A System for Query, Analysis and Visualization
of a Multi-Dimensional Relational Database

A graduate project submitted in partial fulfillment
of the requirement for the degree of
Master of Science in Computer Science

by
Yahui Peng

May 2002
The graduate project of Yahui Peng is approved:

Professor David Salomon

Professor SonPham

G. Michael Barnes, Chair

California State University, Northridge
TABLE OF CONTENTS

Signature page ................................................................. ii
List of Figures ................................................................. v
List of Tables ................................................................. vii
Abstract ............................................................................. viii
CHAPTER 1 OVERVIEW .......................................................... 1
  1.1 Motivation ................................................................. 1
  1.2 Java .......................................................................... 2
  1.3 Data Warehouse ......................................................... 3
  1.4 Objective of the project ................................................ 6
CHAPTER 2 TECHNOLOGY INTRODUCTION .................................. 7
  2.1 Server/Client architecture .............................................. 7
  2.1.1 Three-tier architecture ........................................... 7
  2.1.2 Communication mechanisms ..................................... 8
  2.1.2.1 Message-passing mechanism ............................... 8
  2.1.2.2 Remote procedure call mechanism ...................... 10
  2.1.2.3 Peer-to-peer communications mechanism .......... 10
  2.1.2.4 Message-queuing mechanism .............................. 10
  2.2 RMI ......................................................................... 11
  2.3 OLAP ......................................................................... 14
  2.3.1 Conceptual data model ............................................ 14
  2.3.2 OLAP operations .................................................... 15
  2.3.3 Types of OLAP Servers ......................................... 16
  2.3.4 Schema in ROLAP .................................................. 17
  2.3.4.1 Star schema ..................................................... 18
  2.3.4.2 Snowflake schema ............................................ 20
  2.3.4.3 Comparison of Star and Snowflake schemas .......... 22
  2.4 JDBC ......................................................................... 23
<table>
<thead>
<tr>
<th>FIG</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Data warehouse architecture</td>
<td>5</td>
</tr>
<tr>
<td>2-1</td>
<td>Three-tier architecture</td>
<td>8</td>
</tr>
<tr>
<td>2-2</td>
<td>Message-passing mechanism</td>
<td>9</td>
</tr>
<tr>
<td>2-3</td>
<td>RPC mechanism</td>
<td>9</td>
</tr>
<tr>
<td>2-4</td>
<td>Message-queuing mechanism</td>
<td>11</td>
</tr>
<tr>
<td>2-5</td>
<td>RMI architecture</td>
<td>11</td>
</tr>
<tr>
<td>2-6</td>
<td>RMI Naming service</td>
<td>12</td>
</tr>
<tr>
<td>2-7</td>
<td>Conceptual data model</td>
<td>15</td>
</tr>
<tr>
<td>2-9</td>
<td>Star schema</td>
<td>18</td>
</tr>
<tr>
<td>2-10</td>
<td>Snowflake schema</td>
<td>21</td>
</tr>
<tr>
<td>2-11</td>
<td>JDBC architecture</td>
<td>24</td>
</tr>
<tr>
<td>2-12</td>
<td>JDBC Class Hierarchy</td>
<td>25</td>
</tr>
<tr>
<td>3-1</td>
<td>Database schema</td>
<td>28</td>
</tr>
<tr>
<td>3-2</td>
<td>RMIClass package</td>
<td>35</td>
</tr>
<tr>
<td>3-3</td>
<td>Class diagram of server side</td>
<td>35</td>
</tr>
<tr>
<td>3-4</td>
<td>The layout design</td>
<td>40</td>
</tr>
<tr>
<td>3-5</td>
<td>The rule list (under the View 1D function)</td>
<td>40</td>
</tr>
<tr>
<td>3-6</td>
<td>User activity diagram</td>
<td>41</td>
</tr>
<tr>
<td>3-7</td>
<td>Class diagram of the client side</td>
<td>42</td>
</tr>
<tr>
<td>4-1</td>
<td>Top view</td>
<td>45</td>
</tr>
<tr>
<td>4-2</td>
<td>View 1D display</td>
<td>46</td>
</tr>
<tr>
<td>4-3</td>
<td>The rule list</td>
<td>47</td>
</tr>
<tr>
<td>4-4</td>
<td>Specify the value by clicking</td>
<td>47</td>
</tr>
<tr>
<td>4-5</td>
<td>The rule list</td>
<td>48</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1-1 Comparison of operation databases and data warehouses ....................... 4
Table 2-1 Comparisons of ROLAP and MOLAP ............................................. 16
Table 2-2 A relational table ........................................................................... 17
Table 2-3 Records in the time table ............................................................... 19
Table 2-4 Records in the fact table ............................................................... 19
Table 3-1 Database tables ............................................................................. 29
ABSTRACT

A System for Query, Analysis and Visualization of a multi-Dimension Relational Database

by

Yahui Peng

Master of Science in Computer Science

In this project, a 3-tier distributed system is developed. It contains a small data warehouse implemented by a standard relational database for a business of coffee shops, an application server for both handling Remote Method Invocation (RMI) operations and accessing the database, and a client application component (applet). The client applet provides a Graphical User Interface (GUI) environment for visualizing data in different views by one dimension at a time or multiple dimensions in form of bar chart, line chart, pie chart, ring chart or statistical chart. It is hoped that visualization would help users make decisions better and sooner. To achieve system integration, the technologies of Online Analysis Processing (OLAP), Java Remote Method Invocation (RMI), Swing, Graphics 2D, Drag & Drop, Jar, and Java Database Connectivity (JDBC) were needed and discussed. In addition, the design and the future extension to the project were presented.
CHAPTER 1 OVERVIEW

This paper describes the development of a system for query, analysis and visualization of a multi-dimensional relational database. This is a three-tier distributed system that contains a small business data warehouse, an application server and a client applet. The client applet provides a highly user-interactive environment for viewing data graphically in forms of dimensional hierarchy. With rich graphical expression, users can easily recognize the trend, get to the fact, predict the future and make decisions better and sooner. The client applet provides 2 major functions: view one dimension at a time, and view multi-dimension at a time. Further, the user is able to choose the chart type to view the data, such as bar, pie, ring, line, table, 3D bar or statistical plot.

1.1 Motivation

Information grows increasingly nowadays, and finding useful data in the large database is getting difficult. People learn knowledge by images and symbols faster than by texts and numerical numbers, so visualizing data helps users to learn, to find the useful and interesting patterns, and to get insight into the facts. In addition, the Internet is everywhere. It’s impossible to provide local service only. A distributed system [9, 12, 16, 28] meets the requirements of the shared resources, security, spreading the workload, and the clear scope of responsibilities for each component. That’s why this project developed a distributed system with a database analytical tool that handles data visualization.
1.2 Java

Java compared to other object-oriented languages has two important advantages: security and portability. As a java program runs on top of the Java Virtual Machine (JVM) [23], it makes "write once, run anywhere" possible. With its platform-independent feature, java language meets the requirement of this project. Furthermore, it has rich GUI components [14] that help the user interface (UI) design with functionality and aesthetics. Also, java has a powerful set of application programming interfaces (APIs) such as Java Database Connectivity (JDBC) supporting database connectivity, with clear documentations. Java supports the developers a lot. A Java application's characteristics [17] are simple, architecture neutral, object oriented, portable, distributed, high performance, interpreted, multithreaded, robust, dynamic, and secure.

The most common types of programs [23] written in the Java programming language are applets, applications and servlets. An applet is a program that runs within a Java-enabled browser. An application is a standalone program that runs directly on the Java platform. A servlet can be thought of as an applet that runs on the server side that provides web developers with a simple, consistent mechanism for extending the functionality of a web server. The servlet acts like a CGI program.

Only applications and applets are suitable on the client side. The advantage of applets over applications is that the applet is downloaded dynamically and executed inside the browsers that include the JVM. There is no installation required for applets on the client side. However, the applet cannot access any resources on the local host (such as file system) due to security issues.
1.3 Data Warehouse

A data warehouse [4,5,7,8] is a subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making. Typically, the data warehouse is maintained separately from the organization’s operational databases. There are many reasons doing this [4,20]. The data warehouse supports Online Analytical Processing (OLAP), and the functional and performance requirements are quite different from those of Online Transaction Processing (OLTP) applications traditionally supported by the operational databases. Query throughput and response time of data warehouses are more important than transaction throughput. Table 1-1 is the comparison of relational databases and data warehouses [26].

Fig 1-1 shows a typical data warehousing architecture [4]. It includes tools for extracting data from multiple operational databases and external sources; for cleaning and transforming and integrating this data; for loading into data warehouse; and for periodically refreshing the warehouse to reflect the updates at the sources. The data in the main warehouse is managed by one or more warehouse servers which present multi-dimensional views of data to a variety of front-end tools: query tools, report writers, analysis tools, and data mining tools. Data warehouses provide [26] Decision Support, Data Mining and Query/Reporting. A DBMS that runs these decision-making queries efficiently is sometimes called a Decision Support System (DSS) [20]. OLAP is one of the technologies in the Decision Support System. OLAP extends the spreadsheet model and provides multi-dimensional views of data. Data is modeled into conceptual data cubes. OLAP operations include rollup, drill-down, slice and pivot. These operations manipulate the data cubes to produce different views of data.
There are many commercial database analytical tools in the market of Decision Support Systems [24], and each of them has unique features.

<table>
<thead>
<tr>
<th>Operational Databases</th>
<th>Data warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Mostly updates</td>
<td>1. Mostly retrieves</td>
</tr>
<tr>
<td>2. Many small transactions</td>
<td>2. Long, complex queries</td>
</tr>
<tr>
<td>3. MB-TB of data</td>
<td>3. GB-TB of data</td>
</tr>
<tr>
<td>5. Raw data</td>
<td>5. Summarized, consolidated</td>
</tr>
<tr>
<td>6. Clerical users</td>
<td>6. Decision makers, analysts as users</td>
</tr>
</tbody>
</table>
Fig 1-1 Data warehouse architecture
1.4 Objective of the project

This project has two objectives. First and foremost is for me to learn and gain the experience working with the Java language environment. This includes the building of the event-driven GUI program, the Internet communication between different machines, database design and database connectivity. The second is to develop a database analytical tool to visualize data. Through developing this project, I hope to get more sophisticated with Java API's and with data visualization.

The next chapter introduces the technologies used in the project. Chapter 3 discusses the design of the three tiers in this distributed system. Chapter 4 gives the manual and installation. Chapter 5 summarizes this project.
CHAPTER 2 TECHNOLOGY INTRODUCTION

This project implements a distributed system. The client has the ability to visualize the data from a remote database. In this chapter, I’ll introduce four technologies that are applied to the project. They are Server/Client architecture, Java Remote Method Invocation (RMI), Online Analytical Processing (OLAP) and Java Database Connectivity (JDBC).

2.1 Server/Client architecture

The distributed system has the advantages of resource sharing, accessing remote resources, security, easy maintenance, and clear scopes of responsibilities [9], so it has become popular these days. Client/Server architecture can be described as a two-tier system. One tier is the client and the other is the server. However, it can be extended to any depth to narrow the scope of responsibilities at each level, namely n-tier architecture. This project implements a three-tier distributed system.

2.1.1 Three-tier architecture

This project designed a three-tier server/client architecture. They are a database server, a client browser and an application server. The design is shown in Fig 2-1. The client is a web browser that runs a Java applet downloaded from the web server, and then the applet sends the user requests to the application server and gets the responses from it. The application server can directly access the database server with JDBC APIs, and handles the client requests. In fact, it’s the bridge between the client and the database server. With the middle tier, the strength is to isolate the database from users with security support,
and to provide ease of maintenance. If the behaviors of the database change, only the middle tier needs to be updated.

![Three-tier architecture diagram]

Fig 2-1 Three-tier architecture

2.1.2 Communication mechanisms:

There are four distinct technologies that are commonly used in a distributed environment for handling communications between application components. They are not necessarily mutually exclusive. A distributed system may adopt a combination of these four technologies:

- Message-passing mechanism
- Remote procedure call mechanism (RPC)
- Peer-to-peer communication
- Message-queuing mechanism

2.1.2.1 Message-passing mechanism

Message-passing mechanism [28] consists of the direct exchange of units of data between client and server components that are running on different machines. With message passing, client A sends into the network a message whose destination is server B. Server B receives the message, performs required processing, and sends the message carrying
the results back to client A. The disadvantage is that the developer needs to define the protocol to be used and it may be complicated. A diagram illustrating message passing is shown in Fig 2-2.

Fig 2-2 Message-passing mechanism

Fig 2-3 RPC mechanism
2.1.2.2 Remote procedure call mechanism

Remote procedure call (RPC) mechanism [28] extends the familiar call-programming paradigm from the local environment to the distributed environment. With RPC facility, the calling procedure and the called procedure can execute in different systems over the Internet. RPC mechanisms make it possible to hide the detailed work involved in the remote functions, and make it easy for the application developer to design, implement and maintain a distributed application. RPC usage is shown Fig 2-3. This project uses one of RPC implementations, i.e. Remote Method Invocation (RMI), and I’ll talk about it in more detail in section 2.2.

2.1.2.3 Peer-to-peer communications mechanism

Peer-to-peer communication mechanism [28] is similar to message-passing mechanisms and provides integrated support for using a set of two-phase commit operations (pre-commit and post-commit). It allows client and server application components to communicate asynchronously and still remain a shared context across a dialog that may consist of many interactions.

2.1.2.4 Message-queuing mechanism

Message-queuing mechanism [28] supports asynchronous processing. The source process submits a message and does not wait for a response but immediately performs other work. The destination may or may not be active at the time the message is submitted. If not active, the message-queuing mechanism may wait before delivering the message or it may take steps to activate the destination process. A diagram of message-queuing is shown in Fig 2-4.
2.2 Remote Method Invocation (RMI)

RMI [9] is a RPC mechanism. A RMI application is often comprised of two separate programs: a server and a client. A typical server application creates some remote objects, makes references to them accessible (binds to registry), and waits for clients to invoke methods on these remote objects. A typical client application gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass
information back and forth. The RMI architecture is shown in Fig 2-5. A "remote reference" is a pointer to a proxy object (Stub) in the local heap. Stub contains information that allows it to connect to a remote object, which contains the implementation of the methods. The arguments and the results transmitted over the Internet must be serializable. Serialization is a specification for how to marshall and unmarshall Java objects, i.e., converting objects to the byte stream and back and forth. Therefore, the communication between Stub and Skeleton is handled by an RMI facility that hides the implementations from the developer. A remote object on the client side is treated just like a local object. A client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation. Any return values provided by the implementation are sent back to the proxy and then to the client's program.

But now the question is how the client finds the remote object. On the server side, the server application creates remote objects and binds them to rmiregistry with the service names. The client looks up a remote object by the name in the server registry. If found, it returns the reference (Stub) associated with the remote object.
For example,

**Server Side:**

- Run Java RMI Registry service
  
  ```
  >rmiregistry &
  ```

- Start the application server
  
  ```
  >java RMIConn.DBConnServer & //RMIConn is the package name
  ```

Inside the application server program,

```java
//create remote objects
DBProfileImpl db = new DBProfileImpl();
//binds to registry with "DBProfile" name
Naming.rebind("DBProfile", db);
```

**Client Side**

- Inside the client program,

  ```java
  DBProfile db = (DBProfile)
  Naming.lookup("vrlab.ecs.csun.edu/DBMetaData");
  ```

After executing the client program, it looks up by name on the specific host.

If the name is found, the reference (Stub) of the remote object is returned to the client.
2.3 Online Analytical Processing (OLAP)

Data warehousing and online analytical processing (OLAP) [1,2,4,5,21] are essential elements of Decision Support System (DSS), which has increasingly become a focus of the database industry. Usually, the data warehouse is maintained separately from the organization’s operational database because of their performance requirements and different responsibilities. Data warehouses supporting OLAP, offer the services of Decision Making, Data Mining and Query/Reporting, and perform ad hoc queries. Entity-Relationship diagram and normalization techniques are popularly used in operational databases [4]. However, they are inappropriately applied to the design of data warehouses. Data warehouses focus on the efficient query performance and data loading. Entity-Relationship diagram and normalization can’t satisfy the performance requirements [4].

2.3.1 Conceptual data model

To facilitate complex query processing, the data in a warehouse is typically modeled multidimensionally. For example, in a sales data warehouse, time, location and product may be the dimensions of interest. The dimensions of time, location and product can be further organized into a hierarchy respectively.

Time: year → quarter → month
Location: state → city → store
Product: category→ tem → size

The conceptual data model for this example is shown in Fig 2-7. This data cube has three dimensions, and each dimension has three levels. In the data cube, each of numerical measures depends on the set of the dimensions. For example, the dimensions associated
with a sales measure may be year, city and size. The feature of the conceptual data model is that it performs aggregation efficiently.

2.3.2 OLAP operations

OLAP has four basic operations, pivot, roll-up, drill-down and slice. These operations manipulate the data cube to provide different views of the data.

- Pivot: re-orient the multidimensional view. A simple sample of pivot is that it selects two dimensions (Product and Location) to perform aggregation and forms a new data cube.

- Roll-Up: Take the current data cube and perform a further group-by aggregation on one of the dimensions. The level of the specified dimension is up by one in the hierarchy.

- Drill-Down: the converse operation of roll-up.

- Slice: selection and projection in the specified dimensions.
2.3.3 Types of OLAP Servers

There are three kinds of data warehouses, Relational OLAP (ROLAP), Multidimensional OLAP (MOLAP) and Hybrid OLAP (HOLAP). A Data warehouse might be implemented on a standard relational DBMS, called ROLAP servers. The data is stored in relational de-normalized tables, and all the operations (pivot, roll-up, drill-down and slice) have to map to special queries. In contrast, MOLAP servers store data in a special data structure (array) and implement the OLAP operations through the data structure. A HOLAP server is a combination of both. The following Table 2-1 lists the features of ROLAP and MOLAP [27].

Table 2-1 Comparisons of ROLAP and MOLAP

<table>
<thead>
<tr>
<th>ROLAP</th>
<th>MOLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Fluid dimensionality</td>
<td>1. Bounded dimensionally</td>
</tr>
<tr>
<td>2. Multi-dimensional view</td>
<td>2. Cross-dimensional</td>
</tr>
<tr>
<td>3. Scale-very large</td>
<td>3. Row level calculations</td>
</tr>
<tr>
<td>4. Rapid changing</td>
<td>4. Read-write applications</td>
</tr>
<tr>
<td>5. Data-rich applications</td>
<td>5. Rules-rich applications</td>
</tr>
<tr>
<td>6. Data warehouses</td>
<td>6. Data marts</td>
</tr>
</tbody>
</table>

This project adopts ROLAP. The data is stored in a standard relational database (MySQL, [25]) to implement a conceptual data cube. In Chapter 3, the design of the database is discussed.
2.3.4 Schema in ROLAP

In order to store a conceptual data cube in a few relational tables, we must understand how to organize the tables. In a relational table, a row is also referred to as a tuple or a record; a column is referred to as a field.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Year</th>
<th>City</th>
<th>State</th>
<th>Item</th>
<th>Category</th>
<th>Sales</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2000</td>
<td>Los Angeles</td>
<td>CA</td>
<td>Cappuccino</td>
<td>Coffee</td>
<td>153,266</td>
<td>68,815</td>
</tr>
<tr>
<td>Q2</td>
<td>2000</td>
<td>San Francisco</td>
<td>CA</td>
<td>Espresso</td>
<td>Coffee</td>
<td>256,111</td>
<td>75,583</td>
</tr>
<tr>
<td>Q1</td>
<td>2001</td>
<td>Dallas TX</td>
<td>Orange</td>
<td>Juice</td>
<td>177,213</td>
<td>88,561</td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>2001</td>
<td>Houston TX</td>
<td>Apple</td>
<td>Juice</td>
<td>156,223</td>
<td>96,426</td>
<td></td>
</tr>
</tbody>
</table>

The data type in the field can be ordinal or quantitative. The fields can be partitioned into two types: parameter and measurement. In the above table, all the ordinal fields are parameters such as quarter, year, city, state, item and category; all the quantitative fields are measurements such as sales and profit. The parameters are further grouped into dimensions. Quarter and year belong to Dimension Time; city and state belong to Dimension Location; item and category belong to Dimension Product. A dimension hierarchy can have many levels. In the middle level, you can drill down to the lower level details, or roll up to the upper level summary. For example in the location dimension, cities can drill down to stores or roll up to states. The operation called “slice” is a selection operation such as specifying the state parameter with a value ‘CA’. Pivot operation is to reorient the dimensions, i.e., re-choose the dimensions in a different order to form a new data cube.
There are two schemas commonly used to organize fields in ROLAP: Star schema and Snowflake schema.

2.3.4.1 Star schema

In the Star schema [19], all relational tables are categorized into two types: dimension tables and fact tables, as shown in Fig 2-9. All the measurements along with their aggregates are stored in a single table, i.e., the fact table. Each of the dimension tables stores each of dimension information along with the added level indicator presenting the dimensional level. For example in the time dimension table, level 0 is for months, 1 for quarters, and 2 for years.

![Fig 2-9 Star schema]
Table 2-3 Records in the time table

<table>
<thead>
<tr>
<th>time_ID</th>
<th>level</th>
<th>month</th>
<th>quarter</th>
<th>year</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid0</td>
<td>0</td>
<td>02</td>
<td>Q1</td>
<td>2000</td>
<td>represents a record for the month Feb in 2000.</td>
</tr>
<tr>
<td>tid1</td>
<td>1</td>
<td>null</td>
<td>Q1</td>
<td>2000</td>
<td>represents a record for the first quarter in 2000</td>
</tr>
<tr>
<td>tid2</td>
<td>2</td>
<td>null</td>
<td>null</td>
<td>2000</td>
<td>represents a record for the year 2000</td>
</tr>
</tbody>
</table>

Table 2-4 Records in the fact table

<table>
<thead>
<tr>
<th>time_ID</th>
<th>product_ID</th>
<th>location_ID</th>
<th>sales</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid0</td>
<td>pid0</td>
<td>lid0</td>
<td>12389</td>
<td>1625</td>
</tr>
<tr>
<td>tid1</td>
<td>pid1</td>
<td>lid2</td>
<td>26975</td>
<td>2865</td>
</tr>
<tr>
<td>tid2</td>
<td>pid2</td>
<td>lid3</td>
<td>58596</td>
<td>6421</td>
</tr>
</tbody>
</table>

In Table 2-3, the record with tid2 in the time table stores the dimensional information (the year 2000). In Table 2-4, the record, whose time_ID is tid2, represents the aggregation data of the year 2000. During processing a query, the level indicator must be specified; otherwise, it causes a wrong result. For example, if we are looking for the total sales in 2000, i.e. tid2, only selecting the rows with year ‘2000’ and without providing the level indicator, will also pull out the records with tid0 and tid1 from the time table. After the join with the fact table, the sum of the data exceeds the actual data.
2.3.4.2 Snowflake schema

In contrast to the Star schema, the Snowflake schema [19] eliminates the error-prone processing of level indicators. First, all aggregation records are extracted from the fact table to form sub-fact tables (the tables with shade in Fig 2-10), and then each of the dimensional levels is stored separately into a sub-dimension table. Thus, the fact table is normalized by decomposing at the dimensional level. If we are looking for the total coffee sales for the first quarter in 2000 in CA,

\[
\text{category="coffee" quarter="Q1" year="2000" state="CA"}
\]

There are several ways to get the answer, and one of them is:

1. Get category_ID from the category table with “coffee” selection.
2. Get the time_KEY from the month table by the join of table month, quarter and year with “Q1” and “2000” selection.
3. Get the location_KEY from the store table by the join of table store, city and state with “CA” selection.
4. Get the records from the category fact table where category_ID, location_KEY and time_KEY are selected from 1,2 and 3.
The tables with shade are the fact tables.
2.3.4.3 Comparison of Star and Snowflake schemas

Below is the comparison of Star and Snowflake schemas:

1. The Star is easy to understand and implement. However the Snowflake has better aggregation performance. Roll-up and Drill-down are faster in Snowflake than in Star.

2. The tables in the Star are not normalized. Duplicate data is stored, so the size of the fact table may be big.

3. The tables in the Star are fewer, so it reduces the number of joins. In contrast, the join may involve a number of tables in the Snowflake.

4. The query processing in the star schema has to carry the level indicator, which causes potential errors.

In conclusion, because the project database is small, I chose the Star schema to design the database. To avoid the error-prone level indicators, I modified the star schema to fit this project. The detailed design is discussed in Chapter 3.
2.4 JDBC

From the start, the developers of the Java technology [23] at Sun were aware of the potential of Java working with databases. Starting in 1995 [16], they began working on extending the standard Java library to deal with SQL access to databases. What they first hoped to do was to extend Java so that it would talk to any random database, using only “pure” Java. It didn’t take them very long to realize that this is impossible: there are too many databases in the market, using too many protocols. Moreover, database vendors were all in favor of Sun providing a standard network protocol for database access. It would be useful if Sun provides a pure Java API for SQL access along with a driver manager to allow third-party drivers to connect to specific databases. Database vendors could provide their own drivers to plug into the driver manager. There would be a simple mechanism for registering third-party drivers with the driver manager. At last, Sun delivered Database Connectivity (JDBC) API. The rules for writing drivers were encapsulated in the JDBC driver API. (The JDBC driver API is of interest only to database vendor and database tool providers).

The protocol follows the very successful model of Microsoft’s ODBC, which provides a C programming language interface for database access. Both JDBC and ODBC are based on the same idea: programs written according to the JDBC API would talk to the JDBC driver manager, which, in turn, would use the drivers that were plugged into it to talk to the actual database.
The JDBC consists of the two layers [16] shown in Fig 2-11. The top layer is the JDBC API. This API communicates with the JDBC manager driver API, sending it the various SQL statements. The manager should communicate with the various third-party drivers that actually connect to the database and return the information from the query or perform the action specified by the query.

Fig 2-12 shows the relationships among the key JDBC classes [21]. The Connection class is important because all database operations are performed through a Connection object. A Connection object encapsulates transactions and is used to execute SQL statements against a database. Each Connection object may have multiple Statement objects associated with it, and a program can have multiple Connection objects. The DriverManager class is responsible for loading and maintaining a list of drivers and establishing connection through these drivers. Once a connection has been established, Statement objects can be created which are used to execute queries or updates against a
database. When database queries are executed that produce multiple rows, `ResultSet` objects are used to hold the data rows returned from the query.

Fig 2-12 JDBC Class Hierarchy
CHAPTER 3 DESIGN

To design an object-oriented program is not an easy task, especially for reusability, distributed objects, and easy maintenance. It is almost impossible to design a perfect distributed system the first time. The system is feasible only after several design modifications via an iterative process. This chapter includes five sections: system architecture, database design, Remote Method Invocation (RMI) design, application server design and client applet design.

3.1 System architecture

The system is built in a three-tier architecture with the client, the application and the database server. The architecture is shown in Fig 2-1. As mentioned earlier, the client applet is downloaded from the web server and runs within the client browser. The query request is generated by user interaction in the client applet and is sent to the application server. The communication mechanism between the server and the client is based on RMI. The application server forwards the query request to the local database and gets the result back. The result is wrapped into a remote object, which is sent back to the client applet by the application server. The client applet constructs the graphical display according to the content of the remote result object. This three-tier architecture prevents the user from accessing the database directly: a security concern.
3.2 Database design

I made up a set of business data for a company of coffee shops. The business data is stored in a small data warehouse implemented by a relational database called *MySQL*. *MySQL* is a free open source database available on the web [25].

There are nine parameters (year, quarter, month, category, item, size, state, city and store) and two measurements (sales and profit). The nine parameters are grouped into 3 dimensions: time, product and location. Each dimension has three levels. The fact table stores measurement data such as sales and profit, and the dimension tables store dimension data.

- **Time:** year → quarter → month
- **Location:** state → city → store
- **Product:** category → item → size

3.2.1 Schema design

As mentioned early in the chapter 2, there are two schema options to design data warehouses in a standard relational databases: Star schema and Snowflake schema. The Star is simple and easy to implement; however the Snowflake has better aggregation performance. Since the data set in this project is small, I adopt the Star schema to design the data warehouse. To avoid the error-prone level indicator, I modified the Star schema by removing both the level indicators and the aggregation records from the fact table. The modified Star schema is shown in Fig 3-1. Null is not allowed in any table.
The aggregation data is computed by query processing instead of retrieving it directly from the table in the original Star schema. For example, the total coffee sales for each quarter in 2000 in California State is processed by the query:

```sql
SELECT t.quarter, SUM(f.sales)
FROM fact AS f, time AS t, product AS p, location AS l
WHERE
  f.time_ID=t.time_ID AND f.product_ID=p.product_ID AND
  f.location_ID=l.location_ID AND l.state='CA' AND
  p.category='coffee' AND t.year='2000'
GROUP BY t.quarter
```

The above query, generated by user interaction in the client applet, is sent to the application server. The advantage of the modified Star schema is that it simplifies a visual query without using level indicators. The query gets aggregation data by the operations of GROUP BY and SUM.
3.2.2 Implementation

Here are the steps to build the database:
1. Log in MySQL (make sure you have the permission to create database and table)
2. Create a database called coffeeshop:
   ```
   CREATE DATABASE coffeeshop;
   ```
3. Exit MySQL
4. Execute SetUpDatabase class file

Inside SetUpDatabase class file define all tables. The tables used are listed below:

Table 3-1 Database tables

<table>
<thead>
<tr>
<th>The location table (35 records inserted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE <code>dim_location</code> (</td>
</tr>
<tr>
<td>location_ID smallint unsigned not null</td>
</tr>
<tr>
<td>state            char(2) not null default,</td>
</tr>
<tr>
<td>city             char(20) not null default,</td>
</tr>
<tr>
<td>store            char(20) not null default,</td>
</tr>
<tr>
<td>Primary Key(location_ID) );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The product table (24 records inserted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE <code>dim_product</code> (</td>
</tr>
<tr>
<td>product_ID smallint unsigned not null auto_increment,</td>
</tr>
<tr>
<td>category char(20) not null default,</td>
</tr>
<tr>
<td>item    char(20) not null default,</td>
</tr>
<tr>
<td>size    char(20) not null default,</td>
</tr>
<tr>
<td>Primary Key(product_ID) );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The time table (36 records inserted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE <code>dim_time</code> (</td>
</tr>
<tr>
<td>time_ID smallint unsigned not null auto_increment,</td>
</tr>
<tr>
<td>year     char(4) not null default,</td>
</tr>
<tr>
<td>quarter  char(2) not null default,</td>
</tr>
<tr>
<td>month    char(2) not null default,</td>
</tr>
<tr>
<td>Primary Key(time_ID) );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The fact table (45360 records inserted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE <code>fact</code> (</td>
</tr>
<tr>
<td>fact_ID      int unsigned not null auto_increment,</td>
</tr>
<tr>
<td>time_ID      smallint unsigned not null,</td>
</tr>
<tr>
<td>location_ID  smallint unsigned not null,</td>
</tr>
<tr>
<td>product_ID   smallint unsigned not null,</td>
</tr>
<tr>
<td>sales        int not null,</td>
</tr>
<tr>
<td>profit       int not null,</td>
</tr>
<tr>
<td>Primary Key(fact_ID) );</td>
</tr>
</tbody>
</table>
3.3 RMI design

To design a distributed system is not easy, especially since you need to consider operations on both sides of the communication channel. Design becomes more difficult with multi-tier systems. In this project, the application server and the client applet communicate by the RMI mechanism. The RMI server side (abbreviated as the server side) is the application server, and the RMI client side is the client applet (abbreviated as the client side). I designed the RMI implementations in the following 9 steps:

1. Define the interfaces on the server side.

   These interfaces must extend Java `Remote` interface. All methods must throw `RemoteException`. These interfaces will be exported to the client side that makes sure the behaviors of the remote objects on both sides are the same.

2. Code the classes that implement the interfaces in Step 1.

   These classes are responsible for establishing a database connection, building the database profile and constructing the datasets, respectively. Once the objects of these classes are created and registered to RMI naming server (discussed in Step 3), they wait for the client to connect.

3. Code the application server.

   `DBConnServer` is the application server that must be executed on the host (vrlab.ecs.csun.edu) machine before the client makes requests. The main work of the application server is to create four remote objects, `DBConnImpl`, `DBProfileImpl`, `DataSetImpl` and `DataSetSTImpl`, and to bind them with their service names to the RMI registry that offers naming service to the clients. Below is a code fragment illustrating this step.
public static void main(String args[])
{
    try{
        //create the remote objects
        DBConnImpl db = new DBConnImpl();
        DBProfileImpl dbProfile = new DBProfileImpl();
        DataSetImpl dataset = new DataSetImpl();
        DataSetSTImpl datasetST = new DataSetSTImpl();

        //bind to RMI Naming registry
        //the service names are provided for the client to look up
        Naming.rebind("DBConn", db);
        Naming.rebind("DBProfile", dbProfile);
        Naming.rebind("DataSet", dataset);
        Naming.rebind("DataSetST", datasetST);
    }

    4. Compile java files (four interfaces, four implementations, and the application server).

    >javac DBConn.java  //the interface
    >javac DBConnImpl.java  //the implementation class

    ...  
    >javac DBConnServer.java  //the application server

    5. Create the stubs and the skeletons.

    Once all the interfaces and the remote classes are done on the server side, it is time to
generate the stubs and the skeletons so that the client can actually call objects of those
classes remotely. The stubs and the skeletons know how to communicate with each
other. The stubs need to be copied to the client side in the seventh step. The RMI
compile tool (rmic) is part of the JDK. Only the remote implementation classes need
to be compiled.

    >rmic DBConnImpl
    >rmic DBProfileImpl
    >rmic DataSetImpl
    >rmic DataSetSTImpl

    RMI compilation generates four sets of the stubs and the skeletons

    (DBConnImpl_Stub.class, DBConnImpl_Skel.class, DBProfileImpl_Stub.class,
DBProfileImpl_Skel.class, DataSetImpl_Stub.class, DataSetImpl_Skel.class, DataSetSTImpl_Stub.class and DataSetSTImpl_Skel.class).

6. Start the RMI Naming service and Start the application server

Under Unix:
>rmiregistry &
>java DBConnServer &
or
>java RMIClass.DBClassServer&

Under windows:
>start rmiregistry
>start java DBClassServer
or
>start java RMIClass.DBClassServer

If the current working directory is above the package directory (RMIClass), you need to execute the file by specifying the package name and the class name. The client would connect to the server host on the Java RMI default port 1099 (or any allowable available port) of the RMI Naming service, look up the service name, and get the reference of the remote object if the service name is found.

7. Copy the four interfaces of the server side to the client side, and compile them.

The interfaces are DBClass.java, DBProfile.java, Databse.java and DatabseST.java.

8. Copy the four stubs of the remote classes from the server side to the client side.

The stubs are DBClassImpl_Stub.class, DBProfileImpl_Stub.class, DatabseImpl_Stub.class and DatabseSTImpl_Stub.class. Note that they must be in the same package hierarchy as on the server side. For example, DBClassImpl.class is in the RMIClass package on the server side, so DBClassImpl_Stub.class must be copied to the RMIClass package on the client side. The rules are also followed in Step 7.
9. Create the remote objects on the client side.

```java
DBProfile db;
DataSet dataset;
DataSetST datasetST;
String url = "rmi://vrlab.ecs.csun.edu/";

try{
    //create the remote objects by looking up the names
    db = (DBProfile) Naming.lookup(url+"DBProfile");
    dataset = (DataSet) Naming.lookup(url+"DataSet");
    datasetST = (DataSetST) Naming.lookup(url+"DataSetST");
} catch ...

//call the remote methods
String dbName = db.getDBName();
int dimCount = db.getDimCount();
int[][] data = dataset.getDataArray();
```

After the client gets the references (db, dataset and datasetST) of the remote objects by looking up the service names, the client can invoke the remote methods on these remote objects. These remote objects are now treated like local objects. However, all the passing parameters and the returned objects must be serializable, so the stubs and the skeletons can marshall and unmarshall these arguments and return values.

In this project, I didn’t define any specialized passing/returned class. All the passing arguments and the returned values are serializable by default such as String, String[], int, int[][], and Vector. If you define a customized class passing to or returned from the Internet, it needs to implement Serializable interface. Implementing Serializable is easy; there is no method to implement. The only thing you need to do is to declare it.

For example,

```java
class DataStructure implements Serializable{
```

In the following sections, I’ll discuss the designs of the application server and the client applet separately.
3.4 Application server design

The application server side has one package- RMIConn shown in Fig 3-2, which has 4 interfaces and 5 classes. The four interfaces extend the Remote interface, and must be exported to the client RMIConn package. These interfaces are the key of the RMI mechanism. It makes sure that the behaviors of the remote objects on both sides are the same. Fig 3-3 is the class diagram for the application server.

DBConnImpl is responsible for establishing a database connection. DBProfileImpl is to build the database profile such as the table names, the count of the dimensions, the level count in each dimension. DataSetImpl is to send the query via DBConnImpl object and to wrap the query result into an array data structure for easy use by the client later. DataSetSTImpl is similar to DataSetImpl. The difference is that DataSetSTImpl models multi-dimensional data with a more complicated data structure.

The classes of DBProfileImpl, DataSetImpl and DataSetSTImpl perform their operations with the inner DBConnImpl objects. Either the client gets the database connection directly by creating a remote object, which is looked up in the RMI Naming service with the name ‘DBConn’, or the client manipulates another remote object such as DataSetImpl, and the database connection is handled inside. I adopted the later case, just in case the user might mess up the database.

DBConnServer is the application server. The main work of the application server is to create four remote objects, DBConnImpl, DBProfileImpl, DataSetImpl and DataSetSTImpl, and to bind them to the RMI registry that offers naming service to client request.
3.5 Client Applet Design

This section includes the event-handling design, the layout design, the control design and the class design.

3.5.1 Event-handling design

Event handling is a crucial part of event-driven programs. Java provides several sets of event, event sources and event listeners. An event source may be a GUI component such as a window, a button or a list. An event is generated by user interaction such as mouse clicking and keyboard typing. Event sources have methods that allow you to register event listeners with them. When an event happens to the source, the source sends a
notification of that event to all listener objects that were registered for that event. Taking JButton as an example. Programmers are completely free to designate any object that

```java
JButton bn = new JButton();
bn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        // put code of handling,
        // or call a private method, such as resetData();
    }
});
```

implements the ActionListener interface as a button listener. The listeners can be an object of an existing class, an object of an inner class, or itself (object where the event is generated). In this project, I use anonymous inner classes for event handling, because of the following advantages:

1. Dedication. Each event listener is dedicated.
2. Sharing. The actual handling code is put in a private handling method, and the method of the inner class invokes this private handling method for handling events. The private handling method is shared by all event listeners.
3. Flexibility. If the program needs similar event handling with little modification, there is no need to write down all of them. All you have to do is make the anonymous inner class call the private handling methods in a different order. Moreover, the private handling methods are also shared in the non-event-handling situation.
4. Maintenance. It’s easier to manage classes when there are fewer classes.
3.5.2 Layout design

The layout for the client is shown in Fig 3-4. The buttons of the main functions (View1D and View MD) are located on the top. Below are the setting panel and the display panel. The button panel in Fig 3-4 under the function of View 1D provides the selection of the dimensional attributes, and the selected attributes are listed in order in the rule list. After the selection is done, the user can drill down to view the graphical display. The setting panel under the function of View MD provides the selection of the dimensional attributes, the selection of the measurements and the value assignments on the dimensional attributes. Once the setting is done, after clicking the ‘Show Bar’ button or the ‘Show Circle’ button, the graphical data display is shown below. The layouts of both View 1D and View MD are shown in Fig 3-4 respectively.

3.5.3 Control design

Under the function of ‘View 1D’, there are four controls- Undo, Reset, Up and Down.

- **Undo**: De-select the last listed attribute of the rule list if it is de-selectable.
- **Reset**: reset the button panel and clear the rule list.
- **Up**: Roll up by one level.
- **Down**: Drill down by one level.

Taking an example of Fig 3-5, the current level of the attributes is quarter. The user can undo the attribute selection and drill down as long as the lower level remains (item and city). Otherwise, roll up as long as the upper level remains (state and category). All user visualization requests are gathered via GUI controls, and stored in the Stack and
*DefaultTableModel* objects. The user activity diagram is shown in the upper part of Fig 3-6.

Under the function of ‘View MD’, there are four major control parts- the selection of the dimensional attributes, the selection of the measurements, the assignments to the dimensional attributes and the buttons. In Fig 3-4, the four lists are the implementations of ‘Drag and Drop’ [38]. These lists with the ‘Drag and Drop’ feature facilitate the user interaction. The user activity diagram is shown in the lower part of Fig 3-6.

3.5.4 Class design

The client side has two packages: RMIConn and viewdb. The files under RMIConn are associated with RMI communication. The files are under viewdb are associated with data construction, data visualization, and user interaction.

After the client gets the reference of the remote object (db), which has the information about how to communicate between the client stub and the server skeleton, the client may call the remote methods on this remote object and process the returned data.

```java
DBProfile db = (DBProfile)
    Naming.lookup("rmi://vrlab.ecs.csun.edu/DBProfile");
String name = db.getDBName();
```

Fig 3-7 shows the relationship among all the classes. The heart of the relationship is the GlobalStorage class that contains several remote objects and shared information needed by all components. All attributes and methods of GlobalStorage class are declared static to guarantee that only one copy exists; therefore, there is data integrity for all components accessed.
To clarify the notations in Fig 3-7, generalization refers to an “is a” relationship and dependency refers to a “using” relationship. In addition, a query is constructed by the user interaction in the setting panel. After the query is submitted by the user, Dataset Stub resets the query; thus, the query can be passed to the server host over the Internet. The remote Dataset object is reconstructed according to the query result on the server side. Next the client Dataset Stub retrieves updated data over the Internet. At last, all data-visualization displays on the client side are created.
Main functions

The button panel

The rule list

The setting Panel

The display panel

Main functions

The display panel

Fig 3-4 The layout design

Fig 3-5 The rule list (under the View 1D function)
Fig 3.7 Class diagram of the client side
This chapter includes two major sections: the administrator guide and the client user guide. The former focuses on the server side. The latter focuses on the client side. The source code for this project is available for Professor Barnes in the Computer Science department at CSUN.

4.1 Administrator Guide

4.1.1 System requirement

- A web server (any one you like)
- MySQL database server 3.23 or higher
- Java 2 Standard Edition (J2SE) 1.3 or higher
- JDBC driver, mm.mysql-2.0.4-bin.jar developed by Mark Matthews from MySQL.

4.1.2 Installation

1. Install Database server MySQL, please see Ref.[25].
   And create a user account and set permission.

2. Install J2SE 1.3 or higher.
   Please see Ref. [23].

3. Copy the jar file of JDBC driver into JAVA_HOME/jre/lib/ext
   Thus, the java runtime library includes the JDBC driver for extension.

4. Set up database by executing a java file:
   Open SetUpDatabase.java with your preferred editor and set the ID and password in the second statement in main function,
public static void main(String[] args) {
    SetUpDatabase set = new SetUpDatabase();
    set.url = 
        "jdbc:mysql://localhost/coffeeshop?user=YOUR_ID&password
        =YOUR_PASSWORD";

    Compile and run it.
    >javac SetUpDatabase.java
    >java SetUpDatabase


    Copy two client-side files (test.html and proj.jar) to $HOME/public_html/proj

    ‘proj.jar’ contains the two packages as well as the applet class.

6. Create a server-side directory $HOME/working_proj

    Copy the RMICConn directory of the server side to $HOME/working_proj

7. Start java registry service:

    $HOME/working_proj >rmiregistry &

8. Run application server:

    $HOME/working_proj >java RMICConn.DBConnServer &
4.2 Client User Guide

4.2.1 System requirement:
- A web browser (Netscape or IE)
- Java Plug-In 1.3.1 or higher

4.2.2 Installation:
No installation is required. The applet is downloaded automatically.

4.2.3 User manual:
After visiting http://vrlab.ecs.csun.edu/~yahui/proj/test.html, the screen shows two buttons: View 1D or View MD.
- View 1D: view data with one dimension at a time.
- View MD: view data with multiple dimensions at the same time.
4.2.3.1 Clicking View 1D

The split panel in Fig 4-2 contains the top with setting and the bottom with display. The setting panel consists of a button panel, a rule list and 4 buttons.

- Dimension Panel: the place where users choose the attributes by clicking.
- Rule List: show the chosen attributes in order. Also, it is highlighted with pale purple to notify users the current level.
- Undo: undo the selection.
- Reset: reset the selection and display.
- Up: roll up one level of the rule list.
- Down: drill down one level.

In the bottom display panel, the tab control on the right provides seven different data views: bar plot, pie plot, ring plot, line plot, 3D bar plot, table and statistical plot. With different points of view, the user should be able to recognize the trend easier.

Fig 4-2 View 1D display
Attribute selection:

Select state, category and quarter. The order is shown in the Rule List in Fig 4-3. Since quarter is lower than year in dimensional hierarchy, quarter is selected, so year is then disabled.

![Fig 4-3 The rule list](image)

Click down button to visualize data with the first attribute, i.e., state. The rule list is highlighted with pale purple on the first attribute. To drill further down, the user has to specify the attribute value by clicking in the display panel. After specifying the value, click down button to view the next level, i.e., category. Fig 4-4 shows how to specify the attribute value.

![Fig 4-4 Specify the value by](image)
The user can view data with other plots by clicking the tab. All of the plots provide the drill-down function except the statistical plot. In addition, there are restrictions on the Undo, Up and Down functions (the top right buttons). Taking an example of Fig 3-5,

- **Undo**: Only twice undo is allowed, because the current view attribute is *quarter* and there are only *item* and *city* available to undo.
- **Up**: Roll up by one level to view data with *category* and cancel the coffee value.
- **Down**: Drill down by one level to view with the *item* attribute after user specifies the attribute value for *quarter*.

Another example in Fig 4-5, the data is viewed with the last attribute, *quarter*, so the user can’t undo the selection, but can roll up. Furthermore, the user can’t drill down unless adding more attributes to the rule list. Fig 4-6 shows seven different data graphical display.

<table>
<thead>
<tr>
<th>dimension</th>
<th>attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dim_location</td>
<td>state</td>
<td>FL</td>
</tr>
<tr>
<td>dim_product</td>
<td>category</td>
<td>COFFEE</td>
</tr>
<tr>
<td>dim_time</td>
<td>quarter</td>
<td></td>
</tr>
</tbody>
</table>

Fig 4-5 The rule list
Fig 4-6 Seven data graphical display
Fig 4-6 Seven data graphical display (continued)
4.2.3.2 Clicking View MD

After clicking View MD button, the user should see Fig 4-7. The display consists of a setting panel and a display panel. In the setting panel, user can,

- Select the dimensional attributes.
- Select the quantity attributes.
- Set the query constraints in the assignment text field.
- Reset the setting panel and display panel.
- View BAR or CIRCLE plots.

Fig 4-7 ‘View MD’ display

To view the bar plot, at least one attribute of dimension and at least one attribute of quantity are needed. To view the circle plot, at least two attributes of dimension and two attributes of quantity are needed. In addition, the first and second selected dimensional attributes can’t be of the same dimension. (The first attribute is in the row header, and the others are in the column header.)
For example, choose *city*, *category* and *quarter* from the dimension list, and choose *sales* and *profit* from the quantity list. The assignment is set empty. The bar plot is shown in Fig 4-8. The circle plot is shown in Fig 4-9. In circle plot, the quantity of sales is proportional to the circle size, and the quantity of profit is proportional to color. In such plots, the user is able to view data with multiple dimensions at the same time.
CHAPTER 5 CONCLUSIONS

This project’s goal was to develop a three-tier distributed system that enables users to retrieve data from a remote database and visualize the data with different views. This was an interesting project. It is a common result in software engineering that the final product is not exactly what you designed at first. Usually, only through an iterative process can a robust product be produced. Having iterated through the design many times in this project, I gained experience in integrating the analysis, design, coding and testing for a project. In addition, I learned:

1. Debugging as early as possible, don’t wait in the end.
2. Designing a layered component architecture would save coding and maintenance time.
3. A more efficient approaches for event listeners.
4. To get more sophisticated with Java programming environment.
5. The strength of separating data model from view and control models.

Moreover, there are several enhancements I would to make to my project. They are:

1. Multithreading: The current program is a single thread. The user can only do one thing at a time. After retrieving data over the Internet, the data visualization is computing-intensive and time-consuming. One of the enhancements I would like to make is multithreading, doing several things at a time. One thread is for RMI communication, and the other is to build data visualization.
2. Multi-connection: Only one connection to the database on vrlab.ecs.csun.edu machine is allowed in the current version. It is not necessary that the web server and the database server be on the same machine. By installing the application server on the second database server, the client can access more databases on different machines.

3. EJB: The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. The application server can be included in such architecture to reduce implementation complexity by using existing models.

4. Stand-alone application: Due to security issue of applet limitation, the applet can’t save images or print the reports. With extending the prototype to an application version, these features can be added.

In conclusion, the project achieved the development of a three-tier distributed system that contains a small business data warehouse supporting OLAP, the application server and the client applet. The application server handles both the RMI communication to the client applet and the database connection. The client applet can send the query to the remote database via the application server, and visualize the query result in forms of bar, pie, ring, line, table, 3D bar and statistical plots. The graphical display of data can be shown by either one dimension at a time or multiple dimensions at a time.
REFERENCES

17. Introduction to Java™ Technology,
20. Data Warehousing, Decision Support & OLAP,
22. Data Warehousing & Decision Support,
26. Data Warehousing & Decision Support,