are able to drawn, such as "The promotion of binding products `b` and `c` stimulates product `a`'s sale". With generalized association rules, more information can be analyzed between two or more than two rule sets. More interesting deductions can be drawn in this way, such as, "The promotion of binding product `b` and product `c` together and a 10% lower price of `c` can upgrade product `a`'s distribution 7.5%."

3.2 Generalized item-based association rules

Compared with simple item-based association rules, generalized association rules can become more comprehensive and interesting. More information and con-

straints can be involved when generating these kinds of rules.

The source transaction data we use in this section is also the classical basket data set, but compared with Table 3.1, it contains more information about the purchase records. Besides items bought, each transaction here also contains the transaction's date and the item's quantity and price.

tr.	*cust*	*item*	date	*price*	*q.ty*
1	cust1	ski_pants	17	140	1
1	cust1	hiking_boots	17	180	1
2	cust2	col_shirt	18	25	2
2	cust2	brown_boots	18	150	1
2	cust2	jackets	18	300	1
3	cust1	jackets	18	300	1
4	cust2	col_shirts	19	25	3
4	cust2	jackets	19	300	2

Table 3.5: A `purchase` table for a generalized transaction record

A sample relation is given in Table 3.5. The transaction column (*tr.*) contains the identifier of the customer transaction; the other columns correspond to the customer identifier (*cust*), the type of the purchased item (*item*), the date of the purchase (*date*), the price (*price*) and the quantity (*q.ty*).

The source data sets for generalized association rules contain more information than plain transaction records. Correspondingly, this kind of association rules can sustain user queries with additional constraints. Other than the measurements, the schemas of rules, users can also propose queries according to specific attributes of the items.

Now let us consider some frequent phenomena. Retailers always propose new promotions for product "A" with the intension of stimulating the sale of product "B". But how could we know this kind of associations between pairs of products? To retrieve these associations, preexisting rules needed to be further "mined". In this phase, two steps are required. The first step is to fix the range of items referring to rules with the same expression but different support values. The second step focuses on the detailed information of the fixed items or itemsets. By mapping rules to the source data records, the fluctuation of which attributes affected the sales records could then be revealed. Thus, promotions could be put into practice and new records are then collected. After getting these new records, sales managers may then be concerned about whether or not their former sales promotions have achieved the expected effect. This could be viewed as the decision checking phase. We interpret these two phases by a set of queries with example data similar to the one used in the former section:

Query 8 *Given two source data sets,* `S1` *and* `S2` *shown as Table 3.6, and two thresholds,* `support = 0.04` *and* `confidence = 0.6`. *Get all valid rules from these two data sets named by* `R1` *and* `R2`.

M2MQuery 8

```
CREATE RULE R1 AS
SELECT S1.itemset.item AS BODY
FROM S1
WITH support THRESHOLD = 0.04
WITH confidence THRESHOLD = 0.6
```

In the same way, rule set `R2` is generated from `S2` shown as Table . In this

tid	itemset	
	item	price
1	a	pa1
	b	pb1
	c	pc1
2	...	...
	...	...
3	⋮	⋮

S1: source data set 1

tid	itemset	
	item	price
1	a	pa2
	b	pb2
	c	pc2
	e	pe2
2	...	...
	...	...
3	⋮	⋮

S2: source data set 2

Table 3.6: S1 and S2: two source data sets

body	head	supp	conf
a	b	s11	c11
⋮	...	...	⋮

R1: Rule set from S1

body	head	supp	conf
a	b	s21	c21
⋮	...	...	⋮

R2: Rule set from S2

Table 3.7: R1 and R2: two rule sets

example, assume that we are only interested in rules related to product "a". After generating the two rule sets `R1` and `R2`, the post-processing procedure is used to prune out futile rules.

M2MQuery 9

```
SELECT RULE R1-A
FROM R1
WHERE (a) IN R1.BODY
OR    (a) IN R1.HEAD
```

In the same way, R2-A is generated from R2 shown as Table 3.8.

body	head	supp	conf
a	b	s11	c11
a b	c e	s12	c12
⋮	...	...	⋮

R1-A: subset of R1 which contains "a"

body	head	supp	conf
a	b	s21	c21
a b	c e	s22	c22
⋮	...	...	⋮

R2-A: subset of R2 which contains "a"

Table 3.8: R1-A and R2-A: two subsets of R1 and R2

Query 9 *Get rules which have the same scheme but a increasing support from* R1-A *and* R2-A. *The new rule set is given a name* RI *which abbreviates for rulesets-intersection shown as Table 3.9.*

M2MQuery 10

```
CREATE RI AS
(SELECT BODY, HEAD
 FROM R1-A, R2-A
 WHERE R1-A.BODY = R2-A.BODY
 AND   R1-A.HEAD = R2-A.HEAD
 AND   R1-A.sup  < R2-A.sup)
```

Once the intersectant rule set is generated, we are thinking about factors that cause the fluctuations on rules' support. The following step is to filter source data supporting rules in RI.

Query 10 *Given a rule set* RI *and two source data sets* S1 *and* S2. *Get two subsets of* S1 *and* S2 *storing tuples which support rules in* RI *only.*

body	head
a	b
	c
a	c
b	e
⋮	⋮

RI: Intersected rule set of R1, R2

Table 3.9: RI: rule intersectant part

M2MQuery 11

```
CREATE S1-sup-RI AS
(SELECT *
 FROM S1, RI
 WHERE EXIST
            (    RI.BODY IN S1.itemset
             AND RI.HEAD IN S1.itemset))
```

In the same way, the subset of S2, named by S2-sup-RI is generated shown as Table 3.10. At this stage, by joining S1-sup-RI and S2-sup-RI, a new relation comparing items' prices is then generated shown as Table 3.11.

M2MQuery 12

```
SELECT S1-sup-RI.itemset.item, S1-sup-RI.itemset.price,
       S2-sup-RI.itemset.item, S2-sup-RI.itemset.price
FROM S1-sup-RI, S2-sup-RI
WHERE S1-sup-RI.itemset.item  =  S2-sup-RI.itemset.item
AND   S1-sup-RI.itemset.price <> S2-sup-RI.itemset.price
```

BODY	HEAD	tid	itemset	
item	item		item	price
a	c	1	a	pa1
b	e		b	pb1
			c	pc1
			e	pe1
a	c	2	a	pa1
b	e		b	pb1
			c	pc1
			d	pd1
			e	pe1
⋮	⋮	⋮	⋮	⋮

S1-sup-RI: Subset of S1 which supports RI

BODY	HEAD	tid	itemset	
item	item		item	price
a	c	1	a	pa2
b	e		b	pb2
			c	pc2
			e	pe2
a	c	2	a	pa2
b	e		b	pb2
			c	pc2
			d	pd2
			e	pe2
			g	pg2
⋮	⋮	⋮	⋮	⋮

S2-sup-RI: Subset of S2 which supports RI

Table 3.10: S1-sup-RI and S2-sup-RI: two subsets of S1 and S2

This example includes how to manipulate association rule mining applications, how to post-process on existing rule sets, how to deal with the source data set guided by new generated rule sets, and how to retrieve new information from more than one rule sets. This example depicts the main characters of the M2MQL. It supports post-processing on existing rule sets which enhances the traditional single cycle of data mining applications. It connects the rule sets with the source data set which realizes a reciprocal use between rules and data sets.

item	price1	item	price2
a	pa1	a	pa2
b	pb1	b	pb2
⋮	⋮	⋮	⋮

PriceC: Price comparison

Table 3.11: PriceComparison

3.3 CREATE RULE and SELECT RULE Primitive

The M2MQL syntax is comprised of four basic statements. The main idea behind the language design has been to allow intensive representation and manipulation of rule components as well as the whole rule set generated, which, being flat cells or nested sets, are not easily represented in standard SQL.

The outline of the syntax for these M2MQL extension is shown below:

```
<M2MQL statement> ::=   <CREATE RULE PRIMITIVE - statement>
                    |   <SELECT RULE PRIMITIVE - statement>
```

We consider each M2MQL command in its respective section in the rest parts of this chapter. The generalized syntax is given in details in Appendix A.

The CREATE RULE PRIMITIVE statement is used for rule type-declaration, rule schema-illustration and rule-generation. The SELECT RULE PRIMITIVE statement provides interactive processing between generated rule sets and original data sets thereby realize the reuse of existing rule sets in other data mining queries.

Chapter 4

The Algebra in M2MQL

In this chapter, we describe a set of operators that would be used in our M2MQL internal query expressions. In our approach, we adopt nested relational algebra with a set of extended algebras specified for data mining. First, we do a quick review of nested relations and explain why we chose this type of relation for rule mining. Secondly, nested relational algebra is outlined. The specific data mining algebra is introduced in the third section with definition, operative formula and examples.

4.1 Review of nested relations

A fundamental property of existing relational database theory is that relations must be at least in 1NF ($1^{st} - normal - form$). This property influences the whole database domain and has been long accepted by both researchers and practitioners. But as stated in Codd's publication[Cod71] - *"For presentation purposes, it may be desirable to convert a normalized relation to un-normalized form"*. This statement implies the need for un-normalized relations. Then comes the nested

relation, the nested relational model [Mak77, SS87, AFS89] was developed by Makinouchi, etc, in order to extend the applicability of the flat relational model to non-business applications. Such applications include office automation, computer aided design, image processing, spatial data, text retrieval, expert systems, and geographical and statistical analysis.

4.1.1 Concepts of nested relation

The nested relational model removes the 1NF restriction from the flat relational model, and allows attribute values to be relation-valued as well as atomic. Thus, the nested relational model attribute domains are defined recursively, i.e. an element of a relation-valued domain may be itself relation-valued. The resulting non-first-normal-form relations (NF^2) are called nested relations and are also referred to in the literature as complex objects [AFS89, SS87].
In general, the advantages that result from the use of nested relations in comparison with flat relations are as follows:

1. Increased flexibility of data structures allows a more direct mapping of applications onto the database;

2. The hierarchical structures of nested relations can be used to describe a wide variety of physical database designs;

3. Nested relational query languages can nicely express some of the "awkward" SQL-features, such as grouping;

4. The conceptual gap between relational and nested relational models is minimal. Thus this extension could be easily acceptable by users.

When we are puzzled about how to express data mining tasks with SQL and when we are hindered by the restriction imposed by 1NF databases, nested relations throw a new light on way ahead. We wonder if its relaxed specialities could be utilized in data mining applications? In the coming subsection, we discuss this question in detail.

4.1.2 Nested relation and association rule mining

We adopt nested relations in the association rule mining approach, because:

1. Nested relations minimize redundant data. In association rule mining, using 1NF relations, the original transaction data set, intermediate itemsets and frequent itemsets can generate lots of redundant information to identify different categories. By allowing set-valued attributes, nested relations present the generic information in a more natural and efficient way.

2. Join operations can be more effective realized using nested relations making query processing is more efficient. Assume that the well known Apriori algorithm is used to generated rules, recursive joins are unavoidably needed in the *generate-prune-generate* procedure. Once implemented, we can see that the reduction of joins will reduce the computing time required.

3. Nested relations allow a very flexible interface at the external level, since both flat and hierarchical data can be presented to the user. At the end of association rule mining, associations between two itemsets are revealed. It would be awkward to represent rules in the form of $\{item_a, item_b, ...\} \Rightarrow \{item_i, item_j, ...\}$ as two product tables and one $\Rightarrow$ relation. On the other hand, with support for set-value, one nested relation could represent sets

of rules as tuples with two columns $Itemset_{left}$, $Itemset_{right}$ and additional columns for the rule's parameters.

4. Nested relations can explicitly represent the semantics of the rule mining applications.

Hereinabove, aptness between nested relation and rule mining is explained intuitively. To thoroughly get into association rule mining in nested relations, algebraic operators are introduced in the coming section.

4.2 Nested relational algebra

The operators needed for expressing association rule queries include SELECT, PROJECT, NEST, UNNEST and RENAME. The usual set operators of UNION, DIFFERENCE, INTERSECTION and CARTESIAN PRODUCT are also part of the algebra adopted in our expressions.
In the following illustrations, we use one record of `purchase` record named by `tran-tid1` (Table 4.1) as a running example.

1. **SELECT** σ^e: Definition:

   The *select* operations select tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) with a superscript (^{e}) to denote selection.

   Operative formula:

   $\sigma^e_{(\theta)}(R)$

   The predicate θ appears as a subscript to σ^e. The argument relation is given in parentheses following the σ^e. In general, we allow comparisons using $=$, $\neq$, $<$, $\leq$, $>$, $\geq$ in the selection predicate. Furthermore, we can combine

several predicates into a larger predicate using the connectives *and* ($\wedge$) and *or* ($\vee$).

2. **PROJECT** π^e Definition:

   Traditional project operation π is to returns its argument relation, with certain attributes left. Duplicated rows are eliminated in this way. Here we denote an extended project operation for the nested relation by the Greek letter pi(π) with a superscript (^{e}). By using π^e on attributes in different levels, more flexible relations can be produced.

   Operative formula:

   $\pi^e_{(A_1,A_2,...,A_n)}(R)$

   In this expression, $(A_1, A_2, ..., A_n)$ indicates attributes to be left in the resultant relation and (R) is the relation argument. Rules for predicates' composition and combination are the same as we defined for those in the σ^e operation.

3. **NEST** Γ Definition:

   The NEST operator creates partitions which are based on the formation of equivalence classes. Tuples are equivalent on some attributes if the values of these attributes are the same. All equivalent tuples are replaced with a single tuple in the resulting nested relation. The resulting relation is then defined on all the attributes on which the equivalence is defined, and a nested attribute that is defined on the rest of the attributes in the original relation.

   Operative formula:

   $\Gamma_{C \leftarrow (A_1,A_2,...,A_n)}(R)$

In this expression, $A_1, A_2, ..., A_n$ are attributes in relation R. C is a new attribute name which is one level above $A_1, A_2, ..., A_n$ in the resultant relation.

We can also comprehend the **NEST** operation from another perspective. Assume that relation R is in the form of $(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ where B_i represents some attributes in R. Nesting attribute set $(A_i, A_2, ..., A_n)$ plays a similar role to that of Grouping attribute set $(B_1, B_2, ..., B_m)$. Now let us get down to the differences between these two similar but non-identical operations.

In the operation **NEST**, relation R is partitioned by the formation of equivalence B classes. Tuples with equivalent B attribute set values are replaced by a single tuple, columns of attribute set A are appended as a corresponding tuple value to the single B values with a new attribute name C.

In the operation **GROUPING**, relation R is firstly partitioned by the equivalent B classes. But, in the resultant relation, the values of the remaining columns ($A_i s$) which correspond to the single B values could only be an aggregation result based on attribute $A_i s$ from those tuples of the original relation R.

Here, we note that the NEST operation restructures the nested relation and allows us to view the internal information in the nested structures which was concealed by the aggregation in the Grouping operation. Moreover, this internal information is exactly what would be ubiquitously used in statistics or set comparison through the whole rule generation phases. From this angle, we confirm again the fitness of nested relations adopted in the rule mining tasks.

4. **UNNEST** η Definition:

   The UNNEST operator reverses the effect of the NEST operator. Intuitively, nesting transforms a nested relation into one which is more deeply nested, while unnesting transforms a nested relation into a flatter one.

   Operative formula:

   $\eta_C(R)$

   C is an attribute name which represents nested values in a relation R.

5. **RENAME** ρ^e

   Definition:

   Let A_1, A_2 be two attributes in the relation R. The *rename operator* is denoted as ρ^e that renames attribute A_1 to attribute A_2, for any attribute A_1 and A_2. Its definition is similar to that in the relational algebra.

   Operative formula:

   $\rho^e_{A_2 \leftarrow A_1}(R)$

4.3 Specific data mining algebra

In this section, a set of algebra designed for data mining tasks are proposed. This set of algebra includes POWERSET($\wp$), SET-CONTAINMENT-JOIN($\bowtie_{\subseteq}$), ARITY(ϱ) and CARDINALITY(ϖ).

4.3.1 POWERSET $\wp$

Definition

In a nested relation, this operator generates a power set of values in a specified attribute. For instance, a specified attribute containing n values in a tuple, $\wp$ generates 2^{n-1} subsets as new values in the output relation (ϕ is not included).

Operative formula

$\wp(R.A_i)$

A_i is a specified attribute in relation R.

Example

In `tran-tid1` (Table 4.1, here we abbreviate it as `tt1`), by using $\wp$ operator on the attribute `tidcontent`, an output relation whose tuple containing 7 values is generated.

`powerset-tran-tid1`(Table 4.2) = $\wp$(`tt1`. *tidcontent*)

4.3.2 SET-CONTAINMENT-JOIN $\bowtie_{\subseteq}$

Definition

Set containment join is denoted by $\bowtie_{\subseteq}$. It is a join between the set-valued attributes b and c of two relations $R(a, b)$ and $S(c, d)$ such that,

$S \bowtie_{b \subseteq c} R = T(a, b, c, d) = \{t \mid t \in S \times R \wedge b \subseteq c\}$.

In case attributes on the two sides of the operator $\bowtie_{\subseteq}$ have different schemes, e.g., the left hand side attributes are only one subset of the right hand side ones.

For the generalized cases, we give a more flexible definition for $\bowtie_{\subseteq}$.

$$S \bowtie_{b(b_1,(b_2...(b_i))) \subseteq c(c_1,(c_2...(c_j)))} R = \tag{4.1}$$

$$T(a, b(b_1, b_2...b_i), c(c_1, c_2...c_j), d) = \tag{4.2}$$

$$\{t \mid t \in S \times R \wedge i \leq j \wedge (b_1, b_2...b_i) \in (c_1, c_2...c_j)\}. \tag{4.3}$$

Please note that in Equation 4.3, expressions $(b_1, b_2...b_i)$ and $(c_1, c_2...c_j)$ indicate that attribute b has i different sub level attributes while attribute c has j sub level attributes. We use Table 4.3 to illustrate this kind of relation. Until year 2000, little work had focused on set containment joins. "There is very little previous work on set containment joins. The only reported work of which we are aware is the work by Helmer and Moerkotte", [RPN00]. For more details about set containment joins, refer to [RPN00, HM96, HM97].

Operative formula

$S \bowtie_{b \subseteq c} R = T(a, b, c, d) = \{t \mid t \in S \times R \wedge b \subseteq c\}$.
S, R are two input relations while T is the output relation. The schemas of these three relations are given in Table 4.5:

Example

In this example, relation R is a frequent itemset candidate. Relation S is a source purchase record. To evaluate its statistical parameter, we use $\bowtie_{\subseteq}$ to get all transactions supporting the given itemset. We use Table 4.7 to illustrate this procedure. Its algebraic expression is as follows:

$T = R \bowtie_{R.itemset \subseteq S.tidcontent.itemcontent} S$

4.3.3 Functional operators

CARDINALITY ϖ

Definition

The statement of CARDINALITY is given as [url03]: Cardinality is a notion of the size of a set which does not rely on numbers. In our approach, CARDINALITY ϖ calculates the size of the $(i+1)^{th}$ level tuples in the i^{th} level relation(for the definition of *level*, please refer to 4.3.3).

Operative formula

$\varpi(R.A_i*) = \varpi\ (R)$ or $\varpi\ (R.A_i+)$

$\varpi\ (R.A_i+) = \varpi\ (R.A_i)$ or $\varpi\ (R.A_i.B_j+)$

Here, R is a relation, A_i and B_j are attributes in R. Asterisk "*" in $\varpi(R.A_i*)$ declares that both expressions with or without A_i are valid. Without A_i, $\varpi(R)$ indeed gives the cardinality of the relation. In $\varpi\ (R.A_i.B_j+)$, note that B_j is one of the subsequent attributes of the attribute A_i. The following example gives a straightforward depiction of these notations.

Example

We use Table 4.7 again to illustrate ϖ.

$$\varpi(T.itemset) = 2 \tag{4.4}$$

$$\varpi(T.tid) = 2 \tag{4.5}$$

$$\varpi(T.tidcontent) = 2 \tag{4.6}$$

$$\varpi(T.itemset.item) = 2[whiletid = 3] \tag{4.7}$$

$$\varpi(T.tidcontent.custcontent) = 2 \tag{4.8}$$

$$\varpi(T.tidcontent.itemcontent) = 2 \tag{4.9}$$

$$\varpi(T.tidcontent.custcontent.age) = 1[while\ tid = 3] \tag{4.10}$$

$$\varpi(T.tidcontent.custcontent.career) = 1[while\ tid = 3] \tag{4.11}$$

$$\varpi(T.tidcontent.itemcontent.item) = 3[while\ tid = 1] \tag{4.12}$$

$$\varpi(T.tidcontent.itemcotnent.item) = 4[while\ tid = 3] \tag{4.13}$$

ATTACH α

Definition

Just as its name implies, the ATTACH operator α "attaches" a new attribute, *new attribute name*; for each tuple, the value to be assigned to the new attribute is obtained by the algebraic expression denoted *expression*.

Operative formula

$\alpha(new\ attribute\ name,\ expression)(R)$

In this formula, *new attribute name* is the new "attached" attribute in the result relation while *expression* is defined to evaluate values of the new attribute for each tuple, and R is the operand relation.

Example

T = $\alpha(support, \frac{\varpi(\mathrm{R}.tidlist)(\mathrm{R})}{AllGroup})$ (R)

In this example, R is an operand relation, T is a resultant relation and *support* is the name given to the new attribute in T. *AllGroup* is an integer which indicates

a statistical parameter. $\frac{\varpi(\mathtt{R}.tidlist)(\mathtt{R})}{AllGroup}$ here is an expression used to calculate values for each tuple in the new attribute *support*. Assume that relation R has a schema in the form as (*tidlist*, *itemset*), after α operation, its schema will be modified to (*tidlist*, *itemset*, *support*).

To all appearances, the introduce of $attach(\alpha)$ operation is to modify table schemes horizontally which could release users flexibly representing statistical parameters, e.g., *support* and $confidence$ in the rule mining tasks.

ARITY ϱ

Definition

The statement of ARITY from [url03] is given as: The arity of something is the number of arguments it takes. Here in our approach, ARITY ϱ indicates the depth variable of tuples in a nested relation. With this depth parameter, we can distinguish attributes that are located in different sub relations into various levels. Those attributes in a more outer relation, in a higher level have a small arity value and vice versa.

Operative formula

$\varrho(R.A_i)$

A_i is a specified attribute in relation R.

Example

In Table 4.2(`powerset-tran-tid1`, here we abbreviate it as `ptt1`), we can get the arity value for each attribute:

$\varrho(\mathtt{ptt1.tid}) = \varrho(\mathtt{ptt1.powerset - content}) = 1$

$ptt1.tid$ and $ptt1.powerset - content$ are 1^{st} *level* attributes.

$\varrho(\mathtt{ptt1.itemset - id}) = \varrho(\mathtt{ptt1.itemset}) = 2$

$ptt1.itemset - id$ and $ptt1.itemset$ are 2^{nd} *level* attributes.

$\varrho(\texttt{ptt1.item}) = 3$

$ptt1.item$ is a 3^{rd} *level* attribute.

tid	*tidcontent*
	item
1	a
	c
	d

Table 4.1: (left:)One transaction record `tran-tid1`[abbr:tt1] in a nested relation

tid	*powerset − content*	
	itemset	*itemset*
	−id	*item*
1	1	a
	2	c
	3	d
	4	a
		c
	5	a
		d
	6	c
		d
	7	a
		c
		d

Table 4.2: (right:)Powerset relation `powerset-tran-tid1`
[abbr:ptt1]

a	b			
	b_1	b_2	b_3	b_4

c						*d*
c_1	c_2	c_3	c_4	c_5	c_6	

Table 4.3: Illustration of generalized $\bowtie_{\subseteq}$

a	b
	$b-item$
a_1	b_1
a_2	b_1
	b_2

Relation R:

c	d
$c-item$	
b_1	d_1
c_1	
b_2	d_2
c_1	
b_1	d_3
b_2	
c_2	

Relation S:

a	b	c	d
	$b-item$	$c-item$	
a_1	b_1	b_1	d_1
		c_1	
a_1	b_1	b_1	d_3
		b_2	
		c_2	
a_2	b_1	b_1	d_3
	b_2	b_2	
		c_2	

Relation T:

Table 4.4: (left1,2:) Relation R and S

Table 4.5: (right:) Relation $T = R \bowtie_{b \subseteq c} S$

itemset
item
a
b

Relation R:

tid	*tidcontent*				
	custcontent		*itemcontent*		
	age	*career*	*item*	*price*	*qty*
1	25	student	a	20	1
			b	30	2
			c	5	4
2	34	officer	c	5	1
			e	12	7
3	28	teacher	a	20	4
			b	30	2
			c	5	9
			e	12	4

Relation S:

itemset	*tid*	*tidcontent*				
item		*custcontent*		*itemcontent*		
		age	*career*	*item*	*price*	*qty*
a	1	25	student	a	20	1
b				b	30	2
				c	5	4
	3	28	teacher	a	20	4
a				b	30	2
b				c	5	9
				e	12	4

Relation T:

Table 4.6: (up,left1,2)Relation R : *itemset*, S : *purchaserecord*

Table 4.7: (down:)Relation T : *itemset joined with transactions supporting it*

T=R$\bowtie_{R.itemset \subset S.tidcontent.itemcontent}$S

itemset	*tid*	*tidcontent*				
item		*custcontent*		*itemcontent*		
		age	*career*	*item*	*price*	*qty*
a	1	25	student	a	20	1
b				b	30	2
				c	5	4
	3	28	teacher	a	20	4
a				b	30	2
b				c	5	9
				e	12	4

Relation T:

Table 4.8: Resultant relation T chop from Table 4.7

Chapter 5

Mining On Top of M2MQL

The emphasis of this chapter, is to discuss evaluation issues in M2MQL. We use one running example to depict how to represent one association rule mining query written in M2MQL as a sequence of algebraic expressions sustained by the extended algebra declared in the Chapter 4. First, we describe an example query including its concept, statement and parameters. Three later sections deal with three main phases in solving the association rule mining query. They are the frequency counting phase, frequent itemset generation phase and the rule generation phase.

5.1 Problem statement

In this section, we use association rule mining in classical basket record as our running example. Table 5.1 is the source data set which contains 4 purchase records marked by 4 transaction id numbers (*tid*). Alias information is given in Table 5.1. From this data record, users may not be interested in the plain data but some information may be found which is much more *interesting*. Users

tid	*item*
1	a c d
2	b c e
3	a b c e
4	b e

alias	item
a	ski_pants
b	hiking_boots
c	col_shirt
d	brown_boots
e	jackets

Table 5.1: Source data table with alias

looking for association rules between any two sets of items always propose queries as follows:

Query 11 *Given a transaction record named* `purchase` *(Table 5.1), get all valid rules meeting two thresholds: minsupport= 0.5 and minconfidence=0.6.*

As mentioned in chapter 4, this query can be expressed in M2MQL as below, in which `purchase` is the source data set's name and `R` is a newly generated relation for all valid association rules:

M2MQuery 13

```
CREATE RULE R AS
SELECT tranrecord.item AS BODY
FROM purchase
WITH support THRESHOLD = 0.5
WITH confidence THRESHOLD = 0.6
```

There are two parameters, *support* and *confidence*.

1.*support* = 50%. To be frequent, an itemset must occur in at least 2 transactions;

2.*confidence* = 60%. To be a valid rule, in at least 60% of the transactions containing the *antecedent* (the *body* of the rule) should also contain the *consequence* (the *head* of the rule).

Based on the support-confidence framework, we decompose association rule mining into three phases: the frequency counting phase; frequent itemset generation phase and the rule generation phase.

1. Frequency counting phase: In this phase, one summation of all unitary items and frequencies of each item is calculated. One itemset containing all frequent unitary items is generated with the name *frequent one itemset.*

2. Frequent itemset generation phase: This phase utilizes the *frequent one itemset* to generate all frequent itemsets of different lengths (number of unitary items in one itemset is called the *length* of the itemset).

3. Rule generation phase: The first step of this phase is to calculate all potential rule candidates based on the set of frequent itemsets. Secondly, the given two thresholds are used to filter valid rules. After expurgating redundant rules, the anticipative resultant rule set is created.

In the following three sections, we explain these phases step by step. Each section start with an overview of the algebraic expressions. Then a diagram is used to depict the internal generating procedures. Examples are attached to every algebraic expression in the later part of each sections.

5.2 Frequency Counting Phase

We name tables as `T`i, e.g., the source transaction record stored as a nested relation is named `T1`. Algebraic expressions in this phase are given as below:

1. *AllGroup*= 4.

2. $AllGroup = \varpi(\text{T2}.tid) = 4$

3. T2 = η(T1.*tran record*)

4. T3 = $\Gamma_{tidlist \leftarrow tid}$ (T2)

5. T4 = $\alpha(freq,\ \varpi$(T3.*tidlist*)) (T3)

6. T5 = $\sigma^{e}_{(freq\ >\ minsup)}$ (T4)

Here we use Table 5.2 to depict its internal generating processes. T1 is the source data set as a nested relation. T2 is the source data set as a flat relation. T3 is a new nested relation by nesting *tid* in the source data set. The first step is to

tid	*tran record*
	item
1	a
	c
	d
2	b
	c
	e
3	a
	b
	c
	e
4	b
	e

T1:

tid	*item*
1	a
1	c
1	d
2	b
2	c
2	e
3	a
3	b
3	c
3	e
4	b
4	e

T2:

tidlist	*item*
tid	
1	a
3	
2	b
3	
4	
1	c
2	
3	
1	d
2	e
3	
4	

T3:

Table 5.2: T1 $\Rightarrow$ T2 $\Rightarrow$ T3

count the total number of groups, *AllGroups*. This summation is ubiquitously used in the calculation of measurement *support* and *confidence* in the following steps. From Table 5.2, the parameter *AllGroups* can be obtained as:

1. (step1) T2 = η(T1.*tran record*). In this step, set-valued attribute *tran record* in T1 is unnested, every item is then bounded with its corresponding tid number shown as T2.

2. (step2:) T3 = $\Gamma_{tidlist \leftarrow tid}$ (T2). Operation Γ on attribute *tid* does a similar work as traditional *group by* does for flat relations. In its resultant relation T3, a new set-valued attribute *tidlist* is generated.

3. (step3:) T4 = $\alpha(freq,\ \varpi(\texttt{T3}.tidlist))$ (T3). In this step, a new attribute *freq* is attached to every item. Contents in attribute *freq* are return values calculated by functional operation ϖ on attribute *tidlist*. T4 and T5 are given in Table 5.3. Following the apriori algorithm, we prune out 1-itemsets with frequencies smaller than the given minimum support to retrieve frequent one itemsets. In this example, 1-itemset $\{D\}$ with support 25% ($<$ 50%), a support of less than 50% is pruned out.

items	*tid*	*freq*
{a}	(1,3)	2
{b}	(2,3,4)	3
{c}	(1,2,3)	3
{d}	(1)	1
{e}	(2,3,4)	3

T4:

items	*tid*	*freq*	*support*	*> minsup?*
{a}	(1,3)	2	50%	yes
{b}	(2,3,4)	3	75%	yes
{c}	(1,2,3)	3	75%	yes
{d}	(1)	1	25%	no
{e}	(2,3,4)	3	75%	yes

T5:

Table 5.3: T4 $\Rightarrow$ T5

5.3 Frequent Itemset Generation Phase

The second phase is to generate frequent itemsets of different lengths. Algebraic expressions in this phase are as follows:

1. T6 = $\Gamma_{itemset \leftarrow item}(\pi^{e}_{item}(\texttt{T5}))$

2. T7 = $\wp$ (T6.*itemset*)

3. T8 = $\texttt{T7}_{\texttt{T7}.(itemset) \bowtie_{\subseteq} \texttt{T1}.tran\ record}$ T1

4. T9 = $\Gamma_{tidlist \leftarrow (tid, tran\ record)}$(T8)

5. T10 = $\alpha(supp, \frac{\varpi(\text{T9}.tidlist)(\text{T9})}{AllGroup})$(T9)

6. T11 = $\pi^e_{itemset,\ supp}\ (\sigma^e_{supp\ >\ minsup}\ (\text{T10}))$

Table generation procedure from T6 to T9 is shown in Table 5.4.

1. In T5, {a}, {b}, {c} and {e} are values in the *items* column. These items can not be used or accessed separately.

2. T6 = $\Gamma_{itemset \leftarrow items}(\pi^e_{items}(\text{T5}))$.To make them single set values, the Γ operation is used on attribute *item* in T5 and a new set-value accepted new attribute *itemset* is produced in T6. At this stage, {a}, {b}, {c} and {e} are four independent sets on which set operations, e.g., the *powerset* operation can be applied.

3. T7 = $\wp$ (T6.*itemset*). T7 is generated by the $\wp$ operation. Excluding the null set, all subsets made of any combinations within {{a}, {d}, {c} and {e}} are given as T7.

4. All these subsets are potential frequent itemsets. To judge which itemsets are frequent, it is necessary to determine the frequency for each itemset.
   (step1:) T8 = $\text{T7}_{\text{T7}.(itemset) \bowtie_{\subseteq} \text{T1}.tran\ record}$ T1. This step is to join each itemset with all transaction records which "support" or contain it.
   (step2:) T9 = $\Gamma_{tidlist \leftarrow (tid, tran\ record)}$(T8). As T1 in Table 5.4, T8 can not express "grouped" information. To bind all transaction records to each itemset, the Γ operation is used on attributes *tid* and *tran record*. These two attributes are nested as one new attribute *tidlist* which is shown on top. By doing so, T8 is reformulated by grouping *itemset*, shown as the first

itemset
item
a
b
c
e

T6:

Notation: To limit the length of the table, we use horizontal set to show the contents of the itemsets, convertionally in the vertical layout.

T9 is given as the first three columns of T10 and is not repeated here.

⇓ Powerset(itemset)

itemset
item
a
b
c
e
a b
a c
a e
b c
b e
c e
a b c
a b e
b c e
a b c e

T7:

$\bowtie_{\subseteq}$

tid	*tran record*
	item
1	a
	c
	d
2	b
	c
	e
3	a
	b
	c
	e
4	b
	e

T1:

=

itemset	*tid*	*tran record*
item		*item*
a	1	(a,c,d)
a	3	(a,b,c,e)
b	2	(b,c,e)
b	3	(a,b,c,e)
b	4	(b,e)
c	1	(a,c,d)
c	2	(b,c,e)
c	3	(a,b,c,e)
e	2	(b,c,e)
e	3	(a,b,c,e)
e	4	(b,e)
ab	3	(a,b,c,e)
ac	1	(a,c,d)
ac	3	(a,b,c,e)
ae	3	(a,b,c,e)
bc	2	(b,c,e)
bc	3	(a,b,c,e)
be	2	(b,c,e)
be	3	(a,b,c,e)
be	4	(b,e)
ce	2	(b,c,e)
ce	3	(a,b,c,e)
abc	3	(a,b,c,e)
abe	3	(a,b,c,e)
bce	2	(b,c,e)
bce	3	(a,b,c,e)
abce	4	(a,b,c,e)

T8:

Table 5.4: T6⇒ T7⇒ T8

three columns of T10 in Table 5.5. In T10, one itemset only connected with a set of *tid* list, by accessing T10.tidlist, sets of *tid* as {(1,3)} or {(2,3,4)} can be retrieved.

(step3:) T10 = $\alpha(supp, \frac{\varpi(\texttt{T9}.tidlist)(\texttt{T9})}{AllGroup})(\texttt{T9})$. The expression $\varpi(\texttt{T9}.tidlist)$ calculates the number of transaction records supporting one itemset. By doing $\alpha(supp, \frac{\varpi(\texttt{T9}.tidlist)(\texttt{T9})}{AllGroup})(\texttt{T9})$, a new attribute $(\frac{\varpi(\texttt{T9}.tidlist)(\texttt{T9})}{AllGroup}(\texttt{T9}))$ representing parameter support, named *supp* is attached to each itemset.

(step4:) T11 = $\pi^e_{itemset,\ supp}\ (\sigma^e_{supp\ >\ minsup}\ (\texttt{T10}))$. In this step, only the attributes *itemset* and *supp* which contribute to the following rule generation phase are projected. Also only those itemsets which fulfilled the minimum support threshold are kept in the result relation. T11 is the frequent itemset which is to be used to generate rule candidates.

5.4 Rule Generation Phase

The third phase is to generate all the rules from the frequent itemsets derived in the former phase shown in Table 5.5. In this phase, valid rules over the thresholds *support* and *confidence* can be retrieved in 3 steps:

1. (step1:) generate all pairs of itemsets by set containment join;
2. (step2:) pick out rules above the second threshold *min confidence*;
3. (step3:)eliminate redundant components pairs by nested set difference operation after which all valid rules are derived.

Algebraic expressions for the rule generation phase are shown as follows:

1. T12 = $\rho^e_{(itemset \leftarrow \texttt{Body},\ leftsupp \leftarrow supp)}\ (\texttt{T11})$

itemset	*tidlist*		supp
item	*tid*	*tran record*	
		item	
a	1	(a,c,d)	2
	3	(a,b,c,e)	
b	2	(b,c,e)	3
	3	(a,b,c,e)	
	4	(b,e)	
c	1	(a,c,d)	3
	2	(b,c,e)	
	3	(a,b,c,e)	
e	2	(b,c,e)	3
	3	(a,b,c,e)	
	4	(b,e)	
ab	3	(a,b,c,e)	1
ac	1	(a,c,d)	2
	3	(a,b,c,e)	
ae	3	(a,b,c,e)	1
bc	2	(b,c,e)	2
	3	(a,b,c,e)	
be	2	(b,c,e)	3
	3	(a,b,c,e)	
	4	(b,e)	
ce	2	(b,c,e)	2
	3	(a,b,c,e)	
abc	3	(a,b,c,e)	1
abe	3	(a,b,c,e)	1
bce	2	(b,c,e)	2
	3	(a,b,c,e)	
abce	4	(a,b,c,e)	1

T10:

itemset	*supp*
item	
a	2
b	3
c	3
e	3
a c	2
b c	2
b e	3
c e	2
b c e	1

T11:

Table 5.5: T10 ⇒ T11

2. $\mathrm{T13} = \rho^{e}_{(itemset \leftarrow \mathtt{Head},\ leftsupp \leftarrow supp)}\ (\mathtt{T11})$

3. $\mathrm{T14} = \mathtt{T12}\ {}_{\mathtt{T12.Body}} \bowtie_{\subseteq} {}_{\mathtt{T13.Head}}\ \mathtt{T13}$

4. $\mathrm{T15} = \sigma^{e}_{(conf\ >\ minconf)}\ (\ \alpha(conf,\ \frac{\mathtt{T14}.rightsupp}{\mathtt{T14}.leftsupp})(\mathtt{T14})\ )$

5. $\mathrm{T16} = \pi^{e}_{(\mathtt{Body},\ \mathtt{Head},\ supp,\ conf)}(\ \rho^{e}_{(supp \leftarrow right.supp)}(\mathtt{T15})\ )$

Pairs of Itemsets

Association rules reveal associations between items or itemsets within same frequency itemsets. Association rules' connotation decide the way to generate rules. Given one frequent itemset, say F, containing i_1, i_2, i_3. Any two mutually exclusive subsets of this itemset can be a potential valid rule. It is unavoidably necessary to create all pairs of combinations between its own two subclasses. Generally, to deal with this problem, the *powerset* operation is used on the given itemset, then the *cross product* of the two powersets is found. In this traditional way there exist two time consuming processes, *powerset* and *cross product*. In our approach, we adopt the *set containment join* operation in our rule candidate generation phase. This join between two duplicated copies of one frequent itemset produces all rule candidates as in its resultant relation. The join condition *to be contained* guarantees all rule candidates are included. This procedure is shown in Table 5.6. Note that, in these tables, *Body* and *Head* are set-valued attributes with all frequent itemsets as their sublevel contents. Pairs of itemsets are generated by a *set containment join* which combines frequent itemset table with itself. Based on the support-confidence framework, all rules are originate from frequent itemsets. Each rule is a combination between a frequent itemset and one of its subsets. To find all valid rules, we first generate all possible combinations of each frequent itemset and any of its subsets. In our approach, the set containment

Body *item*	*leftsupp*
a	2
b	3
c	3
e	3
a c	2
b c	2
b e	3
c e	2
b c e	1

T12:

$\bowtie_{\subseteq}$

Head *items*	*rightsupp*
a	2
b	3
c	3
e	3
a c	2
b c	2
b e	3
c e	2
b c e	1

T13:

=

Body *items*	*leftsupp*	*Head* *items*	*rightsupp*
a	2	a	2
a	2	a c	2
b	3	b	3
b	3	b c	2
b	3	b e	2
b	3	b c e	1
c	3	c	3
c	3	a c	2
		b c	2
c	3	c e	2
c	3	b c e	1
e	3	e	3
e	3	b e	2
		c e	2
e	3	b c e	1
a c	2	a c	2
b c	2	b c	2
b c	2	b c e	1
b e	3	b e	3
b e	3	b c e	1
c e	2	c e	2
c e	2	b c e	1
b c e	1	b c e	1

T14:

Table 5.6: T12 $\bowtie_{\subseteq}$ T13 $\Rightarrow$ T14

join operator is used again. Tuples are joined if, and only if, there exists a "be contained" relationship between two operand tuples.

1. In Table 5.6, `T12` and `T13` are duplicate copies of `T11` get from T12 = $\rho^{e}_{(Body \leftarrow \texttt{itemset},\ leftsupp \leftarrow supp)}$ (`T11`) and T13 = $\rho^{e}_{(Head \leftarrow \texttt{itemset},\ leftsupp \leftarrow supp)}$ (`T11`). The *Rename* ρ operation in these two expressions redefines attribute names to distinguish the two operand relations. These attribute aliases also help to convert rules into their final scheme - (`Body, Head, supp, conf`). T14 = `T12` $_{\texttt{T12.Body}} \bowtie_{\subseteq} {}_{\texttt{T13.Head}}$ `T13`.

2. In `T14`, every item or itemset is *joined* with itself and those itemsets containing it. This join matches all possible itemsets together as a set of rule candidates. In `T14`, one tuple indicates one possible rule. For example, frequent one itemset $\{e\}$ is joined with four frequent itemsets, $\{e\}$, $\{b, e\}$, $\{c, e\}$ and $\{b, c, e\}$. These four tuples imply all rules with item e as their rule `Body`. At this stage, the following sequence is used to filter valid rules over the two given thresholds.

Valid Rules

The support of a rule is simply the support of the frequent itemset from which this rule is originated. For example, for rule candidate $b \rightarrow c,\ e$

$$support\ (b \rightarrow c,\ e) = support\ (b,\ c,\ e)$$

The confidence is,

$$confidence\ (b \rightarrow c,\ e) = \frac{support(b,\ c,\ e)}{support(b)}$$

Body item	*leftsupp*	*Head item*	*rightsupp*	*conf*
a	2	a	2	1
a	2	a c	2	1
b	3	b	3	1
b	3	b c	2	0.66
b	3	b e	2	0.66
b	3	b c e	1	0.33
c	3	c	3	1
c	3	a c	2	
c	3	b c	2	0.66
c	3	c e	2	0.66
c	3	b c e	1	0.33
e	3	e	3	1
e	3	b e	2	0.66
e	3	c e	2	0.66
e	3	b c e	1	0.33
a c	2	a c	2	1
b c	2	b c	2	1
b c	2	b c e	1	0.5
b e	3	b e	3	1
b e	3	b c e	1	0.33
c e	2	c e	2	1
c e	2	b c e	1	0.5
b c e	1	b c e	1	1

T15:

Body item	*Head item*	*supp*	*conf*
a	a	2	1
a	a c	2	1
b	b	3	1
b	b c	2	0.66
b	b e	2	0.66
b	b c e	1	0.33
c	c	3	1
c	a c	2	
c	b c	2	0.66
c	c e	2	0.66
c	b c e	1	0.33
e	e	3	1
e	b e	2	0.66
e	c e	2	0.66
e	b c e	1	0.33
a c	a c	2	1
b c	b c	2	1
b c	b c e	1	0.5
b e	b e	3	1
b e	b c e	1	0.33
c e	c e	2	1
c e	b c e	1	0.5
b c e	b c e	1	1

T16:

Table 5.7: T15 and T16

1. In T14, *leftsupp* is the support of the rule's Body, while *rightsupp* gives the support of the frequent itemsets where the rule Body comes from. By calculating $\frac{\texttt{T14}.rightsupp}{\texttt{T14}.leftsupp}$, the confidence parameter for each tuple is generated. Then we use the *attach* α operation to set up a new attribute *conf* for each tuple, that is $\texttt{T15} = \sigma^{e}_{(conf\ >\ minconf)}\ (\ \alpha(conf,\ \frac{\texttt{T14}.rightsupp}{\texttt{T14}.leftsupp})(\texttt{T14})\ )$.

2. In T15, the attribute *rightsupp* is the *support* value of each rule. To show the support parameter for each rule shown in one tuple, we project *rightsupp* but omit *leftsupp*. $\texttt{T16} = \pi^{e}_{(\texttt{Body},\ \texttt{Head},\ supp,\ conf)}(\ \rho^{e}_{(supp \leftarrow right.supp)}(\texttt{T15})\ )$. Table 5.7 shows T15 and T16.

Candidate Set of Rules

In T15 and T16, we note the appearance of the set of redundant pairs caused by the equal join which is permitted by the *set containment join* operator. Self-combination is futile as rules in form of $A \rightarrow A$ cannot reveal connections between different items. These tuples are not of use in the rule generation phase.
There exists various ways to eliminate redundant rules. The first is to relax the join condition, by introducing addtional extended set join conditions, e.g., $\bowtie_{\subset}$ into the algebraic expressions. Anther way is to develop a column based operation which supports subtraction between set values. In T16, if an operation like Head $\leftarrow$ T16.Head $-$ T16.Body were accepted, all redundant rules would be filtered. This topic is not the focus of this chapter, a more detailed discussion is omitted at this stage.

5.5 Summary

In this chapter, association rule mining is represented as a series of algebraic expressions. From the frequency counting phase, through the frequent itemset generation phase to the rule generation phase, each calculation is shown as a sequence of algebraic expressions sustained by the extended algebra on top of the nested relational algebra defined in chapter 5. Here we list all calculations together to make a full inspection.

$AllGroup = \varpi(\mathrm{T2}.tid) = 4$

$\mathrm{T2} = \eta(\mathrm{T1}.tran\ record)$

$\mathrm{T3} = \Gamma_{tidlist \leftarrow tid}\ (\mathrm{T2})$

$\mathrm{T4} = \alpha(freq,\ \varpi(\mathrm{T3}.tidlist))\ (\mathrm{T3})$

$\mathrm{T5} = \sigma^e_{(freq\ >\ minsup)}\ (\mathrm{T4})$ $\mathrm{T6} = \Gamma_{itemset \leftarrow item}(\pi^e_{item}(\mathrm{T5}))$

$\mathrm{T7} = \wp\ (\mathrm{T6}.itemset)$

$\mathrm{T8} = \mathrm{T7}_{\mathrm{T7}.(itemset) \bowtie_{\subseteq} \mathrm{T1}.(tran\ record)}\ \mathrm{T1}$

$\mathrm{T9} = \Gamma_{tidlist \leftarrow (tid, tran\ record)}(\mathrm{T8})$

$\mathrm{T10} = \alpha(supp, \frac{\varpi(\mathrm{T9}.tidlist)(\mathrm{T9})}{AllGroup})(\mathrm{T9})$

$\mathrm{T11} = \pi^e_{itemset,\ supp}\ (\sigma^e_{supp\ >\ minsup}\ (\mathrm{T10}))$

$\mathrm{T12} = \rho^e_{(Body \leftarrow \mathtt{itemset},\ left.supp \leftarrow supp)}\ (\mathrm{T11})$

$\mathrm{T13} = \rho^e_{(Head \leftarrow \mathtt{itemset},\ right.supp \leftarrow supp)}\ (\mathrm{T11})$

$\mathrm{T14} = \mathrm{T12}\ _{\mathrm{T12.Body}\ \bowtie_{\subseteq}\ \mathrm{T13.Head}}\ \mathrm{T13}$

$\mathrm{T15} = \sigma^e_{(conf\ >\ minconf)}\ (\ \alpha(conf,\ \frac{\mathrm{T14}.rightsupp}{\mathrm{T14}.leftsupp})(\mathrm{T14})\ )$

$\mathrm{T16} = \pi^e_{(\mathtt{Body},\ \mathtt{Head},\ supp,\ conf)}(\ \rho^e_{(supp \leftarrow right.supp)}(\mathrm{T15})\ )$

Chapter 6

Conclusions and Future Work

6.1 What we have achieved

Extending current database systems in order to support ad hoc data mining queries has been the focus of many database researchers and commercial vendors. A significantly increasing amount of research has gone into tightly integrating data mining with database systems. However, in recent decades, the best techniques for extending DBMSs are still not powerful enough to "embed" data mining into database systems. Furthermore, current query languages lack the expressiveness to compose complex queries. They require re-using the derived mining results or mixing use the mining results with the original data sets.

This thesis has accomplished the following:

1 We have focused on nested relational algebra for integrating multi-step association rule mining. The introduction of nested relational algebraic operations releases Codd's 1NF restriction. This allows both database and data mining queries be processed in a similar manner.

2 We propose a sequence of algebraic operations for association rule mining. These sequences simplify the internal expressions of the mining tasks compared with existing work.

3 We study a new data mining query language *M2MQL (Mining to Mining Query Language)* based on the formal semantics sustained by the nested relational algebraic operations. We have shown that *M2MQL (Mining to Mining Query Language)* can play a major role in supporting association rule mining applications with examples of typical association rule mining queries.

4 In the *M2MQL*, we introduce *Association Rule Mining Primitives* which gives the flexibility for users to reuse mining results and conduct mining tasks bases on mining results. These primitives are expressed as a series of complex queries in *M2MQL*.

5 We show that *M2MQL* is built on SQL by adding minimal extensions. While ease of use is a great advantage, many traditional benefits of SQL are inherited by *M2MQL*.

6.2 What is ahead

6.2.1 Issues of Query Optimization

Gopalan, Nuruddin and Sucahyo [GNS02, GNS01] discuss how to interpret the internal expressions of association rules. The main difference in our approach is when and where we should execute the *powerset* $\wp$ operation. In [GNS02], $\wp$ is used inside each transaction. But in our approach, we first prune the low -

support items, then use the $\wp$ operator on the reminders of the items, not for each transaction but for whole purchase records. These two schemes may have their own advantages in different scenarios. We should analyze their aptness towards different data set schemas.

The key point here should be the time consuming of "powerset" operation. We are convinced that for different databases, the algorithm proposed in [GNS02] and our own will have different advantages.

6.2.2 Issues of Expanding Table Forms

In our approache, we introduce new operators supporting rule mining tasks. In the rule mining procedure, table forms have been frequently re-formulated. We can note that, the *powerset* operation vertically change the form of the input relation while the *attach* operation does similar work horizontally. How to set up proper data structures to represent tables and relations would be interesting future work.

Appendix A

General Syntax of M2MQL

In this section, first we show the most general formulation of the association rule mining query with M2MQL statements as follows:

```
CREATE RULE PRIMITIVE <rule_set_name> <rule-type>} (
    <column definition list>  /*design the schema of the rule_set*/
) FROM <source_data_table> WHERE <where_condition_list> GROUP BY
clause HAVING condition
```

In the description `CREATE RULE PRIMITIVE` above, `<rule_set_name>` denote the name of the generated rule base. `<rule-type>` indicate information of specific association rules what we have discussed in former chapters. It is mainly used to classify *item-based association rules* and *quantitative association rules*; also to specify whether negative rules are required to generate or not towards *item-based association rules*. Its expanded syntax are given as follows. `<column definition list>` depict the schema of the rule base to be derived which is discussed in detail with its expanded syntax in the following fragment.

```
<rule-type>  ::= <item-based>
               | quantitative
<item-based> ::= item-based
                    /*default value which indicate normal rules*/
               | item-based negative
                    /*indicate negative rules to be generated*/
<column definition list> ::=
    <BODY> <type> COVER|IS|IN
    (
        [column1_name],<column_type> {<column definition list>}
        [column2_name],<column_type> {<column definition list>}
        ...
        [columni_name],<column_type> {<column definition list>}

    ),
    <HEAD> <type> COVER|IS|IN
    (
        /*same as those in BODY shown above*/
    ),
    <SUPPORT> <operation> constant,
    <CONFIDENCE> <operation> constant,
    {<INTEREST> <operation> constant}.

<column_type> ::= SUBTABLE
                | VARCHAR
                | NUMBER
```

`<operation> ::= < | > | = | <= | >=`

Being the kernel of the CREATE RULE PRIMITIVE statement, we lay our emphasis on `<column definition list>`. The `<column definition list>` is a comma-separated list of column identifiers. Conventional association rule consists of four parts `<BODY>`, `<HEAD>`, `<SUPPORT>`, `<CONFIDENCE>`, for user specified measurement, we add an optional column `INTEREST` to add the third threshold into association rules when needed. Taking nested relational database into consideration, `<type>` following each column is to specify the data-type of related column. As shown above, obviously `<BODY>` and `<HEAD>` can be traditional components get from source data set as well as being a nested subtable even itself correlated with additional sub-subtables are also acceptable. In `<BODY>` and `<HEAD>` statements, we introduce three reserved words `COVER`, `IS` and `IN`. The appointment of these three words frees users from providing strict schema of association rules to be mined. Reserved word `COVER` requires a minimum range of rule's `<BODY>` or `<HEAD>` whereas `IN` constrains a maximal valid scope. Intuitively, `IS` fixes an exclusive `<BODY>` or `<HEAD>`.

Appendix B

Syntax and Example for MSQL

B.1 Syntax of MSQL

For an intensive comprehension of these two query language we quota its full syntax here. In the syntax, square brackets denote optionally; productions <Fromlist> and <WhereClause> denote the standard SQL clauses FROM and WHERE which are not further expanded (the keywords BODY and HEAD can be used as correlation variables in the WHERE clause for the SELECT clause and in the HAVING clause for the CLUSTER BY clauses); <TableName> and <AttributeName> denote identifiers, <Number> denotes a positive integer, <real> denotes real numbers.

```
<MSQL Stmt> ::=          <GetRules-Query>
              |          <SelectRules-Query>
              |          <Sat-Violate-SubQuery>
              |          <Encode-Stmt>

<GetRules-Query> ::=     [ Project Body, Consequent, confidence, support]
                         GetRules(C) [ as R1 ]
```

```
                    [ into< rulebase_name > ]
                    [ where < cond > ]
                    [ sql-group-by clause ]
                    [ using-clause ]

<cond>      ::=     <Rule Format Conditions RC>
              |     <Pruning Conditions PC>
              |     <Mutex Condition MC>
              |     <Stratified Subquery Conditions SSQ>
              |     <Correlated Subquery conditions CSQ>

<Rule Format Conditions RC> ::= Body { in | has | is } <descriptor-list>
                              | Consquent { in | is } <descriptor-list>

<Pruning Conditions PC> ::=  confidence <relop> <float-val in [0.0,
1.0]>
                           | support <relop> <integer>
                           | support <relop> < float-val in [0.0,1.0]>
                           | length <relop> <integer>
        <relop> ::=   { < | ≤ | = | > | ≥ }

<Mutex Condition MC> ::=  Where <other-conditions>
                          { AND | OR } mutex (method, method [, method])
                          [{ AND | OR } mutex (method, method [,
method])]
```

B.2 Example

TableB.2 corresponds to the source data encoded in the input format used by MSQL. The boolean attributes are as many as the possible items.

Step 1: selection of the subset of data to be mined

Table B.1: Source table used by MSQL

t_id	ski_pants	hiking_boots	col_shirts	brown_boots	jackets	customer_age	payment
t1	1	0	0	0	0	26	credit_card
t2	0	0	1	1	0	35	credit_card
t3	0	0	1	1	0	48	cash
t4	0	0	0	0	1	29	credit_card
t5	0	0	1	0	1	46	credit_card
t6	0	1	0	1	0	25	cash
t7	1	1	0	1	0	29	credit_card
t8	1	1	0	1	1	34	credit_card
t9	0	1	0	0	0	28	credit_card
t10	1	0	0	0	0	41	credit_card
t11	1	0	0	1	1	36	cash

We are interested only in finding out the clients paying with credit cards. MSQL requires that we have to select a subset of data to be mined before the extraction task. The relation on which we will work is supposed to have been correctly selected from a pre-existing set of data, by means of a view, named RSales.

Step 2: encoding age

MSQL provides some methods to encode the attributes. It is important to note that MSQL is able to do discretization "on the fly", so that the intermediate encoded value will not appear in the final results. The following query will encode

the age attribute.

```
CREATE ENCODING e_age ON RSales.customer_age AS BEGIN (MIN, 9, 0),
(10, 19, 1), (20, 29, 2), (30, 39, 3), (40, 49, 4), (50, 59, 5),
(60, 69,6 ), (70, MAX, 7), 0 END;
```

Step 3: Rules extraction

We want to extract rules associating with a set of items to the customer's age and having a support over 25% and a confidence over 50%..

```
GETRULES (Rsales) INTO (RSalesRuleBase)
            WHERE BODY has {(ski_pants = 1)
                 OR (hiking_boots = 1)
                 OR (col_shirt = 1) OR  (brown_boots = 1)
                 OR (jackets = 1)}
           AND Consequent is {(Age = *)}
           AND support >= 0.25
           AND confidence >= 0.5
USING e_age FOR customer_age
```

This example shows a limitation of MSQL: if the number of items is high, the number of predicates in the WHERE clause increases correspondingly.

Step 4: Crossing-over: looking for exceptions in the original data

Finally , we select tuples from RSales that violate all the extracted rules of size 3.

```
SELECT * FROM RSales WHERE VIOLATES ALL
(SELECTRULES(RSalesRuleBase) WHERE length=3)
```

Step 5: rule manipulation

We select rules with 2 items in the body. As MSQL is designed to extract rules with one item in the head and as it provides access only to the extracted rules (and not to the originating itemsets), we must specify that the total size of rule is 3.

```
SelectRules(RSalesRuleBase) where length = 3
```

Step 6: extract rules with a maximal body

It is equivalent to the situation where there are no couple of rules with the same consequent, such that the body of one rule is included in the body of the other one.

```
SELECTRULES(RSalesRuleBase) AS R1 WHERE NOT EXISTS
(SELECTRULES(RSaleRuleBase) AS R2
                    WHERE R2.body has R1.body
                    AND NOT (R2.body is R1.body)
                    AND R2.consequent is R1.consequent)
```

Appendix C

Syntax and Example for MINE RULE

C.1 syntax of MINE RULE

```
<MineRuleOp> := MINE RULE <TableName> AS
SELECT DISTINCT <BodyDescr>, <HeadDescr> [ , SUPPORT] [ ,CONFIDENCE]
[ WHERE <WhereClause>]
FROM <FromList> [ WHERE <WhereClause>]
GROUP BY <Attribute> <AttributeList>
     [ HAVING <HavingClause>]
[ CLUSTER BY <Attribute> <AttributeList>
     [ HAVING <HavingClause>] ]
EXTRACTING RULES WITH SUPPORT: <real>, CONFIDENCE: <real>

<BodyDescr> := [ <CardSpec>] <AttrName> <AttrList> AS BODY
     /* default cardinality for Body:  1..n */
```

```
<HeadDescr> := [ <CardSpec>] <AttrName> <AttrList> AS HEAD
    /* default cardinality for Head:  1..1 */
<CardSpec> := <Number> ..  (<Number | n>)
<AttributeList> := ,<AttributeName>
```

C.2 Example

Queries in MINE RULE is in form as:

```
MINE RULE SalesRuleBase AS
SELECT DISTINCE 1..n item AS BODY,
                1..1 customer\_age AS head, SUPPORT,
                CONFIDENCE
                FROM Sales
                WHERE payment = 'credit_card'
                GROUP BY t_id
                EXTRACTING RULES WITH SUPPORT: 0.25,
                                      CONFIDENCE: 0.5
```

To have an intensive comprehension for MINE RULE operator, here we cite one go though example to illustrate the whole association rule generating procedure step by step.

Problem Statement: Extracting rules that describe temporally ordered purchases, i.e., the items in the body are purchased previously than the items in the head. The temporal constraint is a condition on the clustering attributes; it is specified

by means of an optional HAVING clause associated to the CLUSTER BY clause. This predicate is used to discard couples of clusters before forming rules.
The refined problem is described as follows:

```
MINE RULE OrderedSets AS SELECT DISTINCT 1..n item AS BODY, 1..n
item AS HEAD,
                SUPPORT, CONFIDENCE
FROM Purchase GROUP BY customer CLUSTER BY date
    HAVING BODY.date < HEAD.date
EXTRACTING RULES WITH SUPPORT: 0.01, CONFIDENCE: 0.2
```

C.2.1 Counting Groups

The first step is to count the groups in the source relation. The number of groups (*AllGroup*) is necessary to evaluate the support of rules. Function *CountAllGrops* projects the source table on the grouping attributes(the list of grouping attributes is call *GBAttrList* and it contains the attributes appearing in the GROUP BY clause); then, it counts the tuples remained after the projection.

$$AllGroup =$$

$$CountAllGroups(Table) = COUNT(\pi(GBAttrList)Table)$$

C.2.2 Making Couples of Clusters

$$\texttt{Clustered} \equiv \tag{C.1}$$

$$MakeClusterPairs(Table) = \tag{C.2}$$

$$\eta(ClusterPairs) \tag{C.3}$$

$$\varepsilon(ClusterPairs;\ MakePairs(Group)) \tag{C.4}$$

$$\sigma(GroupCondition)\Gamma(GBAttrList; Group)Table \tag{C.5}$$

$$MakePairs(Group) = \tag{C.6}$$

$$\rho(BODY, HEAD; BCluster, HCluster)$$

$$(\rho(foreach\ a \in ClAttrList; BODY.a)\Gamma(ClAttrList; BODY)Group)$$

$$\bowtie [Cluster\ Condition]$$

$$\rho(foreach\ a \in ClAttrList; HEAD.a)\Gamma(ClAttrList; HEAD)Group)$$

(C.5): $\sigma(GroupCondition)\Gamma(GBAttrList; Group)Table$

(C.6): $MakePairs(Group)$

After the join, the complex attributes BODY and HEAD are renamed as *BCluster* and *HCluster*. The produced relation has the schema as:

(C.4): $\varepsilon(ClusterPairs;\ MakePairs(Group))$

(C.3): $\eta(ClusterPairs)$

C.2.3 Extracting Bodies

Relation *Bodies* contains all possible bodies contained in the clusters actually present in the source relation *Table*. We need to know the set of bodies in order to evaluate the confidence of rules.

ExtractBodies(Table, AllGroups):

$$ExtractBodies = T12 \tag{C.7}$$

$$T4 = \sigma(Groupcondition)\Gamma(GBAttrList; BGoup)Table \tag{C.8}$$

$$T5 = \eta(BGroup)T4 \tag{C.9}$$

$$T6 = \Gamma(GBAttrList, ClAttrList; BCluster)T5 \tag{C.10}$$

$$T7 = \Sigma(BCluster; \mathcal{P}(BodySet)BCluster)T6 \tag{C.11}$$

$$T8 = \eta(BCluster)T7 \tag{C.12}$$

$$T9 = \Sigma(BodySet; \pi(BSchema)BodySet) \tag{C.13}$$

$$T10 = \pi(GBAttrList, BodySet) \tag{C.14}$$

$$T11 = \Gamma(BodySet; BodyGroups) \tag{C.15}$$

$$T12 = \pi(COUNT(BodyGroups)/AllGroups > minSupport) \tag{C.16}$$

Function *ExtractBodies* proceeds as follows:

C.2.4 Extracting Rules

Function *ExtractRules* is used to extract rules. Set of sub-functions are used.

ExtractRules(Clustered, Bodies, AllGroups):

$$ExtractRules(Clustered, Bodies, AllGroups) = \texttt{Rules} \tag{C.17}$$

$$ExtractRules(Clustered, Bodies, AllGroups) \equiv \quad (C.18)$$

$$\equiv \quad AddConfidence(Bodies, \quad (C.19)$$

$$AddSupport(AllGroups, \quad (C.20)$$

$$CollectRules(DiscardTautologies(\quad (C.21)$$

$$MakeRules(MakeSubsets(Clustered)))))) \quad (C.22)$$

Expand(C.22):

$$\texttt{CoupledRules} \equiv \quad (C.23)$$

$$MakeSubsets(Clustered) \equiv$$

$$\equiv \quad \eta(PairsOfSunsets)$$

$$\pi(GBAttrList, BCLAttrList, HClAttrList, PairsOfSubsets)$$

$$\varepsilon(PairsOfSubsets; MakepairsOfSubsets(BCluster, HCluster))$$

$$Clustered$$

Expand(C.23):

$$MakepairsOfSubsets(BCluster, HCluster) \equiv$$

$$\equiv \mathcal{P}(BODY)BCluster \times \mathcal{P}(HEAD)HCluster$$

Expand(C.22)and(C.23):

$$MakeRules(CoupledSubsets) \equiv \tag{C.24}$$

$$\Sigma(HEAD; \pi(HSchema)HEAD) \tag{C.25}$$

$$\Sigma(BODY; \pi(BSchema)BODY) \tag{C.26}$$

$$\sigma(BadBH = \phi) \tag{C.27}$$

$$\varepsilon(BadBH; BODY \bowtie [\neg MiningCond]HEAD) \tag{C.28}$$

$$CoupledSubsets$$

Expand(C.22)

$$DiscardTautologies(ClusterWithRules) \equiv \tag{C.29}$$

$$\pi(GBAttrList, BODY, HEAD) \tag{C.30}$$

$$(\sigma(Taut = \phi)) \tag{C.31}$$

$$\varepsilon(Taut; \pi(CSchema)BODY \cap \pi(CSchema)HEAD) \tag{C.32}$$

$$(\sigma(CSchema = HSchema \wedge \forall a \in ClAttrList : BODY.a = HEAD.a))$$

$$ClustersWithRules \cup \tag{C.33}$$

$$(\sigma(CSchema \neq HSchema \vee \exists a \in ClAttrList : BODY.a = HEAD.a))$$

$$ClusterWithRules \tag{C.34}$$

Expand(C.22)and(C.30)

$$\texttt{RulesWithoutSupport} \equiv$$

$$CollectRules(GroupsWithRules) \equiv$$

$$\Gamma(BODY, HEAD; GroupSet)GroupWithRules$$

Expand(C.21)and(C.35)

$$\texttt{RulesWithSupport} \equiv \tag{C.35}$$

$$AddSupport(AllGroups, RulesWithoutSupport) \equiv$$

$$\pi(BODY, HEAD, SUPPORT)$$

$$\sigma(SUPPORT \geq minSupport)$$

$$\varepsilon(SUPPORT; COUNT(GroupSet)/AllGroups)$$

$$RulesWithoutSupport$$

Expand(C.20)and(C.35)

$$\texttt{Rules} \equiv \tag{C.36}$$

$$AddConfidence(Bodies, RulesWithSuppport) \equiv$$

$$\pi(BODY, HEAD, SUPPORT, CONFIDENCE)$$

$$\sigma(CONFIDENCE \geq minConf)$$

$$\varepsilon(CONFIDENCE; COUNT(GroupSet)/COUNT(BodyGroups))$$

$$(RulesWithSupport \bowtie Bodies)$$

Table C.1: Source table used by Mine Rule

t_id	customer_age	item	payment
t1	26	ski_pants	credit_card
t1	26	hiking_boots	credit_card
t2	35	col_shirts	credit_card
t2	35	brown_boots	credit_card
t3	48	col_shirts	cash
t3	48	brown_boots	cash
t4	29	jackets	creadit_card
t5	46	col_shirts	creadit_card
t5	46	jackets	creadit_card
t6	25	hiking_boots	cash
t6	25	brown_boots	cash
t7	29	ski_pants	credit_card
t7	29	hiking_boots	credit_card
t7	29	brown_boots	credit_card
t8	34	ski_pants	credit_card
t8	34	hiking_boots	credit_card
t8	34	brown_boots	credit_card
t8	34	jackets	credit_card
t9	28	hiking_boots	credit_card
t10	41	ski_pants	credit_card
t11	36	ski_pants	cash
t11	36	brown_boots	cash
t11	36	jackets	cash

Table C.2: The Purchase Table of a big store

tr.	*cust*	*item*	date	*price*	*q.ty*
1	cust1	ski_pants	12/17/95	140	1
1	cust1	hinking_boots	12/17/95	180	1
2	cust2	col_shirt	12/18/95	25	2
2	cust2	brown_boots	12/18/95	150	1
2	cust2	jackets	12/18/95	300	1
3	cust1	jackets	12/18/95	300	1
4	cust2	col_shirts	12/19/95	25	3
4	cust2	jackets	12/19/95	300	2

Table C.3: Purchase table grouped by customer and clustered by date

cust	*date*	*item*	*tr.p*	*rice*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

Table C.4: Purchase with a complex attribute: Group

customer	**Group**				
	date	*item*	*tr.*	*price*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

Table C.5: Group with renamed attributes before join

BODY.date	**BODY**			
	BODY.item	*BODY.tr.*	*BODY.price*	*BODY.q.ty*
12/17/95	ski_pants	1	140	1
	hiking_boots	1	180	1
12/18/95	col_shirts	2	25	2
	brown_boots	2	150	1
	jackets	2	300	1
12/19/95	col_shirts	4	25	3
	jackets	4	300	2

HEAD.date	**HEAD**			
	HEAD.item	*HEAD.tr.*	*HEAD.price*	*HEAD.q.ty*
12/17/95	ski_pants	1	140	1
	hiking_boots	1	180	1
12/18/95	col_shirts	2	25	2
	brown_boots	2	150	1
	jackets	2	300	1
12/19/95	col_shirts	4	25	3
	jackets	4	300	2

Table C.6: Schema of the result of C.6

BODY.date	HEAD.date	BCluster				HCluster			
		BODY.tr	BODY.item	BODY.pr	BODY.qty	HEAD.tr	HEAD.item	HEAD.pr	HEAD.qty

Table C.7: Schema of the result of (C.5):

cust	ClusterPairs								
	BODY.date	HEAD.date	BCluster				HCluster			
			BODY.tr	BODY.item	BODY.pr	BODY.qty	HEAD.tr	HEAD.item	HEAD.pr	HEAD.qty

Table C.8: Schema of the result table:`Clustered`

cust	BODY.date	HEAD.date	BCluster				HCluster			
			BODY.tr	BODY.item	BODY.pr	BODY.qty	HEAD.tr	HEAD.item	HEAD.pr	HEAD.qty

Table C.9: Purchase with a complex attribute: Group

customer	**Group**				
	date	*item*	*tr.*	*price*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

T4: Group the source table by the group attribute, in order to apply the predicate on groups, named *GroupCondition*, i.e., the `HAVING` clause associated to the `GROUP BY` clause.

Table C.10: T5: unnest complex attribute: Group

cust	*date*	*item*	*tr.p*	*rice*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
cust1	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
cust2	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

T5: Unnest the complex attribute *BGroup*, in order to obtain the source relation without invalid groups.

Table C.11: T6: Table with BCluster

cust	*date*	*BCluster*			
		item	*tr.p*	*rice*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
cust1	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
cust2	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

T6: Group again T2 by the grouping and clustering attributes, together, in order to partition valid groups into clusters, from which bodies are extracted: a new complex attribute, `BCluster`, is created.

Table C.12: BCluster produced by Powerset

BCluster			
BodySet			
item	tr.	pr	qty
ski_pants	1	140	1
hiking_boots	1	180	1
ski_pants	1	140	1
hiking_boots	1	180	1
jackets	3	300	1
col_shirts	2	25	2
brown_boots	2	150	1
jackets	2	300	1
col_shirts	2	25	2
col_shirts	2	25	2
jackets	2	300	1
brown_boots	2	150	1
jackets	2	300	1
col_shirts	2	25	2
brown_boots	2	150	1
jackets	2	300	1
col_shirts	2	25	2
jackets	2	300	1
col_shirts	2	25	2
jackets	2	300	1

Table C.13: T7:BCluster substituted by powerset

cust	*date*	*BCluster*			
		BodySet			
		item	*tr.*	*price*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
		ski_pants	1	140	1
		hiking_boots	1	180	1
cust1	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
		col_shirts	2	25	2
		col_shirts	2	25	2
		jackets	2	300	1
		brown_boots	2	150	1
		jackets	2	300	1
		col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
cust2	12/19/95	col_shirts	2	25	2
		jackets	2	300	1
		col_shirts	2	25	2
		jackets	2	300	1

Table C.14: T8:BodySet get from unnest BCluster

cust	*date*	*BodySet*			
		item	*tr.*	*price*	*q.ty*
cust1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
		ski_pants	1	140	1
		hiking_boots	1	180	1
cust1	12/18/95	jackets	3	300	1
cust2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
		col_shirts	2	25	2
		col_shirts	2	25	2
		jackets	2	300	1
		brown_boots	2	150	1
		jackets	2	300	1
		col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
cust2	12/19/95	col_shirts	2	25	2
		jackets	2	300	1
		col_shirts	2	25	2
		jackets	2	300	1

T8: Unnest complex attribute `BCluster` in order to put its internal attribute

Table C.15: T9: BodySet with *BSchema* only

cust	*date*	*BodySet*
		item
cust1	12/17/95	ski_pants
		hiking_boots
		ski_pants hiking_boots
cust1	12/18/95	jackets
cust2	12/18/95	col_shirts
		brown_boots
		jackets col_shirts
		col_shirts jackets
		brown_boots jackets
		col_shirts brown_boots jackets
cust2	12/19/95	col_shirts
		jackets
		col_shirts jackets

T9: Project *BodySet* on the attributes appearing in the body schema *BSchema* to obtain bodies. For the example used here, the body schema is simply *item*.

Table C.16: T10: cust and BodySet

cust	*BodySet*
	item
cust1	ski_pants
	hiking_boots
	ski_pants
	hiking_boots
cust1	jackets
cust2	col_shirts
	brown_boots
	jackets
	col_shirts
	col_shirts
	jackets
	brown_boots
	jackets
	col_shirts
	brown_boots
	jackets
cust2	col_shirts
	jackets
	col_shirts
	jackets

T10: The relation is projected on the grouping attributes, denoted as *GBAttrList*, and the attribute *BodySet*.

Table C.17: T11(T12:*ExtractBodies* same as T11): Groupby BodySet

BodySet	*BodyGroup*
item	*cust*
ski_pants	cust1
hiking_boots	cust1
ski_pants hiking_boots	cust1
jackets	cust1 cust2
col_shirts	cust2
brown_boots	cust2
col_shirts brown_boots	cust2
col_shirts jackets	cust2
brown_boots jackets	cust2
col_shirt brown_boots jackects	cust2

T11: The relation is grouped by *BodySet*, in order to have in each tuple only one body and in the complex attribute, *BodyGroups*, the set of groups in which the body is presented.

T12: Bodies with insufficient support are discarded according to the valued of minimum support inserted by the user.

Bibliography

[AFS89] S. Abiteboul, P. C. Fischer, and H. J. Schek. *Nested Relations and Complex Objects in Databases.* Lecture Notes in Computer Science 361. Springer-Verlag, Berlin, 1989.

[AIS93a] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. In Nick Cercone and Mas Tsuchiya, editors, *Special Issue on Learning and Discovery in Knowledge-Based Databases*, number 5(6), pages 914–925. Institute of Electrical and Electronics Engineers, Washington, U.S.A., 1993.

[AIS93b] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[AIS93c] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items inlarge databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993.

[AN01] U. Fayyad J. Bernhardt A. Netz, S. Chaudhuri. Integrating data mining with sql databases: Ole db for data mining. In *Proceedings of the 2001 ICDE International Conference of Data Engineering*, 2001.

[ASU86] A. V. Aho, R. S., and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[AU79] A. V. Aho and J. D. Ullman. Universality of data retrieval languages. In *Proceedings of the 6th Annual Symposium on Principles of Programming Languages*, pages 110–117, 1979.

[BMS03] M. Botta, R. Meo, and M. L. Sapino. Incremental execution of the mine rule operator. Technical report, University degli Studi di Torino, Italy, 2003.

[Bus01] J. V. Den. Bussche. Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions. *Theoretical Computer Science*, 254, 2001.

[CHY96] M. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. *Ieee Trans. On Knowledge And Data Engineering*, 8:866–883, 1996.

[Cod71] E. F. Codd. Relational completeness of data base sublanguage. In *Proceedings of Courant Computer Science Symposium 6 on Data Base Systems*, 1971.

[Gei02] I. Geist. A framework for data mining and kdd. In *Proceedings of 2002 ACM SAC Symposium on Applied Computing*, 2002.

[GNS01] R. P. Gopalan, R. Nuruddin, and Y. G. Sucahyo. A seamless integration of association rule mining with databse systems. Technical report, School of Computing, Curtin University of Technology, Bentley, Western Australia, June 2001.

[GNS02] R. P. Gopalan, T. Nuruddin, and Y. G. Sucahyo. Algebraic specification of association rule queries. In *Proceedings of the 2002 SPIE International Society for Optical Engineering - 4th Conference of Data Mining and Knowledge Discovery: Theory, Tools and Technology*, volume 4730, 2002.

[GS02] I. Geist and KU. Sattler. Towards data mining operators in database systems: Algebra and implementation. Technical report, Departe-

ment of Computer Science, University of Magdeburg, Dresden Germany, 2002.

[Han96] J. W. Han. *Data mining techniques.* 1996.

[HFW+96] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, 1996.

[HGN00] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[HLN99] T. Hohnson, L. Lakshmanan, and R. Ng. Towards a toolkit for data analysis and mining. In *Proceedigns of 1999 ACM SIGMOD Workshop on Research Iuuse in DMKD*, 1999.

[HM96] S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. Technical report, University of Mannheim, 1996.

[HM97] S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. In *Proceedings of the 1997 International Conference on Very Large Data Bases*, 1997.

[HS95] M. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. In *Proceedings of 1995 11th International Conference on Data Engineering*, 1995.

[HXW00] C. Zaniolo H. X. Wang. Using sql to build new aggregates and extenders for object- relational systems. In *Proceedings of the 2000 International Conference on Management of DataInternational Conference on Very Large Data Bases*, pages 166–175, 2000.

[IBM01] IBM. White paper nested relational databases. 2001.

[IM96] T. Imielinski and H. Mannila. A database perspective on kowledge discovery. *Communications of the ACM*, 39:58–64, 1996.

[IV99] T. Imielinski and A. Virmani. Msql: A query language for database mininig. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.

[IVA99] T. Imielinski, A. Virmani, and A. Abdulghani. Dmajor - application programming interface for database mining. *Data Mining and Knowledge Discovery*, 3:347–372, 1999.

[Jam01a] H. M. Jamil. Ad hoc association rule mining as SQL3 queries. In *Proceedings of the 2001 ICDM International Conference of Data Mining*, 2001.

[Jam01b] H. M. Jamil. On the equivalence of top-down and bottom-up data mining in relational databases. In *Proceedings of the 2001 DaWaK International Conference of Data Warehousing and Knowledge Discovery*, 2001.

[JLN00] T. Johnson, L. Lakshmanan, and R. Ng. The 3w model and algebra for unified data mining. *The VLDB Journal*, pages 21–32, 2000.

[Joh79] S. C. Johnson. Yacc: Yet another compiler compiler. In *UNIX Programmer's Manual*, volume 2, pages 353–387. Holt, Rinehart, and Winston, New York, NY, USA, 1979.

[Les75] M. E. Lesk. Lex: A lexical analyzer generator. Technical Report 39, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[Mak77] A. Makinouchi. A consideration on normal form of not-necessarily-normalized relation in the relational data model. In *Proceedings of 3rd International Conference on Very Large Data Bases*, pages 447–453, Tokyo, 1977.

[MC98] R. Meo and S. Ceri. An extension to sql for mining association rule. *Data Mining and Knowledge Discovery*, pages 195–224, 1998.

[ME03] R. Meo and R. Esposito. An architechture for the incremental execution of mine rule. Technical report, University degli Studi di Torino, Italy, 2003.

[MPC96] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of 22nd VLDB International Conference on Very Large Data Bases*, pages 122–133, 1996.

[MS03] R. Meo and M. L. Sapino. An extended relational calculus for data mininig. Technical report, University degli Studi di Torino, Italy, 2003.

[MWZ00] T. Morzy, M. Wojciechowski, and M. Zakrzewicz. Data mining support in database management systems. In *Proceedings of the 2000 DaWaK International Conference of Data Warehousing and Knowledge Discovery*, 2000.

[Ran02] R. Rantzau. Frequent itemset discovery with sql using universal quantification. In *Proceedings of DTDM Workshop on Database Technology for Data Mining*, 2002.

[RCIC99] K. Rajamani, A. Cox, B. Iyer, and A. Chadha. Efficient mining for association rules with relational database systems. In *Proceedings of 1999 International Databse Engineerin and Application Syposium*, 1999.

[RG03] R. Ramakrishnan and J. Gehrke. *Database Management Systems.* Mc Graw Hill, 2003.

[Rot88] M. A. Roth. Extende algebra and calculus for nested relational database. *ACM Transactions on Database Systems*, 13(4), 1988.

[RPN00] K. Ramasamy, J. M. Patel, and J. F. Naughton. Set containment joins: The good, the bad and the ugly. In *Proceedings of 26th VLDB International Conference on Very Large Data Bases*, 2000.

[RSMW03] R. Rantzau, L. D. Shapiro, B. Mitschang, and Q. Wang. Algorthms and applications for universal quantification in relatioanl databases. *Information Systems*, 28, 2003.

[Sch] M. H. Scholl. Extensions to the relational data model. In *In P. Loucopoulos and R. Zicari, editors, Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development.*

[SS87] M. H. Scholl and H. J. Schek. Theory and applications of nested relations and complex objects. In *Proceedings of the 1987 International Workshop: Workshop Material*, Darmstadt, West Germany, 1987.

[STA98] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 343–354, 1998.

[SZZA01] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. A sequential pattern qyuery language for supporting instant data mining for e-services. In *Proceedings of the 2001 International Conference on Management of DataInternational Conference on Very Large Data Bases*, 2001.

[TS98] S. Thomas and S. Sarawagi. Mining generalized association rules and sequential patterns using SQL queries. In *Proceedings of the 4th International Conference on KDD and DM Knowledge Discovery and Data Mining*, pages 344–348, 1998.

[TUA+98] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association-rule mining. In *Proceeding of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1998.

[url03] www.planetmath.org, 2003.

[WZ00] H. X. Wang and C. Zaniolo. User defined aggregates in object-relational systems. In *Proceedings of the 2000 ICDE International Conference on Data Engineering*, pages 135–144, 2000.

[ZZ02] C. Q. Zhang and S. C. Zhang. *Association Rule Mining - Models and Algorithms.* 2002.