Personalization for OLAP Querying

A Dissertation submitted to Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of

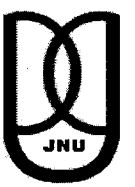
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE & TECHNOLOGY

by

VENKATESWARULU M

Under the esteemed guidance of

Prof. PARIMALA N



SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067 (INDIA)

JULY 2006

जवाहरलाल नेहरू विश्वविद्यालय

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110067 (INDIA)

CERTIFICATE

This is to certify that the dissertation titled **"Personalization for OLAP Querying"**, which is being submitted by **Mr. Venkateswarulu M** to the **School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi,** in partial fulfillment of the requirements for the award of **Master of Technology in Computer Science & Technology** is a bonafide work carried out by him under the supervision of **Prof. Parimala N.** The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.

Counter Signed

Prof. Balasundaram S
Dean, SC & SS
Jawaharlal Nehru University
New Delhi - 110067

Parimala N. 19/7/06

Prof. Parimala N
SC & SS
Jawaharlal Nehru University
New Delhi - 110067

जवाहरलाल नॅहरू विश्वविद्यालय

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110067 (INDIA)

DECLARATION

This is to certify that the dissertation titled **"Personalization for OLAP Querying"**, which is being submitted to the **School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi,** in partial fulfillment of the requirements for the award of **Master of Technology in Computer Science & Technology** is a bonafide work carried out by me.

The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.

19/07/2006

Venkateswarulu M,
M.Tech (Final Semester),
SC & SS, JNU,
New Delhi - 110067.

ACKWNOLEDGEMENTS

An endeavor over a long period can be successful only with the advice and support of many well-wishers. I take this opportunity to express my gratitude and appreciation to all of them.

I would like to sincerely thank my supervisor **Prof. Parimala N**, *School of Computer and Systems Sciences, Jawaharlal Nehru University* for the help, encouragement and support extended by her in successful completion of this dissertation. Her innovative ideas and the valuable discussions I had were very much helpful in keeping the thesis work on the right track.

I would also like to extend my thanks to the Dean, Faculty Members and Technical Staff of *School of Computer and Systems Sciences, Jawaharlal Nehru University.*

I would like to sincerely thank my best friend **Mr. Kiran Kumar Vinnakota** for the help, encouragement and support extended by him during the tenure of my M.Tech course. I would also like to extend my thanks to **Mr. Hareesh Basani** for help extended by him during the implementation of the proposed system.

Last but not the least, my sincere thanks to my parents, family members and friends and classmates for their continuous support, inspiration and encouragement without which this project would not have been a success.

Venkateswarulu Maddi Reddy

ABSTRACT

The OLAP tools that are presently available in the market, initially present all dimensions, dimension members, levels and measures to the analysts for decision making. So, an analyst has to go through all the data to choose the data for analysis. It requires more time and effort. Instead, if OLAP tools allow a user to store his preferences in a profile, then that profile can be used for initial data presentation and personalizing OLAP queries. Now, the time and effort required to select the data for analysis will be less and the users get the most interesting results to his / her queries. Traditional OLAP tools lack of this feature.

In order to overcome this we propose personalization for OLAP querying, which helps the user to create a user profile and use this profile for personalizing the initial data presentation and personalizing OLAP queries posed by the user. A user profile consists of dimensions, dimension members, levels and measures in which the user is interested. Additionally, constraints which restrict the values can also be expressed as a part of the user profile. The user profile is built and store in the system. When the user queries using OLAP tool, then only the relevant dimensions, dimension members, levels and measures as stored in the user profile are displayed. In this manner the user's screen is not filled up with unwanted schema constraints. Further, the constraints in the user profile are used to restrict the data that is to be retrieved.

The resultant data obtained by executing OLAP queries is presented to the user in tabular and graphical formats. Immediately after executing a query, the interface presents the data in tabular format. Here, a user can compare various numerical figures for detailed analysis. Alternatively, the user can view the data graphically. On demand, the tool presents the resultant data in a pie chart format. Using this pie chart, the user can quickly analyze the data.

We provide a personalized graphical OLAP tool, which provides interfaces for creating and editing the user profiles, personalizing the query interface, user profile integration and resultant data presentation.

TABLE OF CONTENTS

		Page No
ACKWNOLEDGEMENTS		**iv**
ABSTRACT		**v**
LIST OF FIGURES		**ix**
1. INTRODUCTION		**1**
1.1	General Concepts	2
1.1.1	Data Warehouse	3
1.1.2	On-Line Analytical Processing (OLAP)	8
1.1.2.1	OLAP Operations	10
1.1.2.2	Types of OLAP Systems	11
1.2	State of the Art	13
1.3	Aim of the Thesis	14
1.4	Organization of Thesis	14
2. PERSONALIZATION		**16**
2.1	Related Work	16
2.2	Proposed Work	16
2.3	User Profile Definition	17
2.4	Personalized Graphical OLAP Tool	17
2.4.1	Creating and Editing the User Profiles	17
2.4.2	Personalized Querying Interface	21
2.4.2.1	Initial Data Presentation	21
2.4.2.2	Profile Integration	21
2.4.3	Resultant Data Presentation	23
3. MULTI DIMENSIONAL MODELING		**24**
3.1	Multidimensional modeling	24
3.1.1	The Star Schema	25
3.1.1.1	Advantages of the Star Schema	27
3.1.2	The Snowflake Schema	28
3.1.2.1	Advantages of the Snowflake Schema	30
3.1.2.2	Disadvantages of the Snowflake Schema	30

3.2 RDBMS Vs MDBMS 30
3.2.1 Benefits of MDBMS over RDBMS 31
3.2.2 Disadvantages of MDBMS over RDBMS 32

4. DESIGN 34
4.1 Overview of System Architecture 34
4.2 The Design 36
4.2.1 Profile Management Module 38
4.2.2 Query Generation Module 38
4.2.3 Query Execution Module 39
4.2.4 Displaying Results Module 39
4.2.5 OLAP Operations Module 40

5. GRAPHICAL USER INTERFACE 42
5.1 The GUI 42
5.2 Creating User Account 42
5.3 Connecting to the Data Warehouse 44
5.4 Changing Password 45
5.5 Creating and Editing User Profiles 46
5.6 Querying 48
5.7 Viewing Resultant Data Graphically 53

6. IMPLEMENTATION 56
6.1 Building Data Warehouse 56
6.2 Implementation of the Interface 59
6.2.1 User Account Management 61
6.2.2 Connecting to Data Warehouse 62
6.2.3 Profile Management 63
6.2.4 Display Querying Window 64
6.2.5 Query Generation 65
6.2.6 Profile Integration 66
6.2.7 Query Execution 68
6.2.7.1 Establishing a Connection 68

6.2.7.2 Query Processing 69
6.2.7.3 Closing the Connection 70
6.2.8 Displaying Results 70
6.3 The Platform 72

7. CONCLUSION **74**
7.1 Features of the System 75
7.2 Future Enhancements 75

REFERENCES **76**

BIBLIOGRAPHY **77**

LIST OF FIGURES

Figure No	Description	Page No
1.1	An Example of Subject Orientation of Data	4
1.2	The Issue of Integration	5
1.3	The Issue of Nonvolatility	6
1.4	The Issue of Time Variancy	7
3.1	Multi-Dimensional Data Model	24
3.2	An Example of Multi-dimensional Data Model	25
3.3	The Star Schema	26
3.4	An Example of Star Schema	27
3.5	An Example of Snowflake Schema	29
3.6 (a)	RDBMS Storage	30
3.6 (b)	MDBMS Storage	30
3.7 (a)	RDBMS Storage	31
3.7 (b)	MDBMS Storage	31
3.8 (a)	RDBMS	33
3.8 (b)	MDBMS	33
4.1	Architecture of the System	34
4.2	Structure Chart for the Complete System	36
4.3	Structure Chart for Profile Management	38
4.4	Structure Chart for Query Generation	39
4.5	Structure Chart for Query Execution	39
4.6	Structure Chart for Query Displaying Result	40
4.7	Structure Chart for OLAP Operations	40
5.1	Invoking Create User	43
5.2	GUI for User Creation	43
5.3	Information Message	44
5.4	Warning Message	44
5.5	Login Form	44
5.6	Password Change Form	46
5.7	Creating and Editing Profiles	47

5.8	Querying Interface	48
5.9	Executing a Query	49
5.10	Roll-up / Drill-down	50
5.11	Level Selection Wizard	51
5.12	Slicing	52
5.13	Slicing Wizard	52
5.14	Resultant Data	53
5.15	Draw Chart Menu Item	54
5.16	Pie Chart	54

CHAPTER 1

INTRODUCTION

The amount of information available to large-scale enterprises is growing quickly. New data is being generated continuously by operational sources such as order processing, inventory control, and customer service systems. To make business decisions, enterprise analysts need to ask complex analytical queries over an integrated view of the data across all sources. Answering such complex queries is very expensive because data from multiple distributed heterogeneous sources must be extracted, cleansed, integrated, and processed at the query time. Further more, the sources may not always be available, they may not support complex analytical queries, and the network connecting them may have unreliable performance.

To deal with these problems, data warehousing has been proposed as an efficient approach to supporting on-demand analysis. Instead of integrating data from sources to answer the complex queries on demand, we integrate data in advance and store it in a data warehouse. Then, we can answer these complex queries locally using the data warehouse without going back to the sources. On the other hand, when the source data undergoes changes, we must perform warehouse maintenance to keep the data in the data warehouse up-to-date.

"A *Data warehouse* is a repository of historical data, collected from multiple heterogeneous data sources, integrated, and organized under a unified schema at a single site in order to facilitate management decision making."

Data warehouse provides the best opportunity for analysis of the data and *OLAP (On-Line Analytical Processing)* is the vehicle for carrying out this analysis. OLAP tools provide a way for managers and decision makers to extract information quickly and easily from data warehouse in order to answer questions regarding their business. A data warehouse without OLAP tools is unthinkable.

Data warehouse users use OLAP tools for decision making process. The OLAP tools that are presently available in the market, initially present all dimensions,

dimension members, levels and measures to the analysts for decision making. So, an analyst has to go through all the data to choose the data for analysis. It requires more time and effort. Instead, if OLAP tools allow a user to store his preferences in a profile, then that profile can be used for initial data presentation and personalizing OLAP queries. Now, the time and effort required to select the data for analysis will be less and the users get the most interesting results to his / her queries. Traditional OLAP tools lack of this feature and hence there is a need for *Personalized OLAP tools*, which enable *Personalization for OLAP Querying*. With the help of these personalized OLAP tools, the effort and time required to make decisions can be reduced.

In this chapter, first we introduce the concepts of *Data Warehousing* and *On-Line Analytical Processing* and then mention the need for *Personalized OLAP tools*.

1.1 General Concepts

The notion of extracting useful knowledge from collected data is not a new idea in the field of information systems technology. Only with the explosive growth of the quantity of data, it has become crucial to explore new techniques for data extraction and analysis. Many organizations possess large amounts of data that is maintained, and stored, but they are unable to capitalize on the valuable information hidden in the data. A data warehouse is designed to manage large volumes of business data and to provide a foundation for analytical processing. The primary goal of data warehouses is to improve the quality of decision making process in the enterprise. Years of research have produced state-of-the-art technology for designing and implementing such warehouses. Furthermore, in parallel with the investigation into design of data warehouses, various techniques for analyzing large amounts of data in the warehouses have been proposed and accordingly, was given a new term called *OLAP (On-Line Analytical Processing)* [1]. In the following sections we describe the general concepts of data warehouse and OLAP.

1.1.1 Data Warehouse

According to William H. Inmon [13], a leading architect in the construction of data warehouse systems, "*A data warehouse is a subject-oriented, integrated, nonvolatile and time-variant collection of data in support of management's decisions*". This short, comprehensive definition presents the major features of a data warehouse. The four key words, *subject-oriented, integrated, time-variant, and non-volatile,* distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems, and file systems. Let's take a closer look at each of these key features.

- **Subject-oriented:**
  The subject orientation of the data warehouse is shown in Figure 1.1. A data warehouse is organized around major subjects, such as customer, supplier, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers. Hence, data warehouses typically provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.

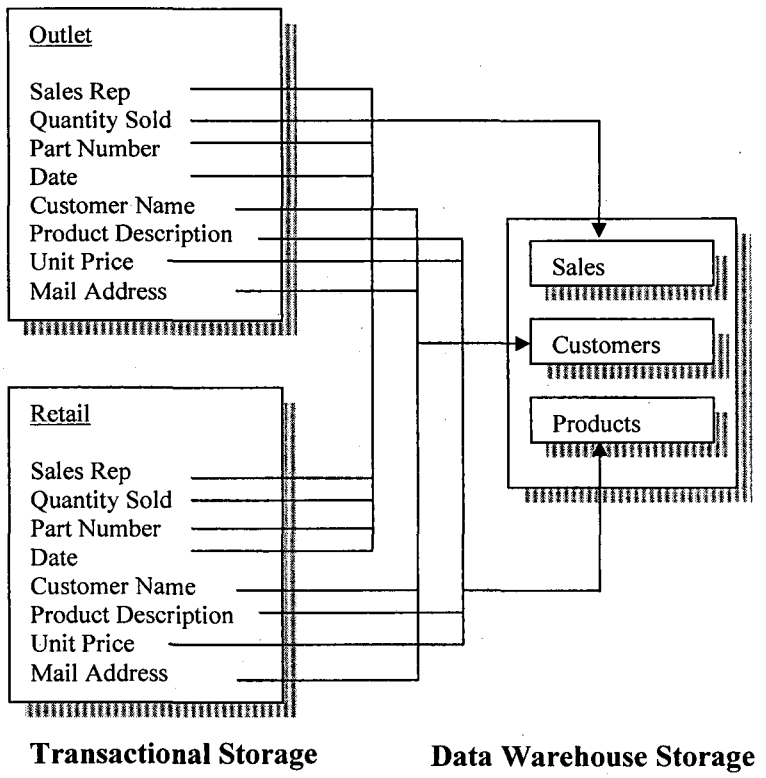


Figure 1.1 An Example of Subject Orientation of Data

- **Integrated**

  The second salient characteristic of the data warehouse is that it is integrated. Of all the aspects of a data warehouse, integration is the most important. Data is fed from multiple, disparate sources into the data warehouse. As the data is fed, it is converted, reformatted, re-sequenced, summarized, and so forth. Figure 1.2 illustrates the integration that occurs when data passes from the application-oriented operational environment to the data warehouse.

  Design decisions made by applications designers over the years show up in different ways. In the past, when application designers built an application, they never considered that the data they were operating on would ever have to be integrated with other data. Such a consideration was only wild theory. Consequently, across multiple applications there is no application consistency in encoding, naming conventions, physical

attributes, measurements of attributes, and so forth. Each application designer has had free rein to make his or her own design decisions. The result is that any application is very different from any other application.

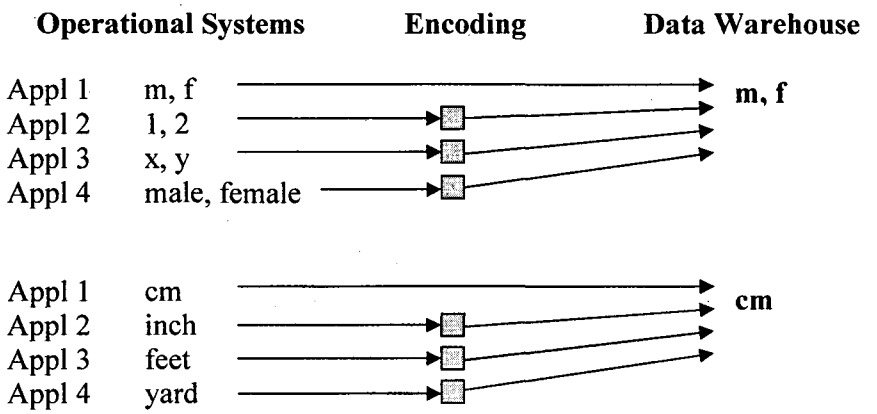


Figure 1.2 The Issue of Integration

Data is entered into the data warehouse in such a way that the many inconsistencies at the application level are undone. For example, as previously shown in Figure 1.2, as far as encoding of gender is concerned, it matters little whether data in the warehouse is encoded as m / f or 1 / 0. What does matter is that regardless of method or source application, data warehouse encoding is done consistently. If application data is encoded as X / Y for gender, it is converted as it is moved to the warehouse. The same consideration of consistency applies to all application design issues, such as naming conventions, key structure, measurement of attributes, and physical characteristics of data.

- **Nonvolatile**

  The third important characteristic of a data warehouse is that it is non-volatile. Figure 1.3 illustrates nonvolatility of data and shows that operational data is regularly accessed and manipulated one record at a time. Data is updated in the operational environment as a regular matter of course, but data warehouse exhibits a very different set of characteristics. Usually (but not always), data warehouse data is loaded at a time and

accessed for further usage. But in general it is not updated. Instead, when data in the data warehouse is loaded, it is loaded in a snapshot, static format. When subsequent changes occur, a new snapshot record is written. In doing so, a historical record of data is kept in the data warehouse.

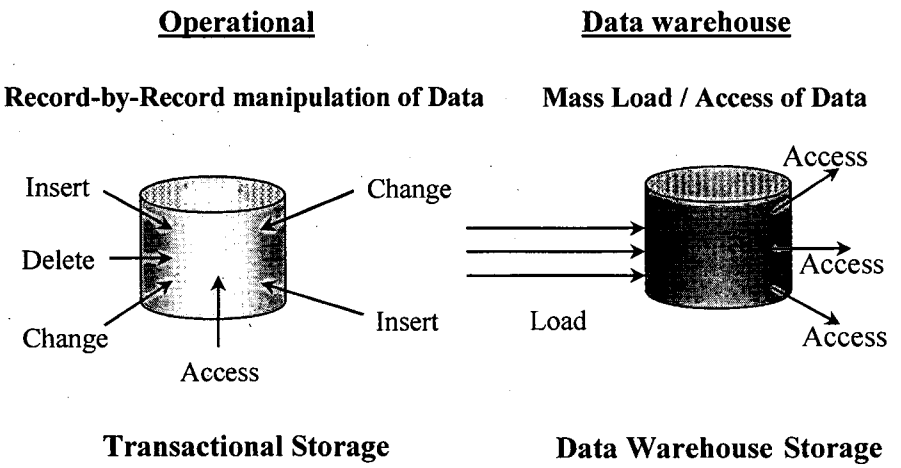


Figure 1.3 The Issue of Nonvolatility

- **Time-variant**

  The last salient characteristic of the data warehouse is that it is time variant. Time variancy implies that every unit of data in the data warehouse is time stamped. In other cases, a record has a date of transaction. But in every case, there is some form of time marking to show the moment in time during which the record is accurate. Figure 1.4 illustrates how time variancy of data warehouse data can show up in several ways.

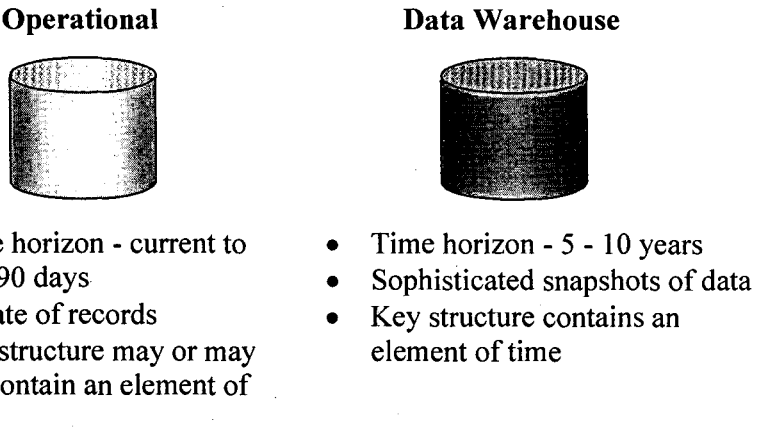


Figure 1.4 The Issue of Time Variancy

Different environments have different time horizons associated with them. A *time horizon* is the length of time data is represented in an environment. The collective time horizon for the data found inside a data warehouse is significantly longer than that of operational systems. A 60 - to - 90 day time horizon is normal for operational systems; a 5 - to - 10 year time horizon is normal for the data warehouse. As a result of this difference in time horizons, the data warehouse contains much more history than any other environment.

Operational databases contain *current-value data,* or data whose accuracy is valid as of the moment of access. Data warehouse data is very unlike current-value data, however. Data warehouse data can be thought of as nothing more than a sophisticated series of snapshots, each snapshot taken at one moment in time. The effect created by the series of snapshots is that the data warehouse has a historical sequence of activities and events, something not at all apparent in a current-value environment where only the most current value can be found.

The key structure of operational data may or may not contain some element of time, such as year, month, day, and so on. The key structure of the data warehouse always contains some element of time. The embedding of the element of time in the data warehouse record can take

many forms, such as a time stamp on every record, a time stamp for a whole database, and so forth.

In sum, a data warehouse is a semantically consistent data store that serves as a physical implementation of a decision support data model and stores the information on which an enterprise needs to make strategic decisions. A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/ or ad hoc queries, analytical reporting, and decision making.

1.1.2 On-Line Analytical Processing (OLAP)

OLAP is a category of software technology that can be used to analyze the data existing in a data warehouse. The functional and performance requirements of On-Line Analytical Processing (OLAP) applications are very different from those of on-line transaction processing (OLTP) applications. Transaction processing systems are judged on their ability to collect and manage data, whereas analytical processing systems are judged on their ability to extract information from data. OLAP allows querying and analyzing of data from many different perspectives. OLAP council [6], [7] defines OLAP in the following way.

"OLAP is a category of Software Technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user. OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities".

The two types of processing namely OLAP and OLTP differ in many aspects as summarized below.

- **Users**

  The number of users in OLTP systems is large where as in OLAP systems this number is relatively small. OLTP is performed mainly by clerks,

where as OLAP is used by management people in decision making process.

- **Data**

  Data in OLTP is current, accurate, and very detailed. In contrast, data stored in data warehouses and manipulated by OLAP is historical, multidimensional and often summarized.

- **Unit of operation**

  Transactions in OLTP are usually short SQL statements, as opposed to OLAP where a knowledge worker deals with very complex nested queries. This creates a necessity for a flexible user interface and even more importantly for an efficient query optimization.

- **Number of accessed records**

  In most cases the number of records accessed by an OLAP server is at least by an order of magnitude larger than in the case of OLTP. Moreover access type in OLTP systems may be read or update or delete, where as in OLAP systems access type is read only.

- **Metrics**

  Transaction throughput is the main performance indicator in OLTP applications, however query throughput and response time are important for OLAP applications.

All these characteristics are strong arguments for physical separation of data warehouses from operational data. Moreover, it is often the case that data warehouses contain data consolidated from heterogeneous sources. The different sources may contain data of varying quality, and/or may use inconsistent representations, codes, formats, which have to be reformatted. In most cases OLTP is developed using E-R model that is application-oriented. Such a model cannot effectively serve for decision support, since a database model for OLAP is

to be subject-oriented as mentioned earlier. *Star* and *snowflake* schemas have emerged as the main candidates of models for efficient OLAP.

1.1.2.1 OLAP Operations

OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data. The following are the key operations of OLAP.

- **Drill Down**

  Drilling down is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized to the most detailed. The drilling paths may be defined by the hierarchies within dimensions or other relationships that may be dynamic within or between dimensions.

- **Roll-up**

  Roll-up is a specific analytical technique whereby the user navigates among levels of data ranging from the most detailed (down) to the most summarized (up) along a concept hierarchy. Roll-up is considered as a process of ascending concept hierarchies. Roll-up is a method of analysis for retrieving higher levels of summary data starting from detailed data.

- **Slice and Dice**

  This is an informal term referring to data retrieval and manipulation. We can picture a data warehouse as a cube of data, where each axis of the cube represents a dimension. Slicing means retrieving a piece (a slice) of the cube by specifying measures and values for some or all of the dimensions. When we retrieve a data slice, we may also move and reorder its columns and rows as if we had diced the slice into many small pieces. A system with good slicing and dicing makes it easy to navigate through large amounts of data.

- **Pivot**

  Pivot is to change the dimensional orientation of a report or page display. For example, Rotating consists of swapping the rows and columns, or

moving one of the row dimension into the column dimension, or swapping an off-spreadsheet dimension with one of the dimensions in the page display (either to become one of the new rows or columns), etc.

- **Drill-across**

  Drill-across is the act of requesting similarly labeled data from two or more fact tables in a single report.

- **Drill-through**

  Drill-through is a specific operation whereby the user views the raw data pertaining to a high level concept from a concept hierarchy of a given dimension.

1.1.2.2 Types of OLAP Systems

There are five different types of OLAP systems based on storage methodology. In all these systems the processing is same and the storage methodology is different. Architectures of these systems are briefly summarized as under.

- **ROLAP**

  ROLAP stands for Relational On-Line Analytical Processing, in which data is stored as rows and columns in relational form. This model presents data to the users in the form of business dimensions. In order to hide the storage structure to the user and present data multi-dimensionally, a semantic layer of metadata is created. The metadata layer supports summarizations and aggregations. We can store the metadata in relational databases.

- **MOLAP**

  MOLAP stands for Multidimensional On-Line Analytical Processing. In this model, online analytical processing is best implemented by storing

the data multi-dimensionally, that is, easily viewed in a multidimensional way.

- **HOLAP**

  HOLAP stands for Hybrid On-Line Analytical Processing. Physical implementation of this model is based on both relational and multidimensional technologies. HOLAP technologies attempt to combine the strengths of MOLAP and ROLAP. For summary-type information, HOLAP leverages cube technology for faster performance. When detail information is needed, HOLAP can drill through from the cube into the underlying relational data.

- **DOLAP**

  DOLAP stands for Desktop On-Line Analytical Processing, is a variation that exits to provide portability for the OLAP users. It creates multidimensional datasets that can be transferred from server to desktop, requiring only the DOLAP software to exit on the target system. This provides significant advantages to portable computer users, such as salespeople who are frequently on the road and do not have direct access to their office server.

- **O3LAP**

  O3LAP stands for Object Oriented On-Line Analytical Processing whose physical implementation is based on object-oriented databases. It combines the advantages of MOLAP and ROLAP.

Defining a schema and selecting an OLAP server is only one step in the process of building and maintaining a data warehouse. But it is important to choose carefully the architecture which fits the needs of knowledge workers. Ideally, creating an integrated enterprise warehouse that collects information about all subjects (e.g., customers, products, revenues, personnel) spanning the whole organization would be the best choice. The problem is that building such a

warehouse is a long and a complex process. For this purpose, different kinds of metadata, including *administrative*, *business*, and *operational* metadata, have to be managed. Consequently, many organizations are settling for *data marts* instead. A data mart is an integrated data resource, usually oriented to a specific purpose or major data subject that may be distributed to support local business needs. Data marts enable faster roll out, since they do not require the enterprise-wide consensus, but they may lead to a complex integration problems in the long run.

1.2 State of the Art

Existing OLAP tools do not have personalization for OLAP Querying. i.e., initially every user is presented with the same data for decision making. Even though a user is interested in some particular data, he/she has to go through the entire data of the data warehouse for making a decision. In this context, the time and effort required to make a decision are more. These reasons gave a motivation for Personalized OLAP tool. i.e., if a user is allowed to express his preferences in a profile, then later on that profile can be used for OLAP querying. In this case, initially the data presented to the user is purely based on the user profile. So, the initial dataset for decision making is reduced. This leads to efficient decision making process with the reduced efforts and time. The following example illustrates the necessity of personalized OLAP tool.

Example:

Let us suppose that we have a data warehouse with 20 dimensions. Further let us assume that each dimension has 5 levels and fact table contains 5 measures. If we use a traditional OLAP tool to analyze the data in the data warehouse, the OLAP tool presents all the dimensions, dimension members, levels and measures to every user. i.e., a user has to choose from the existing 105 options.

On the other hand, let us suppose that a user has a profile, which consists of 5 dimensions, 3 measures and 3 levels in each dimension.

If the OLAP tool supports personalization, then the initial dataset contains only 18 choices for decision making.

The above example illustrates the impact of the personalization in decision making process.

1.3 Aim of the Thesis

Our aim is to build a **Personalized OLAP tool** which provides Personalization for OLAP Querying. The tool has to provide an interface to create user profile. Initially, the profile has to be used for presenting the data to the user for decision making. Later on, the preferences in the profile have to be integrated with the query to get the most interested results.

Keeping this in view, we propose to develop a graphical tool, which provides an interface for creating a user profile. This tool integrates the preferences into the queries to get the desired results. Moreover this tool supports general OLAP operations like drill-down, roll-up, slice and dice. Once the query is built and executed, the results are displayed in various formats to the user. Usage of such a tool requires no knowledge of the underlying data structures, query building, etc to the user. The user is able to build OLAP queries and easily analyze the results using the visual interface.

1.4 Organization of Thesis

The rest of this thesis is organized as follows.

- **Chapter 2, Personalization:** In this chapter, we give a definition of user profile. We discuss about the profile creation, initial data presentation, query generation, profile integration and displaying the results.
- **Chapter 3, Multi Dimensional Modeling:** This chapter gives a description of the multi dimensional modeling. Here we discuss the merits and demerits of multi dimensional models and relational models.

- **Chapter 4, Design:** This chapter presents the over all design of the proposed system. We use structure charts to present the design of various modules of the proposed system.
- **Chapter 5, The GUI:** This chapter describes the graphical view of the proposed system. This acts as a user manual to help the users in using the system.
- **Chapter 6, Implementation:** This chapter deals with the implementation details of the proposed system. Here we present the implementation details of the front-end and back-end. We also discuss about the platform on which we develop the system.
- **Chapter 7, Conclusion:** In this chapter we present the features and future enhancements of the proposed system.

CHAPTER 2

PERSONALIZATION

In this chapter, we discuss about the personalization framework for OLAP querying. In section 2.1 we look at the related work and in section 2.2 we present the proposed work. Section 2.3 presents user profile definition and section 2.4 describes about personalized graphical OLAP tool.

2.1 Related Work

Handling user preferences is an important issue in current information systems, which has motivated many research efforts since many years. In the context of relational databases different methods have been proposed for personalizing queries. [2], [3] proposed some methods for personalization of queries in database systems. In [2], a user can express his / her preferences by degree of interest associated with values of attributes, or more generally with atomic selection or join conditions. It stores user profiles. When a user submits a query, this query is personalized using the preferences stored in his / her profile.

[5] Proposed a personalization framework for OLAP querying. This concentrates in finding most interesting visualization for the resultant data of OLAP queries. Based on the user profile, it finds the most interesting visualization for the resultant data of OLAP queries.

None of the above works concentrated on personalization of initial data presentation and preference integration in OLAP and hence gave a motivation for our present proposed work. We discuss the proposed work in the following section.

2.2 Proposed Work

The proposed work aims at storing user preferences in user profiles and using the profile to initial data presentation and personalization of OLAP queries. In the present work:

1. We propose a definition of user profile in OLAP context.
2. We develop a personalized graphical OLAP tool, which provides:
    - An interface for creating and editing the user profiles.
    - A personalized query interface
    - User profile integration
    - Resultant data presentation

We discuss these in detail in the following sections.

2.3 User Profile Definition

User profile is a set of user preferences. User profile consists of user preferences and interests. These preferences say about the data in which the user is interested. A user profile consists of dimensions, members, levels, values and measures in which the user is interested. We store the user profiles in a database.

2.4 Personalized Graphical OLAP Tool

In the present work, we develop a personalized graphical OLAP tool. This tool facilitates the following.

- An interface to create and edit the user profiles.
- A personalized querying interface.
- User profile integration
- Resultant data presentation.

2.4.1 Creating and Editing the User Profiles

The personalized graphical OLAP tool presents a GUI to create and edit the user profiles. In this GUI, different structures of a data warehouse are displayed. To start with, all the dimensions existing in the data warehouse are displayed. The user can choose any number of the dimensions for analysis. After selecting a

dimension, the user can choose the members and levels he/she is interested in, from the list that is displayed. For each of the members or levels selected by the user, specific values that the user is interested in can also be specified. These can be a set of discrete values or a range of values. Finally, we display a list of measures to the user. User is allowed to choose any of these measures for analysis. All the user preferences will be stored in a database. The following example illustrates the process of building a user profile.

Example:

Consider the following data warehouse schema (discussed in detail in chapter3), which consists of six dimension tables and a fact table.

Dimensions:

1. Customer Dimension

Customer No	Customer Name	Customer Income	Customer Category	Customer Address	City

2. Product Dimension

Product No	Product Name	Product Category	Unit Price	QOH

3. Time Dimension

Date Key	Day	Month	Year

4. Sales Person Dimension

Sales Person Id	Sales Person name	City	Quota

5. Orders Dimension

Order No	Order Date

6. Location Dimension

City Key	State	Region	Country

Fact Table:

Customer No
Product No
Date Key
Sales Person Id
Order No
City Key
Quantity
Total Amount

Now, we see the step wise process of building a user profile for the above schema.

- **Selecting a dimension**

  We display a list of dimensions that are present in the data warehouse to the user. User is allowed to choose any number of dimensions from this list. In this example we display all the dimensions namely **customer, product, sales person, location, time, orders** to the user. Clicking on **Add Dimension** button in the interface adds a dimension to the profile.

- **Selecting levels and Members**

  After adding a dimension to the profile, we display all the members and levels of that dimension. For example, suppose that user selects **customer** dimension. Then we display **customer no, customer name, customer income, customer category, customer address, city** to the user. User can choose any of these members and levels. To add a member or level to user profile, the user can click on **Add Member / Level** button in the interface.

- **Specifying Constraint**

  After adding a member or a level to the profile, we display all the values corresponding to the selected one. The user can choose both discrete values and a range of values. Suppose the user has added **customer income** to his profile. Then we display all the values that exist in the data warehouse for that. These values may be **Rs. 10,000,**

20,000, 4,000 ...etc. Similarly, in the **product** dimension **unit price** may have the values **Rs. 200, 300 ...etc**.

Now, user can specify his preferences in two ways. One way to express the preference is to specify a range. For example, the user can choose a range of incomes and product unit prices as shown below:

Example of a range:

```
CUSTOMER.CUSTOMER_INCOME BETWEEN 1500 AND 20000 AND
PRODUCT.UNIT_PRICE < 2000
```

Another way is to express the preferences in terms of discrete values. The following example illustrates discrete valued preferences.

Example of discrete values:

```
CUSTOMER.CUSTOMER_CATEGORY = 'STUDENT' OR
CUSTOMER.CUSTOMER_CATEGORY = 'ENGINEER'
```

In the above manner a user can specify interested values either in discrete values or in a range of values or in both.

- **Selecting measures**
  Similarly, we display all the measures that exist in the data warehouse to the user. User can choose any of these measures.

  After completing the selection process, the user can save his/her profile.

By using this interface users can create new profiles and edit or delete the existing profiles.

2.4.2 Personalized Querying Interface

Once the user profile has been built, we use it for

a) Initial data presentation
b) Query modification

The selected dimensions, members, levels and measures contribute to the initial data presentation. The constraints are used during profile integration phase for query modification.

2.4.2.1 Initial Data Presentation

As discussed in the previous chapter, the OLAP tools that are presently available in the market, initially present all dimensions, dimension members, levels and measures to the analysts for decision making. So, an analyst has to go through all the data to choose the data for analysis. It requires more time and effort. We provide a personalized query interface to the user, in which we display all the dimensions, dimension members, levels and measures according to the user profile. In the initial data presentation, since only the chosen ones are presented, the data set will be reduced. A user can easily browse through the reduced dataset and make better decisions in less time.

2.4.2.2 Profile Integration

The interface supports querying, general OLAP operations like drill-down, roll-up, slicing and dicing. Once is query is specified by the user, the system internally translates the query to a SQL query. The following is an example of a generated SQL query.

```
SELECT Customer_Name, Customer_Category, Product_Category,
SUM(Total_Amount) FROM Customer, Product, SalesFacts WHERE
SalesFacts.Customer_No = Customer.Customer_No AND
SalesFacts.Product_No = Product.Product_No
```

Now, we gather all the constraints specified in the profile. Suppose we have the following constraints in the profile.

```
Customer.Customer_Income BETWEEN 1500 AND 20000 and
Product.Unit_Price BETWEEN 48 AND 1899
```

We integrate these constraints into the generated query to get the most interested results. The query after personalization is shown below:

```
SELECT Customer_Name, Customer_Category, Product_Category,
SUM(Total_Amount) FROM Customer, Product, SalesFacts WHERE
SalesFacts.Customer_No = Customer.Customer_No AND
SalesFacts.Product_No = Product.Product_No AND
Customer.Customer_Income BETWEEN 1500 AND 20000 AND Product.Unit_Price
BETWEEN 48 AND 1899 GROUP BY Customer_Name, Customer_Category,
Product_Category
```

The constraints shown in **bold** are automatically integrated into the query by the interface. A more detailed example of profile integration during slice operation is shown below:

```
SELECT Customer_Name, Customer_Category, Product_Name,
sum(Total_Amount) FROM Customer, Product, SalesFacts WHERE
SalesFacts.Customer_No = Customer.Customer_No AND
SalesFacts.Product_No = Product.Product_No AND
Customer.Customer_Income BETWEEN 1500 AND 21000 AND
Product.Unit_Price BETWEEN 48 AND 1899 AND ( Customer.Customer_Name
= 'RAJ' OR Customer.Customer_Name = 'KIRAN' OR
Customer.Customer_Name = 'NAIDU' OR Customer.Customer_Name = 'REDDY'
OR Customer.Customer_Name = 'BHASKAR' ) GROUP BY Customer_Name,
Customer_Category, Product_Name
```

In the above example, the constraints that are shown in **bold** are automatically integrated into the query by the interface. These constraints are specified by the user at the time of profile creation and are automatically imposed on all the queries posed by the user. So, the user gets most interested results, which helps the user to make better decisions. In the above example, the slice conditions are shown in *italics.*

Similarly, during the drill-down, roll-up and dice operations, the interface generate different queries based on the user selections.

2.4.3 Resultant Data Presentation

The resultant data obtained by executing OLAP queries is presented to the user in tabular and graphical formats. Immediately after executing a query, the interface presents the data in tabular format. Here a user can compare various numerical figures for detailed analysis. Alternatively, the user can view the data graphically. On demand, the tool presents the resultant data in a pie chart format. Using this pie chart, the user can quickly analyze the data.

The proposed work aims at presenting a fully personalized visual query generation interface to the user. So, a user does not require any prior knowledge of the syntax of the query languages like SQL. The user doesn't require any technical knowledge of the underlying data structures to get the required information. The tool handles complex structures and relationships inherent in data. It allows the user to select the data needed by him and impose conditions (if any), on the data. By using this tool, a user can access the data and can perform all the OLAP operations on this data. The user can view the resultant data in different formats like tables or pie charts.

CHAPTER 3

MULTI DIMENSIONAL MODELING

In this chapter we explain the multi-dimensional data model, its advantages and disadvantages. Data can be stored using Relational DBMS and Multi-dimensional DBMS. We discuss these technologies focusing on their advantages and disadvantages.

3.1 Multidimensional Modeling

Generally operational data is mainly stored in the form of relational tables. In contrast to operational data, data in a warehouse needs a new kind modeling to support the complex and efficient operations of on-line analytical processing (OLAP). Thus, a new kind of modeling called *multidimensional modeling* has been proposed to cater the needs of data warehousing. Currently, multidimensional approach is the most commonly used design approach for modeling a warehouse.

The multidimensional models composed of logical cubes, measures, dimensions, hierarchies, levels and attributes. The simplicity of the model is inherent because it defines objects that represent real world business entities. This logical structure of multi-dimensional data model is shown in Figure 3.1.

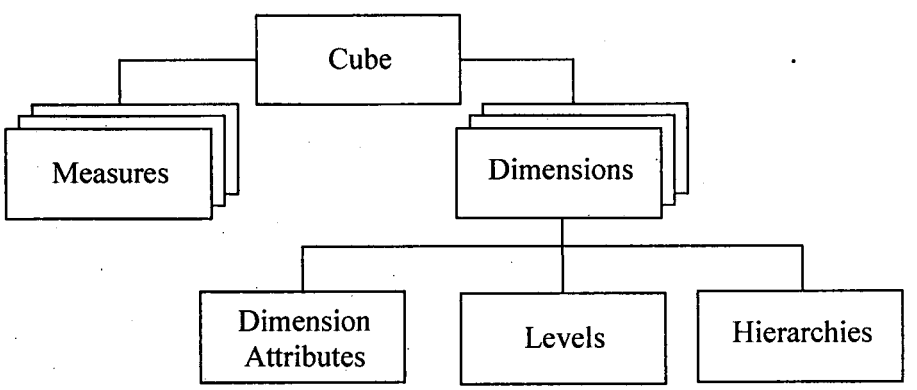


Figure 3.1 Multi-Dimensional Data Model

Multidimensional cube is constructed based on a set of dimensions and measures. It contains collection of cells. Each cell is an intersection among a set of dimension members and measures. These cells can be grouped to form aggregates. The multi-dimensional model, which views the data in the form of a data cube, is shown in Figure 3.2.

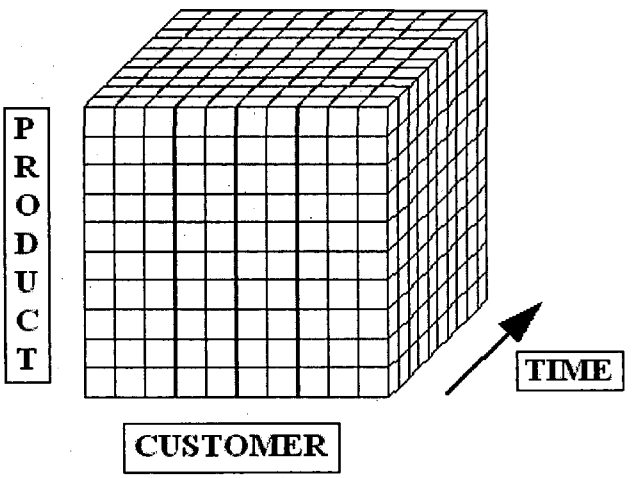


Figure 3.2 An Example of Multi-dimensional Data Model

3.1.1 The Star Schema

The Star Schema [11] is a representation of multidimensional data model. A star schema is a set of tables comprised of a single, central fact table surrounded by de-normalized dimensions. Each dimension is represented in a single table. Star schema represents dimensional data structures with de-normalized dimensions. The data is stored in a central fact table, with one or more tables holding information on each dimension. Dimensions have levels, and all levels are usually shown as columns in each dimension table.

As shown in Figure 3.3 the fact table is in the middle of the star schema and the dimension tables are arranged around the fact table. This arrangement in the multidimensional model looks like a star formation, with the fact table at the core

of the star and the dimension tables along the spikes of the star. The dimensional model is therefore called a star schema [10].

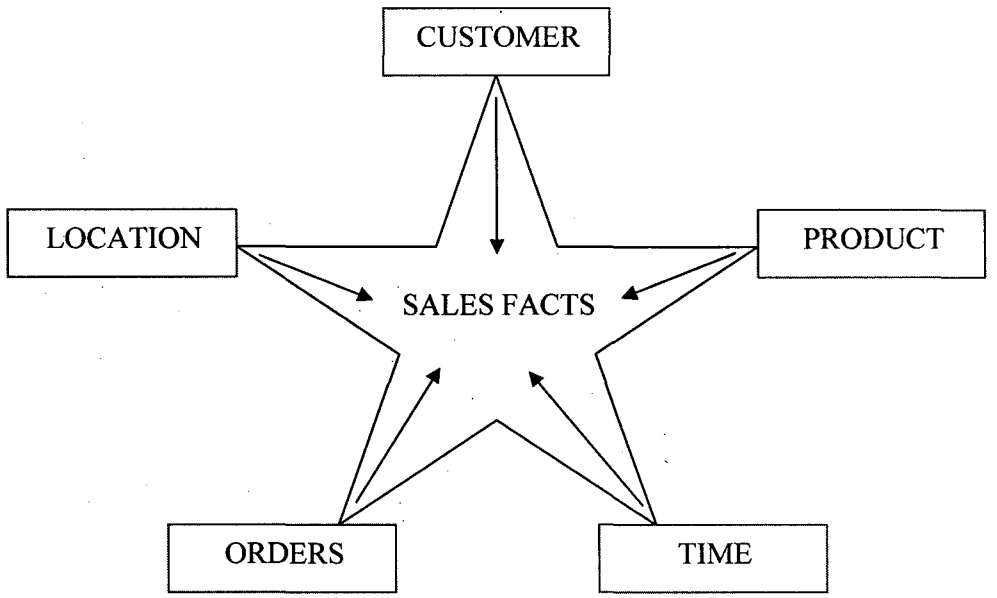


Figure 3.3 The Star Schema

Let us examine the star schema for the sales as shown in Figure 3.3, the sales fact table is in the center. Around this fact table, we have the dimension tables of Customer, Product, Time, Orders and Location. Each dimension table is related to the fact table in a one-to-many relationship. In other words, for one row in a dimension table, there exist one or more related rows in the fact table.

A fact table is a central table which is large, especially in terms of the number of tuples, whereas the dimensional tables are usually relatively small. This asymmetric architecture is very different from what the entity-relationship model is built on. Each tuple in the fact table contains a pointer, in the form of a foreign key, to each of the dimension tables, and a set of measures related to those particular dimensional objects. On the other hand, each dimension table consists of columns that represent attributes of the dimension.

These attributes may or may not correspond to the concept hierarchy of dimension. An example of star schema is shown in the figure 3.4. In the example

we have a "*sales fact table*". This fact table has pointers in the form of foreign keys to the following dimension tables: *customer, product, time, orders*, and *location.* The fact table is highly normalized, where as the attendant dimension tables are kept denormalized.

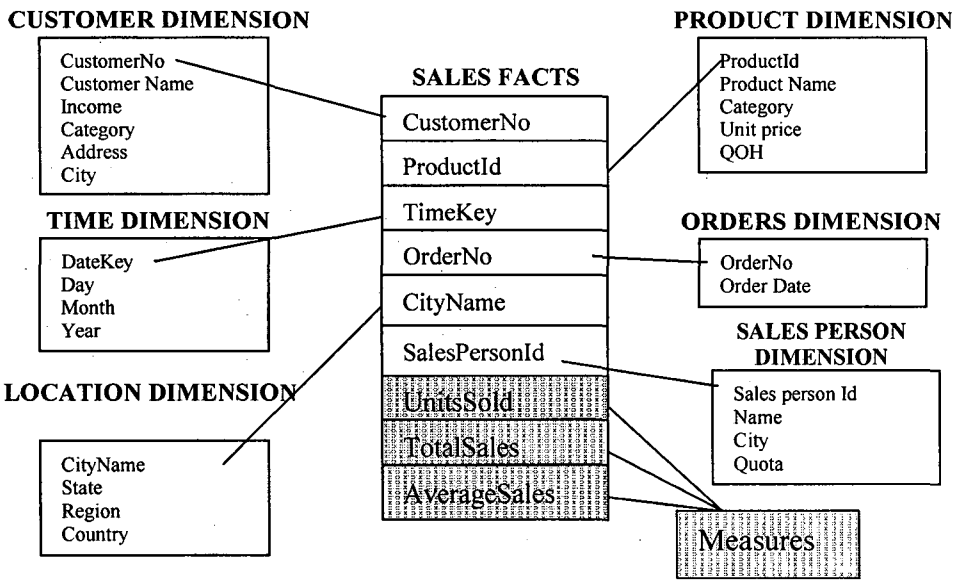


Figure 3.4 An Example of Star Schema

In addition to the dimensions, the star schema collects measures of the business. Since the main motivation for the whole data warehouse technology is to enhance decision support process, obtaining useful measures presents a pivotal issue. All measures of the business are stored in the fact table. In the above star schema, "*UnitsSold*", "*TotalSales*" and "*AverageSales*" are measures (See Figure 3.4).

3.1.1.1 Advantages of the Star Schema

The star schema structure is a structure that can be easily understood by the users and with which they can comfortably work. The structure mirrors how the users normally view their critical measures along their business dimensions. In the following section we describe the advantages of star schema.

- **Easy for Users to Understand**

  The star schema reflects exactly how the users think and need data for querying and analysis. They think in terms of significant business metrics. The fact table contains the metrics. The users think in terms of business dimensions for analyzing the metrics. The dimension tables contain the attributes along which the user normally query and analyze. Because of this reason, the users can easily understand the structure of star schema.

- **Optimizes Navigation**

  The relationships are used to go from one table to another for obtaining the information we are looking for. The relationships provide the ability to navigate through the database using the join paths. If the join paths are simple and straightforward, our navigation is optimized and becomes faster.

  A major advantage of the star schema is that it optimizes the navigation through the database. Even when we are looking for a query result that is seemingly complex, the navigation is still simple and straightforward.

- **Most Suitable for Query Processing**

  The star schema is a query-centric structure. This means that the star schema is most suitable for query processing. Irrespective of the number of dimensions that participate in the query and irrespective of the complexity of the query, every query is simply executed first by selecting rows from the dimension tables using the filters based on the query parameters and then finding the corresponding fact table rows. This is possible because of the simple and straight forward join paths and because of the very arrangement of the star schema. There is no intermediary maze to be navigated to reach the fact table from the dimension tables.

3.1.2 The Snowflake Schema

A Snowflake schema is a set of tables comprised of a single, central fact table surrounded by normalized dimension hierarchies. Each dimension level is

represented as a table. Snowflake schema implements dimensional data structures with fully normalized dimensions. This is the most well known variation of star schema. "Snow flaking" is a method of normalizing the dimension tables in a star schema. When we completely normalize all the dimension tables, the resultant structure resembles a snowflake with the fact table in the middle. The star schema for sales as shown in Figure 3.4 above contains only six tables, where as the normalized version which is the snowflake schema, contains eight tables as shown in Figure3.5.

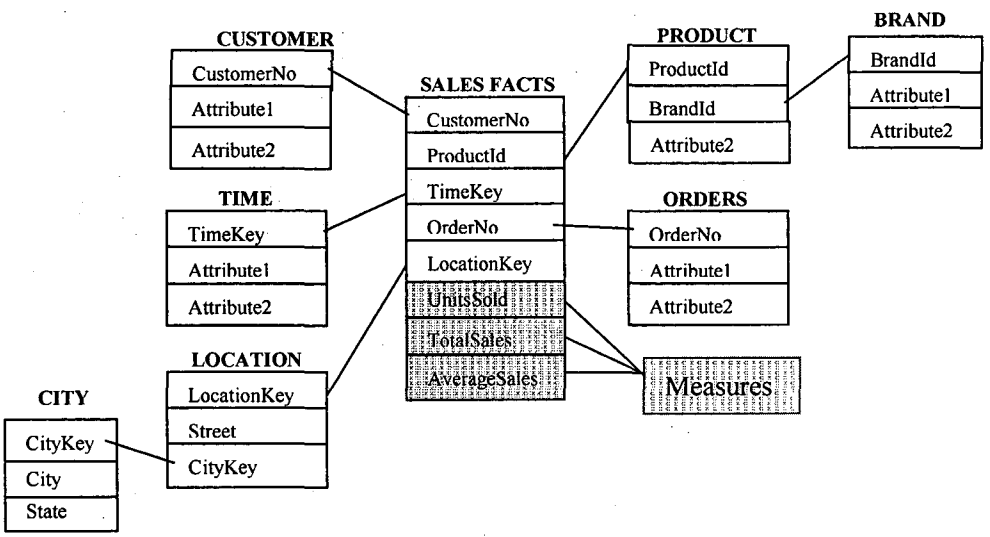


Figure 3.5 An Example of Snowflake Schema

However, the denormalized structure of the dimension tables in star schema offers easier browsing of the dimensions. For this reason Kimball, one of the leading experts in data warehouse technology, strongly discourages using a snowflake schema. "The dimension tables must not be normalized but should remain as flat tables. Normalized dimensions destroy the ability to browse". Snow flaking is not generally recommended in a data warehouse environment. Query performance takes the highest significance in a data warehouse and snow flaking hampers the performance. The difference between star and snowflake schema is handing the dimensions. Star schema is most preferable because of easy to understand and less joins make quick responses.

3.1.2.1 Advantages of the Snowflake Schema

The following are the advantages of the snowflake schema.

- Small savings in storage space
- Normalized structures are easier to update and maintain

3.1.2.2 Disadvantages of the Snowflake Schema

The following are the disadvantages of the snowflake schema.

- Schema is less intuitive and end-users are put off by the complexity
- Browse through the content is difficult
- Degraded query performance because of additional joins

3.2 RDBMS Vs MDBMS

Figure 3.6(a), 3.6(b) and Figure 3.7(a), 3.7(b) show how the data is stored in Relational DBMS and Multidimensional DBMS. In Relational DBMS we store the data in tables and in Multidimensional DBMS we store the data in multidimensional arrays (cubes).

MODEL	COLOR	SALES VOL
MINI VAN	BLUE	10
MINI VAN	RED	30
MINI VAN	WHITE	20
SPORTS CAR	BLUE	20
SPORTS CAR	RED	10
SPORTS CAR	WHITE	40
SEDAN	BLUE	10
SEDAN	RED	15
SEDAN	WHITE	20

Figure 3.6(a) RDBMS Storage

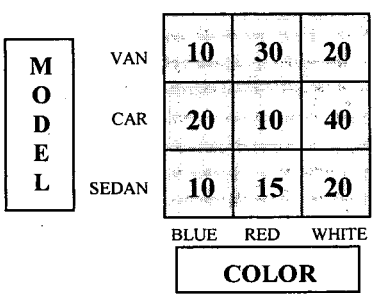


Figure 3.6(b) MDBMS Storage

MODEL	COLOR	DEALER	VOL
MINI VAN	BLUE	CLYDE	10
MINI VAN	BLUE	GLEASON	15
MINI VAN	BLUE	CARR	30
MINI VAN	RED	CLYDE	20
MINI VAN	RED	GLEASON	10
MINI VAN	RED	CARR	30
MINI VAN	WHITE	CLYDE	10
MINI VAN	WHITE	GLEASON	20
MINI VAN	WHITE	CARR	30
SPORTS CAR	BLUE	CLYDE	40
SPORTS CAR	BLUE	GLEASON	30
SPORTS CAR	BLUE	CARR	20
SPORTS CAR	RED	CLYDE	10
SPORTS CAR	RED	GLEASON	20
SPORTS CAR	RED	CARR	30
SPORTS CAR	WHITE	CLYDE	30
SPORTS CAR	WHITE	GLEASON	20
SPORTS CAR	WHITE	CARR	10
SEDAN	BLUE	CLYDE	10
SEDAN	BLUE	GLEASON	20
SEDAN	BLUE	CLYDE	30
.....	.....	.....	....

Figure 3.7(a) RDBMS Storage

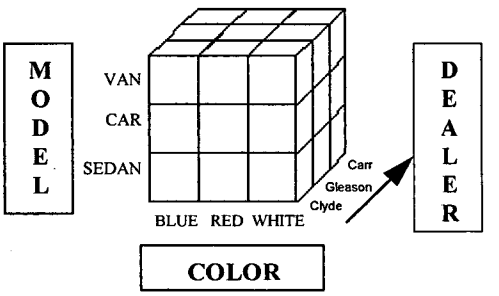


Figure 3.7(b) MDBMS Storage

As shown in Figure 3.6(a), RDBMS structure table needs 9(rows) * 3(columns) = 27(cells) of storage space, and in Figure 3.6(b), MDBMS structure of array needs only 3(dim) * 3(dim) = 9 (cells) of space. When we add one more dimension (Dealership) as shown in the Figure 3.7, the Relational Database requires 27 * 4 = 108 cells where as MDBMS requires 3 * 3 * 3 = 27 cells only to store the data. Thus, we see that array is more efficient and effective means of organizing data.

3.2.1 Advantages of MDBMS over RDBMS

The performance advantages offered by multidimensional technology facilitate the development of interactive decision support applications. Multidimensional DBMS offer several advantages over Relational DBMS. The following are the advantages of Multidimensional DBMS.

- **Ease of Data Presentation & Navigation**
  In MDBMS, a great deal of information is gleaned immediately upon direct inspection of the array. The user is able to view data along presorted dimensions with the data arranged in an inherently more organized, and accessible fashion than the one offered by the relational table.

- **Storage Space**
  Multidimensional databases consume very low space as compared to Relational Databases. Array is more efficient and effective means of

organizing data. From the Figures 3.6 and 3.7, we can say that the storage space required by MDBMS is less than the storage space required by RDBMS.

- **Performance**

  The multidimensional databases achieve high performance levels. It is very useful in OLAP applications. In relational environment performance can be achieved through database tuning (indexing and keys). Database tuning is an expensive activity. With the help of multidimensional databases, user queries can be answered quickly.

- **Ease of Maintenance**

  In multidimensional databases, the data is stored in the same way as it is viewed. So no additional overhead is required to translate user queries into requests for data. In Relational Databases, indexes, sophisticated joins etc. are used which require considerable storage and maintenance.

3.2.2 Disadvantages of MDBMS over RDBMS

In multidimensional databases, we mainly have sparsity problem. An example of sparsity in multidimensional databases is shown in Figure 3.8. In the Figure 3.8 (b) most of the cells are left blank. The figure doesn't have interacting dimensions. Because of this reason, most of the cells are left blank. However in RDBMS that row does not exist. The multidimensional databases in our example requires 9(dim) * 9(dim) = 81(Cells) but in relational databases we only require 9(rows) * 3 (columns) = 27(cells).

LAST NAME	EMP#	AGE
SMITH	01	21
REGAN	12	19
FOX	31	63
WELD	14	31
KELLY	54	27
LINK	03	56
KRANZ	41	45
LUCUS	33	41
WEISS	23	19

Figure 3.8 (a) RDBMS

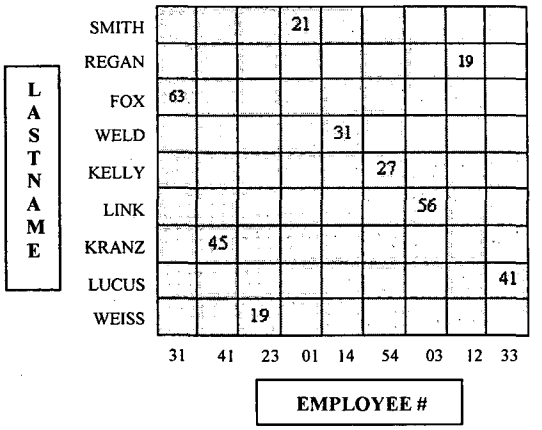


Figure 3.8 (b) MDBMS

In this case, the performance of multidimensional DBMS is less. In relational form, we require a maximum of nine searches to locate a record, where as in multidimensional form, we require 18 searches along two dimensions. If we have interrelated dimensions, then multidimensional models are preferable. Otherwise, relational models are preferable.

From the above discussion, we observe that the multidimensional models are best suited for On-Line Analytical Processing. So, we adopt the multidimensional modeling for implementing the system.

CHAPTER 4

DESIGN

In this chapter we discuss the overall architecture and design of the proposed system. We use structure charts [4] to describe the architecture of the proposed system.

4.1 Overview of System Architecture

The architecture of the proposed system is shown in Figure 4.1. The proposed system provides a multi user environment, in which 'n' number of users (clients) can connect to the OLAP server. The OLAP server gets the data from the Data Warehouse. This server executes the queries and gives the result back to the respective clients.

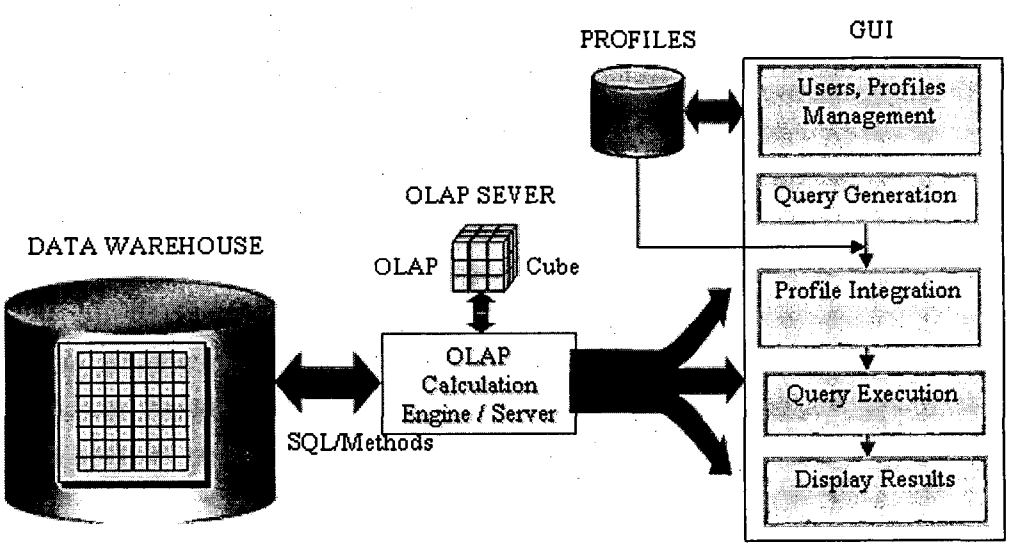


Figure 4.1 Architecture of the system

The user doesn't require any prior knowledge of query languages like SQL, to work with the system. The GUI provides a user friendly interface to work with the system. The proposed system is developed using Java 1.4.2 for front-end and Oracle 9i for back-end. The design details are explained in the following sections.

The **Users, Profiles Management** module provides mainly two functionalities. One is user accounts management. In this we can create a user account and a user can change his / her password. The second one is profile management. With the help of this we can create and edit user profiles. User profile consists of user preferences and interests. These preferences say about the data in which the user is interested. In other words user profile is the set of user preferences viz. dimensions, dimension members, levels, values and measures. This module provides a GUI to create and edit the profiles. By using this GUI a user can create new profile or edit the existing profile.

Query Generation module provides an interface for query building. This interface presents all the user interested dimensions, dimension members, levels and measures according to the user profile. Then the user can choose any of these dimensions, dimension members, levels and measures for querying. According to the user selection, the query will be generated automatically. The generated queries will be passed to the **Profile Integration** module.

Profile Integration module integrates the profile into the query. i.e., the conditions existing in the user profile to get the most possible answers will be integrated into the query by the profile integration module. Then the resulting query will be forwarded to the **Query Execution** module.

Query Execution module first establishes a connection between OLAP server and the system. After establishing the connection, this module passes the query to OLAP server for execution. OLAP servers are multidimensional databases that store summarized data, ready for business analysis. The OLAP server executes the query and the resultant data will be sent to the **Display Results** module.

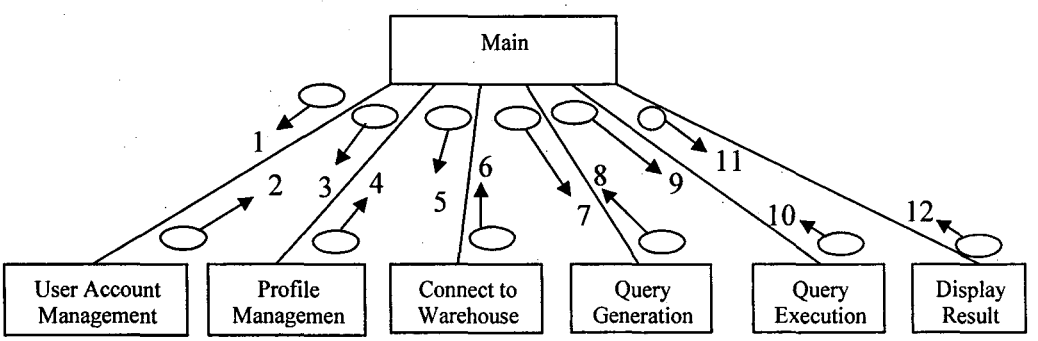
The **Display Results** module displays the resultant data in tabular format. Then, the user can perform drill-down, roll-up, slicing and dicing operations on the resultant data. The resultant data can also be viewed in other format like pie chart. This will help the user to quickly analyze the data.

If the user performs any thing wrong in between, the system raises an exception and the user will be given a warning message.

4.2 The Design

We use structure charts to describe the high-level design of the entire system. These charts act as a blueprint used by an architect to build a house. The structure chart employed in our design, breaks the problem in to major sub problems and we repeat the process for each sub problem until we reach problems that can be solved directly.

The overall system design is shown in Figure 4.2. The Main module controls the execution among **User Account Management, Profile Management, Connecting to Warehouse, Query Generation, Query Execution** and **Displaying Results** modules.



1. Request for user creation, change password
2. User created, password changed
3. Request for Profile Management
4. Profile Management completed
5. Request for warehouse connection
6. Connected to the warehouse
7. User requests
8. Generated queries
9. Generated queries
10. Resultant data
11. Resultant data
12. Display in different formats

Figure 4.2 Structure Chart for the Complete System

The description about the structure chart for the complete system is presented in the following section.

User Account Management module is used to maintain the user accounts. By using this module, we can create user accounts and a user can change his/her password.

Profile Management module is used to maintain the user profiles. With the help of this module a user can create and edit his / her profiles. User profile consists of user preferences and interests. These preferences say about the data in which the user is interested. In other words user profile is the set of user preferences viz. dimensions, levels, values and measures. This module provides a GUI to create and edit the profiles. By using this GUI a user can create new profile, edit the existing profile.

Connecting to Warehouse module is used to connect to the data warehouse. This module presents an interface to the user to enter user name and password. This module verifies the user name and password entered by the user. If the user name and password are correct then this module establishes a connection between the system and the warehouse. After connecting to the data warehouse, a user can query the data warehouse data for decision making.

Query Generation module is the heart of the whole system. This module presents the dimensions, levels and measures in a tree view to the user according to the user profile. The user can choose any of these dimensions, levels and measures. A query will be generated based on the user selections. The generated query will be forwarded to the **Profile Integration** module. The profile integration module integrates the profile into the query and then the integrated query will be sent back to **Query Generation** module. The query generation module passes this query to the **Query Execution** module.

Query Execution module executes the query. The results obtained by executing the query will be sent to the **Display Results** module.

Displaying Results module obtains the results from the query execution module. This module displays the results in different formats like tables and charts. Here a user can further perform the OLAP operations like drill-down, roll-up, slice and dice. The detailed design of these modules is explained with structure charts in the following sections.

4.2.1 Profile Management Module

Figure 4.3 depicts the structure chart for **Profile Management** module. This module first establishes a connection between the data warehouse and the system. Then it will display all the dimensions and measures to the user. User can choose any of the dimensions. After selecting the dimension, this module displays all the dimension members, levels in that dimension. User is free to choose any of these dimension members, levels, measures. A user can also specify constraints on the values. All the user selected data will be stored in a profile. At any time the user is allowed to edit his/her profile.

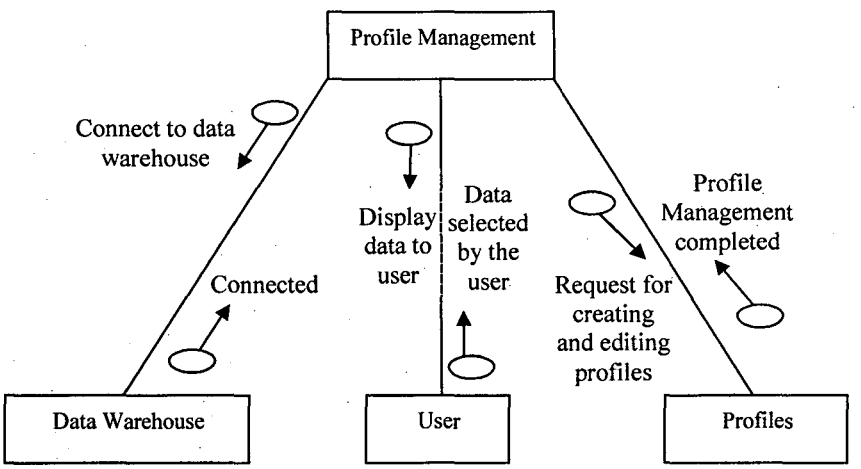


Figure 4.3 Structure Chart for Profile Management

4.2.2 Query Generation Module

The structured chart for **Query Generation** module is shown in Figure 4.4. This module presents the dimensions, dimension members, levels and measures in a tree view to the user according to the user profile. The user can choose any of these dimensions, dimension members, levels and measures. A query will be generated based on the user selections. The generated query will be forwarded to the **Profile Integration** module. The profile integration module integrates the profile into the query and then the integrated query will be sent back to query generation module. The query generation module passes this query to the **Query Execution** module.

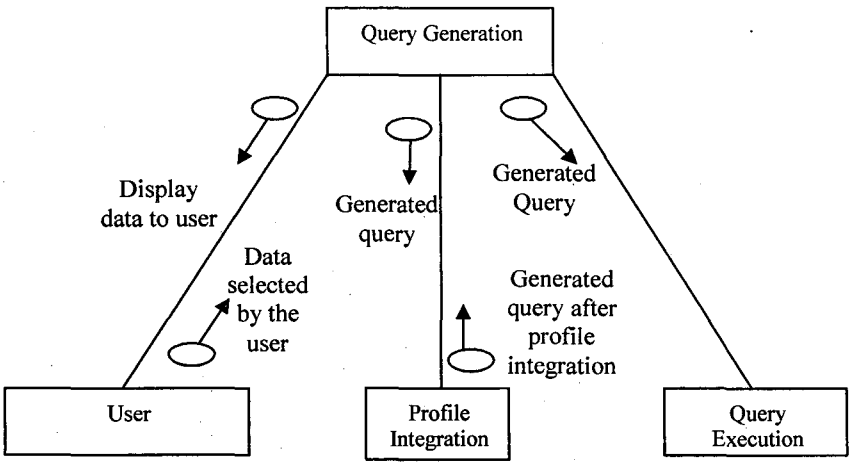


Figure 4.4 Structure Chart for Query Generation

4.2.3 Query Execution Module

The structured chart for **Query Execution** module is shown in Figure 4.5. This module executes the query obtained from the query generation module. The results obtained by executing the query will be sent to the **Displaying Results** module.

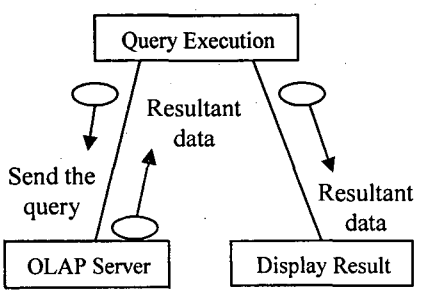


Figure 4.5 Structure Chart for Query Execution

4.2.4 Displaying Result Module

Figure 4.6 depicts the structure chart for **Displaying Result** module. This module displays the resultant data in tabular format. Then, the user can perform drill-down, roll-up, slice and dice operations on the resultant data. The resultant data

can also be viewed in other format like pie chart. This will help the user to quickly analyze the data.

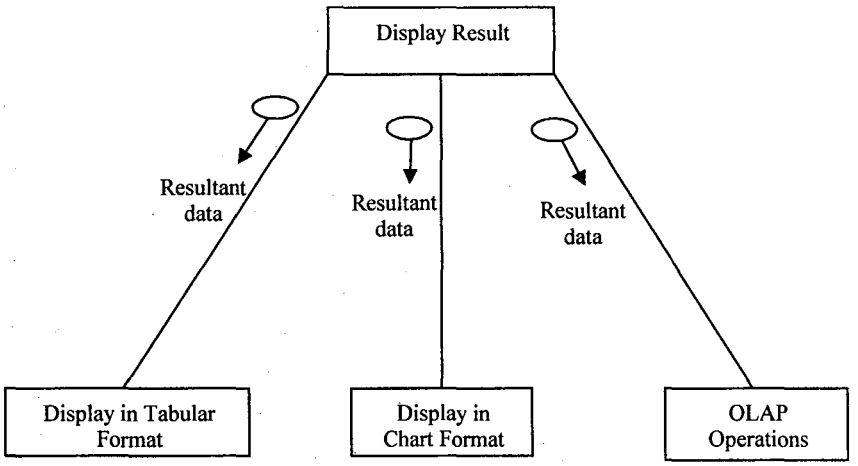


Figure 4.6 Structure Chart for Displaying Result

4.2.5 OLAP Operations Module

The structure chart for **OLAP operations** module is shown in Figure 4.7. By using this module a user can perform the OLAP operations like drill-down, roll-up, slice and dice. This module integrates the conditions specified by the user into the query and the resultant query will be handed over to the query execution module for execution.

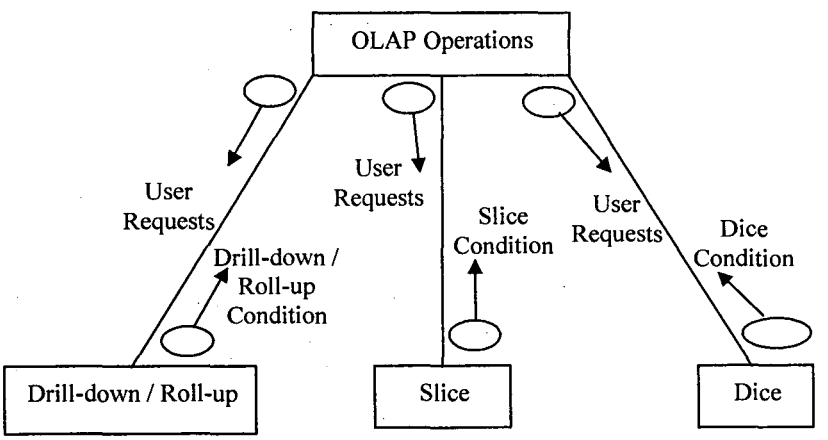


Figure 4.7 Structure Chart for OLAP Operations

The following section describes the functionalities of Drill-down / Roll-up, Slice and Dice modules.

- **Drill-down / Roll-up**

  This module displays all possible levels to which a user can drill-down / roll-up. User is allowed to choose a level. After the user selection, this module generates drill-down / roll-up condition. This condition will be integrated within the previous query and then the resultant query will be sent to the query execution module.

- **Slice**

  This module displays the list of values for the selected level in a tree view. User can select any of these values. This module generates slice condition according to the user selections. This condition will be integrated within the previous query and the resultant query will be handed over to query execution module.

- **Dice**

  This module allows the user to drag the dimensions and levels. With the help of this module a user can analyze the data in different views.

The working model of the proposed system is shown in the Chapter 5 and all the main modules are explained in chapter 6.

CHAPTER 5

GRAPHICAL USER INTERFACE

In this chapter we discuss the Graphical User Interface (GUI) of the proposed system. We have developed the GUI using Java 1.4.2.

5.1 The GUI

This interface is a menu-driven, user friendly, point and click interface. The system is an integrated environment in which a user can perform the following actions.

1. Creating user account
2. Connecting to the data warehouse
3. Change password
4. Creating and editing user profiles
5. Querying
6. Viewing resultant data graphically

We will discuss about these things in detail in the following sections.

5.2 Creating User Account

The interface for creating a user account can be invoked by choosing **Create User** option from the **OLAP Menu**. Alternatively, this can be done by hitting the keys **Ctrl - U**. The process of invoking the interface for creating a user account is shown in Figure 5.1. After choosing the **Create User** option from the **OLAP Menu**, an interface for creating a user account will appear. This interface is shown in Figure 5.2. The user has to enter a user name, password and confirm password to create a new user account. After entering the details, click on **Create User** button. Then, the system verifies the following conditions.

a. Are user name / password / confirm password fields empty?
b. Whether the user name already exists or not?

c. Whether the password and confirm password are same or not?

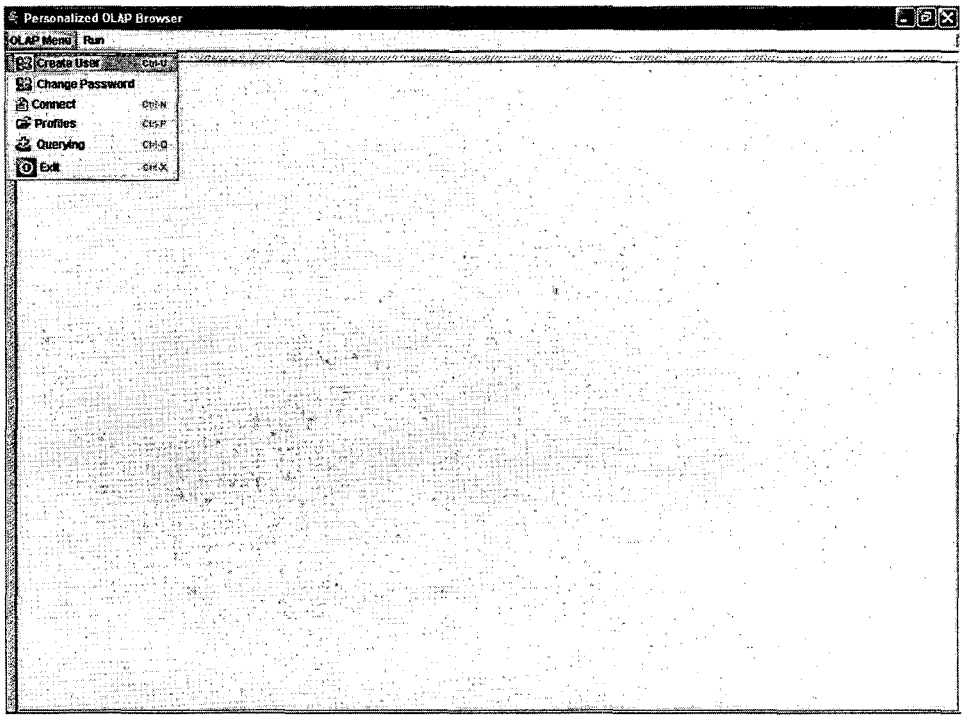


Figure 5.1 Invoking Create User

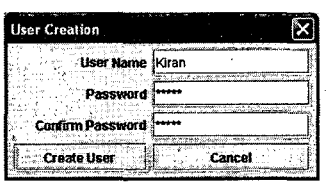


Figure 5.2 GUI for User Creation

Based on the above verification, the system creates a user account, if the following conditions are satisfied.

a. User name, password and confirm password fields are not empty.
b. User name not exists in the users list
c. Password and confirm passwords are same.

After the successful creation of a user account the system generates a message as shown in Figure 5.3. Otherwise, the system generates different types of warning messages based on the situation. The warning message is shown in Figure 5.4.

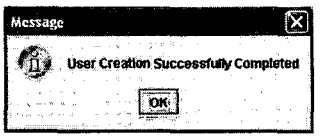


Figure 5.3 Information Message

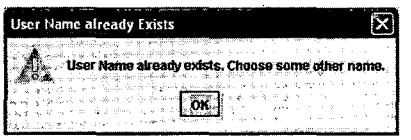


Figure 5.4 Warning Message

5.3 Connecting to the Data Warehouse

In order to access the data warehouse, a user has to connect to the data warehouse. The interface for connecting to the data warehouse is shown in the Figure 5.5. This interface can be invoked by choosing **Connect** option from the **OLAP Menu**. Alternatively, this can be done by hitting the keys **Ctrl - N**.

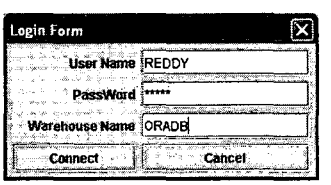


Figure 5.5 Login Form

The user has to enter a user name, password and data warehouse name in order to connect to data warehouse. After entering the details, user has to click on **Connect** button. Then, the system verifies the following conditions.

a. Are user name / password / data warehouse name fields empty?
b. Whether the user name and password are correct or not?

Based on the above verification, the system establishes a connection between the user and the data warehouse, if the following conditions are satisfied.

1. User name, password and data warehouse name fields are not empty.
2. User name and password are correct.

After successfully connecting to the data warehouse the system displays a connection confirmation message. Otherwise, the system generates different types of warning messages based on the situation. Some examples of warning messages are listed below.

1. User name not found / incorrect password. Try again
2. Data warehouse name not found.

5.4 Changing Password

By using this interface a user can change his / her password. To invoke the interface for changing password choose **Change Password** option from the **OLAP Menu**. After choosing the **Change Password** option, an interface for changing the password will appear. This interface is shown in Figure 5.6.

The user has to enter old password, new password and confirm password to change his / her password. When the user clicks on **Change Password** button, the system verifies the following conditions.

a. Whether old password / new password / confirm password fields are empty or not?
b. Whether the new password and confirm password are same or not?
c. Whether the old password is correct or not?

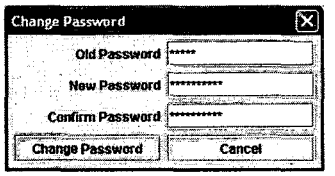


Figure 5.6 Password Change Form

Based on the above verification, the system changes the password, if the following conditions are satisfied.

1. Old password, new password and confirm password fields are not empty
2. New password and confirm password are same
3. Old password is correct.

After successfully changing the password, the system displays a password change confirmation message. Otherwise, the system generates warning message.

5.5 Creating and Editing User Profiles

By using this interface a user can create new profile, edit the existing profile. To invoke this interface **Profiles** option can be chosen from the **OLAP Menu.** Alternatively, this can be done by pressing keys **Ctrl - P**. The interface for creating and editing profiles is shown in Figure 5.7.

In this interface, dimension combo box shows a list of existing dimensions existing in the data warehouse. User can choose any of these dimensions. To add the selected dimension to profile, the user has to click on **Add Dimension** button. User is allowed to choose any number of dimensions. A list of levels for selected dimension will be displayed in the level combo box. User can choose any of these levels. To add a level to the profile, the user has to click on **Add Dimension Member / Level** button.

Every time after selecting a level, the system checks the data type of that level. If the data type of the selected level is either **Date** or **Number**, then the value

combo box is populated with the <, >, <=, >= and **Between** operators. Otherwise, value combo box is populated with the values of the selected level. After choosing a value, the user has to click on **Add** button to add this value to his / her profile.

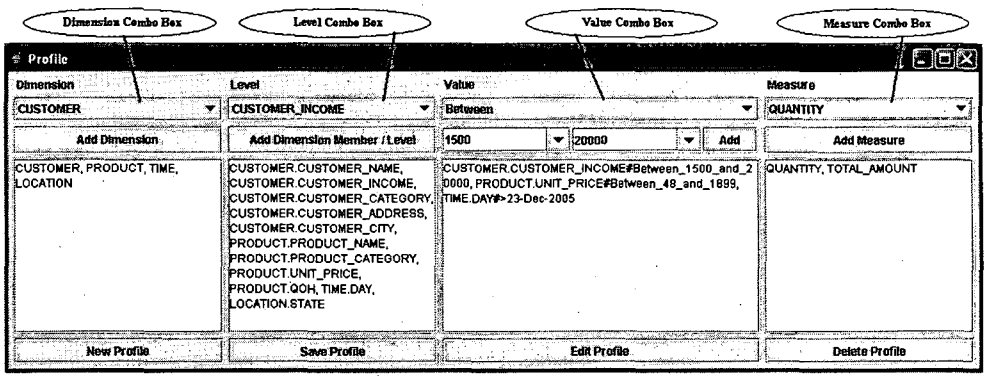


Figure 5.7 Creating and Editing Profiles

For the numeric and date type data, user can choose an operator from the value combo box. If the selected operator is **Between**, then the two combo boxes below the value combo box will be populated with the values of the selected level. User can specify a range by choosing values from these two combo boxes. If the selected operator is other than **Between**, then only one combo box will be populated with the values of the selected level. The other combo box will disappear. After choosing a range, user has to click on **Add** button to store it in the profile.

Measure combo box contains all the measures. User can select any of these measures. To add the selected measure to the profile, the user has to click on **Add Measure** button.

All the selected dimensions, levels, values and measures are displayed in the corresponding text areas below the combo boxes. To save all these preferences in the profile, user has to click on **Save Profile** button. In this interface, a user is allowed to create new profile, edit or delete the existing profile.

5.6 Querying

The interface for querying the data warehouse can be invoked by selecting **Querying** option from the **OLAP Menu**. Alternatively, we can invoke this interface by pressing the keys **Ctrl - Q**. By using this interface, a user can select data for analysis. The interface for querying the data warehouse is shown in Figure 5.8.

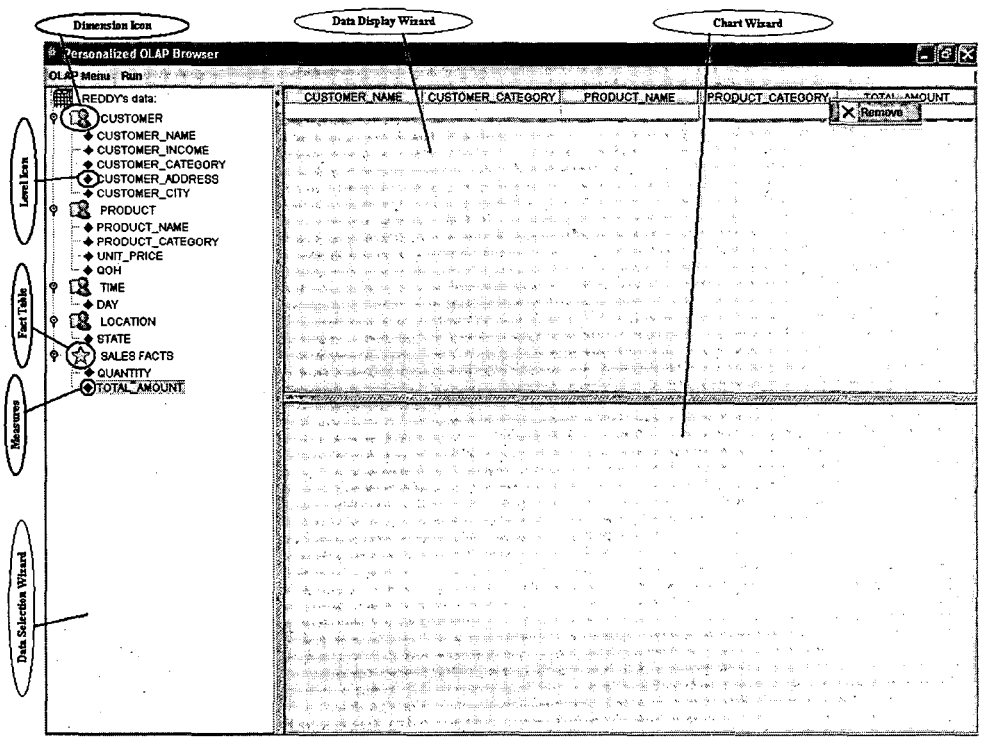


Figure 5.8 Querying Interface

This interface contains three wizards. These are data selection wizard, data display wizard and chart wizard. We discuss about these wizards in the following section.

- **Data Selection Wizard**

  Panel in the left side of the interface is the data selection wizard. This panel consists of a tree, which display all the dimensions, levels and measures according to the user profile. In this wizard, different types of icons are used to distinguish the dimensions, levels and measures. A user can choose the data for analysis by clicking on the leaves of the tree.

- **Data Display Wizard**

  Top-right panel in the interface is the data display wizard. Initially, this wizard contains a table. This table contains no columns or rows. When the user clicks on a leaf of the tree, then that leaf is added to the table as a column. User must select at least one measure for analysis. To remove a column in the table, right click on the corresponding column and select **Remove** option from the popup menu. Like this user can select or remove the attributes for analysis.

- **Chart Wizard**

  The screen area of this wizard is meant for viewing the data in graphical format like pie chart.

After selecting the attributes for analysis from the data selection wizard, the selected attributes will be displayed in a tabular format. To get the resultant data, select **Execute Query** from **Run** menu. Alternatively, this can be done by pressing the keys **Ctrl - E**. The process of selecting **Execute Query** is shown in Figure 5.9.

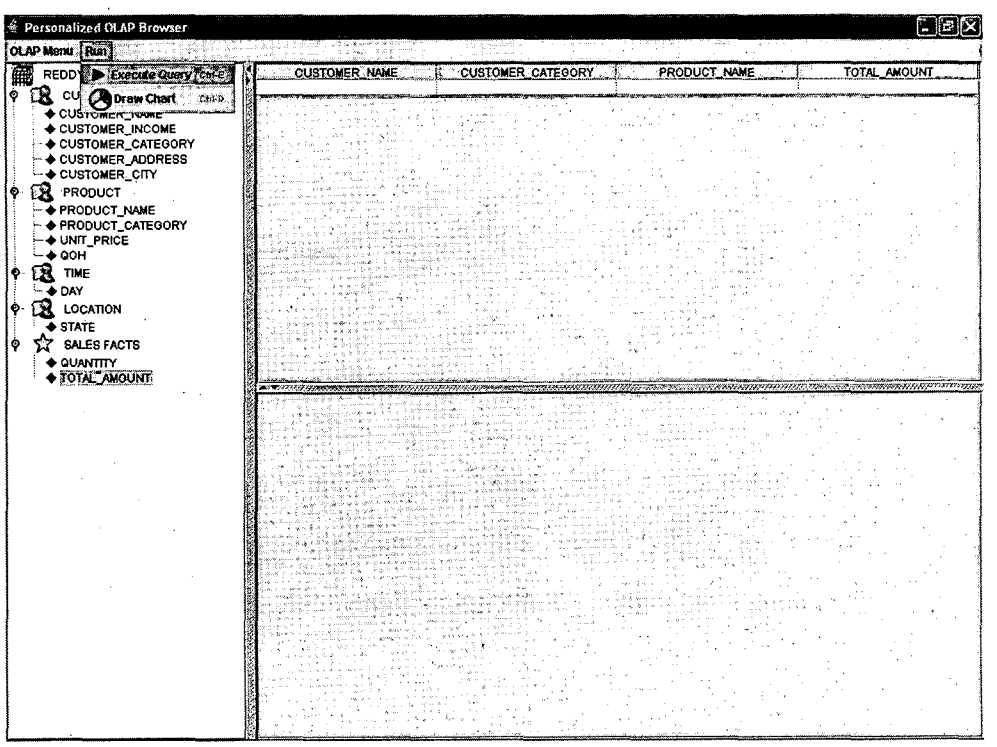


Figure 5.9 Executing a Query

The resultant data obtained by executing query will be displayed in tabular format in the data display wizard as shown in Figure 5.10. Then the user is allowed to perform different OLAP operations on the resultant data. These OLAP operations include the following.

1. Drill-down / Roll-up
2. Slicing
3. Dicing

We discuss about these operations in the following section.

- **Drill-down / Roll-up**

  To drill-down / roll-up to a level in a dimension, right click on the corresponding column of the result table and select **Roll-up / Drill-down** option from the popup menu. The process of selecting roll-up / drill-down option from the popup menu is shown in Figure 5.10.

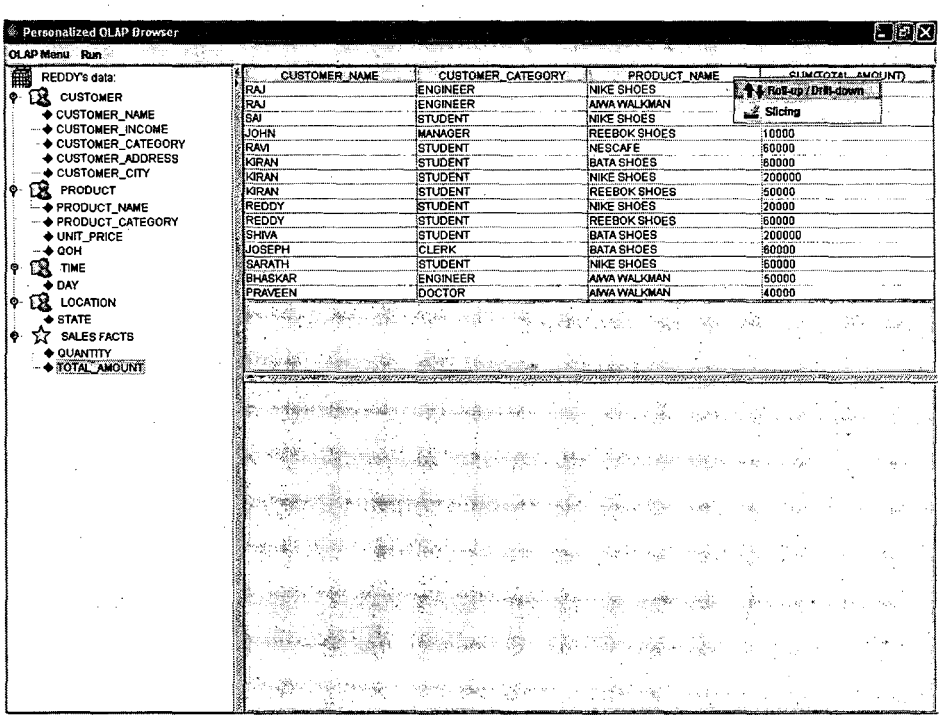


Figure 5.10 Roll-up / Drill-down

After selecting **Roll-up / Drill-down** option from the popup menu, a list of levels to which a user can roll-up / drill-down will be presented as shown in Figure 5.11. If the selected dimension doesn't have any hierarchies, then a message, "No hierarchies defined for the selected dimension" will be displayed.

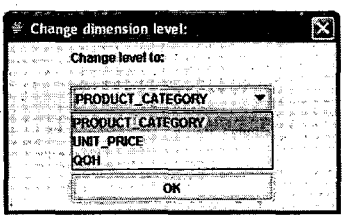


Figure 5.11 Level Selection Wizard

User is allowed to select one of these levels. After selecting a level, the system builds a query and executes it. The resultant data will be displayed in data display wizard as shown in Figure 5.12.

- **Slicing**

  To do slicing, right click on the corresponding table column and select **Slicing** option from the popup menu. The process of selecting **Slicing** option from the popup menu is shown in Figure 5.12.

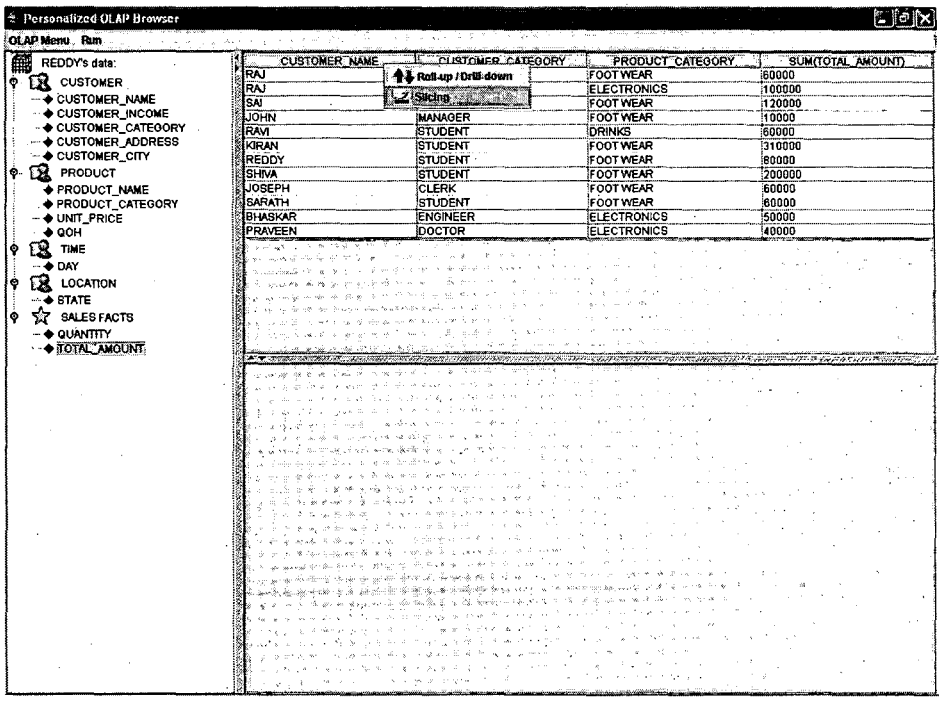


Figure 5.12 Slicing

After selecting **Slicing** option, a window will be displayed, which contains all the distinct values of the selected table column. The window is shown in Figure 5.13. All the values are displayed in a tree view.

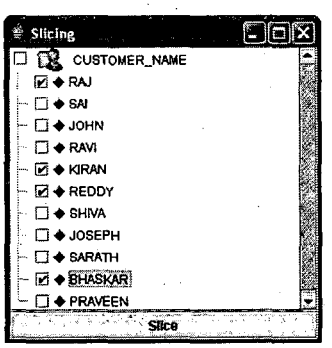


Figure 5.13 Slicing Wizard

Beside every value there exists a check box. User can choose any of these values for slicing. After selecting the values, the system generates a query and executes it. The resultant data will be displayed in the data display wizard as shown in Figure 5.14.

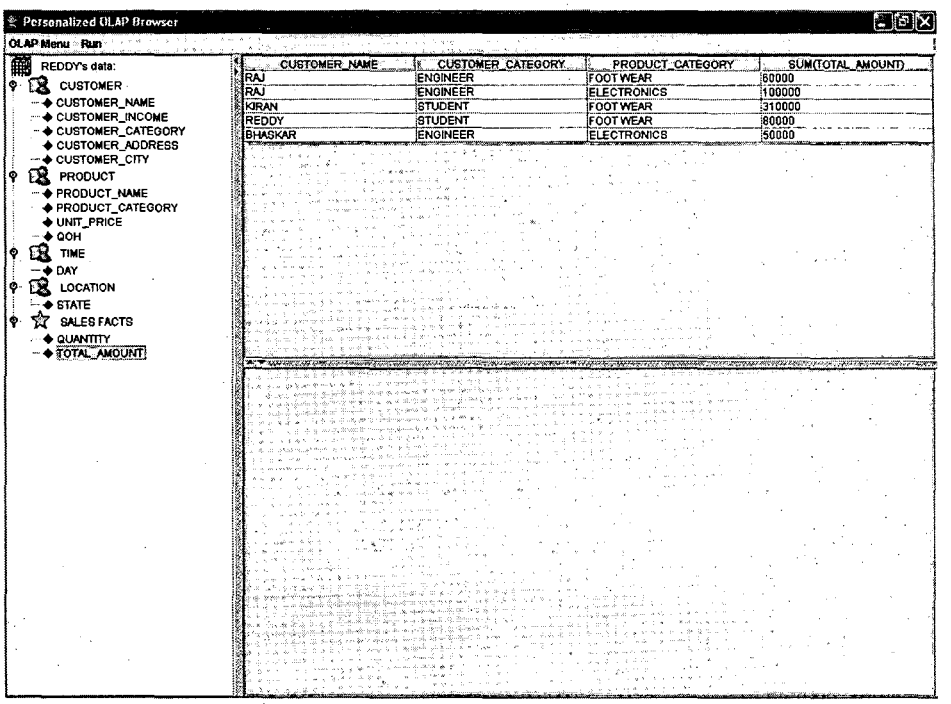


Figure 5.14 Resultant Data

- **Dicing**

  By rearranging table columns, one can analyze the data in different views. In the entire system, we allow users to freely rearrange the table columns. So, when a user wants to perform dicing operation, he / she can comfortably do the dicing.

5.7 Viewing Resultant Data Graphically

As discussed above, after executing a query the resultant data will be displayed in the data display wizard in a tabular format. To view the data in graphical format, **Draw Chart** option can be selected from **Run** menu. Alternatively, this can be invoked by hitting the keys **Ctrl - D**. The process of selecting **Draw Chart** option is shown in Figure 5.15.

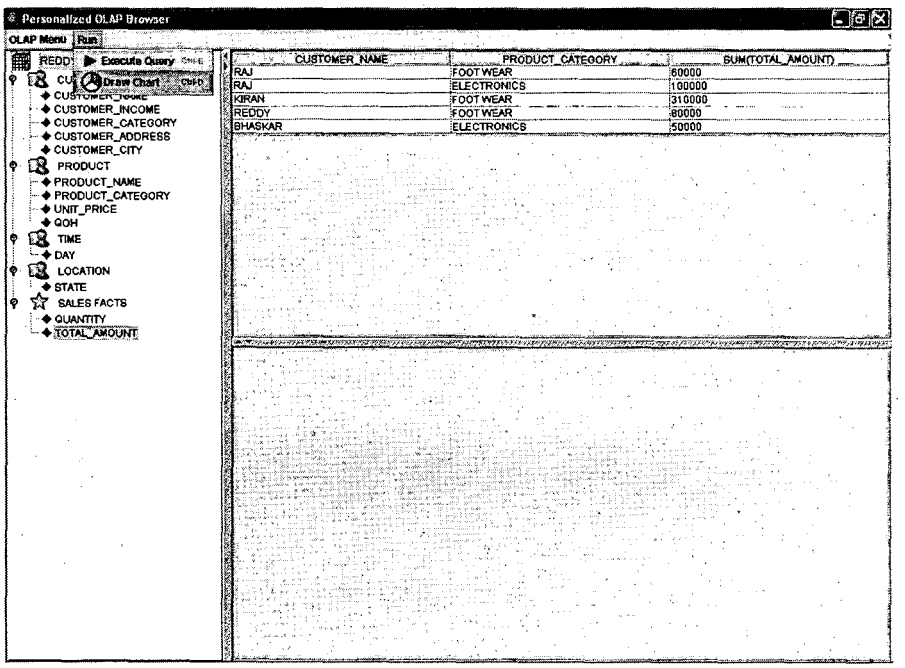


Figure 5.15 Draw Chart Menu Item

Then, a pie chart for the data in the data display wizard will be displayed in the chart wizard. This is shown in Figure 5.16.

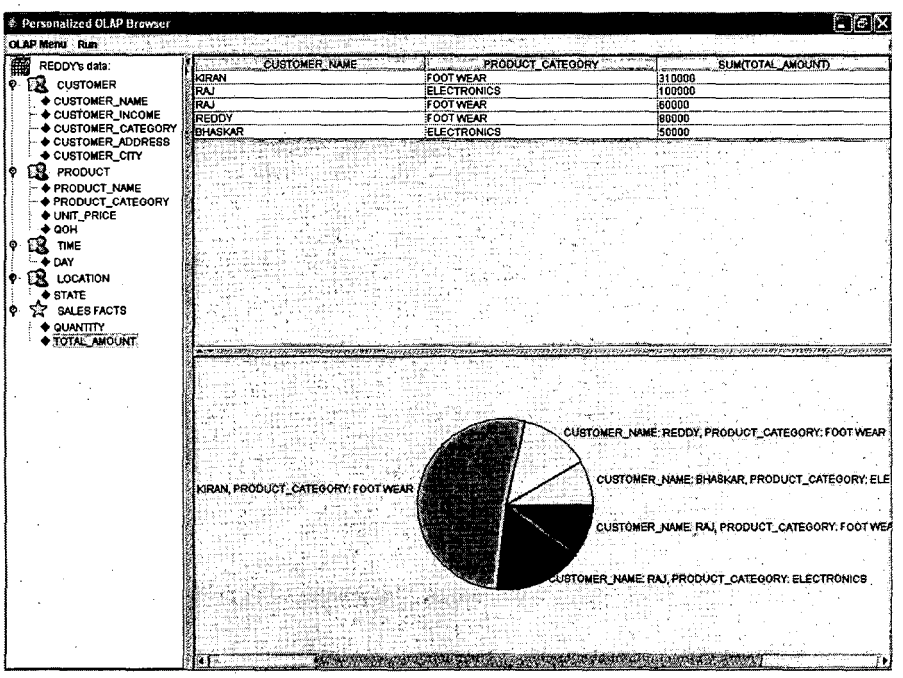


Figure 5.16 Pie Chart

By looking at the pie chart, a user can quickly analyze the data. So, the process of decision making requires less time.

In the next chapter, we discuss about the implementation details of the proposed system.

CHAPTER 6

IMPLEMENTATION

In this chapter we discuss the implementation details of the proposed system. The system is implemented in two stages. First stage concerns about building the data warehouse and second stage is intended to develop the interface. To build the data warehouse, we used Oracle 9i and Java 1.4.2 is used to develop the interface.

6.1 Building Data Warehouse

In section 3.1.1, we discussed the logical design of the data warehouse using star schema. Star schema is best suited to represent the structure of the data warehouse. Logical design is what we draw with a pen and paper before building our data warehouse. To build the data warehouse, we have to map the logical design to the physical design. Physical design is the process of creating the database with the help of SQL statements.

During the logical design phase, we defined a model for our data warehouse consisting of dimensions and fact table. To create dimensions, we need database tables. During the physical design process, we translate the schemas into actual database structures. At this time, we have to map:

- Entities to tables
- Relationships to foreign key constraints
- Attributes to columns
- Primary unique identifiers to primary key constraints
- Unique identifiers to unique key constraints

The multidimensional data model for the example considered in this thesis was presented in section 3.1.1 of chapter 3. The code snippets shown below give its representation in Oracle 9i.

```
CREATE TABLE Customer
(Customer_No NUMBER(5) PRIMARY KEY,
Customer_Name VARCHAR2(30),
Customer_Income NUMBER(8),
Customer_Category VARCHAR2(15),
Customer_Address VARCHAR2(50),
City VARCHAR2(20));

CREATE TABLE Orders
(Order_No NUMBER(5) PRIMARY KEY,
Order_Date DATE);

CREATE TABLE SalesPerson
(Sales_Person_Id NUMBER(5) PRIMARY KEY,
Sales_Person_Name VARCHAR2(30),
City VARCHAR2(20),
Quota NUMBER(6));

CREATE TABLE Product
(Product_No NUMBER(5) PRIMARY KEY,
Product_Name VARCHAR2(20),
Product_Category VARCHAR2(20),
Unit_Price NUMBER(5),
QOH NUMBER(6));

CREATE TABLE Time
(Date_Key DATE PRIMARY KEY,
Day NUMBER(2),
Month NUMBER(2),
Year NUMBER(4));

CREATE TABLE Location
(City_Name VARCHAR2(20) PRIMARY KEY,
State VARCHAR2(20),
Region VARCHAR2(20),
Country VARCHAR2(20));

CREATE TABLE SalesFacts
(Customer_No NUMBER(5) REFERENCES Customer,
Order_No NUMBER(5) REFERENCES Orders,
Sales_Person_Id NUMBER(5) REFERENCES SalesPerson,
Product_No NUMBER(5) REFERENCES Product,
Date_Key DATE REFERENCES Time,
City_Name VARCHAR2(20) REFERENCES Location,
Quantity NUMBER(10),
Total_Amount NUMBER(10));
```

Before we create a dimension, the dimension tables must exist in the database possibly containing the dimension data [9]. For example, if we create a customer dimension, one or more tables must exist that contain the city, state, and country information. In the star schema of the data warehouse, these dimension tables already exist. So, it is a simple task to identify which tables are to be used to build the dimensions. Then we can draw the hierarchies of a dimension.

We can define a dimension as shown below:

```
CREATE DIMENSION products_dim
LEVEL product IS (products.prod_id)
LEVEL subcategory IS (products.prod_subcategory) [SKIP WHEN NULL]
LEVEL category IS (products.prod_category)
HIERARCHY prod_rollup
(product CHILD OF
subcategory CHILD OF
category)
ATTRIBUTE product DETERMINES
(products.prod_name, products.prod_desc,
prod_weight_class, prod_unit_of_measure,
prod_pack_size, prod_status, prod_list_price, prod_min_price)
ATTRIBUTE subcategory DETERMINES
(prod_subcategory, prod_subcategory_desc)
ATTRIBUTE category DETERMINES
(prod_category, prod_category_desc);
```

Alternatively, the *extended_attribute_clause* could have been used instead of the *attribute_clause*, as shown in the following example:

```
CREATE DIMENSION products_dim
LEVEL product IS (products.prod_id)
LEVEL subcategory IS (products.prod_subcategory)
LEVEL category IS (products.prod_category)
HIERARCHY prod_rollup
(product CHILD OF
subcategory CHILD OF
category)
ATTRIBUTE product_info LEVEL product DETERMINES
(products.prod_name, products.prod_desc,
prod_weight_class, prod_unit_of_measure,
prod_pack_size, prod_status, prod_list_price, prod_min_price)
ATTRIBUTE subcategory DETERMINES
(prod_subcategory, prod_subcategory_desc)
```

```
ATTRIBUTE category DETERMINES
(prod_category, prod_category_desc);
```

After creating the dimensions, we create analytical workspaces. Analytic workspaces store data in a multidimensional format where it can be manipulated by the OLAP engine. An analytic workspace is stored as a LOB table in a relational schema. Within a single database, many analytic workspaces can be created and shared among users. There are two methods to create analytical workspaces. Those two methods are discussed below:

- **Analytic Workspace Manager**
  The Analytic Workspace wizard is the easiest method of creating an analytic workspace. The wizard generates a SQL script of DBMS_AWM statements, which we can execute immediately or save for execution later.

- **DBMS_AWM PL/SQL package**
  The DBMS_AWM package enables us to automate the build process in a script and schedule it to run overnight in a batch window. For most production systems, this is the preferred practice.

In the next section, we discuss the implementation details of the interface.

6.2 Implementation of the Interface

We have implemented the interface using Java. Java enables to write applications that are platform-independent. Various modules involved in the implementation of the interface are user accounts & profiles management, display querying window, query generation, profile integration, query execution and displaying the results. The design of these modules is explained in the chapter 4. Here, we present the implementation details of some important classes and methods.

A class named `OlapBrowser` is used to implement the proposed system. This is the main class and contains many sub-classes. We discuss all those sub-classes

later in this chapter. This class is used to display a menu based personalized OLAP browser. The following code snippet shows the implementation details of `OlapBrowser` class.

```
//This class is the main class in implementing the interface.
public class OlapBrowser extends JFrame implements ActionListener
{
        static int HORIZSPLIT = JSplitPane.HORIZONTAL_SPLIT;
        static int VERTSPLIT = JSplitPane.VERTICAL_SPLIT;
        JdbcTest con1 = new JdbcTest();
        int connFlag, profileFlag, queryFlag, flag, rowCount, columnCount;
        JScrollPane left, topRight, bottomRight;
        public OlapBrowser(String title)
        {
                super(title);
                try
                {
                        UIManager.put("Tree.leafIcon",
                                                new ImageIcon("sourceIcon.gif"));
                        UIManager.put("Tree.openIcon",
                                                new ImageIcon("fact.gif"));
                        UIManager.put("Tree.closedIcon",
                                                new ImageIcon("sub.gif"));
                }
                catch(Exception e)
                {
                        System.err.println("Couldn't use system" +
                                                "look and feel.");
                }
                JSplitPane splitPane1, splitPane2;
                JMenuBar menu;
                JMenu OlapMenu, Run;
                JMenuItem menuCreate, menuChangePass, menuConnect;
                JMenuItem menuQuerying, menuExecute, menuChart, menuExit;
                JMenuItem rollUp, slice, remove, menuProfiles;
                JPopupMenu popup;
                /*      Creating instances of the above components      */
                /*      Adding the components to the container    */
                //Rest of the code.....
        }
        //Sub-classes.....
}
```

We discuss each of the modules in the following sections. First we describe the implementation details of **User Account Management** module.

6.2.1 User Account Management

User Account Management module is used to maintain the user accounts. By using this module, we can create user accounts and a user can change his/her password. We present two different interfaces to user to create a user account and to change password. We used `JDialog` to build these two interfaces. These interfaces contain `JLabel`, `JTextField`, `JPasswordField` and `JButton` components. The code snippet shown below is the implementation of **User Creation** interface.

```
//This Function is used to create a user account.
public void createUser ()
{
        JFrame f=new JFrame();
        final JDialog d=new JDialog(f,"User Creation",true);
        /*      Declarations          */
        JLabel luser, lpassword, lConfirmPWD;
        JButton create, cancel;
        final JTextField userText;
        final JPasswordField pass, confirmPass;
        Container container = d.getContentPane();
        container.setLayout(new SpringLayout());
        /*      Creating instances of above components     */
        /*      Adding the components to the Container     */
        container.add(luser);
        container.add(userText);
        //.....
        SpringUtilities.makeCompactGrid(container, 4, 2, 6, 6, 6, 6);
        // some code...
        create.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ev)
                {
                        // code to create a user account
                }
        }
        //Rest of the code
}
```

The implementation details of **Change Password** are shown below.

```
//This Function is used to change user password
public void changePassword()
{
        JFrame f=new JFrame();
        final JDialog d=new JDialog(f,"Change Password",true);
        /*      Declarations          */
        JLabel lOldPWD, lNewPWD, lConfirmPWD;
        JButton change, cancel;
        final JPasswordField pass, newPass, confirmPass;
        Container container = d.getContentPane();
        container.setLayout(new SpringLayout());
        /*      Creating instances of above components     */
        /*      Adding the components to the Container     */
        container.add(lOldPWD);
        container.add(pass);
        //.....:
        SpringUtilities.makeCompactGrid(container, 4, 2, 6, 6, 6, 6);
        change.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ev)
                {
                        // code to change password
                }
        }
        //Rest of the code
}
```

We discuss the implementation details of **Connecting to Warehouse** module in the following section.

6.2.2 Connecting to Warehouse

Connecting to Warehouse module is used to connect to the data warehouse. This module presents an interface to the user to enter user name and password. This module verifies the user name and password entered by the user. If the user name and password are correct then this module establishes a connection between the system and the warehouse. After connecting to the data warehouse, a user can query the data warehouse data for decision making. We used `JDialog` to build this interface. This interface contains `JLabel`, `JTextField`, `JPasswordField` and `JButton` components. The code snippet shown below is the implementation for **Connecting to Warehouse** module.

```
// This Function is used to connect to the data warehouse
public int DwConnect()
{
        connFlag = 0;
        JFrame f=new JFrame();
        final JDialog d=new JDialog(f,"Login Form",true);
        JLabel l1=new JLabel("User Name",JLabel.RIGHT);
        JLabel l2=new JLabel("PassWord",JLabel.RIGHT);
        JLabel l3=new JLabel("Warehouse Name",JLabel.RIGHT);

        final JTextField userName =new JTextField(50);
        final JTextField db=new JTextField(50);
        final JPasswordField pass=new JPasswordField(50);
        pass.setEchoChar('*');
        JButton b = new JButton("Connect");
        JButton b2=new JButton("Cancel");
        d.getContentPane().setLayout(new SpringLayout());
        d.getContentPane().add(l1);
        //.....
        d.setLocation(400,300);
        SpringUtilities.makeCompactGrid(d.getContentPane(), 4, 2, 6, 6, 6, 6);
        //.....
        connect.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ev)
                {
                        // code to connect to the data warehouse
                }
        });
        // Rest of the code.....
}
```

We discuss the implementation details of **Profile Management** module in the following section.

6.2.3 Profile Management

Profile Management module is used to maintain the user profiles. With the help of this module a user can create and edit his / her profiles. We used `JFrame` component to build the interface of this module. This interface contains `JLabel`, `JComboBox`, `JTextArea`, `JScrollPane` and `JButton` components. The

code snippet shown below is the implementation of **Profile Management** interface.

```
// This Function is used to create and edit the profiles
public void Profiles()  throws SQLException
{
        JFrame f=new JFrame("Profile");
        JLabel ldimension, llevel, lvalue, lmeasure;
        JButton addDim, addLevel, addValue, addMeasure, add, edit, save,
                        delete;
        JComboBox dimension, level, value, measure;
        JTextArea dimText,  levelText, valueText, measureText;
        JScrollPane dimPane, levelPane, valuePane, measurePane;
        Container container = f.getContentPane();
        container.setLayout(new SpringLayout());
        //Create the instances of the above components
        //Add the components to the container
        SpringUtilities.makeCompactGrid(container, 5, 4, 6, 6, 6, 6);
        addDim.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ev)
                {
                        // code to add a dimension to profile
                }
        }
        //code to add a level, value, measure
        save.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ev)
                {
                        // code to save profile
                }
        }
        //Rest of the code
}
```

We discuss the implementation details of **Display Querying Window** module in the following section.

6.2.4 Display Querying Window

This interface displays the dimensions, dimension members, levels and measures in tree view. We used a `JTree` component to display the data in a hierarchical

view. This interface contains three wizards as we discussed in chapter5. We used `JSplitPane` and `JScrollPane` components to build these wizards. We named these wizards as *data selection wizard*, *data display wizard* and *chart wizard*. In the data display wizard, we display resultant data in tabular format. We used `JTable` component for this purpose. A user can perform OLAP operations on the resultant data by using the popup menus. We used `JPopupMenu` component to build the popup menus. The implementation details of the **Querying** module are shown below.

```
// This Function is used to display the querying window
public void Querying() throws SQLException
{
        JTree tree;
        DefaultMutableTreeNode root;
        DefaultTreeModel treeModel;
        JTable table;
        JSplitPane splitPane1, splitPane2;
        JScrollPane left, topRight, bottomRight;
        JMenuItem rollUp, slice, remove;
        JPopupMenu popup;
        // Building the tree, separating facts and dimensions with special
        // symbols. & some more functions
}
```

We discuss the implementation details of **Query Generation** module in the following section.

6.2.5 Query Generation

This module generates a query based on the user selections. We used + (string concatenation operator) to build the query. The code snippet shown below is the implementation of the **Query Generation** module.

```
//This Function is used to generate the query
public String generateQuery()
{
        String stmt;
        stmt = "select ";
        for(i=0; i<table.getColumnCount(); i++)
        {
```

```
                int flag = 0;
                for(int n = 0; measures[n] != null; n++)
                {
                        if((tableHead[i].equals(measures[n])))
                                        {
                                stmt = stmt + "sum(" + tableHead[i] + ")";
                                flag = 1;
                                break;
                        }
                }
                if(flag = = 0)
                        stmt += tableHead[i];
                if(i != (tab.getColumnCount()-1))
                        stmt += ", ";
        }
        stmt += " from ";
        //Rest of the code.....
        //.....
        return stmt;
}
```

During the query generation phase large queries may be generated based on the user selection. The following code snippet is an example of generated query.

```
SELECT Customer_Name, Customer_Category, Product_Category, SUM(Total_Amount)
FROM Customer, Product, SalesFacts WHERE SalesFacts.Customer_No =
Customer.Customer_No AND SalesFacts.Product_No = Product.Product_No
```

The generated query is sent to **Profile Integration** module. We discuss the implementation details of **Profile Integration** module in the following section.

6.2.6 Profile Integration

This module integrates the user preferences into the query. We store user preferences in a profile in string format. We divide this string into tokens by using `StringTokenizer` class. We integrate these tokens into the query. The following code is used to integrate the preferences into the query.

```
//This Function is used to integrate the profile into the query
public string integrateProfile(String queryString)
{
```

```
String stmt;
stmt  = queryString;
if(valueText1 = = null)
{
        return stmt;
}
StringTokenizer st= new StringTokenizer(valueText1, ", ");
i = 0;
//Building the Constraints
while(st.hasMoreTokens())
        constraints[i++] = st.nextToken();
constraintCount = i;
for(i = 0; i < constraintCount; i++)
{
        StringTokenizer st1 = new StringTokenizer(constraints[i], "#");
        constraints1[i][0] = st1.nextToken();
        constraints1[i][1] = st1.nextToken();
        StringTokenizer st3 = new StringTokenizer(constraints1[i][0],
                                                  ".");
        String table = st3.nextToken();
        String columnName = st3.nextToken();
        String dataType = new String();
        try
        {
                dataType = con1.getType("select " + columnName + " from
                        " + table);
        }
        catch(SQLException e9)
        {
        }
        if(dataType.equalsIgnoreCase("DATE"))
        {
                //code for DATE type
        }
}
//Integrating the Constraints into the Query
for(i=0;i<parentCnt;i++)
        for(j=0;j<constraintCount;j++)
                if(constraints1[j][0].startsWith(headParent[i]))
                {
                        stmt += constraints1[j][0] + " " +
                                        constraints1[j][1];
                        stmt += " and ";
                        System.out.println(stmt);
                        break;
                }
return stmt;
}
```

During the profile integration phase, preferences in the profile are integrated within the query automatically. The following code snippet is a query generated after profile integration. The code shown in **bold** is automatically integrated within the query during the profile integration phase.

```
SELECT Customer_Name, Customer_Category, Product_Category, SUM(Total_Amount)
FROM Customer, Product, SalesFacts WHERE SalesFacts.Customer_No =
Customer.Customer_No AND SalesFacts.Product_No = Product.Product_No AND
Customer.Customer_Income BETWEEN 1500 AND 20000 AND Product.Unit_Price
BETWEEN 48 AND 1899 GROUP BY Customer_Name, Customer_Category,
Product_Category
```

After integrating the profile into the query, the query is now ready for execution. In the following section we discuss the implementation details of **Query Execution** module.

6.2.7 Query Execution

In this module we execute the query. To execute a query we need to connect to the data warehouse. To connect to data warehouse, we used `Java Database Connectivity (JDBC) thin driver` API. JDBC thin driver is a platform independent driver. So, we can run our application in any environment. `Classes12.jar` file is used to establish a connection between the interface and oracle. We discuss the implementation of **Query Execution** module in the following sections.

6.2.7.1 Establishing a Connection

To establish a connection we need a JDBC driver. We register a driver using the following syntax.

```
DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
```

After registering a JDBC driver we can connect to the data warehouse using the following syntax.

```
conn = DriverManager.getConnection ("jdbc:oracle:thin:@local_host:
                                    1521:dbName", userName, password);
```

6.2.7.2 Query Processing

We can do query processing in two ways. One is by using `Statement` and the other one is `SPLExecutor`. We discuss these two methods of query processing below.

- **Using Statement to Execute Query**

  A statement can be created by using the connection object. The following code snippet creates a statement.

  ```
  Statement stmt = conn.createStatement();
  ```

  Here `conn` is Connection object. By using the `statement` we can execute a query as shown below.

  ```
  ResultSet rset = stmt.executeQuery(queryString);
  ```

  We copy the result set data into objects. These objects are passed to the **Display Result** module.

- **Using `SPLExecutor` to Execute Query**

  If we create analytic workspace, then Oracle OLAP API provides a way to directly manipulate analytical workspace data. It doesn't require any metadata and OLAP API data manipulation classes. The interface uses the **`SPLExecutor`** class to send queries directly to Oracle OLAP for execution. It is explained in [8]. The following code snippet shows the execution of queries using `SPLExecutor`.

  ```
  //Executing Generated Queries
  SPLExecutor Exec  = new SPLExecutor (conn);
  ```

```
try
{
    Exec.initialize();
 }
catch ( SQLException e)
{
    System.out.println ("Cannot initialize the SPL Executor"+ e);
}
String returnVal= gqExec.executeCommand(queryString);
```

`returnVal` contains the resultant data obtained by executing the query. This data is passed to **Display Result** module, which displays the data in different formats.

6.2.7.3 Closing the Connection

After the query execution, we have to close the data warehouse connection. The code shown below is used to close the connection.

```
try
{
       conn.close();
 }
catch ( SQLException e)
{
System.out.println("Cannot close the connection  " + e);
}
```

We discuss the implementation details of the **Display Result** module in the following section.

6.2.8 Displaying Result

Displaying Results module obtains the results from the **Query Execution** module. This module displays the results in different formats like tables and charts. Here a user can further perform the OLAP operations like drill-down, roll-up, slice and dice. We used `JTable`, `JScrollPane` and `ImageProducer`

class to implement this module. The following code shows the implementation of this module.

```
//This Function is used to display the data in tabular format.
public void displayResults(Object[][] tableData[][], Object [] tableHeader[])
{
        JTable chart = new JTable();
        DefaultTableModel chartModel = (DefaultTableModel)chart.getModel();
        chartModel.setDataVector(tableData,tableHeader);
        topRight.setViewportView(chart);
}

//This Class is used to create a pie chart for the resultant data
Public class PieChartProducer implements ImageProducer
{
        int ImageWidth;
        final static int ImageHeight = 300;
        final static int PieWidth    = 200;
        final static int PieHeight   = 200;
        final static int Radius      = PieWidth/2;
        int Inset;
        public PieChartProducer(String s[][], int rows, int columns, String
        chartHead[])
        {
                this.graphics = graphics;
                currentTheta = 0;
                Inset = columnCount * 150;
                ImageWidth = columnCount *400;
                JFrame f = new JFrame();
                f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                f.setVisible(true);
                image = f.createImage(ImageWidth, ImageHeight);
                graphics = image.getGraphics();
                drawPieGraph();
        }
        //Function for drawing pie chart
        public drawPieGraph()
        {
                //code for drawPieGraph
        }
}
```

We can perform different OLAP operations on the resultant data. During the drill-down, roll-up and dicing operations we use the general procedure for generating the queries as we explained in the above sections. Where as the

implementation of slicing operation is little bit different from the general procedure and needs a separate implementation. In the **Slice** operation, we used `CheckRender`, `CheckNode` classes to display a check box by the side of each tree node. The following code is the implementation of `CheckRender` class.

```
//This class is used to display a check node tree
class CheckRenderer extends JPanel implements TreeCellRenderer
{
	protected JCheckBox check;
	protected TreeLabel label;
	public CheckRenderer()
	{
		setLayout(null);
		add(check = new JCheckBox());
		add(label = new TreeLabel());
		check.setBackground(UIManager.getColor("Tree.textBackground"));
		label.setForeground(UIManager.getColor("Tree.textForeground"));
	}
}
```

The following code is the implementation of `CheckNode` class.

```
//This class is used to display a check node tree
class CheckNode extends DefaultMutableTreeNode
{
	protected int selectionMode;
	protected boolean isSelected;
	public CheckNode()
	{
		this(null);
	}
	public CheckNode(Object userObject)
	{
		this(userObject, true, false);
	}
}
```

6.3 The Platform

We used Java programming language to design the graphical user interface and oracle 9i is used to build the data warehouse. Windows XP is the operating system on which this application is developed.

We used Java programming language because of its unique features like platform independence and portability. It has many advantages over other programming languages. Java is a portable, interpreted, high-performance, robust, secure, simple object oriented programming language. The main reason for choosing Java is it's built in support for designing graphical user interfaces. Java swings [12] package that comes with Java offers various widgets (buttons, text boxes, lists etc) which make the interface both graphical and user friendly. This package provides complex structures like table views and tree structure views which are very useful in designing GUI.

Using Java we can write standalone Java applications which can be launched from a browser with Java's WebStart technology, or they can create HTML applications through servlets, JavaServerPages(JSP), and Oracle User Interface XML (UIX), which we can access live data from an Oracle Database.

The unique feature of Java is its portability. Programs written in Java are platform independent because of the 'bytecode' concept introduced by the Sun Microsystems. Java programs on compilation will be converted into bytecodes. These bytecodes will then be executed by the Java Virtual Machine (JVM). The JVM can be implemented either in the hardware or software on a particular machine. Hence the statement 'compile once, run any where'.

The Oracle 9i Enterprise Edition OLAP Option extends the analytic capabilities of the Oracle database by providing new multidimensional data types, and a multidimensional calculation engine. Not only it extends the analytic capabilities of Oracle data warehouses but also supports core Object Oriented features like inheritance, polymorphism, etc. It supports both the object and relational data storage. Oracle 9i is compatible with relational, object and multidimensional data. Irrespective of the type of the data, the Oracle 9i OLAP option will create analytical layer and processing will be done in multidimensional way.

Because of these advantages, we opted to implement the proposed system using Java and Oracle. We used Java 1.4.2 to develop the GUI and Oracle 9i is used as backend.

CHAPTER 7

CONCLUSION

In this thesis we have proposed a framework for personalization of OLAP Querying. There are two aspects to the personalization of queries. The first is to provide a mechanism for building user profiles and the second is to incorporate the user profile in OLAP queries. Both these features have been addressed in this thesis.

The Graphical User Interface (GUI) described in this thesis provides interfaces for user account management, profile management, querying and displaying results.

In this system it is possible to create new user profile or edit the existing profile. While creating a user profile, the dimensions, levels, values and measures can be selectively chosen and conditions imposed thereupon. This user profile is maintained by the system and incorporated in the queries posed by the user.

It is possible in this system to perform all the standard OLAP operations i.e. drill-down, roll-up, slicing and dicing for analyzing the data. The operations however, are restricted to operate on the dimensions, levels, values and the measures of the user profile. Within the profile, the user can choose any of the dimensions, levels and measures for analysis. Based on user selections, a query will be generated automatically which integrates the conditions that are present in the user profile.

It is possible to view the result in different formats like tables and graphs. The result is first displayed in tabular format. If the user wishes to see the result as a Pie chart he can do so by choosing **Draw Chart** option from **Run** menu. This facility, we believe, will support decision making.

We have used Java 1.4.2 to develop the front-end and Oracle 9i is used as back-end.

7.1 Features of the System

- This system provides a user-friendly interface to build user profiles.
- This graphical user interface effectively personalizes the OLAP queries using user profiles.
- The user does not require any prior knowledge of query languages like SQL.
- We can have different types of reports
- This system is platform independent.
- Multiple users can use this system at a time.

7.2 Future Enhancements

- The provision for displaying the data in different charts other than Pie Charts can be incorporated.
- This application can be extended to support web based information delivery and analysis of data.

REFERENCES

1. E.F. Codd, S.B. Codd, and C.T. Salley. *Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate*. E.F. Codd & Associates, White paper, 1993.

2. Georgia Koutrika and Yannis Ioannidis. *Personalization of queries in database systems.* Proceedings of the 20th International Conference on Data Engineering (ICDE'04), pages 597–608. IEEE Computer Society, 2004.

3. Jan Chomicki. *Preference formulas in relational queries.* ACM Transactions on Database Systems, Vol. 28, No. 4, December 2003, Pages 427–466.

4. James A. Senn, "*Analysis and Design of Information Systems*", Second Edition, McGraw-Hill, Inc, New York, 1989.

5. Ladjel Bellatreche , Arnaud Giacometti , Dominique Laurent, *A personalization framework for OLAP queries*, Proceedings of the 8th ACM international workshop on Data warehousing and OLAP, November 04-05, 2005, Bremen, Germany.

6. OLAP Council. *The APB-1 Benchmark.* 1997. Available at http://www.olapcouncil.org/research/bmarkly.htm

7. OLAP Council. *OLAP AND OLAP Server Definitions.* 1997 Available at http://www.olapcouncil.org/research/glossaryly.htm

8. Part No: B10333 - 02. *Oracle OLAP Application Developer's Guide 10g Release 1(10.1),* Oracle Corporation, December 2003.

9. Part No: B14223-02. *Oracle Data Warehousing Guide 10g Release2 (10.2)*, Oracle Corporation, December 2005.

10. Paulraj Ponniah. *Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals.* John Wiley & Sons, New York, 2001.

11. Ralph Kimball, Margy Ross. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses.* John Wiley & Sons, New York, 2002.

12. Sun Microsystems Documentation. *Creating a GUI with JFC/Swing.* Available at: http://java.sun.com/docs/books/tutorial/uiswing/TOC.html

13. William H. Inmon. *Building the Data Warehouse.* Fourth Edition, Wiley Dreamtech India (P) Ltd, New Delhi, 2005.

BIBLIOGRAPHY

1. Andreas S. Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, and Yannis Vassiliou. *Advanced visualization for OLAP*. Proceedings of the 6th ACM international workshop on Data warehousing and OLAP, November 07, 2003, New Orleans, Louisiana, USA.

2. Jennifer Widom. *Research Problems in Data Warehousing.* Proceedings of 4th international Conference on Information and Knowledge Management (CIKM), November, 1995.

3. Jiawei Han, Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2004.

4. Jun Yang. *Temporal Data Warehousing.* Ph.D Thesis, Stanford University, 2001.

5. Nebojsa Stefanovic. *Design and Implementation of On-Line Analytical Processing (OLAP) of Spatial Data.* Master of Science Thesis, University of Belgrade, Yugoslavia, 1993.

6. Part No B10335-02. *Oracle OLAP Developer Guide to the OLAP API 10g.* ORACLE Corporation, 2003.

7. Part No B10979-02. *Oracle Database JDBC Developer's Guide and Reference10g.* ORACLE Corporation, 2004.

8. Part No B10799-01. *Oracle Database Application Developer's Guide-Object –Relational Features10g.* ORACLE Corporation, 2003.

9. Part No B10795-01. *Oracle Database Application Developer's Guide-Fundamentals 10g.* ORACLE Corporation, 2003.

10. Part No B10339-02. *Oracle OLAP DML Reference 10g.* ORACLE Corporation, 2003.

11. Part No B12180-01. *Oracle OLAP Analytical work space Java API Reference 10g.* ORACLE Corporation, 2003

12. Rakesh Agrawal , Edward L. Wimmers, *A framework for expressing and combining preferences,* Proceedings of the 2000 ACM SIGMOD international conference on Management of data, p.297-306, May 15-18, 2000, Dallas, Texas, United States

13. S. Chaudhuri and U. Dayal. *An overview of data warehousing and OLAP technology.* ACM SIGMOD Record, v.26 n.1, p.65-74, March 1997.