

CONFABULATION BASE INFREQUENT WEIGHTED ITEMSET MINING USING FREQUENT PATTERN GROWTH

A PROJECT REPORT

Submitted by

UDHAYAPRIYA M (1513147)

VIGNESHWARAN R (1513156)

VIMAL V RA (1513158)

NAVEENKUMAR P (1513511)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

K.S.R. COLLEGE OF ENGINEERING

(Autonomous)

TIRUCHENGODE



ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2019

K.S.R. COLLEGE OF ENGINEERING
TIRUCHENGODE
ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "CONFABULATION BASE INFREQUENT WEIGHTED ITEMSET MINING USING FREQUENT PATTERN GROWTH " is the bonafide work of "UDHAYAPRIYA M (1513147), VIGNESHWARAN R (1513156), VIMAL V RA (1513158), NAVEENKUMAR P (1513511)" who carried out the project work under my supervision.

SIGNATURE

Dr. A. RAJIV KANNAN M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of CSE,

K.S.R. College of Engineering,

Tiruchengode-637215.

SIGNATURE

Dr. E. BABY ANITHA M.E., Ph.D.,

SUPERVISOR

Associate Professor

Department of CSE,

K.S.R. College of Engineering,

Tiruchengode-637215.

Submitted for the project viva – voce held on

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We feel highly honored to extend our sincere gratitude to our beloved Founder cum Chairman **Lion Dr. K. S. RANGASAMY MJF.,** K.S.R Educational Institutions and our Chairman **Mr. R. SRINIVASAN BBM., MISTE.,** Aarthi Educational and charitable trust for providing all facilities to complete this project work.

We would like to acknowledge the constant and kind support provided by our principal **Dr. P. SENTHILKUMAR M.E., Ph.D.(IITM),** who supported us in all the endeavors and been responsible for inculcating us all through our career.

We feel highly elated to thank our respectable Head of the Department **Dr. A. RAJIV KANNAN M.E., Ph.D., MISTE., MCSI.,** who guided us and was a pillar of support for the successful completion of the project.

We are thankful to our Project Coordinators **Dr. N. S. NITHYA M.E., Ph.D.,** and **Mr. C. THIRUMALAI SELVAN M.Tech., (Ph.D.),** of our department for their valuable suggestions and guidance to our project.

We are the most fortunate in having the opportunity to work under the guide **Dr. E. BABY ANITHA M.E., Ph.D.,** express our sincere thanks to her. This project has brought out the hidden talent within us.

It is a pleasure to express our gratefulness to our beloved parents for providing their support and confidence to us for the completion of the project and our heartfelt thanks to our entire department faculty members, beloved friends, directly and indirectly who helped us during the tenure of the project.

ABSTRACT

High Utility Itemset Mining (HUIM) has emerged as an important with applications to retail-market data analysis, stock market prediction, and recommender systems, etc. Our experiments show that Efficient High Utility Itemset Mining (EHUIM) and High Utility Pattern (HUP) are generally the top two performers in running time, while also consumes the least memory in most cases. In order to compare these two algorithms in depth, we use another synthetic datasets with varying parameters so as to study the influence of the related parameters, in particular the number of transactions, the number of distinct items and average transaction length. This project tackles the issue of discovering rare and weighted itemsets, the Infrequent Weighted Itemset mining (IWIM) problem to facilitate the mining performance and avoid scanning original database repeatedly, a compact tree structure, named Utility Pattern (UP) Tree is used, to maintain the information of transactions and high utility itemsets (HUI). Total Weight Utility (TWU) mining model is not suitable for such databases since the more items a transaction contains, the higher TWU. Total utility (TU) is the aggregate level of satisfaction or fulfillment that a consumer receives through the consumption of a specific good or service. Each individual unit of a good or service has its own marginal utility, and the total utility is simply the sum of all the marginal utilities of the individual units.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF TABLES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 INTRODUCTION OF DATA MINING	1
	1.2 OBJECTIVE OF THE PROJECT	2
2	SYSTEM ANALYSIS	5
	2.1 EXISTING SYSTEM	5
	2.2 DRAWBACKS OF EXISTING SYSTEM	5
	2.3 PROPOSED SYSTEM	5
	2.4 ADVANTAGES OF PROPOSED SYSTEM	6
3	SYSTEM SPECIFICATION	7
	3.1 HARDWARE REQUIREMENTS	7
	3.2 SOFTWARE REQUIREMENTS	7

4	SOFTWARE DESCRIPTION	8
4.1	FRONT END	8
4.2	BACK END	14
5	PROJECT DESCRIPTION	16
5.1	PROBLEM DEFINITION	16
5.2	OVERVIEW OF THE PROJECT	16
5.3	MODULE DESCRIPTION	17
6	SYSTEM DESIGN AND DEVELOPMENT	23
6.1	SYSTEM ARCHITECTURE	23
6.2	UML DIAGRAMS	25
7	SYSTEM TESTING	30
8	CONCLUSION AND FUTURE ENHANCEMENTS	32
	APPENDICES	34
	SCREENSHOTS	34
	SOURCE CODE	41
	REFERENCES AND PUBLICATIONS	49

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
4.1	Front End	8
6.1	System Architecture	24
6.2	Use Case Diagram	26
6.3	Sequence Diagram	27
6.4	Activity Diagram	28
6.5	Class Diagram	29

LIST OF TABLES

TABLE No.	TABLE NAME	PAGE No.
5.1	Admin	19
5.2	Data Items	19
5.3	Profit	20
5.4	Users	20
5.5	Utility Threshold	20
5.6	Weight Utility Threshold	21
5.7	Transaction	21
5.8	Tree	21
5.9	Recorded Items	22

LIST OF ABBREVIATIONS

ACRONYM	ABBREVIATION
EHUIM	Efficient High Utility Itemset Mining
FIM	Frequent Itemset Mining
HUI	High Utility Itemset
HUIM	High Utility Itemset Mining
HUP	High Utility Pattern
IWI	Infrequent Weighted Itemset
IWIM	Infrequent Weighted Itemset mining
TU	Total Utility
TWU	Total Weight Utility
UP	Utility Pattern

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION OF DATA MINING

Data mining is about finding new information in a lot of data. Data mining, the extraction of hidden predictive information from large databases, it is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve.

It helps organization to make full use of the data stored in their databases and when it comes to decision making, this is true in all fields, and is also true in all different types of organizations. Knowledge discovery in databases (KDD) is the process of discovering useful knowledge from a collection of data. This widely used data mining technique is a process that includes data preparation and selection, data cleansing, incorporating prior knowledge on data sets and interpreting accurate solutions from the observed results. Data mining is the process of applying these methods to data with the intention of uncovering hidden patterns.

It has been used for many years by businesses, scientists and governments to sift through volumes of data such as airline passenger trip records, census data and supermarket scanner data to produce market research reports. A primary reason for using data mining is to assist in the analysis of collections of observations of behavior. Such data are vulnerable to co linearity because of unknown interrelations. An unavoidable fact of data mining is that the sub sets of data being analyzed may not be representative of the whole domain, and therefore may not contain examples of certain critical relationships and behaviors that exist across other parts of the domain.

1.2 OBJECTIVE OF THE PROJECT

High utility itemset mining which discovers sets of items occurring together with high combined profits in sales. HUIM algorithms can be chronologically divided into three categories. Algorithms in the first category, such as the Two-Phase algorithm UP-Growth, first generate potential HUI then compute the utilities of each candidate in the second phase, by recursively exploring the tree structure that keeps the transactions. But this type of approach leads to memory and running time overhead, due to the overly large number of candidates generated in the first phase. Algorithms in the second category try to avoid the above problem by utilizing various techniques, data structures, and properties, which limit the number of candidates and prune further the unpromising candidates.

Transactional databases, such as collections of items bought by customers recorded by systems in supermarkets or online Customer Relationship Management systems for online stores, have important reference value to marketers for discovering hidden customer purchase patterns. This can serve as a basis for decisions about marketing activities, such as targeted marketing campaigns, promotional pricing, and products placement, to maximize the revenue or improve the customers' shopping experience.

Frequent Itemset Mining

Frequent Itemset Mining aims at finding the items that frequently co-occur in the same transactions, the goal of HUIM is to identify the items that appear together but also bring large profits to the merchants. Although the managers of large super markets and online shops know each product's profit per unit and the corresponding purchase quantities, it is not advisable to just focus on selling or promoting the most profitable products, because, in real situations, there may not be enough sales of these products. Likewise, it is inadequate to emphasize just popular products that are selling in large quantities, as these might have lower unit profits and would require an order of magnitude increase in purchase quantity to obtain significant total profit gain and will also occupy more space.

Instead, managers should investigate the historical transactions and extract the set of items that have maximum combined profits, by first multiplying the individual profit

with the number of sales of the same item then summing up all the utilities of the items that have appeared together in the actual transactions. The itemsets having top utility values should be preferred.

FIM but it associates weights with the items in transactions - if these weights have unit values it then degenerates to. It is known that HUIM is a more challenging problem than, since the utility of an itemset is neither monotonic nor anti-monotonic, as mentioned above. As a result, techniques cannot be directly applied to HUIM. Started as a phase in the discovery of association rules but has since been generalized, independent of these, to many other patterns, such as frequent sequences, periodic patterns, frequent subgraphs and bridging rules.

The goal of HUIM is the discovery of all the itemsets having support larger than a given threshold, which is called minimum support. Depending on whether an itemset has higher or lower support than this threshold, it is called frequent or infrequent itemset respectively. Unlike the database in is binary and there are no occurrence quantities of items to be maintained. Utilizes the monotonicity of the frequent itemsets, that is, if an itemset is not frequent, then all its supersets are infrequent as well. The problem of was defined to find the rare frequent itemsets but with high profits. The corresponding definitions are already given. Due to the absence of monotonicity and the downward-closure property, the search space for cannot be directly reduced, as in the case of FIM. Early studies suffered from the overhead of generating too many candidates, which consumes vast memory and storage, and takes very long running time to compute the utilities of the candidates, especially when databases contain lots of long transactions or a low minimum utility threshold is set. A few newer approaches use novel pruning strategies that eliminate unpromising itemsets. This can avoid repeatedly scanning the tree structures built for HUI when computing the corresponding utilities of these itemsets, thus decrease memory consumption and computation time.

In order to alleviate this overhead, latter algorithms introduced new techniques, starting from which builds utility list structures and develops a set enumeration tree to directly extract HUI without both candidate generation and additional database rescans

which enumerates an itemset as a prefix extension of another itemset with powerful pruning and, using the TWU properties, it recursively filters out irrelevant items when growing HUI with sparse data.

The Two-Phase algorithm is a classical and one of the earliest approaches for HUIM. It proposed to discover HUI in two phases, defined the total Weight Utility.

Utility Pattern

Utility Pattern-Growth is an improved a compact tree structure, called utility pattern tree , which first uses utility values of single items to order them, then constructs a Tree like prefix tree using this order. For each node in the tree maintains the number of transactions that the itemset corresponding to the path appears, as well as the associated utility for this itemset. Based on this tree structure Growth proposes two strategies, namely unpromising items and Decreasing local node. In the Unpromising item strategy, the low utility items are discarded from the paths during the construction of a local tree. In Decreasing Local Node, the minimum item utilities of descendant nodes are decreased during the construction of a local tree. The authors attest that Growth performs well in databases containing lots of long transactions or when a low minimum utility threshold is specified While pattern growth approaches avoid the level-wise candidate generation and test methodology, accumulated TWU values are still maintained in node utilities. As a result, a large number of candidates are generated in the mining process and the performance is degraded. Therefore, researchers came up with improved methods where the generation of potential HUI is not required.

High Utility Itemset

High Utility Itemset was the first algorithm to discover without candidate generation, it outperformed previous algorithms by then. It employs a novel vertical data structure to represent utility information, called utility list. It needs only a single database scan to create and maintain such utility lists of item sets containing single items. Then, larger itemsets are obtained by performing join operations of utility lists of itemsets with smaller length.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The existing system is focused on two different IWI-support measures: (i) The IWI-support-min measure, which relies on a minimum cost function, i.e., the occurrence of an itemset in a given transaction is weighted by the weight of its least interesting item, (ii) The IWI-support-max measure, which relies on a maximum cost function, i.e., the occurrence of an itemset in a given transaction is weighted by the weight of the most interesting item. Note that, when dealing with optimization problems, minimum and maximum are the most commonly used cost functions. Hence, they are deemed suitable for driving the selection of a worthwhile subset of infrequent weighted data correlations.

2.2 DRAWBACKS OF EXISTING SYSTEM

- The existing system faces the issue of discovering infrequent itemsets by using weights for differentiating between relevant items only.
- Discovering infrequent itemsets is not carried out within each transaction.
- The usefulness of the discovered patterns has not been validated on data coming from a real-life context.

2.3 PROPOSED SYSTEM

In the proposed system, along with existing system approach, UP Tree construction is carried out along with Unpromising to reduce the number of transactions taken for tree construction. The construction of a global UP Tree is performed with two scans of the original database.

In the first scan, Total Utility of each transaction is computed. At the same time, TWU of each single item is also accumulated. By HUI property, an item and its supersets are unpromising to be high utility itemsets if its TWU is less than the minimum utility threshold. Such an item is called an unpromising item.

During the second scan of database, transactions are inserted into a UP Tree. When a transaction is retrieved, the unpromising items should be removed from the transaction and their utilities should also be eliminated from the transaction's TU.

2.4 ADVANTAGES OF THE PROPOSED SYSTEM

- Unpromising items are also included in the result tree.
- Minimum item utilities are utilized to reduce utilities of local unpromising items.
- Mining HUI is effective for pruning candidate itemsets in UP tree construction.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 HARDWARE REQUIREMENTS

This section gives the details and specification of the hardware on which the system is expected to work.

Processor	:	Intel i7 processor
Hard disk	:	1 TB
RAM	:	4 GB
Monitor	:	17" Color
Keyboard	:	Standard 102 keys
Mouse	:	LOGI TECH (3 Buttons)

3.2 SOFTWARE REQUIREMENTS

This section gives the details of the software that are used for development.

Front-End	:	Net Beans 6.8 IDE
Coding Language	:	Java 1.6
Back-End	:	MS-SQL Server 2008
Operating System	:	Windows 10

CHAPTER 4

SOFTWARE DESCRIPTION

4.1 FRONT END

It is a Platform Independent. Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak was intended although the feature set better resembles. The following diagram shows the Java 2 Platform Standard Edition Architecture.

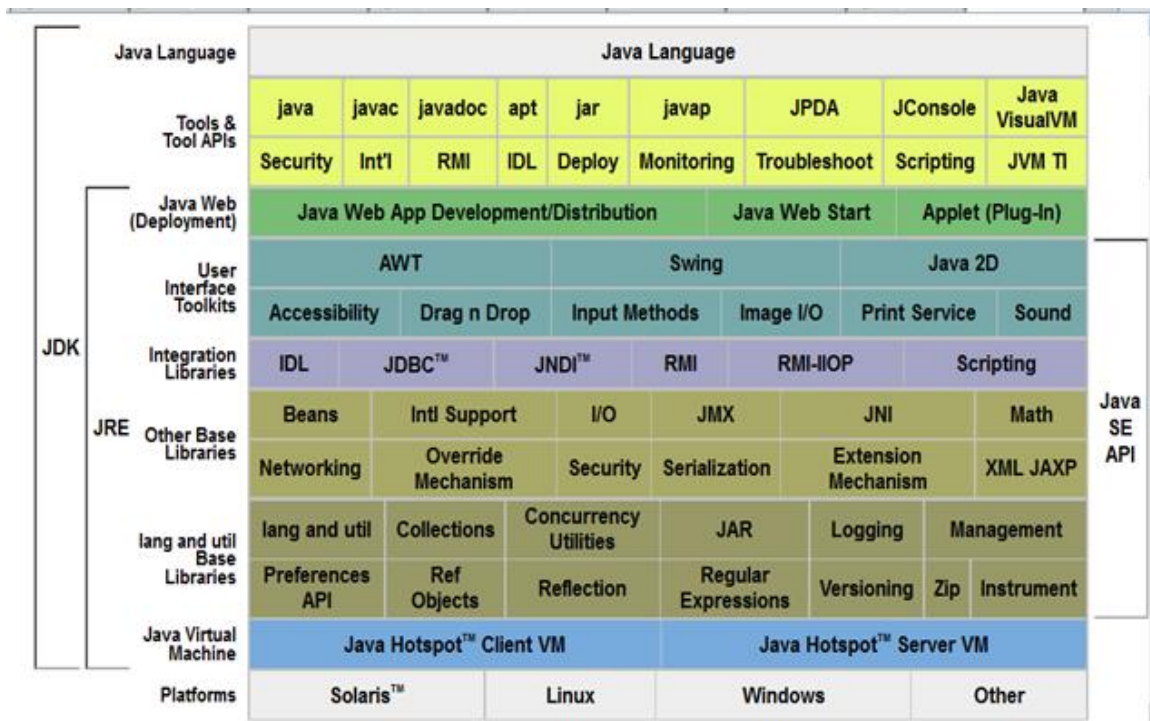


Fig. 4.1 Front End

The following are some of the features which have made Java one of the most popular among the various available programming languages.

Simple

Java was developed by taking the best points from other programming languages. Java therefore utilizes algorithms and methodologies that are already proven. Error prone tasks such as pointers and memory management have either been eliminated or are handled

by the Java environment automatically rather than by the programmer. Since Java is primarily a derivative which most programmers are conversant with, it implies that Java has a familiar feel rendering it easy to use.

Object Oriented

Even though Java has the look and feel it is a wholly independent language which has been designed to be object-oriented from the ground up. In object-oriented programming (OOP), data is treated as objects to which methods are applied. Java's basic execution unit is the class. Advantages of OOP include: reusability of code, extensibility and dynamic applications.

Distributed

Commonly used Internet protocols such as calls for network access are built into Java. Internet programmers can call on the functions through the supplied libraries and be able to access files on the Internet as easily as writing to a local file system.

Interpreted

When Java code is compiled, the compiler outputs the Java Bytecode which is an executable for the Java Virtual Machine. The Java Virtual Machine does not exist physically but is the specification for a hypothetical processor that can run Java code. The byte code is then run through a Java interpreter on any given platform that has the interpreter ported to it. The interpreter converts the code to the target hardware and executes it.

Robust

Java compels the programmer to be thorough. It carries out type checking at both compile and runtime making sure that every data structure has been clearly defined and typed. Java manages memory automatically by using an automatic garbage collector. The garbage collector runs as a low priority thread in the background keeping track of all objects and references to those objects in a Java program. When an object has no more references, the garbage collector tags it for removal and removes the object either when

there is an immediate need for more memory or when the demand on processor cycles by the program is low.

Secure

The Java language has built-in capabilities to ensure that violations of security do not occur. Consider a Java program running on a workstation on a local area network which in turn is connected to the Internet. Being a dynamic and distributed computing environment, the Java program can, at runtime, dynamically bring in the classes it needs to run either from the workstation's hard drive, other computers on the local area network or a computer thousands of miles away somewhere on the Internet.

This ability of classes or applets to come from unknown locations and execute automatically on a local computer sounds like every system administrator's nightmare considering that there could be lurking out there on one of the millions of computers on the Internet, some viruses, Trojan horses or worms which can invade the local computer system and wreak havoc on it.

Java goes to great lengths to address these security issues by putting in place a very rigorous multilevel system of security:

First and foremost, at compile time, pointers and memory allocation are removed thereby eliminating the tools that a system breaker could use to gain access to system resources. Memory allocation is deferred until runtime.

Even though the Java compiler produces only correct Java code, there is still the possibility of the code being tampered with between compilation and runtime.

Java guards against this by using the byte code verifier to check the byte code for language compliance when the code first enters the interpreter, before it ever even gets the chance to run.

The byte code verifier ensures that the code does not do any of the following:

- Forge pointers
- Violate access restrictions
- Incorrectly access classes
- Overflow or underflow operand stack

- Use incorrect parameters of byte code instructions
- Use illegal data conversions

At runtime, the Java interpreter further ensures that classes loaded do not access the file system except in the manner permitted by the client or the user.

Sun Microsystems will soon be adding yet another dimension to the security of Java. They are currently working on a public-key encryption system to allow Java applications to be stored and transmitted over the Internet in a secure encrypted form.

Architecturally Neutral

The Java compiler compiles source code to a stage which is intermediate between source and native machine code. This intermediate stage is known as the byte code, which is neutral. The byte code conforms to the specification of a hypothetical machine called the Java Virtual Machine and can be efficiently converted into native code for a particular processor.

Portable

By porting an interpreter for the Java Virtual Machine to any computer hardware/operating system, one is assured that all code compiled for it will run on that system. This forms the basis for Java's portability. Another feature which Java employs in order to guarantee portability is by creating a single standard for data sizes irrespective of processor or operating system platforms.

High Performance

The Java language supports many high-performance features such as multithreading, just-in-time compiling, and native code usage.

Java has employed multithreading to help overcome the performance problems suffered by interpreted code as compared to native code. Since an executing program hardly ever uses CPU cycles 100 % of the time, Java uses the idle time to perform the necessary garbage cleanup and general system maintenance that renders traditional interpreters slow in executing applications.

Multithreading is the ability of an application to execute more than one task (thread) at the same time e.g. a word processor can be carrying out spell check in one document and printing a second document at the same time.

Since the byte code produced by the Java compiler from the corresponding source code is very close to machine code, it can be interpreted very efficiently on any platform. In cases where even greater performance is necessary than the interpreter can provide, just-in-time compilation can be employed whereby the code is compiled at run-time to native code before execution.

An alternative to just-in-time compilation is to link in native C code. This yields even greater performance but is more burdensome on the programmer and reduces the portability of the code.

Dynamic

By connecting to the Internet, a user immediately has access to thousands of programs and other computers. During the execution of a program, Java can dynamically load classes that it requires either from the local hard drive, from another computer on the local area network or from a computer somewhere on the Internet.

Applets

A web page received from the web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the web browser.

However, client systems will likely need the Java Plug-in and possibly a security policy file in order for the applet to successfully execute in the web browser. Web components are the preferred API for creating a web client program because no plug-ins or security policy files are needed on the client systems.

Application Clients

An application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. It typically has a graphical user interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.

NetBeans 6.8 IDE

NetBeans 6.8 is a tool that is used to edit the java programs that are created and has a very user friendly environment and makes the process to be very simple. Some of the advantages of using this application and its features are explained below.

In the NetBeans environment, every file is stored inside a project. This means that every document, folder, code file (.java) and compiled code (.class) has to be contained by a project. Hence the NetBeans project framework is explained. It is necessary to create a new project to develop a new Java program, but also to edit existing code (i.e., a .java file stored inside a diskette). To create a new project select from the main menu line "File > New > Project ..." You may also select "New > Project ..." by right clicking any point of the package explorer window.

To create or edit Java programs, a Java project should be created. Java project can also store program related information, such as code documentation and other related files. Simple projects should only be created to store documents and other files with no Java code. Plug-in Development projects are used to add new modules and facilities to the NetBeans environment. NetBeans Modeling Framework projects are used to create analysis and design models. Source folders are the folders that store the Java source code files (the .java files). By placing your Java files inside a source folder, they will be automatically compiled.

Another striking feature of this version is that every imported element is duplicated. This means that the programmer may delete the copy of these files that Eclipse manages, and still have the original file. Nevertheless, when importing an existing Eclipse project, the project contents will not be duplicated. Hence, special care needs to be taken when deleting files from NetBeans workspace, especially while deleting those NetBeans imported projects, as no other copies may exist.

The files created using NetBeans may also be exported as normal files (File system), Java Jar files and even Zip files. This is achieved through a process similar to the importing process already detailed, selecting the "Export" option. It is also possible copying, cutting and pasting files and folders from NetBeans to your file system (i.e., Windows explorer)

and vice versa (i.e., selecting a file, pressing CTRL+C, then choosing a destination folder and pressing CTRL+V).

4.2 BACKEND

Microsoft SQL Server is a relational model database server produced by Microsoft. Its primary query languages are T-SQL and ANSI SQL. The OLAP Services feature available in SQL Server version 7.0 is now called SQL Server 2000 Analysis Services. The term OLAP Services has been replaced with the term Analysis Services. Analysis Services also includes a new data mining component. The Repository component available in SQL Server version 7.0 is now called Microsoft SQL Server 2000 Meta Data Services. References to the component now use the term Meta Data Services. The term repository is used only in reference to the repository engine within Meta Data Services

Internet Integration

The SQL Server 2000 database engine includes integrated XML support. It also has the scalability, availability, and security features required to operate as the data storage component of the largest Web sites. The SQL Server 2000 programming model is integrated with the Windows DNA architecture for developing Web applications, and SQL Server 2000 supports features such as English Query and the Microsoft Search Service to incorporate user-friendly queries and powerful search capabilities in Web applications.

Scalability and Availability

The same database engine can be used across platforms ranging from laptop computers running Microsoft Windows® 98 through large, multiprocessor servers running Microsoft Windows 2000 Data Center Edition. SQL Server 2000 Enterprise Edition supports features such as federated servers, indexed views, and large memory support that allow it to scale to the performance levels required by the largest Web sites.

Enterprise-Level Database Features

The SQL Server 2000 relational database engine supports the features required to support demanding data processing environments. The database engine protects data integrity while minimizing the overhead of managing thousands of users concurrently

modifying the database. SQL Server 2000 distributed queries allow you to reference data from multiple sources as if it were a part of a SQL Server 2000 database, while at the same time, the distributed transaction support protects the integrity of any updates of the distributed data. Replication allows you to also maintain multiple copies of data, while ensuring that the separate copies remain synchronized. You can replicate a set of data to multiple, mobile, disconnected users, have them work autonomously, and then merge their modifications back to the publisher.

Ease of installation

SQL Server 2000 includes a set of administrative and development tools that improve upon the process of installing, deploying, managing, and using SQL Server across several sites. SQL Server 2000 also supports a standards-based programming model integrated with the Windows DNA, making the use of SQL Server databases and data warehouses a seamless part of building powerful and scalable systems. These features allow you to rapidly deliver SQL Server applications that customers can implement with a minimum of installation and administrative overhead.

Data warehousing

SQL Server 2000 includes tools for extracting and analyzing summary data for online analytical processing. SQL Server also includes tools for visually designing databases and analyzing data using English-based questions.

CHAPTER 5

PROJECT DESCRIPTION

5.1 PROBLEM DEFINITION

Frequent Pattern Growth algorithm finds frequent item sets without generating any candidate item sets and scans database just twice. Growth algorithm concentrates only the item in the transaction and not the utility of the item. All the products are treated uniformly and all the rules are mined based on the count of the product. So the concept of weighted items was introduced.

Weight association rule mining considers the importance of items, such as transaction databases, items. Quantities of transactions are not taken into considerations yet.

UP Growth achieves better performance than Frequent Pattern Growth and Weight association rule mining. Growth algorithm is mainly composed of two mining phases such as Unpromising and Promising Item. In the first phase is used to eliminate unpromising items and sorting the remaining in a fixed order. In the second phase algorithm is used to construct global tree. It reduces utilities of the nodes that are closer to root of the global tree.

The main purpose of this research is used to eliminate time complexity rate in tree construction process by using two strategies, namely Unpromising items and Promising Items. It also used to reduce number scans to the database. To maintain the information of transactions and HUI.

5.2 OVERVIEW OF THE PROJECT

Frequent Pattern Growth algorithm that combines projection with the use of the tree data structure to compactly store in memory the item sets of the original database.

Weight association rule mining HUIM considers the importance of items, such as transaction databases, items.

Although two-phase algorithm reduces search space by using TU property, it still generates too many candidates to obtain TWU and requires multiple database scans. To

overcome this problem Isolated Items Discarding Strategy to reduce the number of candidates. By pruning isolated items during level-wise search, the number of candidate item sets for TU can be reduced. However, this algorithm still scans database for several times and uses a candidate generation and test scheme to find high utility item sets.

Utility Growth achieves better performance than Frequent Growth and Weight association rule mining. Utility Growth algorithm is mainly composed of two mining phases such as Unpromising utility and Promising Itemsets these two strategies are used to minimize the overestimated utilities stored in the nodes of global tree.

5.3 MODULE DESCRIPTION

- USERS
- ITEMS/PROFIT
- TRANSACTION
- UTILITY
- CALCULATE UTILITY THRESHOLD
- MINIMUM WEIGHT UTILITY THRESHOLD
- CALCULATE WEIGHT UTILITY
- UTILITY PATTERN TREE

USERS

In this module, user id and name are added into 'Users' table. In the list box given, all the user id and name already saved are listed from which the records can be selected and new values can be updated.

ITEMS/PROFIT

In this module, item name and profit are added into 'Profit' table. In the list box given, all the user id and name already saved are listed from which the records can be selected and new values can be updated.

TRANSACTION

In this module, transaction id, user, item and quantity are added into ‘Transactions’ table. In the grid view control, all the records already saved are displayed from which the records can be modified and new values can be updated.

UTILITY THRESHOLD

In this module, threshold value is given and saved in ‘Utility Threshold’ table.

CALCULATE UTILITY THRESHOLD

In this module, transaction id, itemset and transaction utility value is calculated and displayed.

MINIMUM WEIGHT UTILITY THRESHOLD

In this module, threshold value is given and saved in ‘Weight Utility Threshold’ table.

CALCULATE MINIMUM WEIGHT UTILITY THRESHOLD

In this module, transaction id, itemset and transaction weight utility value is calculated and displayed.

UTILITY PATTERN TREE

To facilitate the mining performance and avoid scanning original database repeatedly, we use a compact tree structure, named UP-Tree, to maintain the information of transactions and HUI. Two strategies are applied to minimize the overestimated utilities stored in the nodes of global Utility Pattern Tree. In following sections, the elements of Tree are first defined. Next, the two strategies are introduced.

In an UP Tree, each node consists of a table named header table is employed to facilitate the traversal of Tree. In header table, each entry records an item name, an overestimated utility, and a link. The points to the last occurrence of the node which has the same item as the entry in the Tree. By following the links in header table and the nodes in Tree, the nodes having the same name can be traversed efficiently. If the items are treated non-uniformly .the introduction of the weights made the mining of the association rules possible to be interactive with the users. It can provide a means to apply the knowledge to

the items. System analysis decide the following input design details like, what data to input, what medium to use, how the data should be arranged or coded, data items and transactions needing validations to detect errors and at last the dialogue to guide user in providing input. Accumulated TWU values are still maintained in node utilities. It employs a novel vertical data structure to represent utility information, called utility list. It needs only a single database scan to create and maintain such utility-lists of item- sets containing single items. A large number of candidates are generated in the mining process and the performance is degraded. Therefore, researchers came up with improved methods where the generation of potential HUI is not required. HUI without both candidate generation and additional database rescans which enumerates an itemset as using the TWU properties, it recursively filters out irrelevant items when growing HUI with sparse data.

ADMIN

This table is used to store the user name and password details.

Table 5.1 Admin

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
Username	Varchar	15	Unique
Password	Varchar	15	Not Null

DATA ITEMS

This table is used to store the data item details used during the frequent pattern tree construction.

Table 5.2 Data Items

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
SNo	Int	4	Unique
Row Index	Int	4	Row Index Details
Data Item	Varchar	50	Data Item Details
Occurrences	Int	4	Occurrences Details

PROFIT

This table is used to store the item details with their profit values.

Table 5.3 Profit

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
SNo	Int	4	Unique
Item	Varchar	50	Not Null
Profit	Varchar	50	Not Null

USERS

This table is used to store the user details who make the transactions.

Table 5.4 Users

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
UserId	Varchar	10	Unique
Username	Varchar	50	User Name Details

UTILITY THRESHOLD

This table is used to store the utility threshold details used during high utility pattern tree construction.

Table 5.5 Utility Threshold

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
Utility Threshold	Int	4	Utility Threshold Values

WEIGHT UTILITY THRESHOLD

This table is used to store the utility threshold details used during high utility pattern tree construction.

Table 5.6 Weight Utility Threshold

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
Weight Utility Threshold	Int	4	Weight Utility Threshold Values

TRANSACTIONS

This table is used to store the transaction details from which high utility pattern tree construction is being made.

Table 5.7 Transaction

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
SNo	Int	4	Unique
Transaction ID	Int	4	Transaction ID
UserId	Varchar	10	User ID Details

TREE

This table is used to store the frequent pattern tree record details.

Table 5.8 Tree

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
SNo	Int	4	Unique
Node ID	Varchar	50	Node Id details
Node Level	SmallInt	2	Node Level Details
ParentNodeSNo	Int	4	Parent Node SNO
Frequency	SmallInt	2	Frequency Values
Visited	Bit	1	Visited ID

RECORDED ITEMS

This table is used to store the reordered data item details used during the frequent pattern tree construction.

Table 5.9 Recorded Items

FIELD NAMES	DATA TYPE	SIZE	DESCRIPTION
SNo	Int	4	Unique
Row Index	Int	4	Row Index Details
Data Item	Varchar	50	Data Item Details
Qty	Int	4	Quantity Details

CHAPTER 6

SYSTEM DESIGN AND DEVELOPMENT

6.1 SYSTEM ARCHITECTURE

Efficient discovery of itemsets with high utility like profits deals with the mining HUI from a transaction database although a number of relevant approaches have been proposed in recent years, these algorithm incur the problem of producing a large number of candidate itemsets for HUI and probably degrades the mining performance in terms of execution time and memory space. In this paper, we propose two algorithms, viz., utility pattern growth UP Growth. Improved UP Growth, for mining HUI with a set of effective strategies for pruning candidate itemsets. The information is maintained in a compact tree based data structure utility pattern tree it scan the original database twice to manage data structured way. Proposed algorithms, especially Improved not only reduce the number of candidates effectively but also outperform other algorithms substantially in terms of runtime and memory consumption, especially when databases contain lots of long transactions. HUIM techniques are exploited for finding profitable patterns in transactional databases. In the online shopping scenario, besides recommending the correlated items, the sales managers are also interested in suggestions of HUI that can be placed together for attracting customers' attention and obtain higher profits. This is reflected in a variety of business applications and scientific domains, where HUIM techniques are exploited for finding profitable patterns in transactional databases. But this type of approach leads to memory and running time overhead, due to the overly large number of candidates generated in the first phase. Algorithms in the second category try to avoid the above problem by utilizing various techniques, data structures, and properties, which limit the number of candidates and prune further the unpromising candidates. Likewise, it is inadequate to emphasize just popular products that are selling in large quantities, as these might have lower unit profits and would require an order of magnitude increase in purchase.

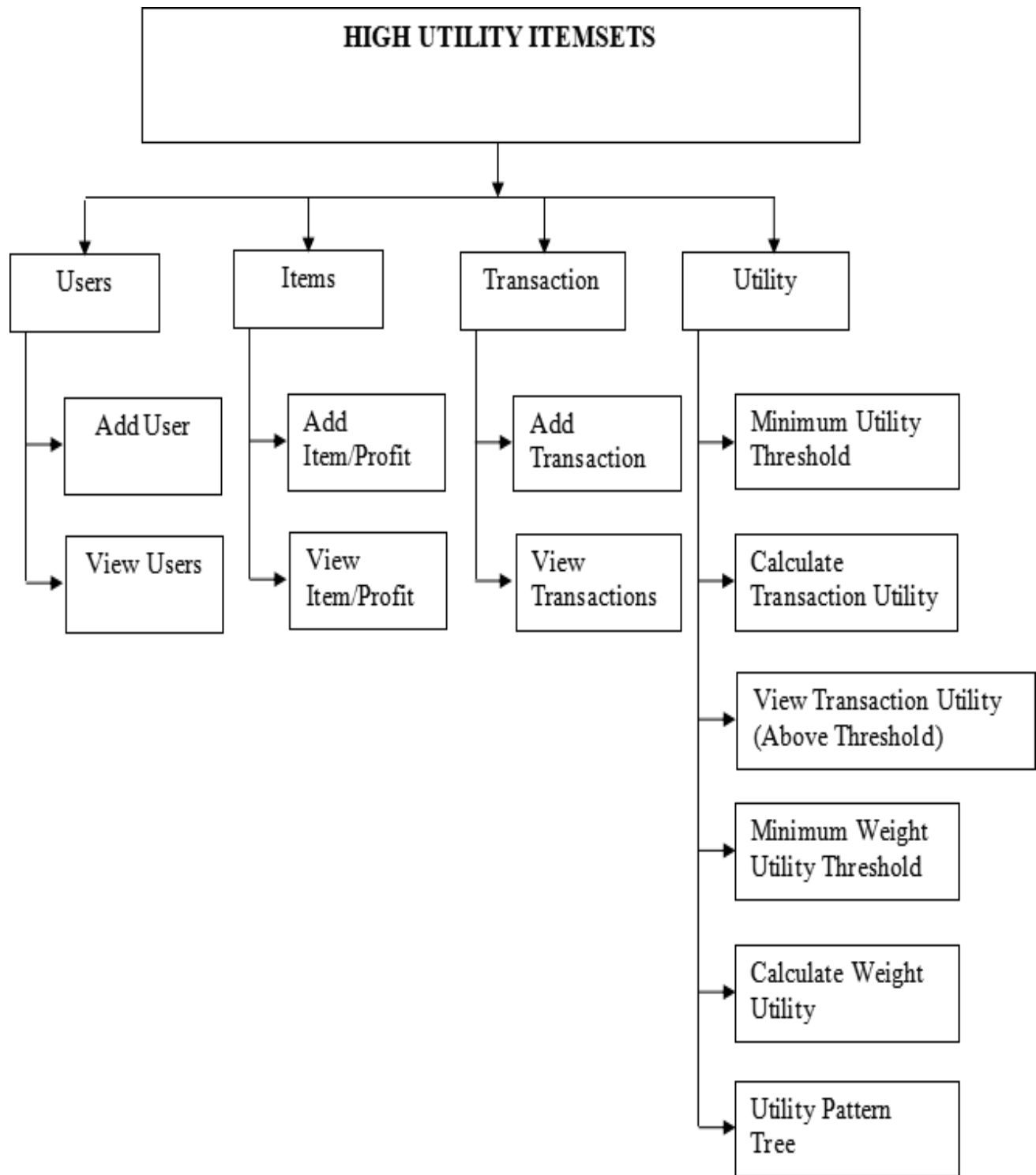


Fig. 6.1 System Architecture

6.2 UML DIAGRAM

The elements are like components which can be associated in different ways to make a complete UML picture, which is known as diagram. Thus, it is very important to understand the different diagrams to implement the knowledge in real life systems.

Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. If we look around, we will realize that the diagrams are not a new concept but it is used widely in different forms in different industries.

We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system.

Use Case Diagram

Use case diagram is a graph of the actors, set of the use cases enclosed in by system boundary, communication (participation) association between the actors and the cases and a generalization among the use cases.

Actors

An actor represents a set of roles that user of a use case play when interacting with the use cases. Actor identified here is Administrator.

Use case

A use case is a description of a set of sequence of action that a system Performs to yield result of value to an actor

- The Login use case is to describe that, the user should choose his/her category whether he/she is user or administrator.
- The use case display the details and the offered with their instruction and also about the payment.
- Rare itemsets provide useful information in different decision-making domains such as business transactions, medical, security, fraudulent transactions and retail communities.

- In a supermarket, customers purchase microwave ovens or frying pans rarely as compared to bread, washing powder, soap. But the former transactions yield more profit for the supermarket.
- A retail business may be interested in identifying its most valuable customers i.e. who contribute a major fraction of overall company profit.

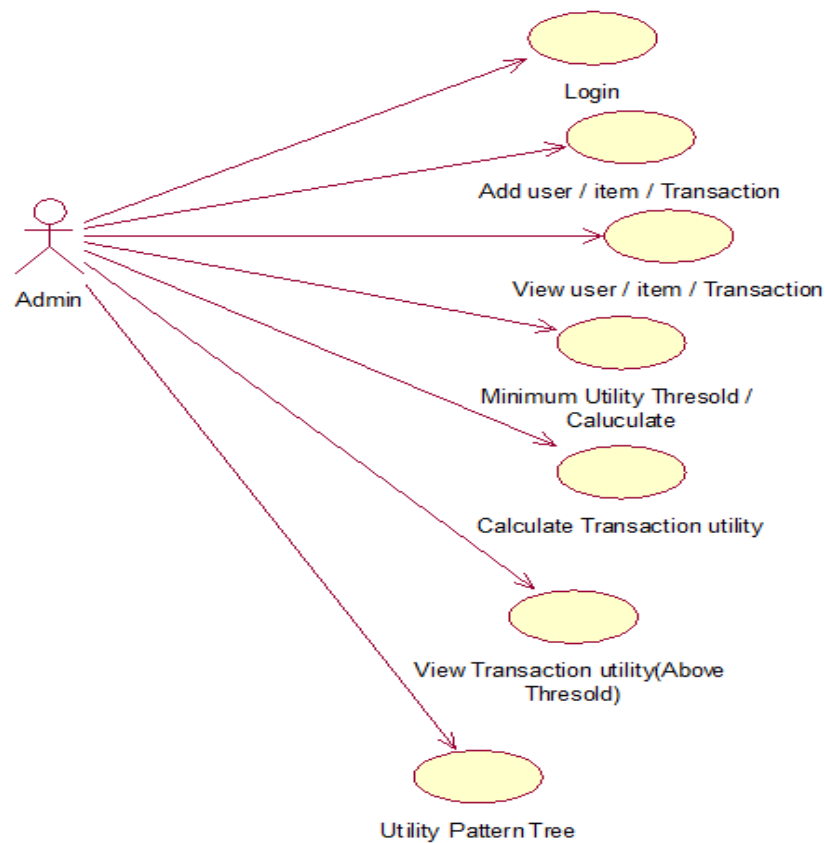


Fig. 6.2 Usecase Diagram

Sequence Diagram

The user has to login and fill up the chat form for their the users need and submit it. For each chat a reference number is given. The administrator views the details with that reference number and he updates the details with additional information.

But the former transactions yield more profit for the supermarket. Similarly, the high-profit rare itemsets are found to be very useful in many application areas.

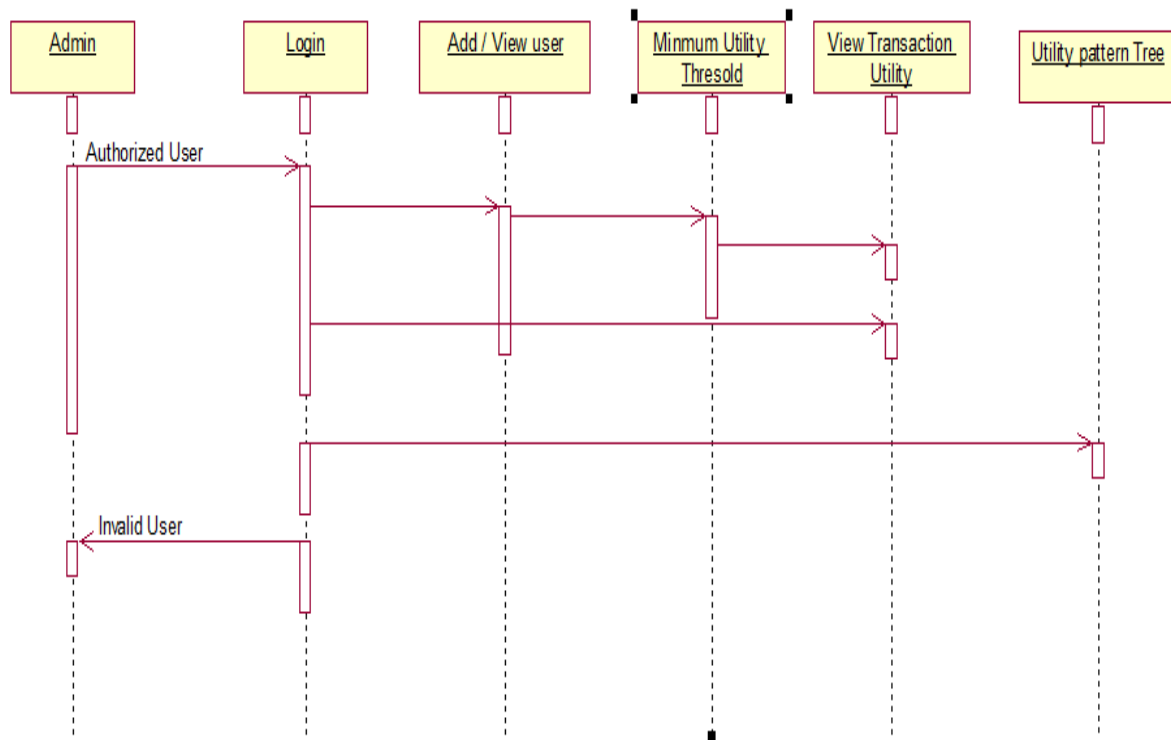


Fig. 6.3 Sequence Diagram

Activity Diagram

The activity diagram describes the sequence of activities with support for both conditional and parallel behavior. The activity diagram is used to describe the various activities taking place in an profit items. Here in our transaction, we have various activities starting from login.

After login, the user selection activity gets performed, where the user can be a user or admin If the user is a users,then they have to enter their password and those details are valid they can access the system. They can register the any user and get the reference number.

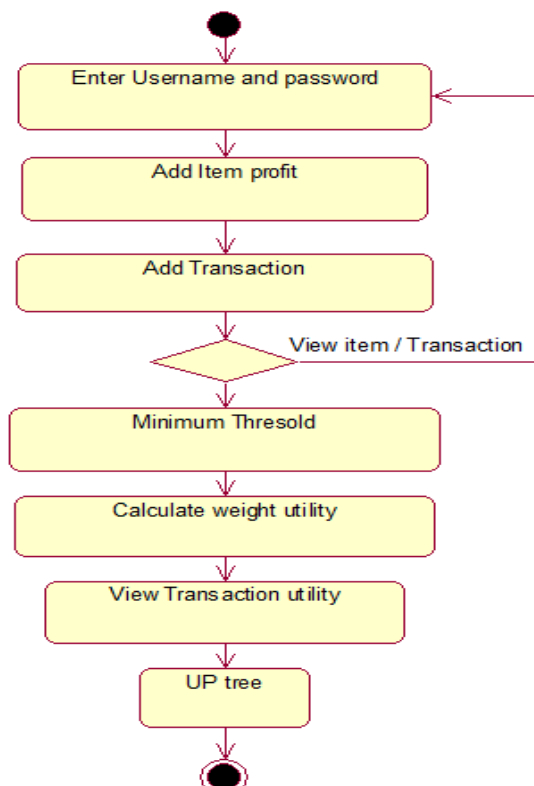


Fig. 6.4 Activity Diagram

Class Diagram

The class diagram involves various classes used in project and their attributes. It also explains various class members details.

The five class in this project are,

1. Admin detail
2. Users detail
3. Items detail
4. Transaction detail
5. Utility

Each class has its own attributes and operations.

Admin class-the attributes defined is user and admin

Users class-It contains the attributes of the profits and own details.

Items class-It contains the attributes of the Transaction.

Transaction class-the attributes of user can be added to the admin class.

Utility-It contains the attributes utility threshold details.

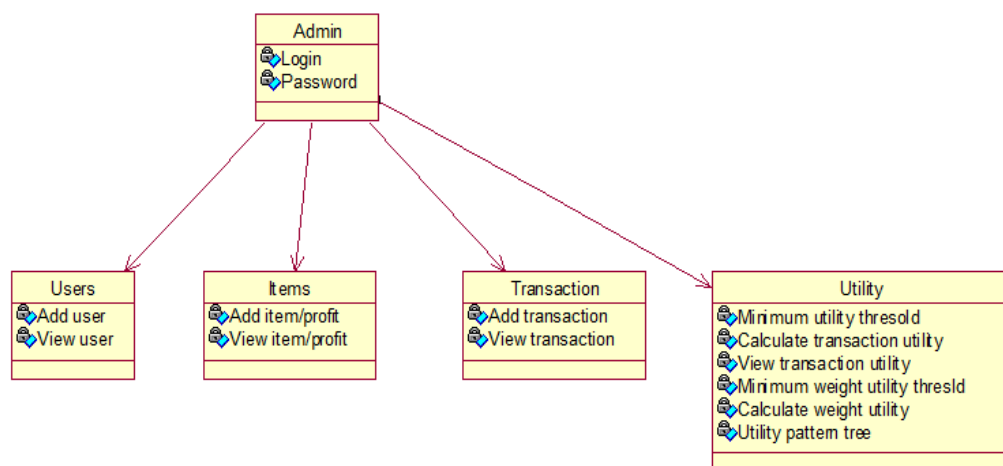


Fig. 6.5 Class Diagram

CHAPTER 7

SYSTEM TESTING

This project has undergone the following testing procedures to ensure its correctness.

- Unit Testing
- User Acceptance Testing
- Integration Testing
- Validation Testing
- White Box Testing
- Black Box Testing

Unit Testing

In unit testing, we have to test the programs making up the system. For this reason, Unit testing sometimes called as Program testing. The software units in a system are the modules and routines that are assembled and integrated to perform a specific function, Unit testing first on the modules independently of one another, to locate errors. This enables, to detect errors in coding and logic that are contained with the module alone. The testing was carried out during programming stage itself.

User Acceptance Testing

In these testing procedures the project is given to the customer to test whether all requirements have been fulfilled and after the user is fully satisfied. The project is perfectly ready. If the user makes request for any change and if they found any errors those all errors has to be taken into consideration and to be correct it to make a project a perfect project.

Integration Testing

Integration testing is a systematic technique for constructing the program structure. While at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design.

In the project, after integrating the all modules like account creation, transaction data update and amount aggregation are tested with their integration and that could integrated and manipulated with respect to the to and fro in between modules.

Validation Testing

It is said that validation is successful when the software functions in a systematic manner that can be reasonably accepted by the customers. This type of testing is very important because it is the only way to check whether the requirements given by user have been completely fulfilled.

White Box Testing

White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Test cases can be derived that

1. Guarantee that all independent paths within a module have been exercised at least once,
2. Exercise all logical decisions on their true and false sides,
3. Execute all loops at their boundaries and within their operational bounds, and
4. Exercise internal data structures to ensure their validity.

Black Box Testing

Black box testing attempts to derive sets of inputs that will fully exercise all the functional requirements of a system. It is not an alternative to white box testing. This type of testing attempts to find errors in the following categories:

1. Incorrect or missing functions,
2. Interface errors,
3. Errors in data structures or external database access,
4. Performance errors, and initialization and termination errors.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

HUP is another algorithm that is able to discover HUI without maintaining candidates, which targets the root cause of candidate generation with the existing approaches. Also, it designs a HUIM approach that enumerates an itemset as a prefix extension of another itemset. It efficiently computes the utility of each enumerated itemset and the upper bound on the utilities of the prefix-extended itemsets. This utility upper bound is much tighter than TWU, and it is further tightened by iteratively filtering out irrelevant items when growing high utility itemsets with sparse data. Moreover, it uses less memory space than tree structures used in the above mentioned algorithms HUP was shown to be more efficient than UP-Growth and Two-Phase. But later algorithms EFIM which is presented in the have been HUP reported.

EFIM is an one-phase HUIM algorithm that proposes efficient database projection and transaction merging techniques for reducing the cost of database scans. EFIM efficiently merges transactions that are identical in each projected database through a linear time and space implementation. As larger itemsets are explored, both projection and merging reduce the size of the database. All items in the database are renamed as consecutive integers. Reuses the same utility-bin array multiple times by reinitializing values before each use.

FIM but it associates weights with the items in transactions - if these weights have unit values it then degenerates to. It is known that HUIM is a more challenging problem than, since the utility of an itemset is neither monotonic nor anti-monotonic, as mentioned above. As a result, techniques cannot be directly applied to HUIM. Started as a phase in the discovery of association rules but has since been generalized, independent of these, to many other patterns, such as frequent sequences, periodic patterns, frequent subgraphs and bridging rules.

FUTURE ENHANCEMENTS

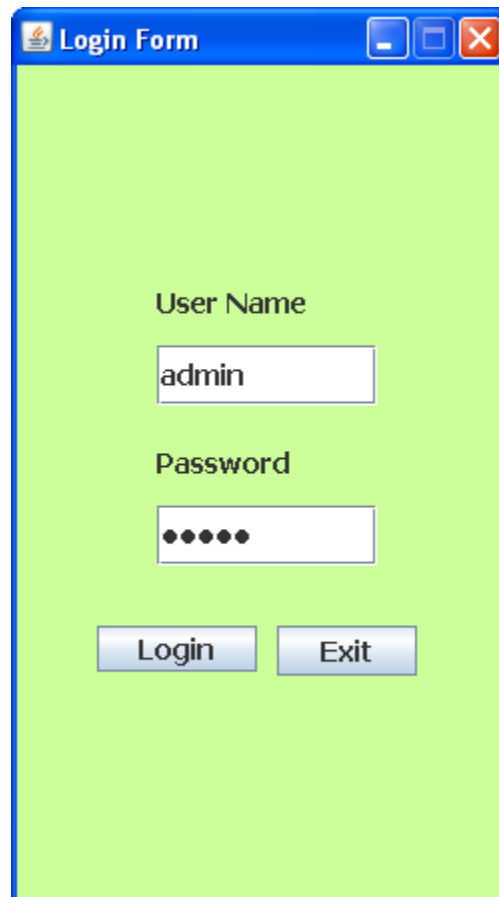
The proposed work become more useful if the below enhancements are made future.

- In Future work, tree will be constructed for both frequent and infrequent moved items in dataset.
- In addition the advanced Mining algorithm can be applied to other applications with the aim to enhance precision for predicting used behaviors.
- UP Growth and HUP for discovering high utility itemset database.
- Future works include the extension of the temporal UP tree to mine noisy patterns and developing more efficient techniques to handle genomic data.
- Frequent pattern growth concentrates only the item in transaction, all the products are treated uniformly.
- Frequent pattern growth that combines production with the use of the frequent pattern tree, the store in the memory itemsets of the original database.

APPENDICES

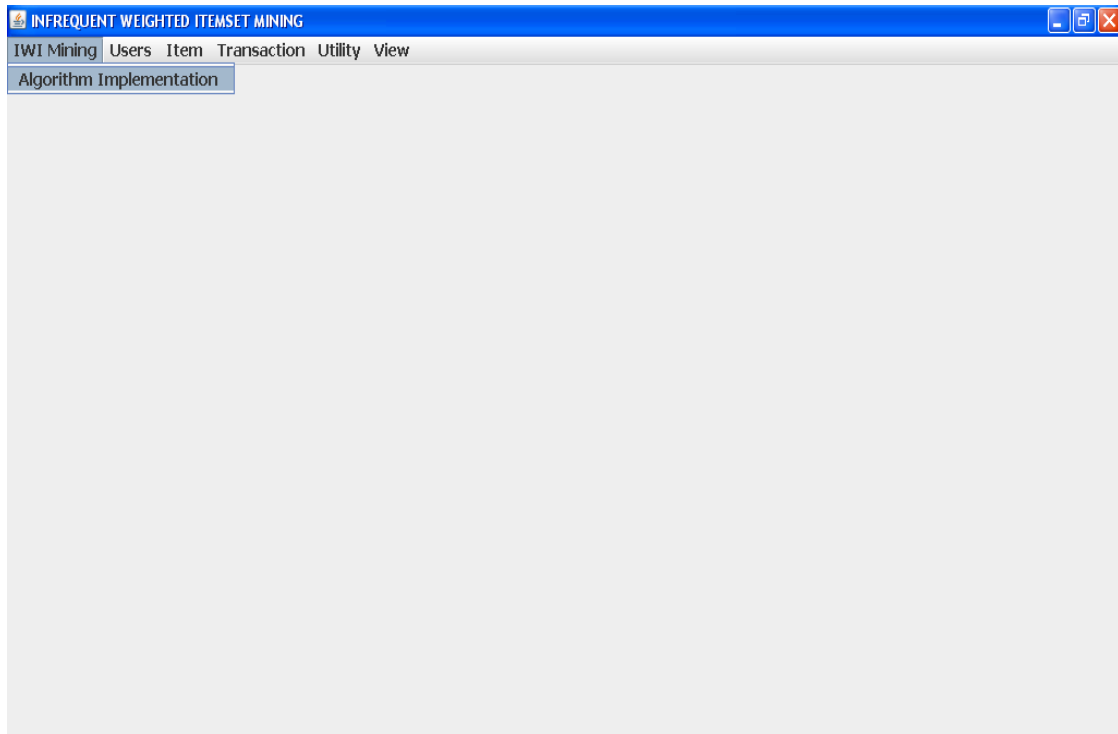
SCREENSHOTS

Login Form

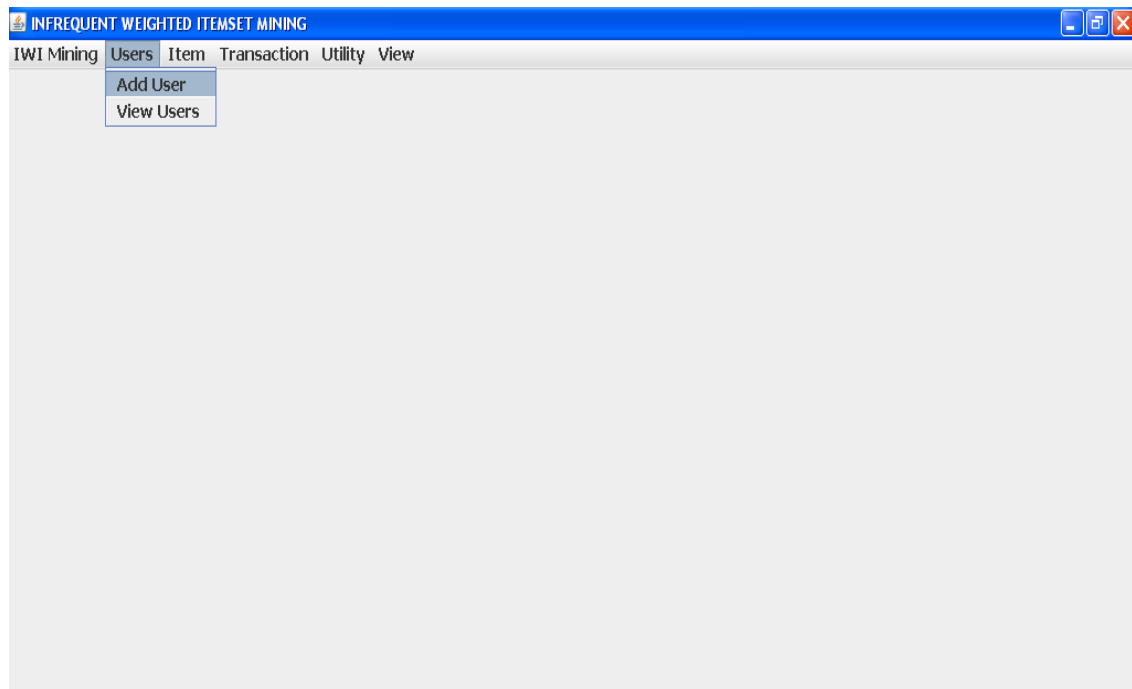


The screenshot shows a window titled "Login Form" with a blue title bar and standard Windows window controls (minimize, maximize, close). The window has a light green background. It contains two text input fields: the first is labeled "User Name" and contains the text "admin"; the second is labeled "Password" and contains six black dots. Below the input fields are two buttons: "Login" and "Exit".

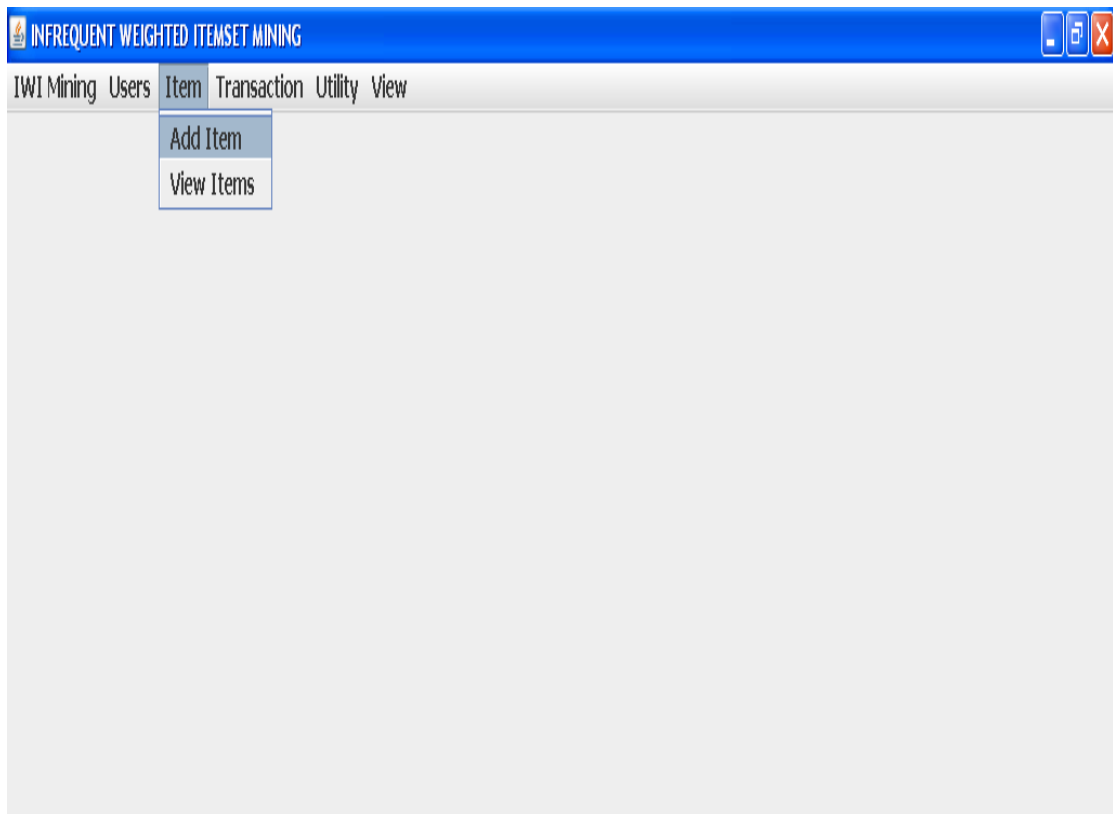
IWI Mining



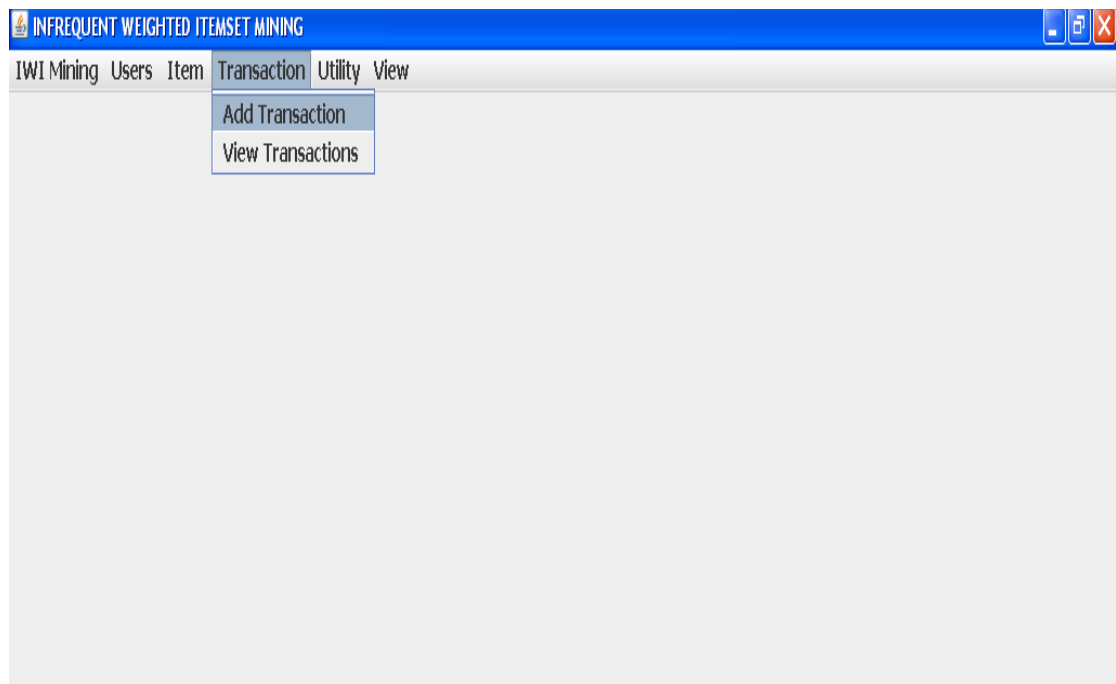
User Menu



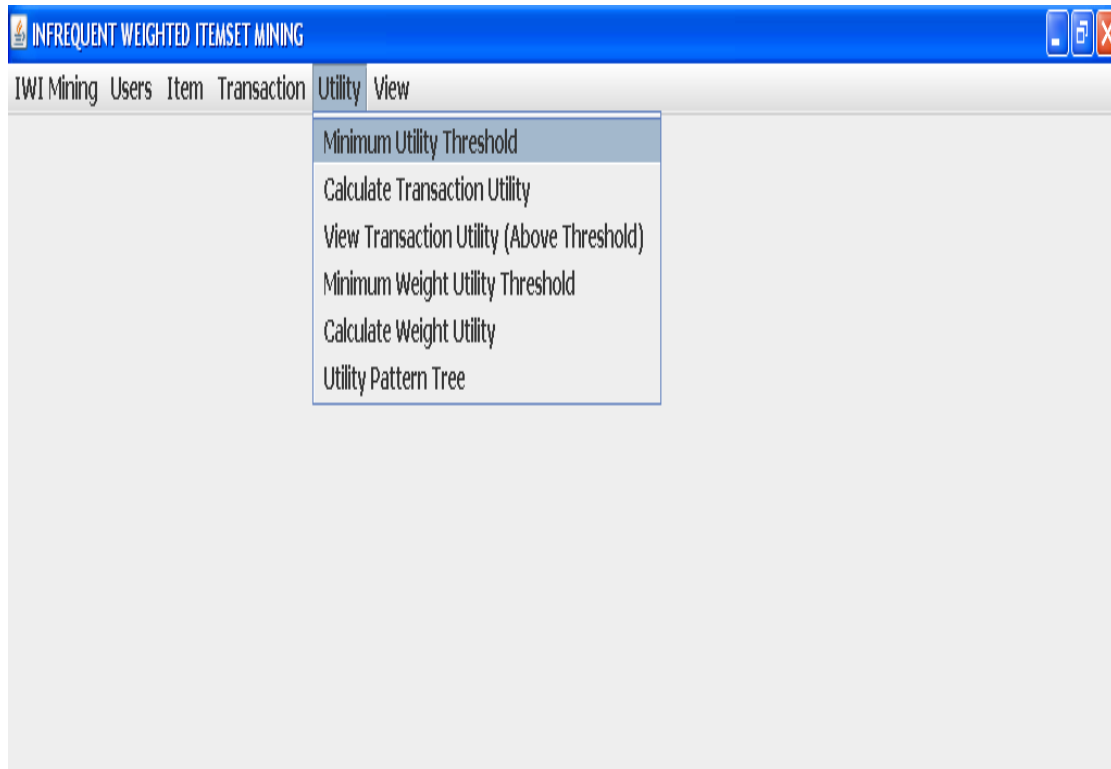
Item Menu



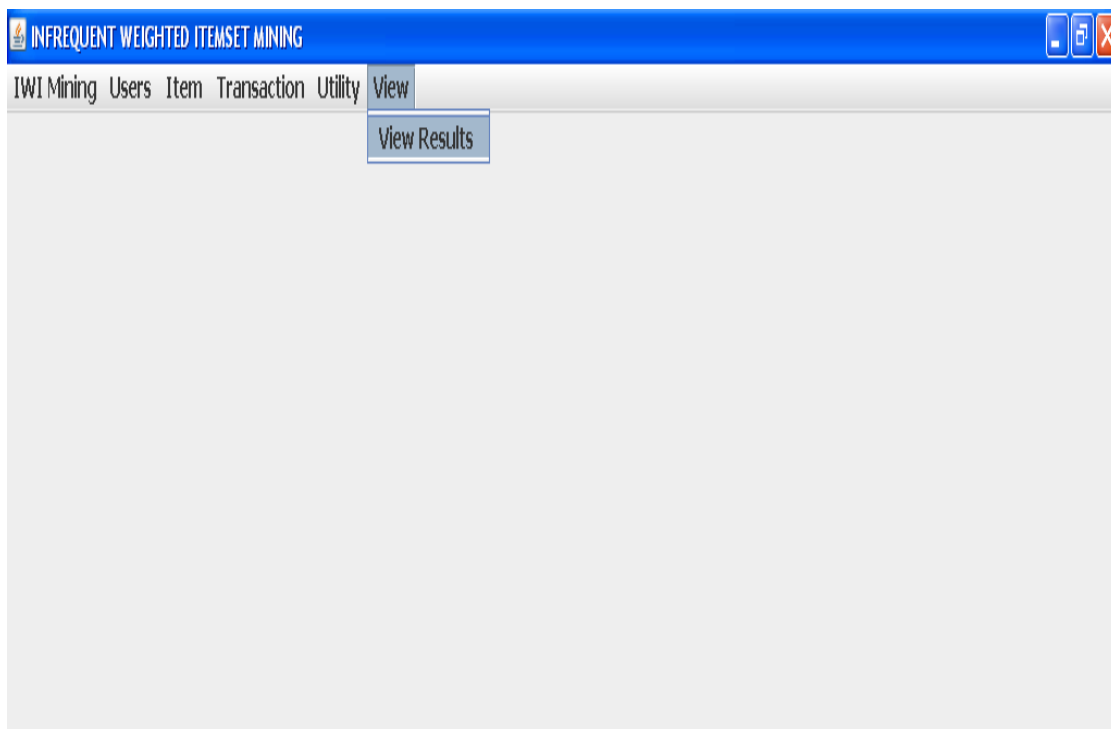
Transaction Menu



Utility



View Menu



Infrequent

Transactions

Support Count - Threshold

3

Entries

Add Data Items

1,11,21,31
23,13,21,14
1,2,23,38
22,33,11,5

Ordered InFrequent Traces

Calculate

InFrequent Traces

Ordered InFrequent Traces

Save

Close

IWI Mining

Sh...

IWI Mining

JTree

- colors
- sports
- food

Order Infrequent

Transactions

Support Count - Threshold

3

Entries

4,22,44,14

Add Data Items

1,11,21,31
23,13,21,14
1,2,23,38
22,33,11,5
4,22,44,14

Ordered InFrequent Traces

Calculate

InFrequent Traces

Ordered InFrequent Traces

33:1
31:1
23:2
22:2
21:2
44:1
14:2
5:1
4:1
13:1
11:2
38:1
2:1
1:2

1 11 21 31
23 21 14 13
1 23 38 2
22 11 5 33
22 14 44 4

Save

Close

IWI Mining

Sh...

IWI Mining

JTree

colors
sports
food

39

Show Mining

Transactions

Support Count - Threshold
3

Entries
4,22,44,14

Add Data Items

1,11,21,31
23,13,21,14
1,2,23,38
22,33,11,5
4,22,44,14

Ordered InFrequent Traces

Calculate

InFrequent Traces

Ordered InFrequent Traces

33:1
31:1
23:2
22:2
21:2
44:1
14:2
5:1
4:1
13:1
11:2
38:1
2:1
1:2

1 11 21 31
23 21 14 13
1 23 38 2
22 11 5 33
22 14 44 4

Save

Close

IWI Mining

Sh...

IWI Mining

Root

22 2~1334
14 1~1339
11 1~1338
5 1~1343
33 1~1348
23 1~1333
21 1~1336
14 1~1341
13 1~1346
1 2~1332
23 1~1337
11 1~1335
21 1~1340
31 1~1345

40

SOURCE CODE

```
import javax.swing.*;
import java.sql.*;
import java.io.*;
import javax.imageio.*;
import java.awt.*;

public class LoginForm extends javax.swing.JFrame {

    public LoginForm() {
        initComponents();

        setSize(250, 448);
        setResizable(false);
        //setUndecorated(true);
        SetResizable (false);
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        // determine the new location of the window
        int w = this.getSize().width;
        int h = this.getSize().height;
        int x = (dim.width-w)/2;
        int y = (dim.height-h)/2;

        // Move the window
        this.setLocation(x, y);
        Image backgroundImage =null;

        try
        {
```

```

        java.io.File f=new File(".");
        backgroundImage=  ImageIO.read(new File(f.getAbsolutePath() +
"\images\\1.jpg"));
// JOptionPane.showMessageDialog(null,backgroundImage.toString());
        Graphics g=jPanel1.getGraphics();
        g.drawImage(backgroundImage,0,0,null);
        jPanel1.repaint();
    }
    catch(Exception ex)
    {
JOptionPane.showMessageDialog(this,ex.getMessage());
    }
    // JOptionPane.showMessageDialog(null,"Amman");
}

```

```

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    btnLogin = new javax.swing.JButton();
    btnClose = new javax.swing.JButton();
    jLabel4 = new javax.swing.JLabel();
    jPasswordField1 = new javax.swing.JPasswordField();
    jTextField1 = new javax.swing.JTextField();
    jLabel3 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();

```

```

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Login Form");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowIconified(java.awt.event.WindowEvent evt) {
        Iconified(evt);
    }
});
getContentPane().setLayout(null);

jPanel1.setBackground(new java.awt.Color(204, 255, 153));
jPanel1.setLayout(null);

btnLogin.setFont(new java.awt.Font("Tahoma", 1, 14));
btnLogin.setText("Login");
btnLogin.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnLoginActionPerformed(evt);
    }
});
jPanel1.add(btnLogin);
btnLogin.setBounds(40, 280, 80, 23);

btnClose.setFont(new java.awt.Font("Tahoma", 1, 14));
btnClose.setText("Exit");
btnClose.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnCloseActionPerformed(evt);
    }
}

```

```

});
jPanel1.add(btnClose);
btnClose.setBounds(130, 280, 70, 25);

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel4.setText("Password");
jPanel1.add(jLabel4);
jLabel4.setBounds(70, 190, 90, 17);

jPasswordField1.setFont(new java.awt.Font("Tahoma", 1, 14));
jPanel1.add(jPasswordField1);
jPasswordField1.setBounds(70, 220, 110, 30);

jTextField1.setFont(new java.awt.Font("Tahoma", 1, 14));
jPanel1.add(jTextField1);
jTextField1.setBounds(70, 140, 110, 30);

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel3.setText("User Name");
jPanel1.add(jLabel3);
jLabel3.setBounds(70, 110, 100, 17);

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel2.setIcon(new
javax.swing.ImageIcon("F:\\MigratingDynamicTask\\images\\1.jpg")); // NOI18N
jPanel1.add(jLabel2);
jLabel2.setBounds(0, 0, 250, 420);

getContentPane().add(jPanel1);

```

```

jPanel1.setBounds(0, 0, 250, 420);

pack();
} // </editor-fold> //GEN-END: initComponents

private void btnCloseActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_btnCloseActionPerformed
//this.dispose();
System.exit(0);
} //GEN-LAST:event_btnCloseActionPerformed

private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_btnLoginActionPerformed

Connection con = null;
Statement st=null;

ResultSet r=null;
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    //Data Source Name -DSN Name
    //Starts->Programs->Control Panel->Administrative tools
    // ->ODBC Data Sources ->Microsoft Access Driver
    con = DriverManager.getConnection("jdbc:odbc:HUIMining", "", "");
    st = con.createStatement();

    r=st.executeQuery("Select Count(*) From Admin Where UserName='" +
jTextField1.getText() + "' and [Password]='" + jPasswordField1.getText() + "'");
    //      JOptionPane.showMessageDialog(this, "Saved");

```

```

int count=0;
if(r.next()) {
    count=r.getInt(1);
}
r.close();

if(count==1) {
    MainForm f=new MainForm();
    f.setSize(900,600);
    f.show();
    this.setVisible(false);

} else {
    JOptionPane.showMessageDialog(this,"Invalid
UserName/Password","Error",0,null );

}

}

catch (SQLException sqle) {
    System.out.println(sqle.getMessage() + " Connection Closed:");
    JOptionPane.showMessageDialog(this, sqle.getMessage());
} catch (ClassNotFoundException sqle) {
    System.out.println(sqle.getMessage() + " Connection Closed:");
    JOptionPane.showMessageDialog(this, sqle.getMessage());
}

```

```

finally {
    try {
        st.close();
        con.close();
    } catch (SQLException sqle) {
        System.out.println (sqle.getMessage () + "Connection Closed :");
    }
    st = null;
}
} //GEN-LAST:event_btnLoginActionPerformed

```

```

Private void Iconified (java.awt.event.WindowEvent evt) { //GEN-
FIRST:event_Iconified
    this.setState (JFrame.NORMAL);
} //GEN-LAST:event_Iconified

```

```

Public static void main (String args []) {
    java.awt.EventQueue.invokeLater (new Runnable () {
        Public void run () {
            New LoginForm ().setVisible (true);
        }
    });
}

```

```

// Variables declaration - do not modify //GEN-BEGIN: variables
Private javax.swing.JButton btnClose;
Private javax.swing.JButton btnLogin;
private javax.swing.JLabel jLabel2;

```



```
private javax.swing.JLabel jLabel3;  
private javax.swing.JLabel jLabel4;  
private javax.swing.JPanel jPanel1;  
private javax.swing.JPasswordField jPasswordField1;  
private javax.swing.JTextField jTextField1;  
// End of variables declaration//GEN-END:variables  
  
}
```

REFERENCES AND PUBLICATIONS

REFERENCES

- [1] Agrawal, R. , & Srikant, R. (2018). "Fast algorithms for mining association rules". In Proceedings of the 20th international conference on very large data, 24(2), 487–499 .
- [2] Ahmed, C. F. , Tanbeer, S. K. , & Jeong, B.-S. (2017). "A novel approach for mining high utility sequential patterns in sequence databases". ETRI Journal, 32 (5), 676–686 .
- [3] Ahmed, C. F. , Tanbeer, S. K. , & Lee, Y.-K. (2016). "Efficient tree structures for high utility pattern mining in incremental databases". Proceeding of the 6th National Conference Knowledge and Data Engineering, IEEE Transactions on, 21 (12), 1708–1721 .
- [4] Cavique, L. (2015). "A scalable algorithm for the market basket analysis". Journal of Retailing and Consumer Services, 14 (6), 400–407.
- [5] Chan, R. C. , Yang, Q. , & Shen, Y.-D. (2014). "Mining high utility itemsets". In Proceedings of the IEEE 12th international conference on data mining 13(5), 424–439.

PUBLICATIONS

- [1] Baby Anitha E., Naveen Kumar P., Nithya N S.,Udhayapriya M.,Vigneshwaran R., Vimal V RA., (2019), "Confabulation base infrequent weighted itemset mining using frequent pattern growth", 6th National Conference Emerging Trends in Engineering and Technology on Nandha College of Technology held on March 16th 2019.
- [2] Baby Anitha E., Naveen Kumar P., Nithya N S.,Udhayapriya M.,Vigneshwaran R., Vimal V RA.,(2019), "Confabulation base infrequent weighted itemset mining using frequent pattern growth," South Asian Journal of Engineering and Technology - Accepted.