Schema Compilation and Query Evaluation in Intension Mining Framework

A thesis
submitted in partial fulfillment
of the requirements for the degree
of

Master of Technology
in
Computer Science & Engineering

by

Ajay Kumar Dhamija
2001MCS038

Under the Guidance of

Dr. SK Gupta

Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

December 2002

Acknowledgement

I am grateful to my guide Dr. SK Gupta for his persistent guidance and constructive criticism , which has enabled me to complete this project. I would like to thank Data Mining Group , working with my guide , for pruductive discussions and valuable inputs in improving my code.

I am indebted to my parents ,my wife and my son for continuous source of inspiration , specially at difficult times , during my M.Tech.

I also acknowledge the special support of my friend Vijay Kr. Yadav , without whose continuous help , the study and this project at IIT , Delhi wouldn't have been possible.

AK Dhamija

Certificate

This is to certify that this project titled **Schema Compilation and Query Evaluation in Intension Mining Framework** submitted by **Ajay Kumar Dhamija** towards partial completion of the requirements for the degree of Master of Technology in Computer Science and Engineering is a record of bona-fide work done by him under my supervision.

This work has not been submitted to any other institute or university for award of any degree or diploma.

Dr. SK Gupta

Abstract

Conventional data mining approaches are based on decoupled architecure and assume the data to be already selected , cleaned and transformed. Dr. SK Gupta has proposed a complete framework for a new approach called Intension Mining which emphasises the integration of Knowledge Discovery with Database Management.

Intension Mining ia a logical extension of incremental algorithm in which user requirements are specified beforehand and saved in the form of *schema* .As per specified requirements , the knowledge is accumulated as the database grows and used for actual mining.This has the advantage of tight coupling of pre-mining, mining and post-mining operations. Intensional algorithm also allows sharing of schema among different users and maximum utilisation of previously mined knowledge , while mining the incremental database.

In this thesis we have defined an SQL like interface for schema definition and query specification so as to communicate with the Intension Mining system and designed and implemented the parser for this.

Contents

List of Figures

Chapter 1

Introduction

A brief Introduction [10] [11] to the concepts used in the project ie. Data Mining , Intension Mining and Parsing is given in following sections.

1.1 Data Mining

Data Mining is a process of non-trivial extraction of implicit,previously unknown and potentially useful information in databases [3]. This may lead to extraction of high level information and interesting regularities in the database.

1.1.1 Data Mining Techniques

Association Rule

Associations are items that occur together in a given event or record [4].Association Tools discover rules of the form: If Item A is part of an event , the x percent of the time (the confidence factor) Item B is part of that event.

Classification

Classificfation assigns instances to a group or class by calculating the value of a categorical variable. A classification model examines a collection of cases for which the group they belong to is already known. It then uses the data to inductively determine the pattern of attributes or characteristics that identifies the group to which the case belongs.This pattern can be used to understand the existing data as well as predict how new instances will be classified.

Clustering

Clustering segments a database into different classes. The goal is to find classes that differ from one another as well as similarities among members.Unlike classification we don't know what the clusters will be when we start , or around which attributes the data will cluster.Consequently , a buisness analyst must interpret the meaning of the clusters.

Regression

Regression uses series of existing values and their attributes to forecast what continuous values will be.

1.2 Intension Mining

Till recently most of the research in the area of KDD had been focussed on Data Mining Techniques and has resulted in development of several useful algorithms to discover different types of knowledge from various types of databases.It is only recently that there has been a shift from algorithm development to more holistic approach

1.2.1 Introduction

The idea behind "Intension Mining" is that quite often the mining requirements can be envisaged earlier than the time of actual mining.This separates the actual mining from the specification of mining requirements to make mining more efficient and structured. The idea was proposed by Dr. SK Gupta and the intension mining architecure [1] has been framed by Ms. Vasudha Bhatnagar , a Research scholor working under Dr. SK Gupta.This is inspired by incremental mining and driven by user's requirements completely.

In conventional data mining approaches [5], the user has to select the data in the database to be mined. This selection is based on mining objectives. The existing mining algorithms operate on clean and formatted data. But real life data may not be clean and NULL values need to be treated appropriately.Thus every mining algorithm requires supporting algorithms for pre-mining and post-mining operations.

The disconnectedness in the various steps of knowledge Discovery has been the chief motivating factor for evolving comprehensive architecure framework for KDD process. Intension mining is a complete platform for data selection , cleaning and transformation along with mining and further analysing and presentation capabilities.This insulates the user from intricacies of knowledge Discovery process and give him a single DBMS like interface to mine interactively.This concept is inspired by DBMS systems , where the database architect stores the logical data requirements of an organisation in the form of a schema or Database Intension. The user can plan his requirements beforehand and input them in the form of knowledge Discovery Schema. The Intension Minking system mines the knowledge and presents the results in required format.This structured approach leads to efficiency and makes the whole process user friendly.

1.2.2 Phases of Intension Mining

Intension Mining is done in three phases (See Figure 1.1 on page 7)

1. Planning Phase
2. Accumulation Phase
3. Mining Phase

Planning Phase

The user anticipates mining requirements and specifies them in the form of knowledge Discovery Schema. The type of knowledge to be mined , the database to be mined , the transformation to be done , cleaning procedure to be used , the specific mining algorithm to be run and finally the presentation tools to be used are specified by the user.The main issue at this level is to design comprehensive Knowledge Discovery Schema Language (KDSL) to communicate the mining requirements to the KDD process.

Accumulation Phase

Directed by the KD Schema , this phase performs all the pre-mining functions for selection , cleaning and transformation on the growing database in an incremental manner [6], Knowledge is accumulated from the growing database in the form of Knowledge Concentrates (KC) , which are used in Mining phase. In this phase there is absolutely no interaction with the user. Accumulation is performed automatically and periodically (frequency is supplied by the user in the knowledge discovery schema) till the user drops the schema.The issues at this level are the interaction between the databsse and the knowledge discovery process to access the data records and the maintenance of knowledge concentrates on the disk.

Mining Phase

This phase is entered when the user invokes a mining query. the user specifies the mining parameters at this stage. the mining process consults the knowledge discovery schema and performs mining by executing specified algorithms on the knowledge concentrates accumulated during the accumulation phase and presents the results in a pre-desired format using presentation tools. The issues at this level are the design of Intension Mining Query Language (IMQL) and the usage of appropriate presentation tools to communicate the discovered knowledge.

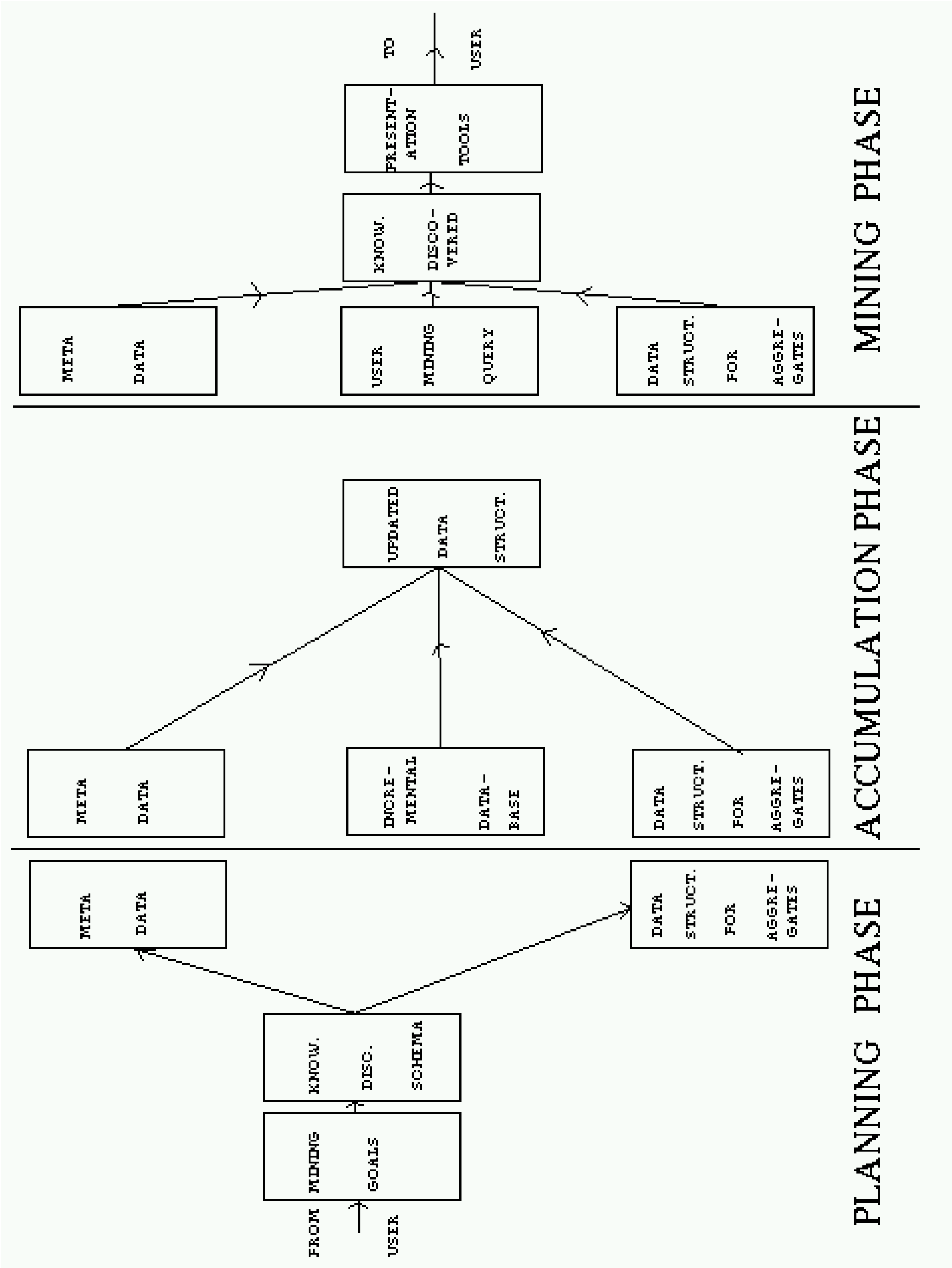

Figure 1.1: Phases of Intension Mining

1.2.3 Intension Mining vs. Incremental Mining

Though Intension Mining strongly depends on Incremental Mining,it differs from it.In incremental mining , the discovered knowledge is maintained and is updated every time incremental knowledge is discovered from the incremental database. The user is transparent to this updation. Subsequently the updated knowledge becomes the discovered knowledge for next iteration. This approach reduces database scans in sitatiuons where the mining exercise is undertaken frequently on the same database. However it doesn't encourage repeated mining , with varying parameters.Whereas experimentation and monitoring is possible in case of intension mining.For example in association rule discovery , "support" can be used to filter the large item sets that are stored for future use in mining.

1.2.4 Intension Mining vs. Data Warehouse

Owing to the rigid architecture , the data warehouse supports a limited variety of operations. However the Intension Mining system is more flexible as the user is free to design different types of functions to satisfy the special mining requirements.

1.2.5 Benefits of Intension Mining

Generic in nature

The proposed scheme is scalable , general enough to cover all types of applications and remains stable with the growth of the database and the organisation.After the initial time lag , during which the accumulation of the existing database takes place . mining , experimenting or monitoring can be performed as desired. Obsolete mining requirements can be removed from the system by dropping the schema.

Reduced Search Space

As the attributes of interest are specified in the planning phase itself , the volume of data to be scanned reduces by a huge factor.

Database Scan Reduction

Knowledge Concentrates are used for Mining purposes , thus avoiding the scan of the database.Hence , planning the mining requirements beforehand shifts the focus of data mining from being I/O inrensive to computation intensive. Mining process becomes faster by reading the reduced data in a specified form .

Tight Coupling of pre-mining operations

By defining all pre mining operations at the schema level , the schema precludes the possibility of any human error in these operations.

Sharing of schema among different users

Ths schema driven approach enables different users to share the schema and the knowledge discoverd in further knowledge discovery process.

Integration with databases

The current SQL interface is too slow to be of any practical use in mining operations. Ths proposed architecture motivates work towards development of integration of database systems and knowledge Discovery systems.

Independent of data source

The schema doesn't take into account the architecure of the data repository.

Clean and Structured approach

Intension Mining is a complete platform for data selection , cleaning , transformation along with the mining and presentation capabilities . This insulates the user from the intricacies of Knowledge Discovery Process.

Concurrent operations

Knowledge Discovery operations are independet of Database Operations . Hence both the operations can by done concurrently.

No need of sampling

Since partial analysis cum aggregation is performed off-line , processing the exhaustive dataset is less expensive. This does away with the need to sampling .

1.3 Lexical Analysis and Parsing

1.3.1 Lexical Analysis

The Lexical Analyser [9] takes a stream of characters and produces a stream of names , keywords , and punctuation marks ; it discards white space and comments between the tokens. A lexical token is a sequence of characters that can be treated as a unit in the grammer of a programming language .

Regular Expressions

Using regular expression rules [9] , we can specify the lexical tokens of a programming language. To resolve the ambiguties in the rules , we use two kinds of disambiguation rules.

1. **Longest match**The longest initial substring of the input that can match any regular expression is taken as the next token.
2. **Rule Priority**For a particular longest initial substring , the first regular expression that can match , determines its token type. This means that the order of writing down the regular expressions rules has significance.

Automata

Regular Expressions are convenient for specofying lexical tokens , but we need a formalism that can be implemented as a computer program. A conversion algorithm first turns each regular expressions into an NFA(Non Determininstic Finite Automaton) with a start edge and ending states.We can define the translation of regular expressions to NFAs by induction. Either an expression is primitive (a single symbol or ϵ) or it is made of smaller expressions. Similarly , the NFA will be primitive or made by combining smaller NFAs. Implementing DFA (Deterministic Finite Automaton) as computer program is easy , but implementing NFA is a bit harder, since most computer don't have good "guessing" hardware . We make a DFA from the NFA , such that each set of NFA states corresponds to one DFA state. Since the NFA has a finite number n of states , the DFA will also have a finite number (at most 2^n) of states.DFA construction is a mechanical task easily performed by computer , so it makes sense to have automatic *lexical analyzer generator like "Flex"* to translate regular expressions into a DFA.

1.3.2 Parsing

The abbreviations mechanism in "Flex" is very convenient , but it is impossible for a finite automaton to recognize balanced parantheses (because a machine with N states can not remember a parenthesis-nesting depth greater than 'N'),so sum and expr as defined below can not be regular expressions.

```
digits=[0-9]+
sum = expr ''+'' expr
expr = ''('' sum '')'' | digits
```

"Flex" manage to implement regular expression abbreviations such as *digits*, because right-hand-side *([0-9]+)* is simply substituted for *digits* wherever it appears in regular expressions , before translation to a finite automaton.This is not possible for *sum-and-expr* language. So the notion of abbreviation does not add expressive power to the language of regular expressions , unless coupled with recursion. Therefore , We use CFG (Context Free Grammer) to specify the language.

CFG

A context-free grammer has a set of productions of the form

```
symbol -> symbol symbol ... symbol
```

where there are zero or more symbols on the right hand side. Each symbol is either a *terminal* or *nonterminal*. No token can ever appear on the left hand side of production.One of the nonterminal is distinguished as start symbol.

Predictive Parsing and LL(1) Grammer

Recursive-descent or predictive parsing works only on grammers where the first terminal symbol of each subexpression provides enough information to choose which production to use.If we can choose the right production for each (X,T) , where X is non terminal ad T is next token, then we can write the recursive-descent parser. All the information we need can be encoded as a two-dimensional table of productions , indexed by nonterminals X and terminals T. This forms the predictive parsing table.Grammers whose predictive parsing tables contain no duplicate entries are called LL(1). This stands for *Left-to-right* , *Leftmost-derivation* , *l-symbol lookahead.*

LL(k) grammer

We can generalise this notion to describe the first k tokens of a string , and to make an LL(k) parsing table whose rows are nonterminals and columns are every sequence of k terminals.

Eliminating Left Recursion

The two pruductions

```
E -> E + T
E -> T
```

are certain to cause duplicate entries in LL(1) parsing table. The problem is that "E" appears as the first right-hand-side symbol in an E-production;this is called *left-recursion.*Grammers with left recursion can not be LL(1).To eliminate left recursion , we can rewrite this using right recursion.

```
E -> T E'
E' ->  + T E'
E' ->
```

LR Parsing

The weakness of LL(k) parsing techniques is that they must predict which production to use , having seen only the first k tokens of the right-hand side. A more powerful technique , LR(k) parsing , is able to postpone the decision until it has seen input tokens corresponding to the entire right-hand side of the production in question (and k more input tokens beyond). LR(k) stands for *Left-to-right parse* , *Rightmost-derivation,k-token lookahead.*

The parser has a stack and an input. The first k tokens of the input are the lookahead , the parser performs two kinds of actions.

Shift:move the first input token to the top of the stack

Reduce: Choose a grammer rule X → A B C;pop C,B,A from stack , push X onto stack.

The decision whether to use shift or reduce is taken by using a DFA! The DFA is not applied to the input - finite automata are too weak to parse CFG - but to the stack [9].

An LR(k) parser uses the contents of its stack and the next k tokens of the input to decide which action to take.In practice K 1 is not used for compilation. This is partly because the tables would be huge , but more because most reasonable programming languages can be described by LR(1) grammers.LR(0) grammers are those that can be parsed looking only at the stack , making shift/reduce decisions without any lookahead. This is too weak to be useful.An LR(1) item consists of a *grammer production , a right-hand-side position*(represented by dot) and a *lookahead symbol.*

LALR(1)

LR(1) parsing tables can be very large , with many states. A smaller table can be made by merging any two states whose items are identical except lookahead sets. The resulting parser is called an LALR(1) parser , for Look-Ahead LR(1). For some grammers , the LALR(1) table contains conflicts (shift-reduce or reduce-reduce) where the LR(1) table has none , but in practice the diference matters little. LALR(1) parsing table requires less memory to represent than the LR(1) table .

The task of constructing LR(1) or LALR(1) parsing tables is simple enough to be automated. Linux Tool *Yacc* reads the grammar specification in the file and generates an LR(1) parser for it. The parsers consist of a set of LALR(1) parsing tables and a driver routine written in the C programming language.

Conflicts

Yacc reports shift-reduce and reduce-reduce conflicts. A *shift-reduce conflict* is a choice between shifting and reducing; a *reduce-reduce conflict* is a choice of reducing by two different rules.By default *Yacc* resolves *shift-reduce conflict* by shifting , and *reduce-reduce conflict'* by using the rule that appears earlier in the grammer.

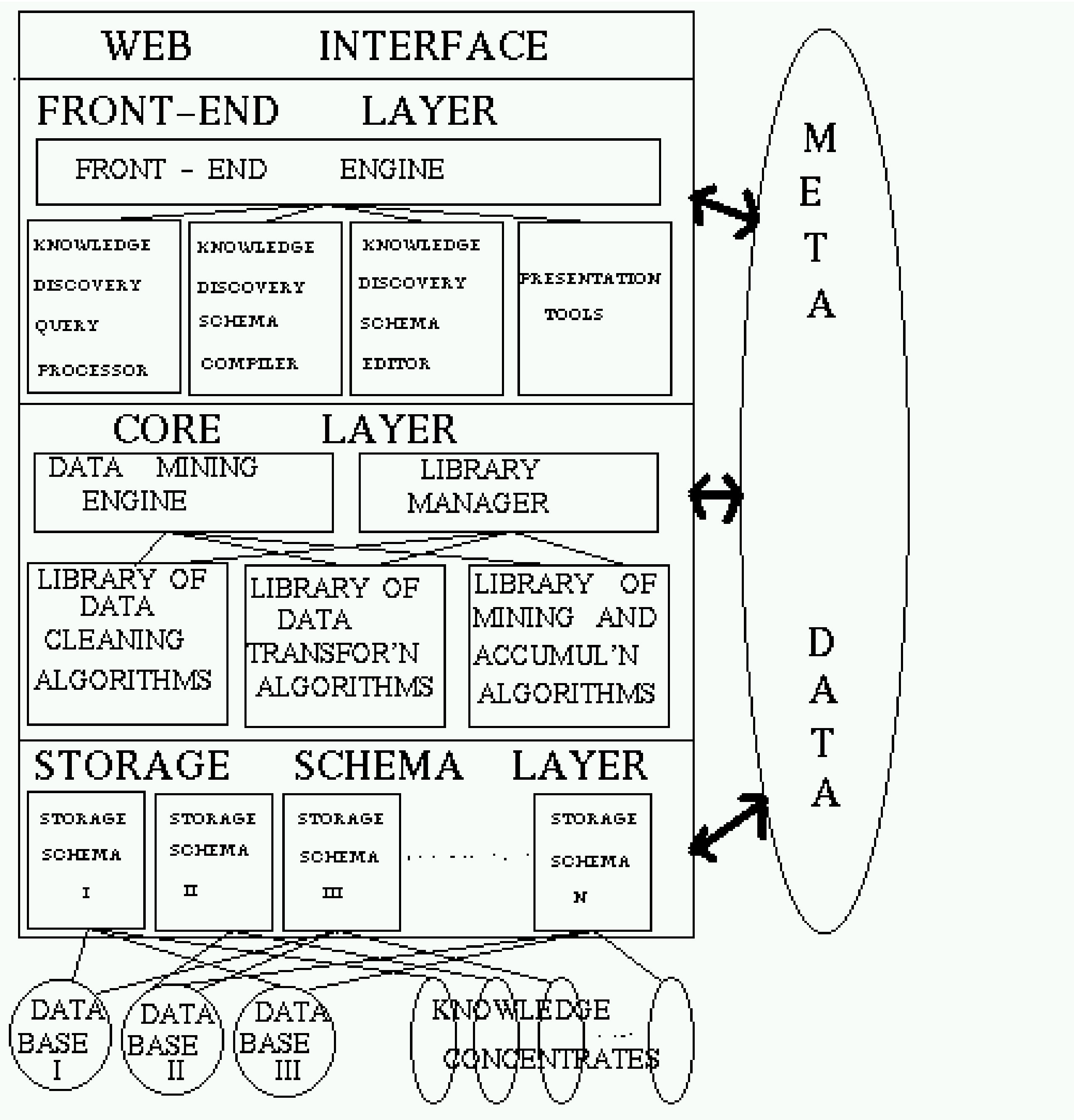

Figure 1.2: Architecure of Intension Mining System

Chapter 2

Problem Definintion

The architecture of the Intension Mining System [1] [2] in Figure 1.2 on page 14.The front end layer [2] in the architecure of Intension Mining system provides the user with the interface to the Knowledge Discovery Process. It typically consists of Knowledge Discovery Query Processor (KDQP) , Knowledge Discovery Schema Editor (KDSE) , Knowledge Discovery Schema Compiler (KDSC) and the presentation Tools. All these componets are coordinated by the front-end Engine (FEE). This layer is responsible for accepting all the user requirements and displaying the messages from the system.Front end layer provides communication facilities , which are used by the knowledge Discovery Schema Compiler for connecting to the destination DBMS (using ODBC etc.) Since FEE does all the communication with the lower layers , installation of the user defined functions in the library is performed by FEE on behalf of compiler. At the mining stage , when the mining process communicates the results , FEE receives the outputs of the mining algorithm and invokes the appropriate presentation / visualisation tool to present the knowledge in desired form. It also manages the library of presentation tools present in Front-end layer. In order to perform all these activities ,FEE maintains a database of its own operations , which contains book-keeping information about schema entries , library contets etc.

KDQP parses the query received form the user , as per grammer specified by Intension Mining Query Language (IMQL) and generates the execution plan and strores it into the external file for further reference in later phases of Intension Mining.Similarly KDSC parses the schema definition received from the user , as per grammer specified by Knowledge Definition Schema Language (KDSL) and creates the metadata , history file and the required directory structre.Metadata contains all the required information , which is required in the later phases of Accumulation and Mining. KDSE facilitates the definition of schema and query through a visual user interface and presentation tools enable to view and analyse the results of query.The main issues at this level are the design of a Knowledge Discovery Schema Language

(KDSL) and the Intension Mining Query Language (IMQL).Accumulation process is started , creates the Knowledge Concentrates (KC) at the frequency specified in the schema.These KCs contain the incremental data.

The Core layer contains the library of Data Cleaning Functions , Data Transformation Functions and the library of mining and accumulation algorithms.This also contains a library manager to manage these libraries and maintain the indexes of these library functions for faster access.Data Mining Engine interacts Meta Data , schema stored and performs actual mining after applying the merging operator on the KCs (already selected on the basis of query definition).

The Storage Schema Layer maintains the stores schemas and the KCs .Metadata maintains the compiled entry along with indexes of pre-mining functions and other mining algorithms library functions.

Comprehensive Implementation of the Intension Mining framework is envisaged by Dr. SK Gupta and the Data Mining Group at Computer Science Dapartment , IIT Delhi and different tasks are assigned to different students and the integration aspects are being handled by NSIT Students (Gauhar and Prenayan). The task of develpment of intension mining algorithms is given to Padmaja,Mahesh et al.I have been assigned the task of Schema Compilation and Query Evaluation in Intension Mining Framework.KDSL and IMQL have already been specified by Ms Vasudha Bhatnagar et al. [1][2] These have been adopted with minor modifications.

KDSL handles the framing of user mining requirements and communicating the same to the system in the form of Knowledge Discovery Schema (KD Schema). This , then becomes input to the KDS Compiler . The compiler validates the details specified by the user and then stores the requirements in the Schema-base in the form of KD Schema , Which can be accessed later in other phases.This schema contains all the relevent details required for mining like Database to be used , Mining knowledge type and the algorithm to be used , pointers to pre-mining functions and other parameters required at the time of mining. IMQL handles the specification of user query. KDQP then parses this query and generates the execution plan , which contains the schema to be used and other run time parameters required at the time of query execution.

In this thesis , I addressed the problem of specification of Kowledge Discovery Schema through Knowledge Discovery Schema Language (KDSL) and Knowledge Discovery Schema Compiler (KDSC) to compile this language.Besides this I also addresssed the specification of query through Intension Mining Query Language (IMQL) and Knowledge Discovery Query Processor (KDSP) to parse this. After that two Intensional Algorithms namely *cactus* and *iceberg* were to be integrated to give Integrated Intension Mining System. Finally Client and Server sockets were to be created to communicate the messages (receiving schema and query definitions , sending valdation query for validating database , relations and attributes , pre-mining functions and mining algorithm and receiving responses to these validation queries) with other layers.A log was required to be generated everytime the KD Schema is accessed.

Chapter 3

KDSL and Schema Compiler

3.1 Knowledge Discovery Schema Language

3.1.1 Introduction

KDSL should be rich enough to capture all the user requirements along with the focussing information. Data Mining covers a wide spectrum of tasks covering data clustering , mining association rules and data classification. This makes the design of KDSL a fairly challenging task [7].

3.1.2 Design Considerations

1. The set of relevent data should be specified in KD Schema.
   User must first select the specific subset of data , which he is interested in. If he can not identify precisely the relevant set of data , a superset of the data can be collected and certain mechanisms can be developed to help identify or rank the relevent set of attributes.

2. The kind of knowledge to be mined should be specified in schema.
   Since mining can be performed in many different ways on any specific data , huge amounts and different kinds of knowledge may be generated by unguided dicovery , much of which may not be of user's interest.Therefore along with specifying relevant set of data , we also specify the kinds of knowledge to be dicovered. This leads to guided discovery of desired kind of knowledge on a relevent set of data and represents constrained search for the desired knowledge.Along with it specific mining algorithm needs to be specified.

3. Background knowledge such as hierarchy innformation may be specified to improve the efficiency of disvovery process.

4. Selection , cleaning and transformation functions' information may be specified.

User should provide the selection information to select the subset of data and cleaning information if he desires to deal with noisy or missing data. In addition if the user wants the data to be in a specific format before mining , he should be able to specify the transformation information.

5. Other sundry paramaters like thresholds (confidence , support , number of clusters , number of classes etc) ,periodicity , priority , start time etc. can be specified flexibly.

3.1.3 Syntax of KDSL

KDSL [1] adopts a SQL like syntax to facilitate high level data mining and integration with the relational query language SQL. KDSL has been adopted from Ms Vasudha Bhatnagar's work [2] [1] with minor modifications. KDSL is defined in extended BNF grammer, where "[]" represents 0 or one occurence , "{}" represents 0 or more occurrences and "{{}}" represents 1 or more occurrences.

$< KDSL >$::
Schema Name is$< schema\_name >$
Accumulate $< knowledge\_type >$
Use Database $< database\_name >$
{**Use hierarchy**$< hierarchy\_name >$ **for** $< attribute >$}
related to {{$< attribute\_list >$ **from**$< relation >$}}
[**Use selection** $< selection\_function >$]
[**Use cleaning** $< cleaning\_function >$]
[**Use transformation** $< transformation\_function >$]
Use Mining $< mining\_algorithm >$ **with** $< algorithm\_parameters >$
[**With parameters** {$< kind\_of > = < parameter\_value >$}]

In $< KDSL >$,"**Schema Name is**$< schema\_name >$" is to specify the name of the schema . It can be any variable name.

The statement "**Accumulate** $< knowledge\_type >$" is specification of the kind of knowledge to be discovered. It can be either classification , clustering or association rules.

The statement "**Use Database** $< database\_name >$" directs the mining process to a specific database "$< database\_name >$", and the optional statement "{**Use hierarchy**$< hierarchy\_name >$ **for** $< attribute >$}", assigns "$< hierarchy\_name >$" hierarchy to attribute "$< attribute >$".

The statement "**related to** {{$< attribute\_list >$ **from**$< relation >$}}" is modified version of adopted KDSL and selects list of relevant attributes "$< attribute\_list >$" for mining from multiple relations mentioned in "**from**$<$

$relation >$" from the selected database.

The optional statement "**Use selection** $< selection\_function >$" , "**Use cleaning** $< cleaning\_function >$" and "**Use transformation** $< transformation\_function >$]" specify the selection function , cleaning function and the transformation if at all required by the user.

The statement "**Use Mining** $< mining\_algorithm >$ **with** $< algorithm\_parameters >$" is used to specify the algorithm used to mine the knowledge. It should correspond to the "$< knowledge\_type >$" defined earlier.

The statement "**With parameters** $\{< kind\_of > = < parameter\_value >\}$" is modified version of adopted KDSL and specifies various kinds of parameters like priority , periodicity , start_time etc. and the knowledge dependent parameters like confidence , support , no of clusters etc.

An example of knowledge Discovery schema defined using KDSL to mine association rules from Officers Selection Data is given as follows.

```
Schema Name is Officers
accumulate association rules
use database Selection_data01
related to grade , IO_marks , psych_marks, GTO_marks from gpsheet
use selection selection_function01
use cleaning cleaning_function01
use transformation trasformation_function01
use mining cactus
with priority parameter = 1,periodicity parameter=2000,start_time parameter=0700
```

3.2 Knowledge Discovery Schema Compiler

The KD compiler takes the knowledge discovery schema definintion (defined in KDSL) as input , validates the information given by the user and generates the compiled schema as metadata and creates directory structure and the history file [2].

3.2.1 Compilation process

The compilation process consists of following aspects.

Access on the specified database schema

For validation of the database , an understanding is required between Intension Mining system and the Database management system.The validation of database may require communication over the network if the database is located at a different machine. The compiler has to establish the connection with the DBMS.This is done through front-end engine [2].The compiler scecks whether we have got the access to read the database schema of the specified database.Permission to peek into the database schema should already have been granted to the compiler by the database.These give rise to resolution of issues like network security and database security.

Validation of relation name and attributes list

The compiler reads the database schema and checks whether the relations and attributes specified in the schema actually exist in the database schema.If they do not exist , an error is reported.If the attribute is specified as * , the compiler reads all the attributes of the specified table form the database schema and stores them.

Validation of hierarchy information

The compiler checks whether the hierarchy information specified in the schema actually exist in the database schema.If it does not exist , an error is reported.

Getting authorisation form DBMS for accumulation

The compiler seeks authorisation from DBMS for data access layer required by the accumulation process.The compiler creates metadata only if the user gets authorization from the DBMS.

Noting User parameters

Compiler writes various user parameters into the metadata.

Finally the necassary data structures are created by the compiler and the compiled entry(metadata) is stored.This metadata has all the information supplied by the user in addition to the information generated for internal usage by accumulation and mining processes.

3.2.2 Functional Design

Functionality

1. **Socket programming** Client sockets are required to access the server sockets for getting the schema definition in the form of messages.Server sockets are required to send messages used for validation of database,algorithms and pre-mining functions etc.

2. **Grammer Specification** Grammer needs to be specified for query schema definition and compilation to be used by Linux tools *flex* and *yacc*.

3. **Integration with the intensional algorithm** Grammer specification should facilitate the integration of schema definition with the intensional algorithm.

4. **Creating Directory Structure** After compilation , appropriate directory structure need to be gnerated , which could be used in later phases like accumulation and mining phases.The directory structure is given as follows

```
<home-directory>             -> home directory name
---<IMIN>                    -> Intension Mining root directory
------<schema>               -> schema directory for schema details
------query.log              -> file for logging during query parsing
------<schema8>              -> direcory for schema named as schema8
---------<KC>                -> directory containing KCs for schema8
------------kcs_file         -> file contains extracted KCs for schema8
------------other KC files   -> all the KC files
---------<output>            -> directory containing outputs for schema8
---------<query>             -> directory containing query def for schema8
------------query.txt        -> file contains query def in text form
------------query.sav        -> file contains query def as system file
---------<temp>              -> temp directory for schema8
---------schema8.sch         -> file contains schema def for schema8
---------schems8.algo        -> algorithmic specific file for schema8
---------schema8.history     -> history file for schema8
------schema_base            -> system file to store all schemas
------schema.log             -> file for logging schema compilation
```

5. **Creation of Metadata** After compilation , the compiled entry (metadata) need to be saved ,which could be used in later phases like accumulation and mining phases.

6. **Error Handling** Parsing Errors and other program errors are required to be handled systematicaly with error codes clearly and separately defined.

7. **Creating Logs** Logs need to be generated everytime schema is accessed in schema compilation .

Modules and Pseudocode

See Structure Chart Figure 3.1 on page 23

1. **main:** This is the main routine , which starts the program
   Pseudocode:

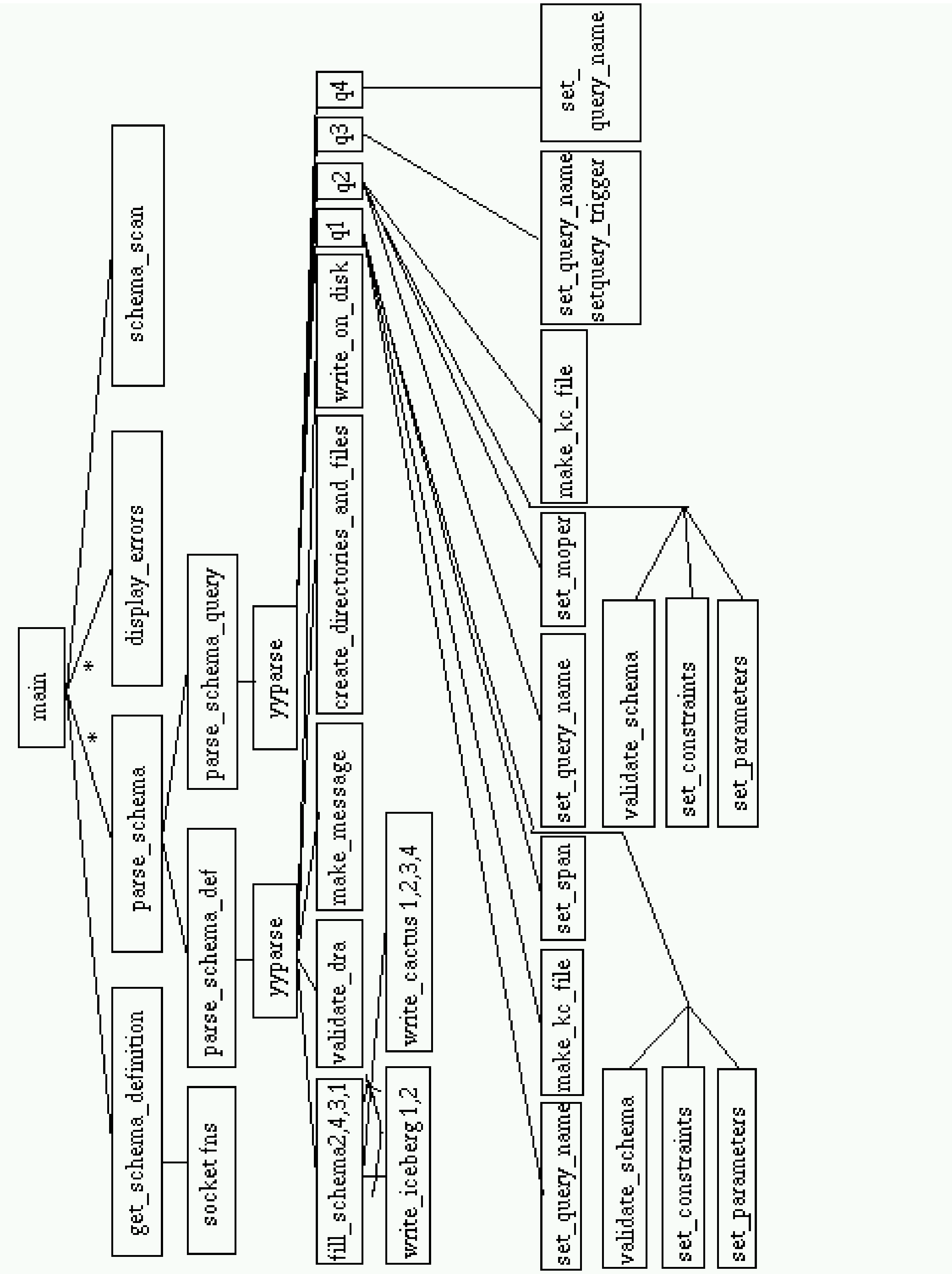

Figure 3.1: Structure Chart

```
begin
  call get_schema_definition
  call parse_schema
  call display_errors
  if required call drop_schema
  if reuired call  display_schema
end
```

2. **get_schema_definition:** This gets the schema definition from server through sockets
   Pseudocode:

```
begin
  open client socket
  connect to server socket
  receive string as schema definition
end
```

3. **parse_schema:** This parses the schema definition and sets error values
   Pseudocode:

```
begin
  call parse_schema_def
  set errror values
end
```

4. **parse_schema_def:** This calls yyparse (creted by *yacc*) to parse the definition and then sets error values
   Pseudocode:

```
begin
  call yyparse
  set errror values
end
```

5. **yyparse:** This parses the schema definition token by token and performs desired actions
   Pseudocode:

```
begin
  get next token from lexer
  parse the token to specified grammer
  call fill_schema2
```

```
  call fill_schema4
  call fill_schema3
  call fill_schema5
  call fill_schema1
  call validate_dra
  call make_message
  call create_directories_and_files
  call write_on_disk
end
```

6. **fill_schema_2:** This caters for hierarchy information if any.
   Pseudocode:

```
begin
   fill the hierarchy name into schema structure
   fill the corresponding attribute
end
```

7. **fill_schema_4:** This fills attribute name for a relation.
   Pseudocode:

```
begin
   fill attribute name  into schema structure
   fill number of attributes
end
```

8. **fill_schema_5:** This fills relation name .
   Pseudocode:

```
begin
   fill the relation name  into schema structure
   fill number of total attributes in that relation
end
```

9. **fill_schema_5:** This fills parameters' names and values .
   Pseudocode:

```
begin
   fill the paremeters' name  into schema structure
   fill value of that parameter
end
```

10. **fill_schema_1:** This fills schema name ad pre-mining functions names and other parameters.
    Pseudocode:

```
begin
   fill the schema name into schema strcture
   fill the knowledge type parameter into schema strcture
   fill the database name into schema strcture
   fill the selection function name into schema strcture
   fill the cleaning function name into schema strcture
   fill the transformation function name into schema strcture
   fill the algorithm  name into schema strcture
   fill the algorithm  name into schema strcture
   write algorithm specific data into external file
   set values for total no of relations, parameters and hierarchies
   call make_message_dra
   call make_messgae_fns
end
```

11. **make_message_dra:** This prapares the message to be sent for validation of database , relation and attributes.
Pseudocode:

```
begin
  fill message_dra with database_name , relations names and attributes names.
end
```

12. **make_message_fns:** This prapares the message to be sent for validation of pre-mining functions.
Pseudocode:

```
begin
  fill message_fns with selection function name
  fill message_fns with cleaning function name
  fill message_fns with transformation function name.
end
```

13. **validate_dra:** This validates database , relation and attributes.
Pseudocode:

```
begin
  connect to database
  validate existence and access of database , relation and attributes
end
```

14. **make_message:** This prapares the message containing schema information to be saved in a text file.
Pseudocode:

```
begin
  fill message with all the schema information
end
```

15. **create_directories_and_files:** This creates the necessary directory structure and files.
    Pseudocode:

    ```
    begin
      create the directory structure for compiled schema
      create history (.history) file
    end
    ```

16. **write_cactus:** This writes the algorithmic specific data into external file (.algo file), which together with general schema file makes complete schema description..
    Pseudocode:

    ```
    begin
      write cactus data into external file
    end
    ```

17. **write_iceberg:** This writes the algorithmic specific data into external file (.algo file) , which together with general schema file makes complete schema description.
    Pseudocode:

    ```
    begin
      write iceberg data into external file
    end
    ```

18. **write_on_disk:** This writes the prepared message contaning schema information into a system file (schema_base) . Along with this the prepared message containing all the schema information is also written into external file (.sch file).
    Pseudocode:

    ```
    begin
      write schema details into external file
    end
    ```

19. **log_schema:** This writes the log after schema compilation is done successfully . Schema log contains user id(uid), process id(pid) ,schema name(schema_name), date of access(date) and error no(error).
    Pseudocode:

```
begin
  write schema log record on schema log file
end
```

3.2.3 Implementation

Platform

Implementation is done on Pentium-III machine , with 256 KB RAM and Linux OS. The programming language used is *C++* and the tools used are linux Tools like *Flex* and *yacc* [12].

Data Structure

A structure *schema* saved the data elements of a schema.It contains many parameters like database name , relations name etc.A class *Schema* is created , which contains the schema structure and the associated functions for operations like saving the schema , reading the schema , dropping the schema .A class *parser* was derived from Schema , which contains the functions required for parsing and handling the parsing errors.The details as as follows

```
struct schema
{
  char name[ID_NAME];                                    /* schema name */
  int knowledge_type;                          /* knowledge to be mined */
  char db_name[ID_NAME];                               /* database name */
  struct ar {                    /* stores the relations and attributes */
    int no_attr;               /* no of attributes in current relation */
    char attr[NO_OF_ATTR][ID_NAME];                   /* attribute name */
    char rel_name[ID_NAME];                            /* relation name */
  } rel_attr[NO_OF_REL_ATTR];
  int no_rel;                                  /* no of total relations */
  char sel_fn[ID_NAME];                      /* selection function name */
  char sel_fn_pathname[FILE_NAME_LENGTH];/* selection function pathname */
  char clean_fn[ID_NAME];                     /* cleaning function name */
  char clean_fn_pathname[FILE_NAME_LENGTH];/*cleaning function pathname */
  char trans_fn[ID_NAME];               /* transformation function name */
  char trans_fn_pathname[FILE_NAME_LENGTH];/*transformation fn pathname */
  char algorithm[ID_NAME];                      /* algorithm to be used */
  char algorithm_pathname[FILE_NAME_LENGTH];/*pathname of algorithm file*/
  int no_param;                                /* no of parameters used */
  struct p {                                      /* to store parameter */
    char  name[ID_NAME];                              /* parameter name */
    double value;                                    /* parameter value */
```

```
  } param[NO_OF_PARAM];
  int no_hier;                                /* no of hierarchies used */
  struct h {                          /* to store hierarchy information */
    char  name[ID_NAME];                              /* hierarchy name */
    char attr[ID_NAME];                               /* base attribute */
  } hier[NO_OF_HIER];
} ;

class Schema             /* to store the schema definintion and methods */
{
 private:
  char          mesg[MESSAGE_LENGTH];           /* schema saved as text */
  char          mesg_dra[MESSAGE_LENGTH];/*database,relations,attributes*/
  char          mesg_fns[MESSAGE_LENGTH];            /* pre-mining  fns */
  char          mesg_algo[MESSAGE_LENGTH];            /* algorithm name */
  struct schema kd_schema;           /* to store the schema definintion */
  char          file_name[FILE_NAME_LENGTH];
  int           error;             /* stores error while parsing schema */
 public:
  Schema();
  ~Schema();

  int         return_error();
  void        reset_error();
  void        make_path();
  void        validate_dra();/*validates database,relations ,attributes */
  int         read_from_file(char *);            /* reads stored schema */
  int         read_from_file_in_temp(char *);    /* reads stored schema */
  void        write_on_file();                /* writes  schema on disk */
  void        drop_schema(char* );     /* drops schema from schema base */
  void        create_directories_and_files();
  void        create_dir(char * );/*creates schema associated directory */
  void        create_files();        /* creates schema associated files */
  void        delete_files();   /* deletes schema  files while dropping */
  void        read_and_print_schema(char *);
  void        fill_schema_1(char *,int,char *,char *,char *,char *,char *,int,
                         int,int);/* fills various fields of the schema */
  void        fill_schema_pathnames();
  void        fill_schema_2(int ,char * ,char * );
  void        fill_schema_3(int ,char * ,double );
  void        fill_schema_4(int ,int ,char * );
  void        fill_schema_5(int ,int ,char * );
  void        make_message();
  void        make_message_dra();
```

```
  void        make_message_fns();
  inline void display_message() {cout << mesg << " || " << mesg_dra << endl;};
};

class parser : public Schema /* handles parsing schema definition or query */  {
 private:
  int   err_line_no;                    /* error occurs at this line no */
  int   err_char_no;                   /* error occurs at this position */
  char  err_yy_text[ERR_YYTEXT_LENGTH]; /* error occurs while parsing   */
  int   error;          /* error type */

 public:
  int parse_schema(char *,int );   /* parses query of schema definition */
  void parse_schema_def(char * );           /* parses schema definition */
  void parse_schema_query(char *);                      /* parses query */
  void set_error(int err)                       {error=err;}
  int  get_error()                              {return error;}
  void display_errors();
};
```

Sockets Implementation

For details of Sockets Implementation refer Appendix A on page I.

Project Files

For details of Project Files refer Appendix B on page IV.

3.2.4 Results

The proposed grammer for was successfully parsed and integrated into the Intension Mining system.
The following schema definition is received via server socket message and then parsed ,then messages to verify database,relations and attributes , pre-mining functions and the specific algorithm used are generated and sent. After receiving the proper feedback,compiled schema is saved as metadata and Directory structure and history file are created accordingly.

```
Received connection from 127.0.0.1
IMQL:>Server Message:  schema is schema8 accumulate classification use database
db2 related to a1,a2, a3,a4,a5 from rel3 use selection selfn use cleaning
cleanfn use transformation transfn use mining algorithm iceberg with number
of attributes=4 , categorical attributes = 2 , numerical attributes = 1,
```

```
continuous attributes = 1 and  dimensions = 3, 4, 3,2;

Parsing Schema Definition ....

Message Sent:db2:1:rel3:5:a1 a2 a3 a4 a5 :
Received connection from 127.0.0.1
Server Message: OK
Message Received:OK
Message Sent:selfn:cleanfn:transfn:
Received connection from 127.0.0.1
Server Message: ~/IMIN/lib/selfn:~/IMIN/lib/cleanfn:~/IMIN/lib/transfn
Message Received:~/IMIN/lib/selfn:~/IMIN/lib/cleanfn:~/IMIN/lib/transfn
Message Sent:iceberg
Received connection from 127.0.0.1
Server Message: ~/IMIN/lib/algo
Message Received:~/IMIN/lib/algo
 creating /home/dhamija/IMIN ...
 creating /home/dhamija/IMIN/schema ...
 creating /home/dhamija/IMIN/schema/schema8 ...
 creating /home/dhamija/IMIN/schema/schema8/KC ...
 creating /home/dhamija/IMIN/schema/schema8/output ...
 creating /home/dhamija/IMIN/schema/schema8/query ...
 creating /home/dhamija/IMIN/schema/schema8/temp ...
/home/dhamija/IMIN/schema/schema_base could not be opened for reading during write
Schema schema8 saved
IMQL:>Error = 0


Schema to be scanned :?schema8
Schema schema8 found
schema found as->schema8 1 db2 selfn ~/IMIN/lib/selfn cleanfn ~/IMIN/lib/cleanfn
 transfn ~/IMIN/lib/transfn iceberg ~/IMIN/lib/algo

no of relations=1 and attributes are ->

no of attributes=5
relation 1 rel3 5 attributes-->a1,a2,a3,a4,a5,

no of parameters=0
parameters->

no of hierarchies=0
hierarchies->
```

3.2.5 Log Generation

Logs are generated while schema compilation .Sample of generated logs is shown as follows.

Schema Compilation log

```
uid pid  schema name date                          error
-------------------------------------------------------
500 1050 schema8     Wed Dec 18 23:35:22 IST 2002  2109
522 1107 schema3     Wed Dec 18 23:37:53 IST 2002  0
```

3.2.6 Testing

Unit Testing

Test stubs and test drivers were written to test each unit in isolation.Some of the test drivers are mentioned in files *def_server.cpp* ,*algo_server.cpp* ,*dra_server.cpp* and *path_server.cpp* and test stubs are mentioned in *externals.h*

Integration testing and Test Cases

Comprehensive Test Cases were prepared to test each individual way , the language can be written using the specified grammer. Following are some examples

1. `schema is schema8 accumulate classification use database db2 related to a1,a2, a3,a4,a5 from rel3 use selection selfn use cleaning cleanfn use transformation transfn use mining algorithm iceberg with number of attributes=4 , categorical attributes = 2 , numerical attributes = 1, continuous attributes = 1 and dimensions = 3, 4, 3,2;`

2. `schema is schema8 accumulate classification use database db2 related to a1,a2,a3,a4,a5 from rel3 use selection selfn use cleaning cleanfn use transformationtransfn use mining algorithm cactus with number of attributes=5 , domain names and sizes = car 2,colour 3,cat 3,cup 4,cub 2 and attribute values = car1 car2 colour1 colour2 colour3 cat1 cat2 cat3 cup1 cup2 cup3 cup4 cub1 cub2 , largest and smallest domain sizes = 4,2;`

3. `schema is schema8 accumulate classification use database db2 related to a1,a2,a3,a4,a5 from rel3 use selection selfn use cleaning cleanfn use transformationtransfn use mining algorithm iceberg with number of attributes=4 , categorical attributes = 2 , numerical attributes = 1, continuous attributes = 1 and dimensions = 3, 4, 3,2;`

4. ```
   schema is schema9 accumulate association_rules use database db1
   use h1 for a1 h2 for a2 related to a1,a2,a4,a5 from rel1, a6,a7 from rel2
   useselection selfn use cleaning cleanfn use transformation transfn use
   mining algorithm rosa_meo with parameters priority = 1,periodicity=2,
   start_time=now;
   ```

5. ```
   schema is schema6 accumulate clustering use database db3 related to a1,a2
   fromrel1, a4,a5,a6 from rel2, a7,a8,a9,a10,a11,a12,a13 from rel3 use
   selection selfn use cleaning cleanfn  use transformation transfn use
   mining algorithm rosa_meo ;
   ```

Chapter 4

IMQL and Query Processor

4.1 Intension Mining Query Language

4.1.1 Introduction

IMQL should be rich enough to capture all the user query requirements along with the focussing information. Data Mining covers a wide spectrum of tasks covering data clustering , mining association rules and data classification. This makes the design of IMQL a fairly challenging task [2].

4.1.2 Design Considerations

1. The following set of relevent operators [2] need to be defined which allow the user to experiment with the knowledge discovery process. All these operators are to be based on some given metric , and can handle user given constraints.These operate on KCs [2] which have been desirable merged(as specified in the query).

   (a) **GETLITS :** Get all large item-sets for a support level from the specified KC

   (b) **RANKLITS :** Mine for first large item-sets which have item I1 and not I2.

   (c) **STATLITS :** Get the desired statistics for the large item-sets discovered for a given support level.This is user defined operator that operates over the results of "GETLITS" or "RANKLITS" operators.

   (d) **COMPLITS :** Mine and compare large item-sets of size $\geq n$ with support s_1 for periods date1 to date2 and date3 to date4.

   (e) **GETRULES :** Get all association rules for a support level and the confidence level. It invokes "GETLITS" with the given support , and then uses the specified metric to generate the association rules. The metric can be varied by the user as long as it is supported by the KC. A new metric may require modification in the design of KC

(f) **RANKRULES :** Mine for first n high confidence association rules which have item "I" as antecedent with support and given metric

(g) **STATRULES :** Get the statistics for the association rules which have item "I" as antecedant with support s_1 and confidence c_1 for periods date1 to date2.This is a user defined operator similar to "STATLITS". It operates over the results of "GETRULES" or "RANKRULES" operators.

(h) **COMPRULES :** Mine and compare association rules which have item "I" as antecedant with support s_1 and conviction c_1 for periods date1 to date2.

2. Manual monitoring is supported by "COMPLITS" and "COMPRULES" operators defined above.The user can define audit conditions for monitoring. Falling of support (confidence) level of an item-set (rule) below a certain point , presence or absence of item-set (rule) , significant deviations in the structural or measure components of item-sets [8] are some examples of audit conditions.

4.1.3 Syntax of IMQL

IMQL [2] adopts a SQL like syntax to facilitate high level data mining and integration with the relational query language SQL.IMQL has been adopted from Ms. Vasudha Bhatnagar's work [2] [1] . IMQL is defined in BNF as follows.

1. **IMQL** $\rightarrow$ DO m-oper PROC proc-name AFTER e-oper rest-of-the-query

2. **IMQL** $\rightarrow$ DO e-oper rest-of-the-query

3. **IMQL** $\rightarrow$ MONITOR SCHEMA schema-name DEFINE TRIGGER trigger-name AFTER e-oper PERFORM procedure-name ON CONDITION cond-name rest-of-the-query

4. **IMQL** $\rightarrow$ trigger-command

5. rest-of-the-query → ON SCHEMA schema-name SPAN span-list WITH CONSTRAINTS cons-list USING PARAMETERS param-list

6. trigger-command → t-oper TRIGGER trigger-name

7. t-oper → ENABLE | DISABLE | CANCEL

8. m-oper → COMPLITS | STATLITS | COMPRULES | STATRULES | USER-DEF-EXP-OPER

9. e-oper → g-oper | order r-oper

10. g-oper → GETLITS | GETRULES | USER-DEF-EXP-OPER

11. r-oper → RANKLITS | RANKRULES

12. order → INCREASING | DECREASING

13. cons-list → cons | cons AND cons | cons OR cons

14. cons → cons-name1 relop arith-expr1 | cons-name2 setop set-expr | agg relop arith-expr2 | NONE

15. cons-name1 → LITSSIZ | ANTESIZ | CONSSIZ | NRULES | NLITS | USER-DEF-CONS-NAME1

16. cons-name2 → LITS | CONS | ANTE | USER-DEF-CONS-NAME2

17. param-list → metric relop arith-expr2

18. metric → SUPPORT | CONF | CONVIC | INTEREST | CORREL | USER-DEF-METRIC

19. span-list → * | FROM date TO date | FROM date TO * | date AND date

20. cond-name → NEWLITS | NEWRULES | USER-DEF-CONS-NAME1

21. agg → Sum(cons-name2) | Max(cons-name2) | Min(cons-name2)

22. arith-expr1 → arithmetic expression evaluating to an integer

23. arith-expr2 → arithmetic expression evaluating to an real value

24. set-expr → set of items in the relevent domain

25. date → standard date format (monddyyyy)

26. setop → $\subset$ | $\subseteq$ | $\supset$ | $\supseteq$

27. relop → $<$ | $>$ | $\leq$ | $\geq$ | $=$ | $\neq$

The rule "**IMQL** → DO e-oper rest-of-the-query" executes experimentation operator e-oper (get operator g-oper i.e.GETLITS | GETRULES | USER-DEF-EXP-OPER or rank operator r-oper i.e. RANKLITS | RANKRULES with order as INCREASING | DECREASING) performed on schema schema-name with start of span and end of span dates mentioned in span rule and constraints mentioned as cons-list in constraints rule and using parameters mentioned in param-list of parameters rule .

The rule "**IMQL** → DO m-oper PROC proc-name AFTER e-oper rest-of-the-query" executes procedure proc-name over the results of experimentation operator e-oper performed on schema schema-name with start of span and end of span dates mentioned in span rule and constraints mentioned as cons-list in constraints rule and using parameters mentioned in param-list of parameters rule .It finally executes manual monitoring operator m-oper(COMPLITS | STATLITS | COMPRULES | STATRULES | USER-DEF-EXP-OPER

The rule "**IMQL** → MONITOR SCHEMA schema-name DEFINE TRIGGER trigger-name AFTER e-oper PERFORM procedure-name ON CONDITION cond-name rest-of-the-query" defines trigger trigger-name for schema schema-name which executes procedure procedure-name on condition defined in rule cond-name after executing experimentation operator e-oper performed on schema schema-name with start of span and end of span dates mentioned in span rule and constraints mentioned as cons-list in constraints rule and using parameters mentioned in param-list of parameters rule .

The rule "**IMQL** → trigger-command" performs t-oper (ENABLE | DISABLE | CANCEL) for trigger trigger-name

The rule "cons-list → cons | cons AND cons | cons OR cons" defines list of constraints coupled with logical operators

The rule "cons → cons-name1 relop arith-expr1 | cons-name2 setop set-expr | agg relop arith-expr2 | NONE" defines the constraints of 4 types namely cons-name1(LITSSIZ | ANTESIZ | CONSSIZ | NRULES | NLITS | USER-DEF-CONS-NAME1) using relational operator relop($<$ | $>$ | $\leq$ | $\geq$ | $=$ | $\neq$) , cons-name2 (LITS | CONS | ANTE | USER-DEF-CONS-NAME2 using set operator setop ($\subset$ | $\subseteq$ | $\supset$ | $\supseteq$) , aggregates constraint agg and finally no constraint by specifying NONE, Where LITSSIZ is size of large item set , NRULES ia number of rules , ANTE stands for antecedant,LITS is large item set

The rule "param-list → metric relop arith-expr2" specify parameters metric (SUPPORT | CONF | CONVIC | INTEREST | CORREL | USER-DEF-METRIC) along with comparison operator (relop) and comparison value arith-expr2

Some examples of IMQL for experimenting are as follows.

```
do complits proc procedure1 after getlits on schema schema8 span from mar112001
to jan112002  with constraints ante <<= {i11,i12,i13} and litssiz >= 2.0 using
parameters support >= .74 and confidence <=.7;

do statlits proc procedure2 after increasing rankrules on schema schema9 span
from mar112001 to jan112002  with constraints null using parameters support >=
.5 and confidence <=.84;

monitor schema schema6 define trigger my_trigger after getlits perform
proc_my_trigger on condition newlits  with constraints null using parameters
support >= .2 ;

cancel trigger my_trigger;
```

4.2 Knowledge Discovery Query Processor

4.2.1 Query Processing Process

The Query Processor validates the information given by the user and executes the query by refering the Metadata [2].

Validation of Schema

Schema is validated in the schema_base,so that the required data can be accessed.

Generation of Execution Plan

After parsing the execution plan is prepared and stored in the external file.

Experimental Queries

Various kinds of operators as defined above are defined to experiment with the data by varying different kinds of parameters repeatedly.

Monitoring Queries

Both automatic moitoring and Manual monitoring are supported by using triggers.

4.2.2 Functional Design

Functionality

1. **Socket programming** Client sockets are required to access the server sockets for getting the query in the form of messages.

2. **Grammer Specification** Grammer needs to be specified for Query specification and processing to be used by Linux tools *flex* and *yacc*.

3. **Integration with the intensional algorithm** Grammer specification should facilitate the integration of query specification with the intensional algorithm.

4. **Creation of Execution Plan** After parsing the query , the query plan of the query needs to be saved , which could be used by query execution phase.

5. **Error Handling** Parsing Errors and other program errors are required to be handled systematicaly with error codes clearly and separately defined.

6. **Creating Logs** Logs need to be generated everytime schema is accessed in processing Query .

Modules and Pseudocode

See Structure Chart Figure 3.1 on page 23

1. **main:** This is the main routine , which starts the program
   Pseudocode:

```
begin
  call get_query_definition
  call parse_query
  call display_errors
end
```

2. **get_query_definition:** This gets the query specification from server through sockets
   Pseudocode:

```
begin
  open client socket
  connect to server socket
  receive string as query specification
end
```

3. **parse_schema:** This parses the query specification and sets error values
   Pseudocode:

```
begin
  call parse_schema_query
  set errror values
end
```

4. **parse_schema_query:** This calls yyparse (creted by *yacc*) to parse the query and then sets error values
   Pseudocode:

```
begin
  call yyparse
  set errror values
end
```

5. **yyparse:** This parses the query specification token by token and performs desired actions
   Pseudocode:

```
begin
  get next token from lexer
  parse the token to specified grammer
  set query type
  call set_query_name, set_query_name_q3, set_query_name_q4, set_query_name_q5
     depending on the type of query
  call validate_schema
  call set_span
  call make_kc_file
  call appropriate procedure to set constranits  for query depending on the
     type of query
  call appropriate procedure to set parameters for query depending on the
     type of query
  call write_on_file
  call write_on_file_as_text
  call set_errors
  call query logger
end
```

Refer Subsection 4.1.3 on page 35 for four types of query

6. **validate_schema:** This reads the schema_base file and validates the existece of schema on which the query is based
Pseudocode:

```
begin
  read schema_base file for schema_name
  report success or failure
end
```

7. **set_span:** This sets the span dates as specified in query so that appropriate KC files can be extracted from the KC directory.
Pseudocode:

```
begin
  set start date for span
  set end date for span
end
```

8. **make_kc_file:** This takes span dates and extracts the KC files[1] can be extracted from the KC directory.
Pseudocode:

[1]Naming convention of KC file is $< schema\_name > \_ < algorithm\_name > \_ < seq\_no > \_monddyyyy$

```
begin
  start from date=start_date
  loop till date < end_date
      extract name of KC file
      increment date with the increment specified in the query
  end loop
end
```

9. **write_on_file:** This writes the query specification on external system file
   Pseudocode:

```
begin
  write query structure on external file by using ''fwrite''
end
```

10. **write_on_file_as_text:** This writes the query specification on external text file
    Pseudocode:

```
begin
  write query specification as text  on external file by using ''fprintf''
end
```

11. **log_query:** This writes the log after query specification is parsed successfully . Query log contains user id(uid), process id(pid) ,schema name(schema_name), date of access(date) and error no(error).
    Pseudocode:

```
begin
  write query log record on query log file
end
```

12. **make_message:** This prapares the message containing query specification to be saved in a text file.
    Pseudocode:

```
begin
  fill message with query specification
end
```

13. **set_con1:** Sets relational constraints for various query types
    Pseudocode:

```
begin
   set constraint type
   set constraint name
   set set operator
end
```

14. **set_cons2:** Sets set constraints depending on query type
    Pseudocode:

```
begin
   set constraint type
   set constraint name
   set logical operator
end
```

15. **set_no_of_id:** Sets no of set members for set type constraint
16. **set_cons2_2:** Sets members for set type constraint
17. **set_cons2_3:** Sets operator for set type constraint
18. **set_cons2_4:** Sets total number of constraints
19. **set_param:** Sets metric and value for parameter metric
    Pseudocode:

```
begin
   set metric type
   set relational operator
   set value compared with
end
```

20. **set_para2_3:** Sets operator for parameter metric
21. **set_param_1:** Sets total number of parameters
22. **set_m_oper:** Sets monitoring operator
23. **set_gr:** Sets 'g' or 'r' operator
24. **set_query_trigger:** Sets trigger and triggering condition
    Pseudocode:

```
begin
   set trigger name
   set trigger procedure
   set triggering condition
end
```

4.2.3 Implementation

Platform

Implementation is done on Pentium-III machine , with 256 KB RAM and Linux OS. The programming language used is *C++* and the tools used are linux Tools like *Flex* and *yacc* [12].

Data Structure

A class *Query* is created to save the query execution plan , read the query etc.

```
struct span_dates                                /* span dates for query */
{
  char from_date[10];
  char to_date[10];
};

struct cons                               /* constraint are of two types */
{                          /* one with relational operator and the other */
  int type;                                         /* with set operator */
  union  u
  {
    struct cc1                                /* relational constraint */
    {
      cons_name_1 cons1;                            /* constraint name */
      relop opr;                                /* relational operator */
      double arith_expr;                        /* value compared with */
    } c1;
    struct cc2                                       /* set constraint */
    {
      cons_name_2 cons2;                            /* constraint name */
      setop opr;                                       /* set operator */
      int no_members;                      /* no of members in the set */
      char members[NO_OF_MEMBERS][ID_NAME];      /* set members' names */
    } c2;
  } un;
};

struct cons_list       /* constraint and the associated logical operator */
{                             /* first constraint has token operator Nil */
  struct cons constr;    /* and all others have logical operator same as */
  logop opr;                                        /* what precedes them */
};
```

```
struct p                          /* metric and its value used for query */
{
  metric_type metric;                                     /* metric used */
  relop opr;;                                     /* relation operator */
  double value;                                  /* value compared with */
};

struct pp                     /* metric with associated logical operator */
{
  struct  p para;                                       /* metric clause */
  logop opr;                                         /* logical operator */
};

struct query_q1        /* elementry query fields to be used in all types */
{                                                          /* of queries */
  char schema_name[ID_NAME];                              /* schema name */
  struct span_dates span;                                  /* span dates */
  int no_of_constraints;                      /* no of constraints used */
  cons_list constraint[NO_OF_CONSTRAINTS];        /* list of constraints */
  int no_of_parameters;                            /* no of parameters */
  struct pp param[NO_OF_PARAMETERS];                       /* parameters */
};


struct query_q2               /* first type of query with g-operator or */
{                                                       /* or r-operator */
  struct query_q1 q1;
  int gr_oper;            /* to specify whether g-operator or r-operator */
  union uu
  {

    g_oper opr;                                          /* get operator */
    struct operr
    {
      order ord;                     /* order used fro ranking operators */
      r_oper opr;                                    /* ranking operator */
    } oper;
  } un1;
};

struct query_q3              /* second type of query with procedure and */
{                                                          /* m-operator */
  struct query_q2 q2;
```

```
  char proc_name[ID_NAME];                              /* procedure name */
  m_oper opr;                                      /* monitoring operator */
};

struct trigger         /* third type of query contains trigger , trigger */
{                                  /* procedure and triggering condition */
  struct query_q2 q2;
  char name[ID_NAME];                                     /* trigger name */
  char t_proc[ID_NAME];                              /* trigger procedure */
  cond_name cond;                                 /* triggering condition */
};

struct trigger_command/* Fourth type of query , contains trigger and the */
{                                                           /* t-operator */
  char name[ID_NAME];                                     /* trigger name */
  t_oper opr;                                         /* trigger operator */
};

struct q_query                             /* used to store parsed query */
{
  int type;                                              /* type of query */
  char name[ID_NAME];                                      /* schema name */
  char schema_name[ID_NAME];                               /* schema name */
  union
  {
    struct query_q2 q2_query;                /* query of  (first )type q2 */
    struct query_q3 q3_query;               /* query of ( second )type q3 */
    struct trigger t1_query;                 /* query of ( third )type q4 */
    struct trigger_command t2_query;        /* query of ( fourth )type q5 */
  } un2;
};


class Query              /* contains stored query and associated methods */
{
 private:
  struct q_query qry;                              /* stores parsed query */
  char mesg[MESSAGE_LENGTH];   /* query in text form , to be saved later */
  int error;                                          /* error in parsing */

 public:
  void set_error(int err)                       {error=err;}
  int  get_error()                              {return error;}
  inline void display_error()   { cerr <<   "Error = " << error << endl;};
```

```
  void set_query_type(int i);                         /* sets query ytpe */
  void set_query_name(char * );         /* sets schema name for q2 query */
  void set_query_name_q3(char *,char * );/*sets schema name for q3 query */
  void set_query_name_q4(char * );      /* sets schema name for q4 query */
  void set_query_name_q5(int ,char * ); /* sets schema name for q5 query */
  void set_span(int ,char *,char *); /* sets span dates as per query type*/
  void set_cons2_2(int,int ,char *);  /* sets members for set constraint */
  void set_cons2_3(int,int );   /* sets operator for set type constraint */
  void set_para2_3(int,int );      /* sets operator for parameter metric */
  void set_cons2_4(int);                   /* sets number of constraints */
  void set_cons1(int ,int,int,int,double);
    /* sets relational constraint and constraint types as per query type */
  void set_cons2(int ,int,int,int);
    /* sets set constraint with their logical operator as per query type */
  void set_no_of_id(int,int );
                       /* sets no of set members for set type constraint */
  void set_param(int,int,int,double);
                             /* sets metric & value for parameter metric */
  void set_param_1(int);                        /* sets no of parameters */
  void set_m_oper(int );                     /* sets monitoring operator */
  void set_gr(int,int ,int,int);      /* sets get and ranking  operator  */
  void set_query_trigger(char * ,char * ,int);
                                /* sets trigger and triggering condition */
  void make_message();/* makes message,which becomes query saved as text */
  void write_on_file();   /* writes parsed query on external binary file */
  void write_on_file_as_text(); /* writes parsed query on ext  text file */
  void read_from_file();   /* reads query from external binary disk file */
  void read_and_print_query();
             /* reads query from external binary disk file and prints it */
};
```

Sockets Implementation

For details of Sockets Implementation refer Appendix A on page I.

Project Files

For details of Project Files refer Appendix B on page IV.

4.2.4 Results

The proposed grammer for IMQL was successfully parsed and integrated into the Intension Mining system.
The following query is received via server socket message and then parsed

,then message to verify the schema name is sent. After receiving the proper feedback,query is further parsed and stored in the external file as execution plan.

```
Received connection from 127.0.0.1
IMQL:>Server Message:  do getrules on schema schema8 span from mar112001 to oct112002  and frequency  = 6 months with constraints ante <<= {i11,i12,i13} and litssiz >= 2.0 using parameters support >= .9 and confidence <=.6;

Parsing Query ....

Schema schema8 found
ls: /home/dhamija/IMIN/schema/schema8/KC/schema*: No such file or directory
KC files not present
IMQL:>Error = 2109

Schema schema8 found
Query  saved
Query  read
Query  saved as text
```

4.2.5 Log Generation

Logs are generated while query processing.Sample of generated logs is shown as follows.

```
uid pid  schema name date                          error
-------------------------------------------------------
550 1152 schema8     Wed Dec 18 23:35:22 IST 2002  0
567 1108 schema5     Wed Dec 18 23:37:53 IST 2002  2306
```

4.2.6 Testing

Unit Testing

Test stubs and test drivers were written to test each unit in isolation.Some of the test drivers are mentioned in files *def_server.cpp* ,*algo_server.cpp* ,*dra_server.cpp* and *path_server.cpp* and test stubs are mentioned in *externals.h*

Integration testing and Test Cases

Comprehensive Test Cases were prepared to test each individual way , the language can be written using the specified grammer. Following are some examples

1. do complits proc procedure1 after getlits on schema schema8 span from mar112001to jan112002 with constraints ante <<= {i11,i12,i13} and litssiz >= 2.0 using parameters support >= .74 and confidence <=.7;

2. do getlits in schema kds_assoc span * with constraints litsiz >= 3 using parameters support >= .8;

3. do getrules on schema kds_assoc span from jan111999 to jun111999 with constraints ante <<= {i11,i12,i13} using parameters support >= 0.9 and confidence >= 6;

4. do rankrules on schema kds_assoc span from jan111999 to * with constraints ante <<= {i11,i12,i13} and litsiz >= 3 and nrules = 3 using parameters support >= 0.9 and support >= 6;

5. do complits proc my_comparison after getlits on schema kds_assoc with constraints none using parameters support >= 0.9 ;

6. do statlits proc procedure2 after increasing rankrules on schema schema9 span from mar112001 to jan112002 with constraints null using parameters support >= .5 and confidence <=.84;

7. do statrules proc my_statistics after rankrules on schema kds_assoc with constraints nrules=k using parameters support >= 0.9;

8. monitor schema schema6 define trigger my_trigger after getlits perform proc_my_trigger on condition newlits with constraints null using parameters support >= .2 ;

9. cancel trigger my_trigger;

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Whereas Earlier Works have already indicated that Intension Mining System is much more efficient than the existing approaches, this work establishes the strong footing for Implementation of Intension Mining Framework.It neatly integrates all the steps of Knowledge Disvovery process.Our work is an attempt to develop complete Intension Mining System. Further work on this system is required to yield better results.

5.2 Future Work

1. Error repair mechanisms (Global as well as Local approaches) like Burke-Fisher error repair may be introduced while parsing.

2. Query Execution Plans need to be optimised for Intension Mining system to be efficient.A comprehensive IMQL is required to mine knowledge form KCs efficiently.

3. In this project Relational Databases are assumed for specifying knowledge. The KDSL can also be designed for Object Oriented Databases , Legacy Databases , MM Databases etc.

4. More and more intensional Algorithms need to be designed and integrated into the intension mining system. Integrated approach towards knowledge discovery will be realised , once the library of intensional algorithms is rich enough.

5. On the similar lines , library of Selection,Transformation and cleaning functions need to be made richer.

6. IMQL currently supports ”a month period” as span granularity.It can further be designed to cover days and different combinations of specifying spans

7. More Work is required to be done on integration aspects like standardisation of interfaces so that intensional algorithm designers and other programmers are not constrained to code in a specific language

8. Library of Presentation Tools need to be developed for effective visualisation.

9. A system of administration needs to be developed within the scope of Intension Mining , which maintains its own database containing bookkeeping information about all the schema entries , contents of libraries etc.This database can be used for maintenance purposes.

10. Aspects like Network security and Database security need to be looked into, while accessing the database.

Appendix A

Sockets Implementation

A socket is a way to talk to other computer. To be more precise, it's a way to talk to other computers using standard Unix file descriptors.

Types of Socket

There are two types of internet sockets.

1. **Stream Sockets** They're error free; they will arrive in the same order ;they use TCP.These are used in our code.

2. **Datagram Sockets** They use UDP ; they're connectionless we build a packet with the destination information and send it out.

Structures

Structures used in socket programming to hold information about the address.

```
1. struct sockaddr holds socket information.

struct sockaddr{
unsigned short  sa_family;    /* address family */
char         sa_data[14];  /* 14 bytes of protocol address */
};

2. struct sockaddr_in helps to reference to the socket's elements.

struct sockaddr_in {
short int      sin_family;  /* Address family */
unsigned short int   sin_port;   /* Port */
struct in_addr      sin_addr;   /* Internet Address */
```

```
unsigned char      sin_zero[8]; /* Same size as struct sockaddr */
};

3. sin_zero is set to all zeros with memset() or bzero()

struct in_addr {
unsigned long s_addr;
};

4. hostent structure is used to get remote host information.

struct hostent
{
  char *h_name;                 /* Official name of host.  */
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of addresses from name server.  */
#define h_addr  h_addr_list[0]  /* Address, for backward compatibility.  */
};

This structure is defined in header file netdb.h.
```

Conversions

There are two types of byte ordering: most significant byte and least significant byte.This former is called "Network Byte Order"

There are two types to be converted : short and long.

```
htons() -> "Host to Network Short"
htonl() -> "Host to Network Long"
ntohs() -> "Network to Host Short"
ntohl() -> "Network to Host Long"
```

IP Addresses

```
1: inet_addr() converts an IP address into an unsigned long.
e.g.
dest.sin_addr.s_addr = inet_addr("195.65.36.12");

2: inet_ntoa() converts string IP addresses to long.
e.g.
char *IP;
ip=inet_ntoa(dest.sin_addr);
```

```
inet_addr() returns the address in Network Byte Order
```

Important Function Used

```
1: socket() :- int socket(int domain,int type,int protocol);

socket() gives a socket descriptor that can be used in later system
calls or it gives you -1 on error (this is usefull for error checking routines).

2: bind() :- int bind(int fd, struct sockaddr *my_addr,int addrlen);

bind() is used when we care about your local port (usually when you use
listen() ) and its function is to associate a socket with a port (on our machine).
 It returns -1 on error.

3: connect() :- int connect(int fd, struct sockaddr *serv_addr, int addrlen);

connect() is used to connect to an IP address on a defined port. It returns-1 on
error.

4: listen() :- int listen(int fd,int backlog);

listen() is used if we're waiting for incoming connections, this is, Aftercalling
listen() we call accept() in order to accept incoming connection.

5: accept() :- int accept(int fd, void *addr, int *addrlen);

When someone is trying to connect to our computer, we use accept() to get the
connection.

6:send() :- int send(int fd,const void *msg,int len,int flags);

This function is used to send data over stream sockets or CONNECTED datagram
sockets.

7: recv() :- int recv(int fd, void *buf, int len, unsigned int flags);

recv() returns the number of bytes read into the buffer and it'll return -1 on
error.

8: close() :- close(fd);

close() is used to close the connection on your socket descriptor.
```

Appendix B

Project Files

parse_schema_def.y → Contains grammer and main function
socket.h → Socket client / server and other definitions
error_codes.h → Error codes and messages
misc.h → Miscellaneous functions,defs for KC,Log etc
externals.h → External stubs and functions for validations
schema.h → Schema class definitions and member functions
query.h → Query class definitions and memner functions
parser.h → Parser class definition and member functions
cactus.h → Cactus algorithm related functions
iceberg.h → Iceberg Algorithm related functions
def.h → Definitions for lexing and parsing
def_server.cpp , algo_server.cpp ,path_server.cpp dra_server.cpp →
Routimes for test drivers

Bibliography

[1] Vasudha Bhatnagar , SK Wasan , SK Gupta & DVLN Somayajulu , An architectural Framework for KDD Process , communicated to SIGMOID , 2000.

[2] SK Gupta , Vasudha Bhatnagar ,SK Wasan & DVLN Somayajulu ,Intension Mining : A New Paradigm in Knowledge Discovery,Technical Report No. IITD/CSE/TR2000/001 , Mar 2000.

[3] G Piatetsky Shapiro and WJ Frawley , Knowledge Discovery in Data Bases , AAAI/MIT Press , 1991.

[4] Rakesh Agrawal and R. Srikant , Fast Algorithm for Mining Association Rules , Proceedings of 20^{th} VLDB Conferece , Santiago , Chile , 1994.

[5] Ming Syan Chen , Jiawei Han and Philip S Yu , Data Mining : An Overview from Database Perspective , IEEE TKDE , Dec 1996.

[6] David W Cheung , J Han and others ,Maintenance of Discovery Association Rules in Large Databases : An Incremental Updating Technique , Proceedings of Internationsl Conference on Data Engineering , Louisiana , Feb 1996.

[7] J Han , Y Fu and others DMQL : A Data Mining Query Language for Relational Databases , SIGMOID Workshop on Research Issues on Data Mining and Knowledge Discovery , June , Montreal , Canada , 1996.

[8] V Ganti , J Gehrke and R Ramakrishnan and WY Loh . FOCUS : A framework for measuring differences in Data Characteristics in Proceedings of 18^{th} Symposium on PODS ,1999.

[9] Andrew W. Appel , Modern Compiler Implementatin in Java, Cambridge University Press , First Edition 2001

[10] Pujari Arun K , Data Minig Techniques , Universities Press (India) Limited , First Edition 2001.

[11] Jiawei Han , Micheline Kamber and Simon Fraser University , Data Mining:Concepts and Techniques , Morgan Kaufmann Publishers , First Edition 2001.

[12] Kernighan Brian W. and Pike Rob , The UNIX Programming Environment , Prentice Hall India Pvt Ltd. Second Indian Reprint.

[13] R Srikant and Rakesh Aggarwal , Mining geberalized associations rules , Proceedings of the 21^{st} International Comference on Very Large Databases , Zurich , Switzerland , Sep 1995.

[14] Ming Syan Chen , J Han and Philip S Yu , Data Mining : An overview from Database perspective , IEEE TKDE , Dec 1996.

[15] P Michaud : Clustering Techniques , Future Generation Computer systems , 1997.

[16] J Han , Y Fu et al, DMQL : A Data Mining Query Language fopr Relational Databases , SIGMOID Workshop on Research Issues on Data Mining and Knowledge Discovery , June , Montreal , Canada , 1996.

[17] J Ross Quinlan , ID3 algorithm , Bagging and Boosting C4.5

[18] Gabar Malli , On-line classification with a Lazy model based algorithm

[19] Roa Meo , A New Approach for Discovery of Frequent Itemsets , Proceedings of 1^{st} International Conference on Data Warehousing and Knowledge Discovery , Florence , Italy , Aug 1996.

[20] Database Mining : A Performance Perspective , IEEE TKDE ,, Vol 5, N 6 , dec 1993 (914-925)