NOTE TO USERS

This reproduction is the best copy available.

UMI®

INVENTORY DATA WAREHOUSE SYSTEM WITH LINEAGE TRACING FOR SMALL RETAIL CHAIN

XIAOMING LUO

A THESIS

IN

JOHN MOLSON SCHOOL OF BUSINESS

PRESENTED IN PARTIAL FULFILLMENT OF
REQUIREMENTS
FOR THE DEGREE OF M.SC. (ADMINISTRATION)
CONCORDIA UNIVERSITY
MONTREAL CANADA

June 2005
©Xiaoming Luo, 2005

Concordia University

Abstract

Inventory Data Warehouse System with Lineage Tracing for Small Retail Chain
Xiaoming Luo

A data warehouse is one of the best solutions to integrate information from distributed, heterogeneous data sources. Different sized companies have different types of information management systems. The data warehouses, which can integrate these data sources together to provide summary views, are critical for the business managers to make operation management decisions.

Besides the data views of the data warehouses, sometimes, the managers also need lineage trace functions to get original data sources that contribute to the data views. Currently, more than 80% of retail companies are small sized businesses or business chains that consist of small sized companies. They need a data warehouse that integrates all kinds of data sources with lineage trace functions to improve their operations management.

Based on the analysis of the requirements of small businesses/business chains, we use the data warehouse [1, 2, 5, 7] and lineage trace [6, 9, 10] approaches to propose an information management data warehouse solution for the operation management of the small businesses/ business chains. In the same time, we develop an "Inventory data warehouse management system with lineage trace for small business chains" demo system. In this thesis is that we main focus on the implementation and demonstrating how to develop a local information

management component, how to integrate remote heterogeneous data sources to

a data warehouse, how to develop a data warehouse management system with the

latest Java data engine technologies and the best performance/cost rate database

technologies, and also show how lineage tracing is used in the data warehouse

and how to implement it as well.

Key words: data warehouse, data view, lineage trace, and data source.

Acknowledgements

I would like to express my special gratitude to all the people who gave me the great help during this thesis.

I am greatly indebted to my supervisor, Dr.Dennis Kira, who gave me a lot of helpful suggestions and encouraged my interest in my study. With his patient advice and constant help, I learned much useful knowledge and completed my thesis successfully.

I am pleased to thank Dr. Fassil NEBEBE and Dr. El Sayed ABOU ZEID as the examiners for the thesis. They gave me many useful comments during the work.

Finally, my special thanks are also due to the faculties and staffs in the MIS Department, John Molson Business School at Concordia University, who provided the large support during my master program's study.

Content

The List of Figures

1 Data Warehousing

1.1 Introduction

A *data warehouse* [1] is a repository storing integrated information for efficient querying and analysis. Information is extracted from heterogeneous sources [7, 3] as it is generated or updated. The information is then translated into a common data model and integrated with existing data at the warehouse. The data source can, for example, be accounting information, operation information, inventory information, customer information, etc. Data warehouse builds cross-reference information between these different data sources to enable data analysis. It groups data into subject areas so that user can find data earlier and faster than traditional procedure, such as query repeat information at remote data sources. It maintains historical data for trend analysis. Since data warehouse is not used for end user transaction processing, it can afford the resources to run computation intensive query for data analysis. When a user query is submitted to the warehouse, the needed information is already there, with inconsistencies and differences already resolved. This makes it much easier and more efficient to run queries over data that originally came from different sources. Key advantages of data warehousing include:

- Since query execution does not involve data translation and communication with remote sources, complex queries can be executed easily and efficiently.

1

- End users can use a single data model and query language.

- System design becomes simpler. For example, there is no need to perform query optimization over heterogeneous sources, a very difficult problem faced by other approaches.

- Information sources may be unreliable and may purge data. On the other hand, information at the warehouse is under the control of the warehouse users; it can be stored safely and reliably for as long as necessary.

From a business perspective, data warehouse provides a single and consistence data source. It makes the data collection process much easier and faster for the users. The user can answer different business questions by issuing queries to the data warehouse.

Potentially, better business decisions can be made in a shorter period of time.

Since the early 1990s, data warehouses have been at the forefront of information technology applications as a way for effectively business planning and decision-making. More and more companies are using data warehousing as a strategy tool to help them win new customers, develop new products, and lower costs. Searching through mountains of data generated by corporate transaction systems can provide insights and highlight critical facts that can significantly improve business performance.

The goal of a data warehouse is simply to provide quality data for the decision making of the organization, by providing techniques to get data in a form that they can use to make strategic decisions and solve business problem [2].

2

The warehouse is there to help decision maker do their job. It will allow them to ask questions not yet considered, and these questions should challenge corporate strategy.

The data is store and optimized for the purpose of analytical processing, turning meaningless data into useful information.

It provides an opportunity to exploit trends and influences across the entire organization by providing a consistent view and providing a solid base for business intelligence and allowing for complex business modeling.

The data warehouse is a business solution issue to a technical issue, but with benefit to both the IT department as well as the business [4, 5].

1.2 Basic Architecture of Data Warehouse

A data warehousing system collects data from multiple distributed sources and stores the integrated information as materialized views in a local data warehouse. Users then perform data analysis and mining on the warehouse views. Figure 1 shows the basic architecture of a data warehousing system. [6]

3
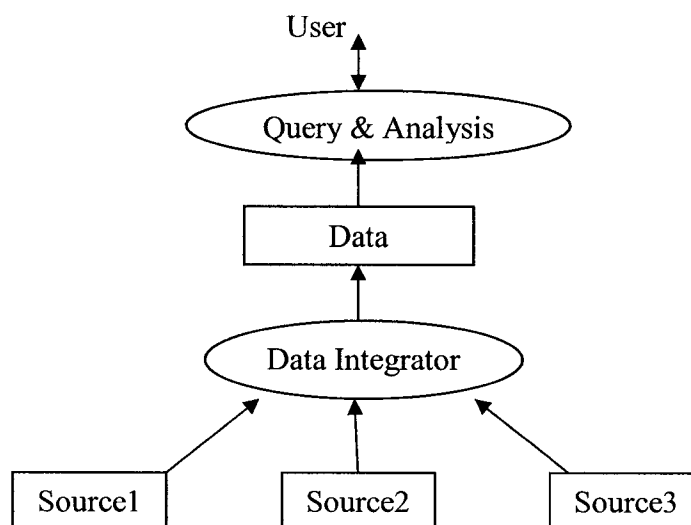
Figure 1 The Basic architecture of a data warehouse system

1.3 Data Warehouse Software

1.3.1 Introduction

Developing a Data Warehousing project, a warehousing team will require several different types of tools. These software products generally fall into one or more of the categories illustrated in Figure 2 and described below [8].

- **Extraction and transformation.** The warehouse team requires tools that can extract, transform, integrate, clean, and load data from source systems into one or more data warehouse databases.

- **Warehouse storage.** Software products are also required to store warehouse data and their accompanying metadata. Relational database management systems in particular are well suited to large and growing warehouses.

- **Data access and retrieval.** Different types of software are required to access, retrieve, distribute, and present warehouse data to its end users.

4
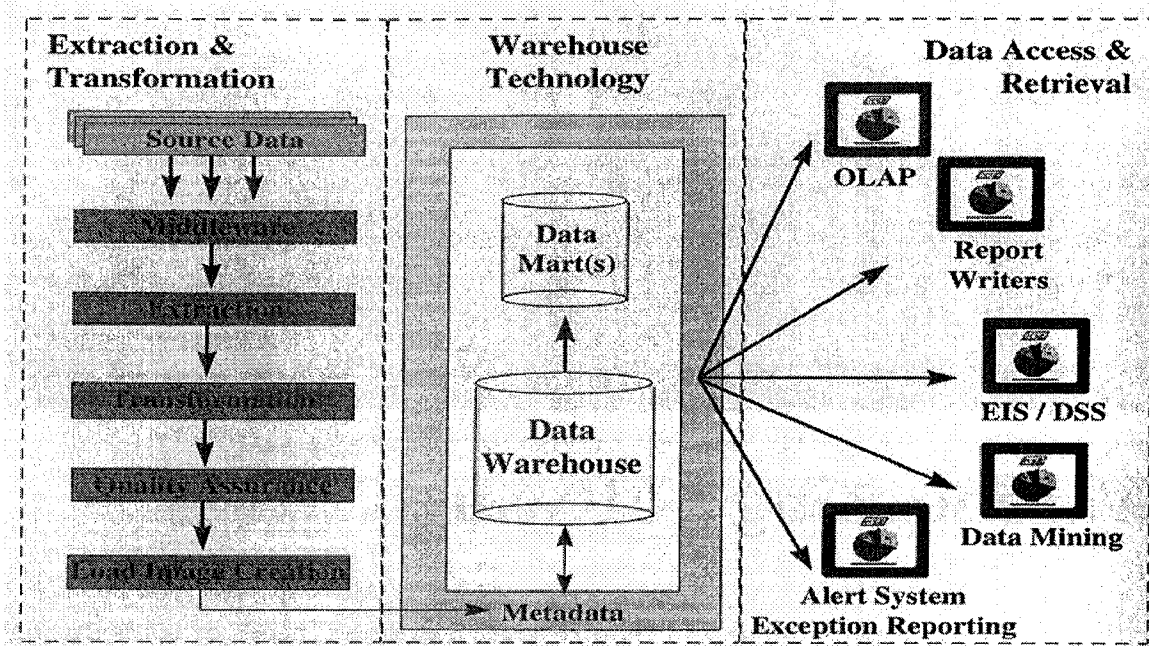
Figure 2 The Data Warehouse Software Components

1.3.2 Extraction Methods

There are two primary methods for extracting data from source systems (see Figure 3) [8]

- **Bulk extractions.** The enter data warehouse is refreshed periodically by extractions from the source systems. All applicable data are extracted from the source systems for loading into the warehouse.

- **Change-Based Replication.** Only data that have been newly inserted or updated in the source systems are extracted and loaded into the warehouse.

5
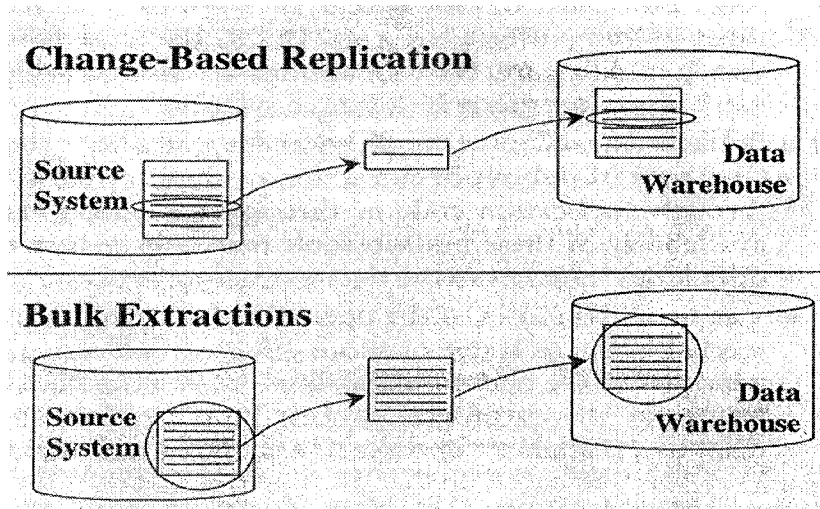
Figure 3 The Extraction Options

1.3.2 Transformation Methods

Most transformation tools provide the features illustrated in Figure 4: [8]

SOURCE SYSTEM	TYPE OF TRANSFORMATION	DATA WAREHOUSE
Address Field: #123 ABC Street XYZ City 1000 Republic of MN	**Field Splitting**	**No:** 123 **Street:** ABC Street **City:** XYZ City **Country:** Republic of MN **Postal Code:** 1000
System A, Customer Title: President **System B: Customer Title:** CEO	**Field Consolidation**	**Customer Title:** President and CED
Order Date: August 05, 1998 **Order Date:** August 08,1998	**Standardization**	**Order Date:** August 05, 1998 **Order Date:** August 08, 1998
System A, Customer Name: Jone W.Smith **System B: Customer Name:** Jone William Smith	**Deduplication**	**Customer Name:** Jone William Smith

Figure 4 The Data Transformations

1.4 The Data Warehouse Difference Among Small, Medium, and Large Sized Retail Companies

In the section, we will compare differing data warehouse solutions for small, medium, and large sized retail companies in the terms of operation managements, information management system's (IMS) characteristics.

7

1.4.1 Operational Management Differences

This section will outline the operation management differences that are the most related to the data warehouse system development.

The ownerships of different sized businesses are different: most of large sized corporations are public; and small and middle-sized companies are private, which lead to different operation management results. Public corporations have more money resources at lower cost, more professional staff, more market share, and cheaper goods supplier than small and middle-sized companies; consequently, the large sized corporations are in good position in the marketing competition and take more market share.

Most of the large sized corporations are multinational corporations that consisted of multi divisions that distributed in different locations, and they use centralized organizations structure. In each country, they have supplier centers to supply all their branches in the country. At the same time, they have buyer centers in the location where the merchandises are produced, like China.

Most of the large sized corporations have research and development centers, which study the corporation's expansion strategies, operation management refinements, market strategies, and product line development, so the corporations are always in the leading edge in their area.

The headquarters of large corporations directly manage all its divisions. These divisions have very close connections with their headquarters, and staff in the headquarters can access all their divisions' information, not just information provided by some simple data views. Furthermore,, the information of the

8

corporations are very sensitive because of the competition in the market. So advanced security networks and data warehouses are chosen to connect all its departments and divisions together, and there is a huge information flow between them.

In a small business chain, every business is independent in finance and share same brand with other members, the head offices of this business chain only charges an annual fee and a percentage of their annual total sales from its members.

Medium sized businesses operation management mix both styles that we introduced above, in which some of them use an independent style, some of them use a centralized management style, and others are between the two styles.

1.4.2 Information Management Data warehouse Systems Characteristic Differences

Because of operation management differences among the small sized businesses, middle sized businesses, and huge sized corporations the information management data warehouse systems are different. Normally, information management data warehouse for big corporation has the characteristics listed below:

- High data volume:

    There is a huge sized database management system for huge sized corporation because of the huge amount management operation information, huge business size, and big budget for marketing, research, and product development. Normally, the size of the database is about Trillion bytes. For

9

example, data base size of Amazon.com is 10 TB; Best Buy's data size is 2.7 TB; Colgate is 2 TB of data; and Telecom Italia Mobile has 10 TB of data in its system. But the rapidly accelerating growth of a data warehouse can easily become too huge to go out of the controllability of the existing information system. These systems are actively used every day to manage the core business of the companies that operate them, contributing impressive returns on investment, large cost savings, and promoting industry-leading business performance. Algorithms that work efficiently on 100 GB data warehouses do not necessarily succeed on 10 TB data warehouses, which are 100 times as large.

- Stability in heavy workload:

  It is one thing to have a data warehouse that contains terabytes of data and supports a few batch reports or a few online queries—especially if there are weeks or months between updates. It is quite another to have a large data warehouse that supports a demanding, dynamic workload consisting of many reports, many queries, many analyses and many updates—all going on at the same time. Whether such a system can exhibit *stability* under heavy load is a crucial issue in large-scale data warehousing. What is stability? It is consistent availability and consistently good performance in the presence of continuing, demanding, diverse and changing use.

- High data transfer speed:

  Close connections leads to a huge amount information exchange, the huge sized data volume make querying data very resource consuming jobs. Even

10

with million bytes per second bandwidth, optimization query approaches in a data warehouse is still very important. Otherwise, we will face the demands for bigger and bigger communication bandwidth.

- High security requirements:

  Because of competition between the corporations in the market, most of the corporation's data, such as their market, research, and product development strategies, are very sensitive. So their data warehouse solutions have high security requirements.

Because of above characteristics, the data Warehouse solution for huge sized corporation mostly is developed in DB2, Oracle, and SyBase with T1~ T4 optical communication channels.

Compare with the big corporation, small businesses have smaller sized data volume, lower communication bandwidth, relatively lower security requirements. So their warehouse solution mostly is developed in MS Access and low bandwidth communication technologies.

1.4.3 Summary of the Differences Among All Types of business

The summary of the operations management and their IMS (information management system) characters are listed in the below:

Type of Business	Operations Management	Connection between the headquarter and branches	Data volume between the headquarter and branches	The Security requirement of their IMS	The Stability requirement of their IMS	The data transfer speed requirement of their IMS
Large	Directly	Closely	Huge	High	High	High

11

Sized Corporat ions	managemen t					
Middle Sized company	Semi- directly managemen t	Between above and below	High	Middle	Middle	Middle
Small sized Retail Chain	Independent in finance	Loosely	Low	Low	Low	Low

12

The summary of the IMS solutions among the different types of business are listed in the below:

Type of Business	Platform (HW)	Platform (SW)	The Database component	Telecommunication channel
Large Sized Corporations	Huge sized computer	Unix for large user amount	DB2, SyBase, Oracle, Informix (high user amount)	T4 or satellite channel (bandwidth ~GBIT)
Middle Sized company	Small computer/or multi CPU NT Server	Unix (middle user amount) NT(high user amount)	Oracle middle user amount) MS SQL Server (with NT server)	Optical channel (bandwidth ~MBIT)
Small sized Retail Chain	PC	Windows NT/XP	MS SQL server (low user amount) MS Access 2000	DSL (bandwidth ~30K)

1.5 Small Business Chain Data Warehouse

Normally, more and more businesses are consisting of loosely connected small business chains, and every branch of the company is distributed in different location. E-management, which widely uses computer hardware and software, network, database technologies to improve their operation management, is the computerized operation management. E-management makes businesses more and more predictable, efficient, and controllable. In general, each branch of the small business chain has several hundreds transaction a day, and daily record size is not greater than 1million bytes. The database program for it should be low cost and easy to use and easy to maintain. Therefore, people with basic computer skills can pick it up quickly. The cost for the whole management, which includes development cost and maintenance cost, should be in the range that the business owner can accept.

13

MS Access, the Microsoft Office database management framework, provides cheap and powerful development tools to develop small business database management system. It has windows style interface, and anyone who has some experience in Windows Operating Systems can use it. The database management system developed with MS Access is easy for new users to learn, to use, and to maintain. Management system developed with MS Access can be installed in the MS windows server system, or in PC, which is the cheapest and widest used hardware platform and software operating system in the world. At same time; the MS Access has reasonable reliability and security. In a word, the Management system developed with MS Access is a very good solution for small businesses. Access provides a powerful set of tools for organizing, accessing, and sharing your information. The database systems that are mostly implemented in MS access system can reach a high performance/cost rate.

Most of the branches in the business chain are running independently, the headquarter staff only need to know a statistical result rather than all the detail information about it in most situations, and they will make management decision based on this information. Some statistical results show that in about 80% cases, only 20% information is accessed, in other word, repeat rate for the accessed records is very high for long term usage, so saving accessed remote records on local computers can decrease lots of network communication traffic. For most of business chain, it is possible for its member to choose different data source solutions because of the business size, security requirements, and their budget. The data warehouse system for them should have open connection interfaces,

14

which allows different number of local data sources be connected to, and allow different types of data sources be connected to the system.

A warehouse system that integrates all local MS Access databases, which is the best database management system for the local small businesses, is the best solution for a small business chain. The data warehouses provide data forms and statistical functions to staff at headquarters for management, and run at the lowest cost and highest performance/cost rate. It saves records, which it acquired from the remote data sources, on the local view data sources. When user queries information, the data warehouse first searches local data source; if it cannot find the information, it will search the remote data source. If it finds the information, it will show the data in the data form and save the results in view data source. After running for a considerable time, about 80% of records can be fond in the local data source; the network communication traffic will be decreased dramatically. In a word, we don't need big sized network communication bandwidth as a simple distributed database. Every elements of the business in the business chain can choose its database management system according to its operations requirement, and the data warehouse solution for the business chain can integrate all the data sources of each member.

The warehouse system, which we will develop, is a java application that integrate all local MS Access database together to aid the management in the headquarter. It uses Microsoft ODBC data source management technologies, which can connect the different numbers of data sources that use different types of data

15

management system. The system provides a good solution for small business

chains.

16

2 Lineage Tracing System

2.1 Introduction

In a data warehousing system, we can define views and query the view information from the source data. In many cases, the warehouse view contents alone are not sufficient for in-depth analysis. It is often useful to be able to "drill through" from interesting (or potentially erroneous) view data to the original source data that derived the view data. For a given view data item, identifying the exact set of base data items that produced the view data item is termed the *data lineage* problem [10].

In this chapter, we discuss the lineage-tracing problem and summary algorithms for lineage tracing introduced by Dr. Cui [6, 9, 10]. After surveys on the data warehouse with lineage trace, we will propose a solution for information data warehouse management system of the small business chain. Following this, we will develop a demo system to realize an inventory data warehouse management with lineage tracing functions. Enabling lineage tracing in a data warehousing environment has several benefits and applications, including in-depth data analysis and data mining, authorization management, view update, efficient warehouse recovery.

17
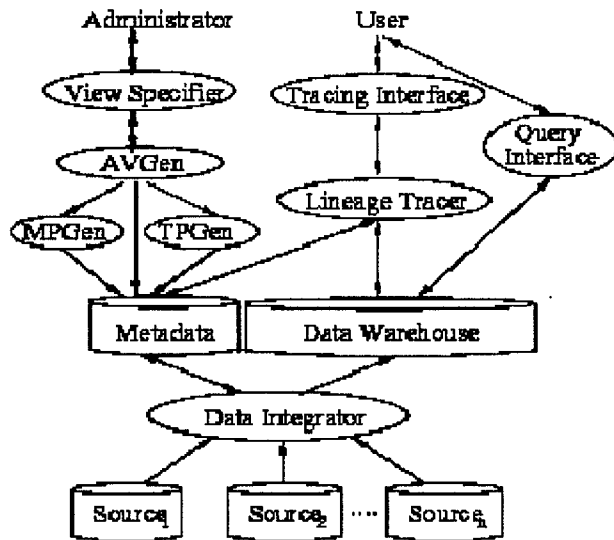
Figure 5 The lineage prototype

Figure 5[6] illustrates the architecture of the lineage tracing system in the context of a data warehousing system. When a view is defined through the View Specifier, and the view definition specifies that the view should be traceable, then the Auxiliary View Generator (AVGen) automatically generates the auxiliary views. The Maintenance Procedure Generator (MPGen) and the Tracing Procedure Generator (TPGen) then generate the maintenance procedures and lineage tracing procedures for the user view as well as its auxiliary views, and store them as part of the Metadata. When a user issues a request through the Tracing Interface, the Lineage Tracer is activated and calls the appropriate sequence of tracing procedures. The lineage results are then returned to the user as tables. If the user further requests to see the derivation process, the lineage tracer combines the lineage results and the view definition to generate a derivation tree for the user, showing the complete lineage information.

18

The warehousing environment introduces some additional challenges to the lineage tracing problem, such as how to trace lineage when the base data is distributed among multiple sources, and what to do if the sources are inaccessible or not consistent with the warehouse views. At the same time, the warehousing environment can help the lineage tracing process by providing facilities to merge data from multiple sources, and to store auxiliary information in the warehouse in a consistent fashion.

2.2 The Lineage Tracing Procedure

In general, to compute the lineage of a view data item, we need the view definition and the original source data, and perhaps some auxiliary information. A view definition provides a mapping from the base data to the view data. Given a state of the base data, we can compute the corresponding view according to the view definition. In many cases, not only are the views themselves are useful for analysis, but knowing the set of source data that produced specific pieces of view information also are very useful. However, determining the inverse mapping— from a view data item back to the source data that produced it—is not as straightforward as creating the view. To determine the inverse mapping accurately, we not only need the view definition, but we also need the base data and some additional information. In a data warehousing system, we can define, compute, and store auxiliary materialized views over source date in the warehouse to answer queries about the source data.

In the thesis, we focus on the lineage problem for relational Select-Project-Join views with aggregation (ASPJ views) in a data-warehousing environment. We

19

first transform the view definition into a normal form composed of aggregate-project-select-join sequences, called ASPJ segments. The lineage of tuples in a view defined by a single ASPJ segment can be computed using relational queries over the sources, called tracing queries, which are parameterized by the tuple(s) being traced. To trace the lineage of a view defined by multiple levels of ASPJ segments, we logically define an intermediate view for each segment, and recursively trace through the hierarchy of intermediate views top-down. At each level, we use tracing queries for a one-level ASPJ view to compute the lineage for the current traced tuples with respect to the views or base tables at the next level below.

2.3 Auxiliary Materialized Views

As mentioned in the above section, we may want to store auxiliary materialized views over source data in the warehouse to enable lineage tracing. There are two ways when we trace the lineage of tuples in a view. Either we can re-compute the relevant portion of the aggregation when tracing a tuple's lineage, or we can define an *auxiliary materialized view* over the node specifically for lineage tracing. In a data warehousing system, if we choose to store auxiliary views over source data, we may perform lineage tracing without querying the sources again. So using auxiliary views in the warehouse to answer queries about the source data is an integrated and efficient way. However, efficient incremental maintenance of multi-level aggregate views generally requires materializing the same intermediate views we need for lineage tracing.

20

Another type of auxiliary view is motivated by the fact that in a distributed multi-source data warehousing environment, querying the sources for lineage information can be difficult or impossible: sources may be inaccessible, expensive to access, and/or inconsistent with the views at the warehouse. By storing additional auxiliary views in the warehouse based on the source tables, we can reduce or entirely avoid source accesses for lineage tracing. This topic will be given more details in next chapter.

2.4 Lineage Trace Difference among Small, Medium Size Business, and Big Corporation

The data views of warehouses are developed for most frequently searching functional requirements, most of which are types of summaries. Lineage trace is designed for the views, in which if users of the data views want to know more details about the original sources that contribute the views, the trace functions can give them satisfactory answers. The data warehouses for big corporations provide more functions to review the data than the ones of the small and medium size business chains; consequently, the views and lineage trace for big corporation are more complicate and hard to implement, and consuming resources. How to reduce the resource consumption during the tracing procedure is a very challenging job. In Chapter 3, we will discuss the detailed algorithms that Dr. Cui introduced to store some auxiliary data for efficient maintenance and lineage tracing of complex views. Data warehousing and lineage trace implementation algorithms aim at reducing search expenses and communication traffic. The algorithms follow the line of thought that what we can do at the local

21

site view databases of the data warehouses should not do at remote site data sources; if we can calculate at remote site data sources, we should not query intermediate results from remote sites data sources and calculate at the local sites data sources. Even with these optimizing algorithms, data warehousing and lineage tracing for large corporations need more powerful data source management systems, more powerful hardware platforms that can host the systems such as IBM Main Frame, and bigger bandwidth communication channels like 16M BPS bandwidth.

Compared with big corporations, data warehousing and lineage tracing for small and medium sized businesses require simpler and smaller data sources managements and less powerful hardware platforms, and smaller bandwidth communication channels.

The differences between the lineage trace algorithms implemented in both kinds of businesses are the magnitude of calculation jobs: for the big corporation, a query may need to search 4 billion records in a second; for the medium size business may need to search 400 Million records in a second, but for small business may only need to search 4 Million records. All these non-function requirements determine how we choose software, hardware, and communication channels. Every 1% reduction of resource consuming during queries means huge money saving in the large corporations. From this point, the data warehousing and lineage tracing algorithms are more important for the large companies than for small to middle sized businesses. So more and more researches join to develop more efficient data warehousing and lineage tracing algorithms.

22

3 Strategies of Auxiliary Views for Lineage Tracing

3.1 Introduction

Recall that in a distributed multi-source warehousing environment, querying the sources for lineage information can be difficult or impossible: sources may be inaccessible, expensive to access, expensive to transfer data from, and/or inconsistent with the views at the warehouse. By storing auxiliary views in the warehouse we can reduce or entirely avoid source queries during lineage tracing. In addition, as we saw in the Chapter 2, auxiliary views corresponding to intermediate ASPJ segments in the view definition can be useful for efficient lineage tracing.

3.2 Auxiliary View

As will be seen in the following subsections, there are a wide variety of possible auxiliary views to maintain, with different performance tradeoffs in different settings. The remainder of this paper focuses on auxiliary view schemes and their relative performance for the restricted case of SPJ views. As future work we will first extend our results to one-level ASPJ views, which we expect to be relatively straightforward, and then to the full generality of multi-level views.

Given an SPJ view $V = v(D) = \pi_A (\sigma_c(T_1 \infty,...\infty T_m))$, and a tuple set $T \in V$ to be traced, Figure 3.2 [9]shows the generic form of its tracing query. We assume that all local selection conditions in the view (conditions that involve a single base table) are pushed down to the T_1's, so σ_c contains join conditions only.

23

Since the size of T tends to be small, in some cases we also push down the semi-join and rewrite the tracing query as in Figure 6.[9] The auxiliary views we consider are based on the forms of these two query trees. (Of course since the traced tuple set T is not available until tracing time, we cannot define or maintain auxiliary views on sub-queries involving T.)

We research seven schemes for storing auxiliary views to support tracing the lineage of T cording to v. For each scheme we specify the lineage tracing procedure, as well as the maintenance procedures for the auxiliary views and the original view, since they are all factors in overall performance.
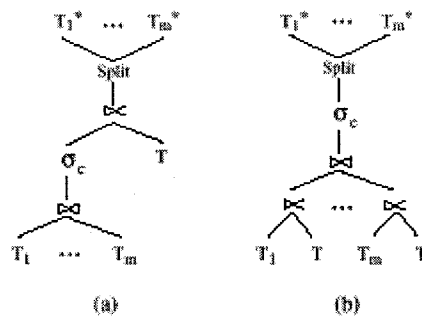
Figure 6 The Tracing queries

In the maintenance procedures, we use δ to denote the delta tables, but with insertions and deletions combined, and we use $\overset{+}{\bigcup}$ to denote application of the delta tables. We refer to the original view v as the user view when we need to distinguish it from the auxiliary views.

3.3 Strategy 1: Store Nothing (∅)

The extreme case is to store no auxiliary views for lineage tracing.

1. Auxiliary views: None

2. Lineage tracing: $TQ_{T,v} = Split_{T_1,...,T_m} (\sigma_c ((T_1 \propto T) \infty ... \infty(T_m \propto T))$

3. Maintenance of auxiliary views: None

4. Maintenance of v [GMS93]: $\delta V = \pi_A (\sigma_c((\delta T_1 \infty (T_2 \overset{+}{\bigcup} \delta T_2)\infty...\infty($

$$T_m \infty \delta T_m) \overset{+}{\bigcup} T_1 \infty \delta T_2 \infty (T_3 \overset{+}{\bigcup} \delta T_3)\infty...\infty(T_m \infty \delta T_m) \overset{+}{\bigcup} ...$$

$$\overset{+}{\bigcup} T_1 \infty...\infty T_{m-1} \infty \delta T_m))$$

We consider tracing a tuple set T rather than a single tuple t for generality: it sets the stage for generalizing our results to multi-level views, and in practice we expect that a warehouse tracing package might permit multiple tuples to be traced together for convenience and efficiency.

This scheme retrieves all necessary information from source tables every time a user poses a lineage tracing query. It incurs no extra storage or maintenance cost, but leads to poor tracing performance. This scheme is included primarily as a baseline to compare with other, more attractive, schemes.

3.4 Strategy 2: Base Tables (BT)

If we can trace the lineage of any tuple in a view without querying the sources, then we say that the view is self-traceable. Self-traceable views can be traced correctly even if source tables are inaccessible or inconsistent with the warehouse

25

views. One easy way to make a view self-traceable is to store in the warehouse a copy of each source table that the view is defined on (after local selections), and issue the tracing queries to the local copies instead of to the source tables during lineage tracing. We refer to these source copies as the base tables (BTs) for v.

1. Auxiliary views: $BT_i = T_i$, $i = 1...m$

2. Lineage tracing: $TQ_{T,v} = Split_{T_1,...,T_m}(\sigma_c((BT_1 \propto T) \infty ... \infty(BT_m \propto T))$

3. Maintenance of auxiliary views: $\delta BT_i = \delta T_i$, $i = 1...m$

4. Maintenance of v: Same as scheme $\varnothing$ replacing T_i with $BT_i = T_i$, $i = 1...m$

Storing base tables can improve user view maintenance as well as lineage tracing, and maintaining the base tables is fairly easy. However, base tables can be large, even after applying local selections, and much of the source data may be irrelevant to any view tuple's lineage if joins are selective.

3.5 Strategy 3: Store Lineage Views (LV)

An alternative way of improving tracing query performance is to store an auxiliary view based on the left subtree in Figure 8(a), which we call the lineage view (LV) for v, since it contains all lineage information for all tuples in the user view.

1. Auxiliary views: $LV = \sigma_c(T_1 \infty ... \infty T_m)$

2. Lineage tracing: $TQ_{T,v} = Split_{T_1,...,T_m}(LV \propto T)$

26

3. Maintenance of auxiliary views: $\delta \mathrm{LV} = \sigma_c (\delta T_1 \infty (T_2 \overset{+}{\bigcup} \delta T_2)\infty...\infty($

$$T_m \infty \delta T_m) \overset{+}{\bigcup} T_1 \infty \delta T_2 \infty (T_3 \overset{+}{\bigcup} \delta T_3)\infty...\infty(T_m \infty \delta T_m) \overset{+}{\bigcup} ...$$

$$\overset{+}{\bigcup} T_1 \infty...\infty T_{m-1} \infty \delta T_m)$$

4. Maintenance of v: $\delta \mathrm{V} = \pi_A (\delta \mathrm{LV})$

The LV scheme significantly simplifies the tracing query and thus reduces tracing query cost. However, lineage views can be large and are usually expensive to maintain. On the other hand, like base tables, lineage views can be helpful in maintaining the user view.

3.6 Strategy 4: Store Split Lineage Tables (SLT)

For views whose joins are many-to-many, lineage views as defined in Section A.1 can be very large, and thus not efficient when performing the semi-join with T during lineage tracing. One solution is to split the lineage view and store a set of tables instead, which we call the split lineage tables (SLTs). Note that we use lineage view LV as defined in Section A.1 in the following definitions.

1. Auxiliary views: $SLT_i = \pi_{T_i}$

2. Lineage tracing: $TQ_{T,v} = Split_{T_1,...,T_m} (\sigma_c ((SLT_1 \propto T) \infty ... \infty(SLT_m \propto T))$

3. Maintenance of auxiliary views: $\delta SLT_i = \pi_{T_i} (\delta \mathrm{LV})$, i = 1...m

4. Maintenance of v: $\delta \mathrm{V} = \pi_A (\delta \mathrm{LV})$

27

Split lineage tables contain no irrelevant source data, since every tuple in SLT_i, i = 1::m, contributes to some view tuples. Furthermore, the size of the split lineage tables can be much smaller than the lineage view. Their maintenance cost is similar to that of the lineage view. Note that although we do not materialize the lineage view LV in the SLT scheme, we still compute δ LV, in order to maintain the user view V and auxiliary views SLT_i, i = 1...m. The disadvantage of SLT is that lineage tracing queries may be more expensive.

3.7 Strategy 5: Store Partial Base Tables (PBT)

Reconsidering the BT scheme, another way to reduce the size of the base tables is to store the semi-join of each source table T_i with the user view V; we call this semi-join result the partial base table (PBT) for T_i according to v.

1. Auxiliary views: $PBT_i = T_i \propto V$, i = 1...m

2. Lineage tracing: $TQ_{T,v} = Split_{T_1,...,T_m} (\sigma_c ((PBT_1 \propto T) \infty ... \infty (PBT_m \propto T))$

3. Maintenance of auxiliary views: $\delta PBT_i = \delta T_i \propto (V \overset{+}{\bigcup} V) \overset{+}{\bigcup} T_i \propto \delta V)$, i = 1...m

4. Maintenance of v: Same as scheme $\varnothing$

For views with selective join conditions, the PBT scheme replicates much less source data than the BT scheme, with several benefits: It reduces the storage requirement, as well as the cost of refreshing the auxiliary views. It also reduces the tracing cost, because the tracing query operates on a much smaller table. However, partial base tables do not help with the maintenance of the user view.

28

Instead, the user view needs to be maintained first. The partial base tables are then relatively cheap to maintain based on the user view's contents and changes.

3.8 Strategy 6: Storing Base Table Projections (BP)

When source tables have known keys, we can store in our auxiliary views key attributes from the source tables together with other necessary attributes, which we call the base table projections (BPs). This scheme improves tracing query performance (over storing nothing) while reducing view maintenance and storage costs (over storing full source replicas).

1. Auxiliary views: where A_i includes the key attributes K_i, attributes that are projected into V ($T_i \cap$ V), and attributes involved in v's join conditions ($T_i \cap$ C)

2. Lineage tracing: $T^*_i = T_i \propto (\sigma_c ((BP_1 \propto \mathrm{T}) \infty \dots \infty (BP_m \propto \mathrm{T})), v^{-1} \mathrm{D(T)}$

   $= \langle\ T^*_1 \dots T^*_m\ \rangle$

3. Maintenance of auxiliary views: $\delta BP_i = \pi_{A_i} (\delta T_i)$, i = 1…m

4. Maintenance of v: Same as scheme $\varnothing$ replacing T_i with BP_i, i = 1…m

Note that the semi-joins in the tracing procedure are key-based. This scheme can be especially useful when a source table has wide tuples but the view projects only a small fraction. During lineage tracing, the stored information identifies by key which source tuples really contribute to a given view tuple, then the detailed source information is fetched from the source using the key information. Maintenance of the user view is easy. However, in the BP scheme we do need to query the sources which have their drawbacks as discussed earlier.

29

3.9 Strategy 7: Storing Lineage View Projections (LP)

Again assuming base tables with known keys, we can store a projection over the lineage view that includes only base table keys and user view attributes. We call this view the lineage view projection (LP). Note that we use lineage view LV as defined in the following definitions.

1. Auxiliary views: LP= $\pi_{A \cup K_1 \cup \ldots K_{m_i}}$ (δ LV), where A is the set of attributes in V, and K_i is the set of key attributes of table T_i, i = 1...m

2. Lineage tracing: $T^*_i = T_i \propto (LP \propto T)$, $v^{-1} D(T) = \langle T^*_1 \ldots T^*_m \rangle$

3. Maintenance of auxiliary views: δ LP= $\pi_{A \cup K_1 \cup \ldots K_{m_i}}$ (δ LV), i = 1...m

4. Maintenance of v: δ V= $\pi_A (\delta LP)$

Compared with the BP scheme, the LP scheme further simplifies the tracing query and improves tracing performance. However, the maintenance cost for the lineage view projection is higher than for the base table projections. LP also requires a source query as the last step of the tracing process, with the disadvantages previously discussed.

3.8 Self-Maintainability and Self-Tractability

Self-traceable views can be traced correctly even if the sources are inaccessible or inconsistent with the warehouse views. Analogously, view self-maintainability ensures that views can be maintained without querying the sources. In cases where the sources are inaccessible, we must ensure that the user views together with our auxiliary views are both self-traceable and self-maintainable. Table A.1 summarizes these properties with respect to the seven schemes introduced so far.

We also consider self-maintainable extensions of three of the schemes, LV, SLT, and PBT, calling the extensions LV-S, SLT-S, and PBT-S.

Table A.1: Scheme self-trace ability and self-maintainability

scheme	$\varnothing$	BT	LV	SLT	PBT	BP	LP	LV-S	SLT-S	PBT-S
Self-traceable?	no	yes	yes	yes	yes	no	no	yes	yes	yes
self-maintainable?	no	yes	no	no	no	yes	no	yes	yes	yes

31

4 Inventory Data Warehouse for Small Business Group

4.1 Introduction

In this chapter we will describe a warehouse solution for small business chain. A lot of company has many branches located in different parts of the world; each one of these has not more than 10 thousand transactions per a business day, for which MS Access is the best solution. As these branches operate more independently, their headquarter staff need management operation reports, such as inventory summary report, sales summary report rather than accessing detail operation information. Base on these requirements we will develop a data warehouse inventory operation management system with lineage trace functions for small business chain. In addition, we will demonstrate that the lineage trace function is very useful for a data warehouse system.

In this demo system that we will develop is aimed at demonstrating how to develop a local information management system with MS Access, how to integrate remote data source to a data warehouse, how to use java data base technologies to develop a data warehouse with user friendly interfaces, as well as how to implement a lineage trace.

We choose an inventory system as our local information case to develop a business model. The system has several relational, operational local databases as data sources, which are DB1, DB2,...,and DBn; and they are distributed in the network (intranet or internet); and one component integrates these systems to a data warehouse which is showed in the Figure 7. In the chapter, we will introduce

32

how we develop the local database. Source databases can be various types of databases, such as MS SQL Server, Oracle, Access, etc. In our system, we use Microsoft Access, which is the best solution for small business, to store our source data. The local data sources are independent data base systems, which have their own access interface for adding, deleting, updating, and brewing. Once the user of the data warehouse that integrates these sources needs information from one of the data sources, he can access the data sources by the warehouse data view functions.
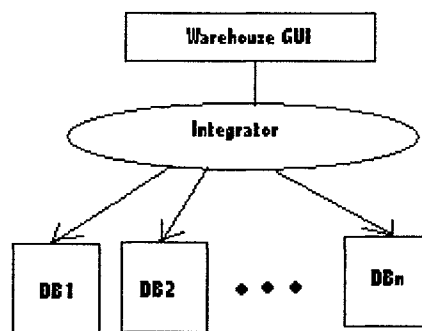
Figure 7 The data warehouse architecture

4.2 Functional & Non-Functional Requirement for Source

Database

The local resource databases are inventory management systems. The number of the daily transactions is between several hundred. The system can be used quickly by persons possessing some basic Windows program knowledge. The computer system should be PC Server and Windows NT. A simple backup strategy can solve the reliability problem. Security requirement can be met by a solution based on MS Windows system.

The user function requirement includes:

- Source data management:

    ➢ Add a new product/ update an existing product's information

    ➢ Add a new employee /update an exist employee's information

    ➢ Add a new customer /update an exist customer's information

    ➢ Add a new Supplier /update an exist Supplier's information

    ➢ Add a new purchasing order/ update an exist purchase order's information

    ➢ Add a new sales order/ update an exist sales order's information

    ➢ Add new shipper / update an exist shipper's information

    ➢ Add new shipping method / update an exist shipping method's information

    ➢ Check product inventory quantity in the stock.

    ➢ Statistic function for inventory management

4.3 Database Schema in Source Database

Products (**ProductID**, ProductName, CategoryID, QuantityPerUnit, UnitPrice)

Categorys (**CategoryID**, CategoryName, Description)

Orders (**OrderID**, SupplierID, OderDate)

OrderDetails (**ODID**, OderID, ProductID, OrderPrice, Quantity, IsDirty)

Sales(SaleID, CustomerID, SaleDate)

SalesDetails (**SDID**, SaleID, ProductID, SalePrice, Quantity, IsDirty)

Shipping(**ShippingID,** SaleID, ShiperID, ShippedDate, ShippingMethod)

Customers (**CustomerID**, CompanyName, ContecrName, Address,Phone)

34

Shippers (**ShoperID**, CompanyName, Phone)

Suppliers (**SupplierID**, CompanyName, ContecrName, Address,Phone)

Employee (**EmployeeID**, FirstName, LastName, Birthday, position, SIN)

4.4 ER Diagram of the Database

We use entity relationship models to design the database to store all information

used in his application. The Figure 8 shows the ER-diagram.

35
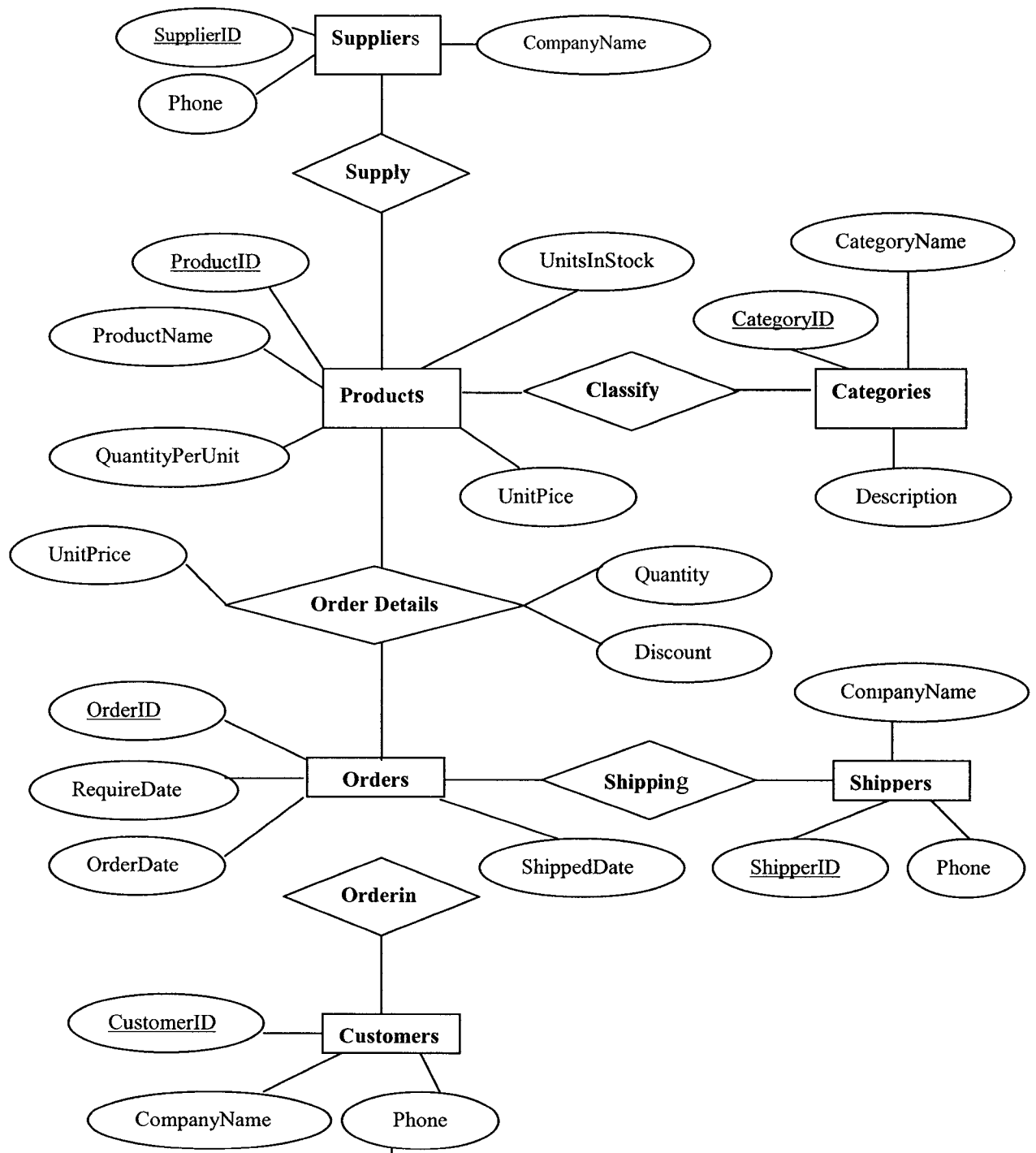
Figure 8 The Database ER-diagram

36

4.5 Functional & Non-functional Requirement for the Warehouse Component

This system is used by the manager of headquarter to check the order and inventory information about all the branch stores.

- Data warehouse set up and query:
    - ➤ Data warehouse set up
    - ➤ The user can view a category of product ordering value information.
    - ➤ The user can view a category of product sales value information
    - ➤ The user can view a category of product inventory stock value information.

- Data warehouse lineage tracing:
    - ➤ The user can trace back the order detail information for a specific category goods ordering.
    - ➤ The user can trace back the inventory stock items detail information for a specific category goods inventory.
    - ➤ The user can trace back the sales detail information for a specific category good sale.

- Data warehouse maintenance:
    - ➤ The system simulates the update data process. That is, when source data are updated, the data in data warehouse also are accordingly updated.

37

4.6 The Data Warehouse System Architecture

The system includes four components, which are user interface, data warehouse, Extraction & Transformation, and source databases. The first three parts together are combined to called data warehouse integrator. The system architecture is shown in Figure 9.
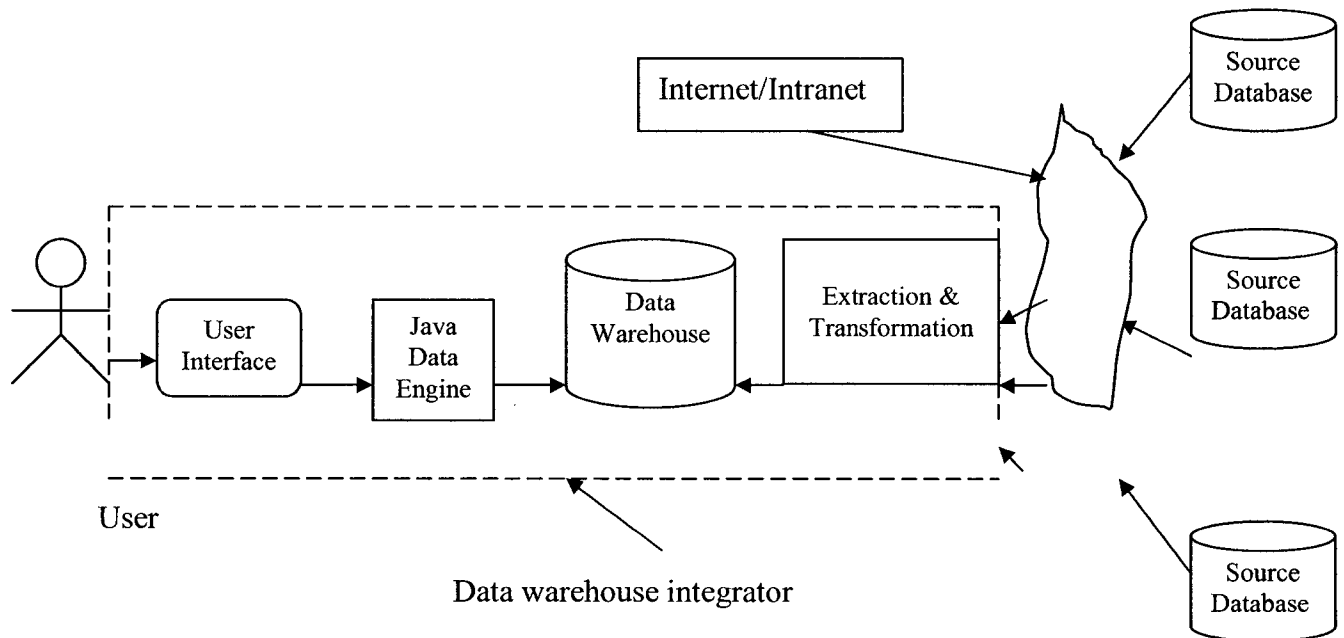
Figure 9 The System Architecture

4.7 Data Warehousing Schema

InventoryView (CategoryID,CategoryName, OrderPrice, OrderAmount,

SalesAmount, Inventoryamount, StoreID)

SalesView (CategoryID,CategoryName, SalesAmount, StoreID)

OrderView (CategoryID,CategoryName, OrderAmount, StoreID)

InventoryTraceView(ProductID, productName, totalOrdering, totalSales,

#InStock)

38

SalesTraceView(ProductID, productName, price, quantity, salesdate, sales value)

OrderTraceView(ProductID, productName, price, quantity, supplier, orderdate,)

4.8 Data warehouse Integrator Architecture

This section will give the inventory data warehouse integrator's architecture, which is presented by user case diagram, class diagram, and chart flow diagram.

4.8.1 User Case Diagram

The user can do four things: view the inventory, view the sales report, review supply report, and lineage trace any original resource for a record that he/she is interested in investigating. The below Figure 10 show these user cases diagrams. The user cases are view the inventory, view sales report, view supply report, and lineage trace using the "jdbc engine" user case to get the data, and use "display engine" to show the result. The "display engine" uses the user case "UidDefaultModel" to collect the table data.
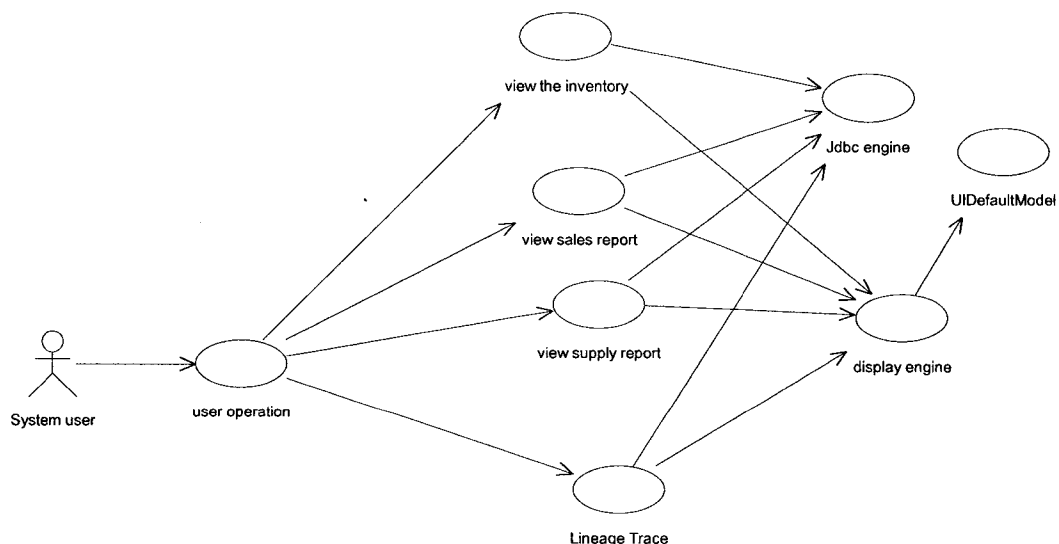
Figure 10 The user case diagram

39

4.8.2 The Classes Diagram

The class diagram is given below Figure 11. The first class is displayFrame, which is the first menu that user accesses with the data warehouse system. In the class displayframe, there is JMenu, JtextArea, Jlabel, Jpanel, and tableFrame objects. JMenu class has JmeniItem class objects. The viewBean Class has resultSet and ResultMrtaData objects. The tableFrame class is used for query result display, which include the viewBean class, MouseEvent class, Jpanel class Objects. The Jpanel class has a ScrollPanel object, which has a Jtable class object.
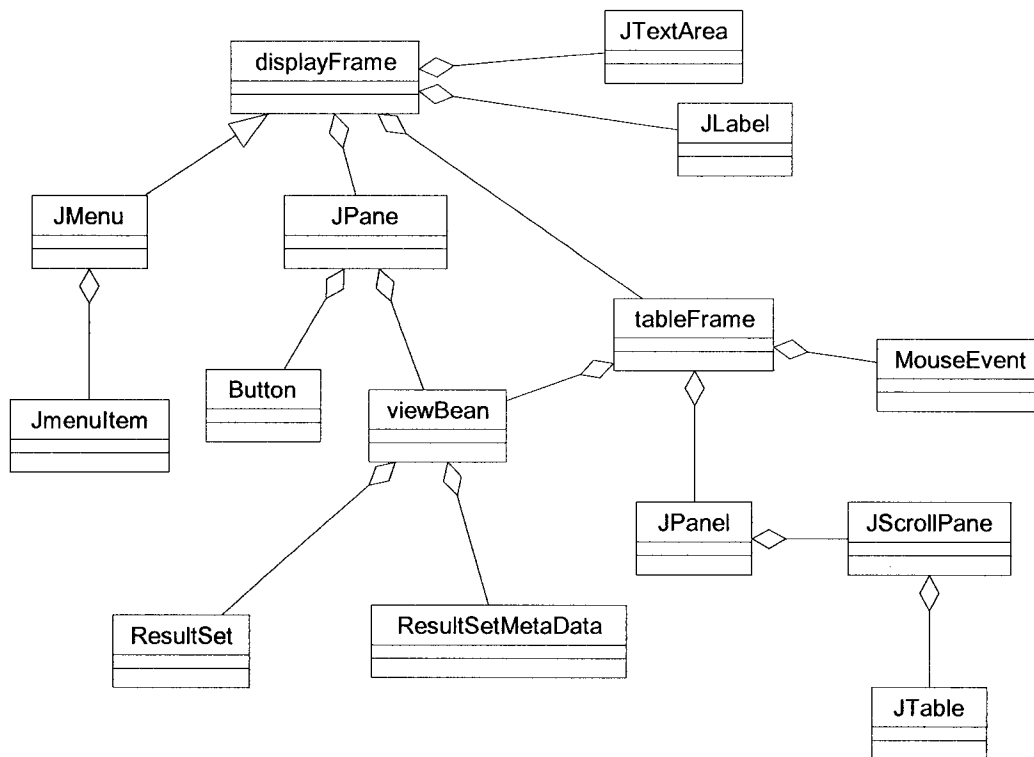
Figure 11 The Class diagram

40

4.8.3 System Flow Chart Diagram

This section gives the system dataflow chart diagram of the data warehouse integrator, which show in the Figure 12; and it includes view menu, view bean, table model, table display, and lineage trace modules. View menu receives data from user, which is embedded SQL request; and system transfers it to the viewBean, which is called JDBC engine, put the data into TableModel, which is called table data engine. After the dataset is transferred to Tabledisplay, called table display engine, system can call Lineage Trace to get original data source. The lineage trace data flow will go JDBC engine again to search, prepare, and display data.
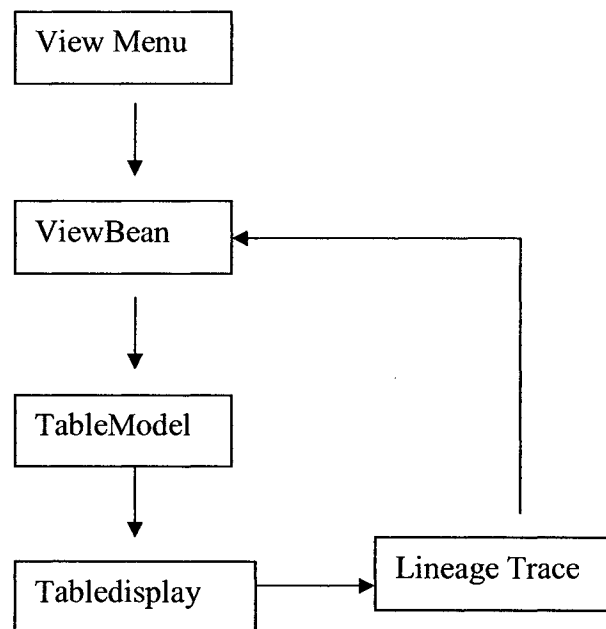
Figure 12 The flow Chart Diagram

41

5 Testing Plan for Inventory Data Warehouse System

This chapter will give the testing plans for both the local inventory management component and the data warehouse integrator component. After the functional testing plans, the non-functional testing plan will be outlined.

5.1 Testing Strategies and Procedure

We can approach any of the testing in three phases:

- Modeling the software's environment:

  A tester's task is to simulate interaction between software and its environment. Testers must identify and simulate the interfaces that a software system uses and enumerate the inputs that can cross each interface.

- Selecting test scenarios

  Many domain models and variable partitions represent an infinite number of test scenarios, each of which costs time and money. Only a subset can be applied in any realistic software development schedule. Testing criteria is the *coverage*: covering code statements (executing each source line at least once) and covering inputs (applying each externally generated event). These are the minimum criteria that testers use to judge the completeness of their work; therefore, the test set that many testers choose is the one that meets their coverage goals.

- Running and evaluating test scenarios

42

Having identified suitable tests scenarios, testers convert them to executable form, often as code, so that the resulting test scenarios simulate typical user action. Because manually applying test scenarios is labor-intensive and error-prone, testers try to automate the test scenarios as much as possible.

Scenario evaluation, the second part of this phase, is easily stated but difficult to do (much less automate). Evaluation involves the comparison of the software's actual output, resulting from test scenario execution, to its expected output as documented by a specification.

5.2 Testing Plans for Functional Requirements of Local Inventory Management Component

The section will describe the detailed testing plans regarding all the functional requirements of the local inventory management component. The test environment is: Windows NT server for 30 users, which system includes the hardware and software. The local inventory management component is installed in the server. Users access the system from 30 work stations, which are Windows XP/2000.

5.2.1 Testing Plan for the Add/view/Modify Employee Module

In the menu of the Add/view/Modify Employee module, the user can choose any of the records to test modifying an employee profile and creating a new employee profile into the system. Testing the View/Modifying employee's profile is described below:

43

- Modifying information: choose a profile of a specific employee, whose ID is "5", the Name is "Daley Brian", the title is "clerk", and the phone number is "(514)767-5456";

- Scenario: by changing his title to be "senior clerk", and pressing the "enter" key;

- Verification: close the menu and reopen it, find employee by ID=5, the user can find that the employee's profile was changed as he wanted.

Testing the adding an employee profile into the system is performed as below:

- Creating a employee's profile: the user wants to add a employee whose name is "Milliner Brian", the title is "clerk", and the phone number is "(514) 767-5450", into the system;

- Scenario: by clicking the button "*", filling all fields as above information ", and pressing the "enter" key;

- Verification: close the menu and reopen it, the user can find the employee profile that he/she wants to add in; or in the sale menu, click draw down pine box of Employee ID, he can find the employee profile he/she added in.

5.2.2 Testing Plan for Add/view/Modify Supplier Module

In the menu of the Add/view/Modify Supplier module, users can choose any one of supplier records to test modifying a supplier profile and creating a new supplier profile into the system. Testing the View/Modifying supplier profile function is performed as below:

- Modifying information: choose a specific supplier, whose supplier ID is "44", the company Name is "Globe Import & Export", the contact represented is

44

"Daccache Ruth", the phone number is 416-239-6767"; the address is "3044 Bloor Street West,", the city is "Etobicoke", the province is "ON", the country is "Canada", and the posted code is "M8X 1C4",

- Scenario: by changing the phone number to "416-595-1221", and pressing the "enter" key;

- Verification: close the menu and reopen it, find the supplier by ID=44, the user can see that the information was changed as he wants.

Testing the adding a supplier's profile is described below:

- Adding information: a user wants to add a supplier whose company Name is "Textile Globe Import & Export", the contact representative is "Jack Brown", the phone number is 819-539-6747"; the address is "3044 lasalle Street West,", the city is "Etobicoke", the province is "ON", the country is "Canada", and the posted code is "M8X 1C4",

- Scenario: by clicking the button "*", filling all fields as above information, and pressing the "enter" key;

- Verification: close the menu and reopen it, the user can find the supplier he likes to add; or in the order menu, click draw down pine box of supplier ID he can find the supplier he has added.

5.2.3 Testing Plane for Add/view/Modify Customer Module

In the menu of the Add/view/Modify Customer module, users can test the modifying a customer's profile and creating a new customer's profile into the system. Testing the View/Modifying the customer's profile function is described below:

45

- Modifying information: choose a specific customer, whose ID is "21", the company Name is "Action Retail Outfitters", the contact represented is "Gloria Francis", the phone number is "204-784-2600"; the address is "2599 Pembina Highway", the city is "Winnipeg", the province is "MB", the country is "Canada", and the posted code is "R3T 2H5",

- Scenario: by changing the phone number to be "204-595-1265", and pressing the "enter" key;

- Verification: close the menu and reopen it, find the customer by ID=21, the user can find the information was changed as he wanted.

Testing the adding a customer function is like below:

- Adding information: the user want to add a profile of a customer whose company Name is "Textile Globe Import & Export", the contact represented is "Jack Brown", the phone number is 819-539-6747"; the address is "3044 lasalle Street West,", the city is "Etobicoke", the province is "ON", the country is "Canada", and the posted code is "M8X 1C4", to the system.

- Scenario: by clicking the button "*", users can fill all fields as above information, and press the "enter" key.

- Verification: close the menu and reopen it, the user can find the customer profile he like to add in; or in the sales menu, click draw down pine box of customerID he can find the customer he added in.

5.2.4 Testing Plan for the Add/view/Modify Product Module

In the menu of the Add/view/Modify Product module, user can choose any of product item records to test modifying a product item's profile and creating a

46

new product item's profile into the system. Testing the View/Modifying s product profile is like below:

- Modifying information: choose a specific product item, in which the product's ID is "164", the Product Name is "Lauan 3.6mm 4' x 8'", the Category ID is "soft wood for internal", the product Description is "3.6mm 4' x 8' "; the quantity per units is "24", and the Unit price is "$174.91",

- Scenario: Users will change the Unit price into "123.99" and press the "enter" key;

- Verification: close the menu and reopen it, he/she will find the product item by product ID=164, the user can find the information is changed as he desired.

Testing the creating a product item profile functions is like below:

- Adding information: The product item profile is: the product name is "Brick 3.6mm 4' x 8'", the category ID is "pata", the product description is "3.6mm 4' x 8' "; the quantity per units is "24", and the unit price is "$74.91",

- Scenario: by clicking the button "*" in the navigation bar, filling all fields as above information, and press the "enter" key;

- Verification: close the menu and reopen it, the user can find the product item he/she liked to add in; or in the sales menu, click draw down pine box of Product ID he can find the product item he added in.

5.2.5 Testing Plan for the Add/view/Modify Purchase Order

Module

In the menu of the Add/view/Modify Purchase Order module, users can choose any of purchase order records to test modifying a purchase order or creating a new purchase order into the system. Testing the View/Modifying purchase order function is like below:

- Modifying information: open the purchase order form, choose a specific purchase order, in which the purchase order ID is "PO101"; the employee ID is "5"; the Supplier ID is "Alba Cash & Carry Import-Export Ltd"; the order Date is "6/21/2004"; in the order, there is one product "Camden 32 1/2 X 80 1/2"; its Quantity is "4750", its price is "$5.83".

- Scenario: add one more product item to the order: for instance; a product whose name is "B/N shelving 12" Maple", its quantity is "550", its price is "$725", and press the "enter" key;

- Verification: close the menu and reopen it, find the purchase order ID="PO101", the user can find the information is changed as he wanted.

The testing of the creating a purchase order functions is listed below:

- Adding information: the user want to add a product whose purchase order ID is "PO133"; the employee ID is "5"; the supplier is "Alba Cash & Carry Import-Export Ltd"; the order Date is "6/21/2004"; in the order there is one product item "Camden 32 1/2 X 80 1/2"; its quantity is "4750"; and its price is "$5.83",

48

- Scenario: by clicking the button "*", filling all fields as above information, and press the "enter" key.

- Verification: close the menu and reopen it, the user can find the purchase order he want to add in.

5.2.6 Testing Plan for the Add/view/Modify Sales Order Module

In the menu of the Add/view/Modify Sales module, users can choose any of the sales records to test modifying a sale order profile and creating a new sales order into the system. Testing the View/Modifying sales order function is described below:

- Modifying information: open the sales order form, choose a specific sales order, such as a product whose sales ID is "S1100"; the Customer ID is "Art's Nursery Wholesale & Retail"; the employee ID is "5"; the order Date is "10/2/2004"; in the order, there is one product item "Camden 36 1/2 x 80 1/2" sold; its quantity is "4500"; and its price is "$8.86";

- Scenario: add one more product item in to the order: for instance, a product "B/N shelving 12" Maple", its quantity is "350", and its price is "$10.99", and press the "enter" key;

- Verification: close the menu and reopen it, find the sales order ID=" S1100", the users can find the information was changed as they liked.

The testing of the creating a sales order functions is like below:

- Adding information: the users want to add a product sales order whose ID is "S1133", the employee ID is "5"; the Customer is "Atcom Wholesale Retail & Repair"; the sales date is "6/21/2004"; in the sales order there is one

49

product, "Camden 32 1/2 X 80 1/2"; its quantity is "450"; and its price is "$7.99".

- Scenario: by clicking the button "*", filling all fields as above information, and press the "enter" key;

- Verification: close the menu and reopen it, the users can find the sales order they like to add in.

5.3 Test Plan for Non Functional Requirement of Inventory Management Component

The section will outline the non-functional testing plans about all the non-functional requirements of the local inventory management component. The test environment is: Windows NT server for 30 users, which system includes the hardware and software. The inventory data warehouse management component is installed in the server. Users access the system from 30 workstations, which are Windows XP/2000.

5.3.1 Test Plan for the Robust Testing

When the inventory management system has run for a period of 6 months and the size of the database increases to about 50M bytes, users can do the robust testing, in which the user test if the system works properly when it is fully loaded. The users need to submit queries at a rate of 10 queries/min, and check if the results are correct. The queries should cover all the types in the previous sections. The second testing is the concurrent testing, in which database is sized to 50M, 10

50

users submit query at a rate of 1 query /min concurrently and then check if the results are correct. The queries result should be the same as the fully loaded testing. The third testing is the communication over load testing, in which, the communication channels are in over load and the system performance are slowed down, repeat the two previous testing and analyze the results to see how much slowdown the system can tolerant. For the three kinds of testing, we need to develop some special testing tools.

5.4 Functional Test for Inventory Management Data Warehouse Integrator Component

This component is developed in the object-oriented methodology; so the testing strategies are the while and black box testing. For every basic module, we use the while box testing method first, such as the code review, and then using the black box testing method to do the functional testing. The modules that will be tested include the view menu, the view bean, the data engine, the table display, and the lineage trace engine. This section will describe the testing plans for all these modules.

5.4.1 Testing Viewing Menu

We will describe testing plans for all modules in the consequence that the data flow chart (Fig. 4.7.3) outlined, before the testing we need to develop some tubs that replace next modules for testing the current one. In testing the view menu, we can see when the view button is clicked; the Embedded SQL Strings are correctly generated. After the grammars checking of the strings, we continue to

51

test the next module combined with the previous ones, if any error occurs; we need to check the module and retest till it gives satisfactory results.

5.4.2 Testing JDBC Engine

In this testing, we test if the system creates ODBC data source, the JDBC engine finds ODBC object, and it transfers the embedded SQL strings, submits queries, and returns the data correctly. First, we use very simple SQL sentences such as "select * from product;" to check if it work properly, and then using some complicated SQL sentences to check. The results should be the same as that we receive from the database query windows with the same SQL sentences. When any error is found, we need to modify the program and test again till getting the correct results.

5.4.3 Testing Table Data Engine and Table Display Engine

In this section we test if the system can transfer the data results from the JDBC engine to the table data engine and then display results correctly. In first step, we create a demo dataset to test the display engine, and then the table data engine; in second step, test the two modules together; and in third step, we link the three modules together to test if they work well, if not, we need to go back to review code and test again till they work well.

5.4.4 Testing the Lineage Trace and System Testing

In the section, we test if the lineage trace module works properly and integrates it into the system and do the system testing. Step one test is to consider if the data

52

source name and category ID are generated correctly; step two checks the meta data's parsing; step three tests the embedded SQL sentence generation, and final step is checking the lineage trace functions. Any error occurring in each step will lead us to check, to modify, and to retest the system until the bugs is fixed. The whole procedures of testing, recoding, and retesting will continue until the whole system meeting all the requirements.

5.5 Non Functional Testing for the Inventory Management Data Warehouse Integrator Component

In this section we describe the stress test to see if when the remote site data source's size, the local view database size, and the workload of the communication channels have reached their design limits, the view and the trace functions work properly. Normally, we need to test the system at maximum 80% of the limits. The testing methods are the same as the functional testing of the component and the difference is all testing conditions are fully loaded.

In the testing, we can set about 10 remote data resources (inventory management systems distributed in a wide area network (LAN)). Each of which has 10 users to submit queries at a rate of 1/min currently, the bandwidth of communication channel drop down to 10% (for example, normal bandwidth is 30KBPS, the current value is set to 3K BPS, which can be adjusted by the administrator of the network or using a dial up connection). All data views have to search from the remote data sources at rate of 2 searches/min. Because of the remote users updating frequently, view maintenances are also busy at updating the data. We can record all of the data view, the lineage tracing results, all the remote data

53

records at different time. From the analyst of these records, we can check if the

system works properly and the data view of the data warehouse display correct

results, and the reaction time of the system meet the system's requirements.

54

6 The Implementation of the Local Data Resource

Management

6.1 Introduction

The local resource in our system is the inventory management system of a branch of the company. Because of the performance/cost considerations, the analyses results from section 1.5, and the budget in the business, we choose PC networks, PC servers, MS Windows NT server computer operation systems, Windows 2K/XP client systems, and the MS Access database as our inventory management solution. In the inventory Access databases, we use tables to store our information, forms to view/enter our data, queries to search the data that we are interested in, and reports to display a query result in a specific format.

The interface of the system includes forms that we add /view /update a product item's information, an employee's profile, a customer's information, a Supplier's profile, a purchase order, a sales order, a shipper's information, and a shipping method's information. Besides these, the interface of the system provides reports to output statistical results for inventory management. The system was developed with the MS Access. In the chapter, we will show design ideas for how to use some Access functions to develop the system.

55

6. 2 Design of the Main Menu Form

The main menu has entries to all of the forms and reports. It is the first menu of the system. It contains a lot of button, each of which will lead us to a functional form or a report.

The code that creates the menu is written in VBA and listed below:

```
On Error GoTo Form_Open_Err

    ' Minimize the database window.

    DoCmd.SelectObject acForm, "Switchboard", True

    DoCmd.Minimize

    DoCmd.Hourglass False

    Set dbs = CurrentDb()

    Set rst = dbs.OpenRecordset("My Company Information")

    If rst.RecordCount = 0 Then

      rst.AddNew

      rst![Address] = Null

      rst.Update

      MsgBox "Before using this application, you need to enter your company
name, address and related information."

        DoCmd.OpenForm "My Company Information", , , , , acDialog

      End If

      rst.Close

      dbs.Close

      ' Move to the switchboard page that is marked as the default.
```

56

```
    Me.Filter = "[ItemNumber] = 0 AND [Argument] = 'Default' "

    Me.FilterOn = True

Form_Open_Exit:

    Exit Sub

Form_Open_Err:

    MsgBox Err.Description

    Resume Form_Open_Exit

End Sub
```

6.3 The Design of Adding /Viewing a Product Form

A form is a type of database object that is primarily used to enter or display data
from a database. To easily view, enter, and change data directly in a table, we use
a form to do the job. When we open a form, Access retrieves the data from one or
more tables and displays it on the screen with the layout you choose in the Form
Wizard or with the layout that you created on your own in Design view.

In lots of situations, we need to enter data in both two tables with primary-
foreign key relationships simultaneously in one form; and one table is master, the
other is child that contains variable number records. The SQL query language for
child table is "select * from child when master. Primary = child. Foreign key."
When the master table switches to another record, the child table also displays
the content that relates to this record.

We use forms to display the master table data, and use an embedded data sheet in
the form to display the child table's content. This kind of interface has solved the
problem that a form may contain a variable number of items. When the master

57

table, which shows sales table contents, displays one record, the child table shows only records from Salesdetail table that have same ID with the one in the master table.

In the GUI interface of the add /view / update product form, we have a navigation bar at the bottom of the form to let us review product items, find a specific product item, add a new product item into the system, or check the product's total ordering, total sales, and total inventory at cost. The interface involves three tables that have a relationship: product, orderingDetail, and SalesDetail. We show a product item from table product in the master form, and total inventory at cost from table orderingDetail and SalesDetail in the embedded datasheet, in which records have the same product ID with the one in the master form. The master form has labels, text fields, buttons; and an embedded data sheet in the main form has the fields of the product name, the total ordering, the total sales, and the inventory information.

The SQL query for the view product item in main form is:

SELECT DISTINCTROW from Product;

The SQL query for the view of inventory in the embedded datasheet is:

SELECT DISTINCTROW [OrderDetail].[ProducteID],

[SalesDetails].[ProductID], [Products].[ProductID], [Products].[ProductName],

Max([OrderDetail].[Price]) AS OrderedPrice, Sum([OrderDetail].[Quantity]) AS

Ordered, Min([SalesDetails].[Price]) AS SalesPrice,

Sum([SalesDetails].[Quatity]) AS Sales

58

FROM (Products INNER JOIN SalesDetails ON

[Products].[ProductID]=[SalesDetails].[ProductID]) INNER JOIN OrderDetail

ON [Products].[ProductID]=[OrderDetail].[ProducteID]GROUP BY

[OrderDetail].[ProducteID], [SalesDetails].[ProductID], [Products].[ProductID],

[Products].[ProductName];

6.4 The Interface of Entry /Viewing /Updating an Purchase

Order

This form lets users to enter, review, or modify a purchase order. Normally, a

purchase order may contain a lots of product items. We use master form with an

embedded datasheet to show purchase order with product items, which is the

same approach as the product form. The SQL query language for the child form

is "select * from child when master. Primary = child. Foreign key. ". In the data

schema, they are shown in the two tables: the Order and the orderDetail.

In the view mode, the SQL query for the main form is: select * from sales; for the

sub form is: select * from [ordeTail] when [orderTail].POID = [order].

PurchaseOrderID

In the update mode, the SQL query for the main form is: update * from sales; for

the sub form is update * from ordeTail.

In the add mode, the SQL query for the main form is: insert (POID, EmployID,

SupplyID, orderDate) into sales: for the sub form is: insert (POID, ProductID,

quantity, Price) into ordeTail.

59

6.5 The Interface of Entry /Viewing /Updating an Sales Order

This form lets users to record a sales transition. Normally, a sales contract contains a lots of product items. The master form with an embedded-datasheet given in the previous sections gives a very good solution for the type of problems. The SQL query language for the child datasheet is "select * from child when master. Primary = child. Foreign key". The data in the child datasheet comes from the two tables: the Sales and the salesDetail.

In the view mode, the SQL query for the main form is: select * from orders; for the sub form is: select * from [salesTail] when [salesTail].POID = [sales]. PurchaseOrderID

In the update mode, the SQL query for the main form is: update * from orders; for sub form is select * from salesTail.

In the add mode, the SQL query for main form is: insert (POID, EmployID, SupplyID, orderDate) into sales; for the sub form is: insert (POID, ProductID, quantity, Price) into salesTail.

6. 6 Report Review Form

In the main menu, clicking button "Preview Reports" will lead users to the menu of the report review menu (Figure 23 Report Review Menu). In the menu, there are four choices: "Preview the Inventory summary", "Preview the Products supplying Report", "Preview the Products Sales Report", and "return main menu". Pseudo code to generate the menu is the same as the one used in the main menu.

60

6.6.1 The Summary Report of Product Inventory

From the report review menu (Figure 23), when users click the first menu button, the menu of "Preview the Inventory summary", the system generates the list of the total inventory at cost. Inventory managers and staff members can get all their information from it, the managers can decide which product items need to be re-order; which items have too many in the stock and need some sales promotions to speed up their sales, and slow down the purchasing of the item as well. The SQL sentences to generate the report is listed below:

"SELECT DISTINCTROW [OrderDetail].[ProducteID],

[SalesDetails].[ProductID], [Products].[ProductID], [Products].[ProductName],

Max([OrderDetail].[Price]) AS OrderedPrice, Sum([OrderDetail].[Quantity]) AS

Ordered, Min([SalesDetails].[Price]) AS SalesPrice,

Sum([SalesDetails].[Quatity]) AS Sales,

(Sum([OrderDetail].[Quantity])- Sum([SalesDetails].[Quatity])) AS [#Instock],

((Sum([OrderDetail].[Quantity])- Sum([SalesDetails].[Quatity]))*

Max([OrderDetail].[Price])) AS [Inventory Value]

FROM (Products INNER JOIN SalesDetails ON

[Products].[ProductID]=[SalesDetails].[ProductID]) INNER JOIN OrderDetail

ON [Products].[ProductID]=[OrderDetail].[ProducteID]

GROUP BY [OrderDetail].[ProducteID], [SalesDetails].[ProductID],

[Products].[ProductID], [Products].[ProductName];"

61

6.6.2 The Summary Report of Product Supplying

From the Report review menu (Figure 23), when users click the first menu button, the menu of "Preview the Products supplying Report", the system generates reports that give all kinds of supplying information, which are classified by the supplier ID. For every supplier, the report displays the product price, the product quantity, and the order data. Utilizing this report, the inventory managers can determine as to who are the important suppliers. The SQL code to generate the report is:

"SELECT [productsupply Query].SupplierName, [productsupply Query].PhoneNumber, [productsupply Query].Address, [productsupply Query].City, [productsupply Query].PostalCode, [productsupply Query].StateOrProvince, [productsupply Query].Country, [productsupply Query].ProductName, [productsupply Query].Price, [productsupply Query].Quantity, [productsupply Query].OrderDate, Suppliers.SupplierID, ([productsupply Query].Quantity*[productsupply Query].Price) AS [sub total], sum(([productsupply Query].Quantity*[productsupply Query].Price) As [total Supplying]

FROM [productsupply Query] INNER JOIN Suppliers ON [productsupply Query].Suppliers_SupplierID = Suppliers.SupplierID;"

And The SQL code that generate the productquery is:

"SELECT [productsupply].[SupplierID] AS SupplierID, [productsupply].[OrderDate], [productsupply].[ProducteID], [productsupply].[Price], [productsupply].[Quantity], [Suppliers].[SupplierID] AS

62

Suppliers_SupplierID, [Suppliers].[SupplierName], [Suppliers].[ContactName],

[Suppliers].[PhoneNumber], [Suppliers].[Address], [Suppliers].[City],

[Suppliers].[PostalCode], [Suppliers].[StateOrProvince], [Suppliers].[Country],

[Products].[ProductID], [Products].[ProductName]

FROM Suppliers INNER JOIN (Products INNER JOIN productsupply ON

[Products].[ProductID]=[productsupply].[ProducteID]) ON

[Suppliers].[SupplierID]=[productsupply].[SupplierID];"

6.6.3 The Summary Report of Product Sales

From the Report review menu(Figure 23), when users click the third menu
button, the menu of "Preview the Products supplying Report", the system
generates reports that give all kinds of sales information, which are classified by
the supplier ID. For every customer, it gives the product items sales price, the
quantity, and the sales data. From the report, managers can identify who are the
important customers and what are the best-sold items. The SQL code to generate
the report is:"SELECT [sales Query].CompanyName, [sales
Query].PhoneNumber, [sales Query].Address, [sales Query].City, [sales
Query].Province, [sales Query].Country, [sales Query].PostCode, [sales
Query].SalesDate, [sales Query].Price, [sales Query].Quatity, [sales
Query].ProductName, Customers.CustomerID, sales.SalesrID,

([sales Query].Quatity* [sales Query].Price) AS [SubTotal],

Sum([sales Query].Quatity* [sales Query].Price) AS [Total Sales]

63

FROM ([sales Query] INNER JOIN Customers ON [sales

Query].Customers_CustomerID = Customers.CustomerID) INNER JOIN sales

ON Customers.CustomerID = sales.CustomerID;"

64

The Implementation of the Inventory Management Data

Warehouse System

7.1 Introduction

This inventory data warehouse management system provides hierarchical management functions for business retail chains, which consist of small retail companies. The implementation of the component includes the mapping remote inventory management component to local data warehouses and the implementing of the java data engine, the data display engine, the lineage trace engine, and the interfaces of all data review and lineage trace modules.

7.2 Mapping the Remote Data Sources into Local ODBC Objects

Open Database Connectivity (ODBC) is a Microsoft's software package for database accessing. It provides a framework for desktop-based tools (word processors, spreadsheets, report writers, Internet/Intranet tools, for example) to transparently access data sources.

In our system, the remote data sources are located in different areas. We use network technologies to map a remote network node to a virtual driver on the local server in the network of the head office, and remote data sources on the node are virtual data files in the driver. We create an ODBC object for each virtual data file, and then, we create a user DSN for each object.

65

A system DSN will be available to the whole system so that any users, who have the system accounts, will be able to access the data source.

7.3 Implementation of Java Data Engine

In the system, we develop a Java data engine that gets in a SQL string, executes a query, returns the query result, and restores all results back to their data types. The algorithm of the Java data engine is:

Begin:

1. Set parameters "class name", which is the driver's name, to the engine;

2. Set "URL", which is the name of ODBC data source, to the engine;

3. Set "User name", which is the name of data source's user, to the engine;

4. Set "Password", which is the password of data source's user, to the engine;

5. Create a connection to the data driver manager, at same time set parameters such as URL, user name, and password, to the connection;

6. Create a statement for the connection;

7. Let the statement to execute the query;

8. If it is a reviewing query, return the query result to a result set and a meta data set;

9. If it is a updating query, just execute the query, and return a Boolean result, if not successful, throw an exception;

10. If it is a creating query, just execute the query, and return a Boolean result, if not successfully, throw a exception;

66

11. Put the result to the Java data engine object. The other object can access

the result by calling the object's functions.

End

7.4 Table Data Display Engine Implementation

The class tableFrame is the data display engine in our programs. All the forms

that need to display table data are reusing the module. The results displaying of

the data searching and the lineage trace are also reusing the engine. The flow

chart of the main functions of the module is showed below Figure 13:
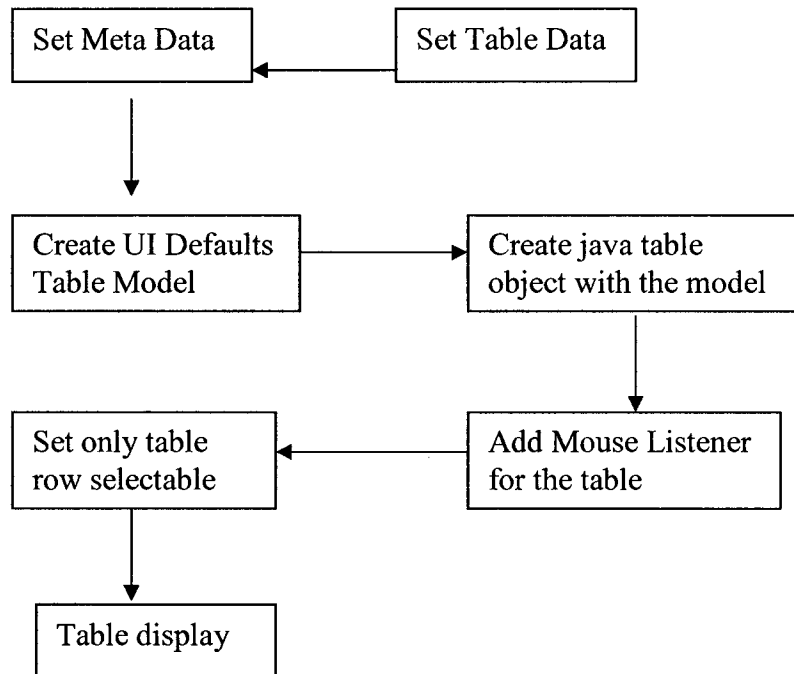
Figure 13 The flow chart of display engine

7.5 The Lineage Trace Engine Implementation

The lineage trace module gets three parameters: category ID, store ID, and the

query string, which are used to generate the lineage trace query string. From

67

Store ID, the lineage trace engine gets the remote data source's ODBC data source name; from the query string, the lineage trace engine generates the table list of the remote data source. Combining them together, the SQL query string is generated for the lineage trace. And then, the java data engine is called to search the lineage data. The local data sources are searched firstly, if the required data is not found; then remote data sources are searched and the system returns results, at mean time, the results will be saved in local data source. The chart flow, which outlines the whole procedure of the program, is listed below Figure 14:

```
┌──────────────┐   ┌──────────┐   ┌──────────────┐
│ Category ID  │   │ Store ID │   │ Formal Query │
└──────────────┘   └──────────┘   └──────────────┘
                         │               │
                         ▼               ▼
                  ┌──────────────┐ ┌──────────────┐
                  │ Remote ODBC  │ │ Meta data for│
                  │ data source  │ │ data source  │
                  │ name         │ │              │
                  └──────────────┘ └──────────────┘
                         
         ┌─────────────────────────────────┐
         │ Lineage trace query generate     │
         └─────────────────────────────────┘
                         │
                         ▼
         ┌─────────────────────────────────┐
         │ Query execution (first search    │
         │ local view, if not found, query  │
         │ remote data source               │
         └─────────────────────────────────┘
               │                   │
               ▼                   ▼
      ┌──────────────────┐ ┌──────────────────┐
      │ Return the lineage│ │ Save the         │
      │ trace result      │ │ auxiliary data   │
      └──────────────────┘ └──────────────────┘
```

Figure 14 The Lineage Trace flow chart

68

7.6 Data Warehouse Implementation

This part outlines the implementation of the data warehouse integrator, the interfaces of the system, and internal modules.

7.6.1 The Main Menu Implementation

This menu provides entries of all the functions of the inventory data warehouse system, which integrates all its inventory data sources together and provides summary views on the total inve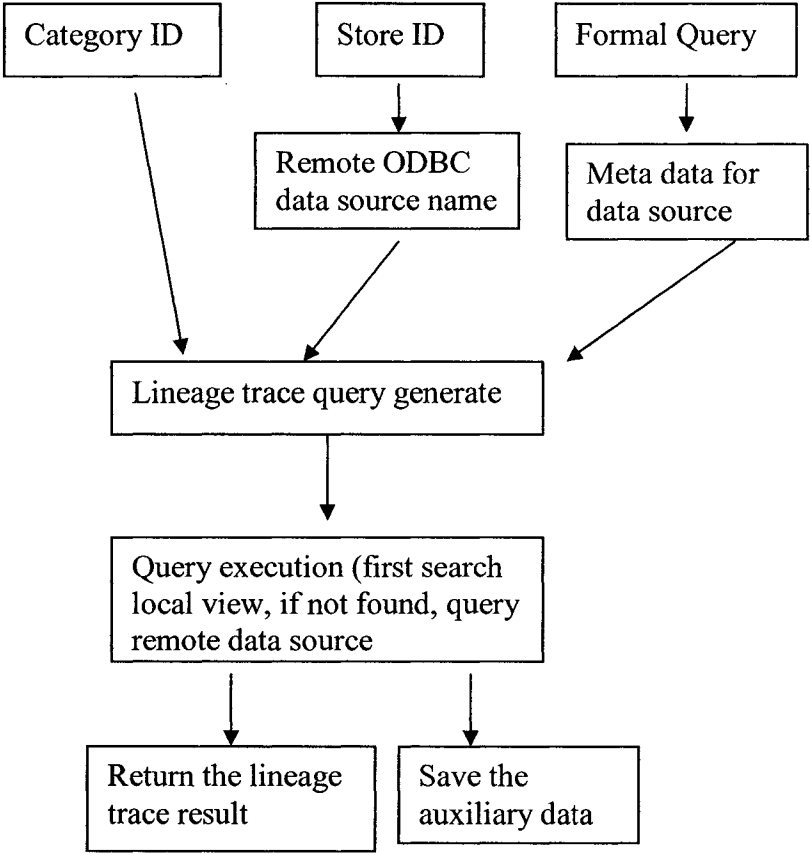ntory at cost, total sales, and total supplying. The interface is a form that includes some panels. First panel has a richtext field that gives some description for the system. The second panel has three buttons that will active three functional views: a sales information view, an inventory information view, and a supplying information view. The menu is like picture Figure 28,

7.6.2 The Sales Information View Menu Implementation

From the main menu, when the user clicks the button, "Sale Info. View", the system will give the category of all the goods sales information of all the branches. It has an attribute, which is named "storeID", to tell which store records come from. The SQL query is:

"SELECT DISTINCTROW [Categories].[CategoryID], [Categories].[CategoryName], Sum([OrderDetail].[Price] * [OrderDetail].[Quantity]) AS [Sum Of Sales] , [StoreID]=str_storeID FROM (Categories INNER JOIN Products ON [Categories].[CategoryID] = [Products].[CategoryID]) INNER JOIN OrderDetail ON [Products].[ProductID]

69

=   [OrderDetail].[ProducteID]   GROUP   BY   [Categories].[CategoryID],

[Categories].[CategoryName];";

The query information is saved into the local view database as well as displayed in the form. The table in the form has some characters: the width of the column is changeable; the record row can be highlighted and trigger an event, which function is used as the trigger of the lineage trace function. The view result is displayed as Figure 29.

7.6.3 The Implementation of the Inventory Information View Menu

From the main menu, when the user clicks the button, "Inventory Info. View", the system will go to view the category of all the goods inventory information of all the sub-branches. It has an attribute, which named "storeID", to tell which store a record comes from. The SQL query is:

"SELECT DISTINCTROW [Categories].[CategoryID],

[Categories].[CategoryName], Max([OrderDetail].[Price]) AS [Price],

Sum([OrderDetail].[Quantity] * [OrderDetail].[Price]) AS [Order amount],

Sum([SalesDetails].[Quatity] * [StoreID] = str_storied, [SalesDetails].[Price])

AS [Sales Amount], (([Order amount]-[Sales Amount])* [Price])

AS [Inventory Value] FROM ((Categories INNER JOIN Products ON

[Categories].[CategoryID] = [Products].[CategoryID]) INNER JOIN

OrderDetail ON [Products].[ProductID] = [OrderDetail].[ProducteID]) INNER

JOIN SalesDetails ON [Products].[ProductID] = [SalesDetails].[ProductID]

GROUP BY [Categories].[CategoryID], [Categories].[CategoryName];

70

The query information is saved into the local database view as well as displayed in the form. The table in the form has the same characters as the previous form: the width of the column is changeable; the record row can be highlighted and trigger an event for lineage trace. The result is displayed as Figure 30.

7.6.4 The Supply Information View Menu Implementation

From the main menu, when the user clicks the button, "Inventory Info. View", the system will go to view the category of all the goods inventory information from all the sub-branches. It has an extra attribute, which named "storeID", to tell which store the record comes from. The SQL query is:

"SELECT DISTINCTROW [Categories].[CategoryID],

[Categories].[CategoryName],

Sum([OrderDetail].[Price]*[OrderDetail].[Quantity]) AS [Order Amount],

[StoreID]= str_storeID FROM (Categories INNER JOIN Products ON

[Categories].[CategoryID] = [Products].[CategoryID]) INNER JOIN

OrderDetail ON [Products].[ProductID] = [OrderDetail].[ProducteID] GROUP

BY [Categories].[CategoryID], [Categories].[CategoryName];";

The query information is written into the local database view as well as displayed in the form. The table in the form has the same characters as previous forms: the width of the column is changeable; the record row can be highlight and triggered an event. The result is displayed as Figure 31.

71

7.6.5 The Sales Information Lineage Trace Implementation

In the table display form showed in Fig. A.4.5, if that user highlights a row and double clicks the highlight area, system will find the categoryID and story name, and then call the lineage trace functions to find the original data set that contributed the record. Normally, if the data source doesn't exist in the view database, it will be searched in the remote database. From the formal query, we get basic table list of original data source; together with categoryID and StoreID, we can generate lineage trace query listed below:

"SELECT [Products].[ProductName], SalesDetails.Price, SalesDetails.Quatity, sales.SalesDate, ([SalesDetails.Price]*[SalesDetails.Quatity]) AS [sales Amount] FROM (Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID) INNER JOIN (sales INNER JOIN SalesDetails ON sales.SalesrID = SalesDetails.SalesrID) ON Products.ProductID "+

   "= SalesDetails.ProductID WHERE (((Categories.CategoryID)="+

   CategoryID+));";

The lineage trace engine submits the query to the java data engine, first searches the warehouse local view data source, if it not exist, then searches the remote data source; after finding it, the system calls the data display engine to display it; at same time, saves it in the warehouse local view data sources. The result is displayed in the Figure 32.

72

7.6.6 The Inventory Information Lineage Trace Form

Implementation

In the table display form showed in Figure 30, if the user highlights a row and double clicks the highlighted area, the system extracts categoryID, story name, and query contents, and then calls the lineage trace engine to find the original data set that contributed the record. Normally, if the data source doesn't exist in the view database, it will query the remote database. From the formal query content, we get the table list of original data source; together with categoryID and StoreID, and then generate lineage trace query listed below:

"SELECT DISTINCTROW Products.ProductName, "+

    "Max(OrderDetail.Price) AS [Price], "+

    "Sum(SalesDetails.Quatity) AS [sales], Sum(OrderDetail.Quantity) "+

    "AS [order], ([order]-[sales]) AS [#Instock] "+

    "FROM ((Categories INNER JOIN Products ON Categories.CategoryID "+

    "= Products.CategoryID) INNER JOIN OrderDetail ON Products.ProductID "+

"+

    "= OrderDetail.ProducteID) INNER JOIN SalesDetails ON

Products.ProductID "+

    "= SalesDetails.ProductID GROUP BY Categories.CategoryID, "+

    "Categories.CategoryName, Products.ProductID, Products.ProductName, "+

    "Products.CategoryID, SalesDetails.ProductID, SalesDetails.Quatity, "+

    "OrderDetail.ProducteID, OrderDetail.Price "+

        "HAVING ((((Categories.CategoryID)=")="+ CategoryID+));";

73

The lineage trance engine submits the query to the java data engine, the engine searches the warehouse view data source firstly, if it does not find, then searches the remote data source, after finding it, calls the data display engine to display it, at same time; saves it in the warehouse view data sources. The result is displayed in the Figure 32.

7.6.7 The Supply Information Lineage Trace Form Implementation

At the table display form Figure 31, when the user highlights a row and double clicks it, the system extracts categoryID, storedID, and query content, and call the lineage trace engine to find the original data set that contribute the record. If the data source doesn't exist in the local view database, the system will query from the remote database. From the query content, we get basic table list of original data source; together with categoryID and StoreID, we can generate lineage trace query as below:

"SELECT Products.ProductName, OrderDetail.Price, "+

"OrderDetail.Quantity, Suppliers.SupplierName, Orders.OrderDate "+

"FROM Suppliers INNER JOIN ((Categories INNER JOIN Products ON "+

"Categories.CategoryID = Products.CategoryID) INNER JOIN "+

"(Orders INNER JOIN OrderDetail ON Orders.PurchaseOrderID = "+

"OrderDetail.POID) ON Products.ProductID = OrderDetail.ProducteID) "+

"ON Suppliers.SupplierID = Orders.SupplierID "+

"WHERE (((Categories.CategoryID)=" + CategoryID+ "));";

74

The lineage trance engine submit the query to the java data engine, which search the warehouse view data source first, and then search the remote data source and call the data display engine to display the record, at same time; save it in the warehouse view data sources. If it does not find the lineage, throw an exception. The result is displayed in the Figure 34.

7.7 Data warehouse Maintenance

In the local view database, there are 6 local view tables for saving the query data. For every one of them, the maintenance engine regularly checks to see if every table's dirty field is true. If yes, the engine updates the records in local view database. The SQL query for the maintenance of "supplying information view" is:

"SELECT DISTINCTROW [Categories].[CategoryID],

[Categories].[CategoryName], Sum([OrderDetail].[Price] *

[OrderDetail].[Quantity]) AS [Sum Of ordrs] , [StoreID]=str_storeID FROM

(Categories INNER JOIN Products ON [Categories].[CategoryID] =

[Products].[CategoryID]) INNER JOIN OrderDetail ON [Products].[ProductID]

= [OrderDetail].[ProducteID] and [OrderDetail].[IsDirty]=true GROUP BY

[Categories].[CategoryID], [Categories].[CategoryName];";

The SQL query of the maintenance of "sales information view" is:

"SELECT DISTINCTROW [Categories].[CategoryID],

[Categories].[CategoryName], Sum([SalesDetail].[Price]*[

SalesDetail].[Quantity]) AS [Sales Amount], [StoreID]= str_storeID FROM

(Categories INNER JOIN Products ON [Categories].[CategoryID] =

[Products].[CategoryID]) INNER JOIN SalesDetail ON [Products].[ProductID]

= [SalesDetail].[ProducteID] and [SalesDetail].[]IsDirty] = true GROUP BY

[Categories].[CategoryID], [Categories].[CategoryName];";

The SQL query of the maintenance of "inventory information view" is:

"SELECT DISTINCTROW [Categories].[CategoryID],

[Categories].[CategoryName], Max([OrderDetail].[Price]) AS [Price],

Sum([OrderDetail].[Quantity] * [OrderDetail].[Price]) AS [Order amount],

Sum([SalesDetails].[Quatity] * [StoreID] = str_storied, [SalesDetails].[Price])

AS [Sales Amount], (([Order amount]-[Sales Amount])* [Price])

AS [Inventory Value] FROM ((Categories INNER JOIN Products ON

[Categories].[CategoryID] = [Products].[CategoryID]) INNER JOIN

OrderDetail ON [Products].[ProductID] = [OrderDetail].[ProducteID]) INNER

JOIN SalesDetails ON [Products].[ProductID] = [SalesDetails].[ProductID] and

([SalesDetails].[IsDirty] =true or [OrderDetail].[IsDirty] =true))GROUP BY

[Categories].[CategoryID], [Categories].[CategoryName];

The SQL query of the maintenance of "inventory information trace view" is:

"SELECT DISTINCTROW Products.ProductName, "+

    "Max(OrderDetail.Price) AS [Price], "+

    "Sum(SalesDetails.Quatity) AS [sales], Sum(OrderDetail.Quantity) "+

    "AS [order], ([order]-[sales]) AS [#Instock] "+

    "FROM ((Categories INNER JOIN Products ON Categories.CategoryID "+

    "= Products.CategoryID) INNER JOIN OrderDetail ON Products.ProductID

"+

76

"= OrderDetail.ProducteID) INNER JOIN SalesDetails ON

Products.ProductID "+

"= SalesDetails.ProductID GROUP BY Categories.CategoryID, "+

"Categories.CategoryName, Products.ProductID, Products.ProductName, "+

"Products.CategoryID, SalesDetails.ProductID, SalesDetails.Quatity, "+

"OrderDetail.ProducteID, OrderDetail.Price "+

"HAVING ((((Categories.CategoryID)=")="+ CategoryID+)) and

([SalesDetails].[IsDirty] =true or [OrderDetail].[IsDirty] =true));";

The SQL query of the maintenance of "sales information trace view" is:

"SELECT [Products].[ProductName], SalesDetails.Price, SalesDetails.Quatity,

sales.SalesDate, ([SalesDetails.Price]*[SalesDetails.Quatity]) AS [sales Amount]

FROM (Categories INNER JOIN Products ON Categories.CategoryID =

Products.CategoryID) INNER JOIN (sales INNER JOIN SalesDetails ON

sales.SalesrID = SalesDetails.SalesrID) ON Products.ProductID "+

"= SalesDetails.ProductID and ([SalesDetails].[IsDirty] =true WHERE

((((Categories.CategoryID)="+ CategoryID+));";

The SQL query of the maintenance of "supplying information trace view" is:

source; together with categoryID and StoreID, we can generate lineange trace

query like:

"SELECT Products.ProductName, OrderDetail.Price, "+

"OrderDetail.Quantity, Suppliers.SupplierName, Orders.OrderDate "+

"FROM Suppliers INNER JOIN ((Categories INNER JOIN Products ON "+

"Categories.CategoryID = Products.CategoryID) INNER JOIN "+

77

"(Orders INNER JOIN OrderDetail ON Orders.PurchaseOrderID = "+

"OrderDetail.POID) ON Products.ProductID = OrderDetail.ProducteID) "+

"ON Suppliers.SupplierID = Orders.SupplierID "+

"WHERE (((Categories.CategoryID)=" + CategoryID+ "));";

;

7.8 Development Tool and Running Environment

The running environment of the inventory data warehouse system is Java virtual machine, with which, program developed in java can run on any operating system such as Unix, Linux, windows, and Macintosh. Further more, java program has more security than other program language, like C++, VB, etc. So we develop the system in java, and use the development tools Jbuilder 8 and JDK1.4 package.

78

8 Conclusion

In this chapter, we will give some comments on the system developed in the thesis from professional inventory staff members. And then, we will discuss some shortcomings of the demo system and the work to be done in the future to improve the system.

8.1 Comments on the System from a Professional

In this section, we will give some comments on the inventory management data warehouse system from an inventory professional after she saw the demo of the system.

The comments are given by a senior auditor who has extensive experience in inventory management. The comments will be described in two main parts: the local inventory management component and the data warehouse integrator component. Based on her opinions, the data views in the inventory data warehouse of retail business chains should give their branches' total inventory at cost, total sales, and total purchase, which are important for a company to report GST/QST and other related accounting and management decision. Furthermore, the information of total sales for each product line is very useful for management to analyze the sales trends and to make purchase decisions.

For small to middle sized retail companies, both the computer knowledge of the staff and the budgetary knowledge on the e-management are limited. So, the program developed for them must be easy to use and easy to maintain. The system, which includes hardware and software, should be as simple as possible.

79

The interface of the program should be similar to the MS Word, Excel or other popular software so that it's easy for people to pick up. The basic maintenance can be handled by a staff member without special qualification in computer skills after a short period of training.

In the following sections, we will give more detailed comments: comments on the local inventory management component in section 8.1.1 and comments on inventory management data warehouse integrator component in section 8.1.2.

8.1.1 Comments on the local Inventory Management Component

This component gives most of the necessary functions of inventory operation management, which include purchase ordering, goods sales, and inventory stock checking. There are forms for inventory manager and staff to change/add an employee information, a customer's /supplier's information, a product item information, a purchase order, a sales order. Besides that, there are also reports to review total inventory value, total sales, and total purchase order.

The interface design of the program follows the design patterns of a Microsoft windows program, so it is easy to learn and operate. The sales and purchase order form give very convenient interfaces for a staff to enter a contract, in which the number of product item can be fit for any situation.

The system chooses MS Access to develop an inventory management component of a small sized business, which is a high performance/cost rate solution. It meets the most of requirements of a small business operation management such as transactions volume requirements, data size requirements, security requirements, and develop/maintenance cost requirements. Furthermore, in the solution, the

80

hardware/software choosing for the system such as hardware, software, and communication channels are in the budget of the type of businesses.

8.1.2 Comments on Data Warehouse Integrator Component

This part of the demo system gives basic data views of inventory data warehousing management system, which meets most of management functional requirements of small business chains. It puts most frequently used data search functions into data views and put less frequently data search functions into lineage trace. For example, the summary on the total sales, total ordering, and total inventory at cost are what managers of most businesses concern most, we put them into data view functions; and checking detail of sales, purchasing, and inventory stocking is not very often, so we put them into lineage trace. This kind of design can simplify the interface of the system and reduce the workload of the system running.

8.2 Shortcoming of Inventory Data Warehouse System

The local inventory system's functions can meet most of the small sized business chains' operation management requirements, in which, each branch is a club member, which shares same brand with other members, pays annual fees, and pay a percentage of annual total sales, like McDonald. Each business may have different operations management function requirements on the system; such as inventory management can write off a specific product item stock quantity. Maybe it needs more functions from the system for customers to choose. At mean time, some functions of the program may need to be redesign when it used

81

in a specific company. For example, they may need more details on the total sales data views in a specific time period, like total sales per a year, per a month, a week, per a day, or in a rush hour time period (3:PM to 7:00 PM on Thursday, Friday).

In the sales and purchase form of local inventory management component, product item stocking quantities and prices being given in a temporary window for seconds when user enter a product item ID is much better for using.

8.3 Future Work

The works needed to improve the system developed in the thesis are listed below. First of all, more case studies on the operations management of the types of the businesses are necessary to deeply understand their operations management to improve our business models for the types of the businesses; Consequently, we can get more complete requirements of the software for the types of businesses to refine the system to help the operations management work efficiently.

Secondly, studying the program's user characteristics, which includes the cultural background and the average computer skills background of the staff in the type of businesses, can help us develop more user-friendly interfaces for the program. Thirdly, to handle the tasks of the inventory operations management well, the program may need functional changeable by the customer because of the specific functional requirements for the system, so customers function choosing able/changeable (XP) design solution on the system may be convenient.

Fourthly, an extensive testing on the data warehousing in real commercial environment by the inventory professionals, which is called β-testing, and more studies of the lineage trace algorithms working efficiency at full loading for the inventory data warehouse operations management system of the types of the businesses are necessary. With the analyses on the testing result, we can optimize the system design, warehousing algorithms, and lineage trace algorithms.

Finally, studying the securities and reliable requirements of the types of businesses well can help us give good performance/cost rate security and reliability/ fault tolerance solutions to the system, such as choosing more reliable computer systems (like Unix), more robust databases (like Oracle, DB2, MS SQL Server, Informix), high reliable communication systems, and a more safety network.

APPENDIX User Guide

In this section, we will give user a guidance of the system installation and user menu. A.1 and A.2 are telling user how to install the system in a computer network of a business, which include the installing of the system includes the installation of local inventory system, building the ODBC connection, and data warehouse integrator; A.3 and A.4 are guiding user how to use the system.

A.1 Install the Local Inventory Management System

The installation of local inventory system into computer network of a business consists of hardware platform choosing, software platform setup, and local inventory component installation.

First of all, we need to choose a PC server where a MS Windows NT operator system is running on and all terminal PCs that inventory staff run the local inventory management component should have MS Windows 2000/ CE/XP in the computer network of the business. In the NT system, we need to install a MS Access database system and make the program shareable. And then, we copy our inventory management program in to a specific folder in the server PC, such as c:\inventory and make a short cut to the inventory management program in very terminal PC of local business. At mean time, we need to copy data warehouse integrator component into the PC terminal in the business chain headquarter department, and make short cut of the "datamining.bat".

A.2 Setup ODBC Data Source for Every Remote Inventory Management System

In section 7.3, we have already talked about the theory of how to match a remote data source into a local ODBC object by create a DSN for every remote inventory management system.

The steps for creating a DSN are as follows:

1. From the desired tab User, press the "Add" button to begin creating a new Data source, like Figure 15.
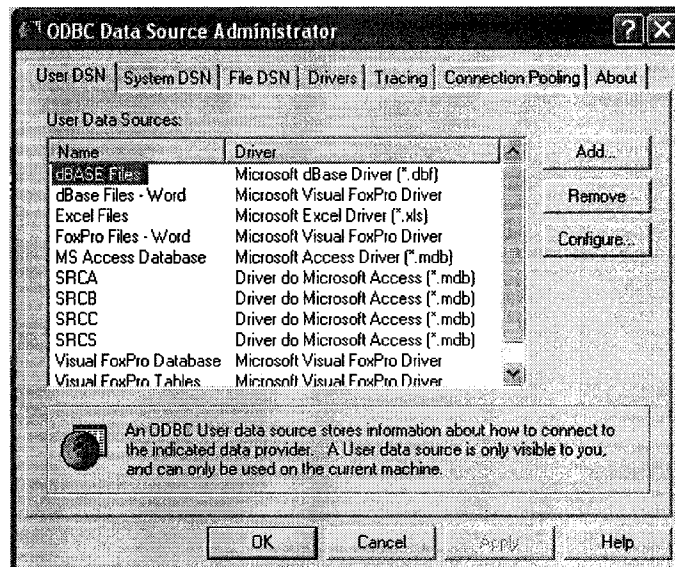
Figure 15 step 1 create data source

2. Choose the data source driver type, in our example it is "Driver do Microsoft Access (*.mdb)", like the Figure 16
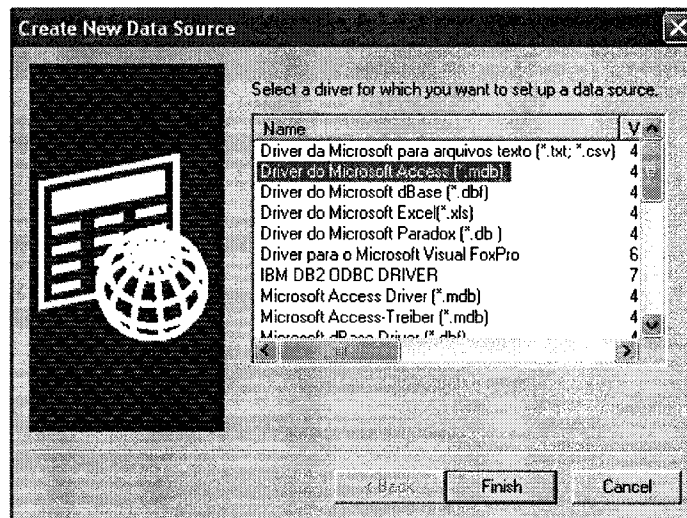

Figure 16 step 2 choose data source type

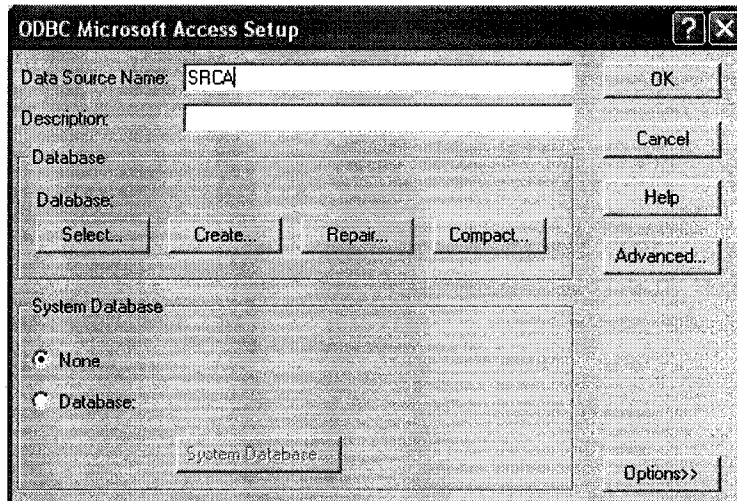3. Give a name to the data source, like the Figure 17

85

Figure 17 step 3 Assign a name to the data source type

4. From above menu, click button "Select" to select database menu, change drives name to the driver where the data source is located, change directory to the folder where the data resource, and then highlight it, press the button "OK". Like below Figure18.
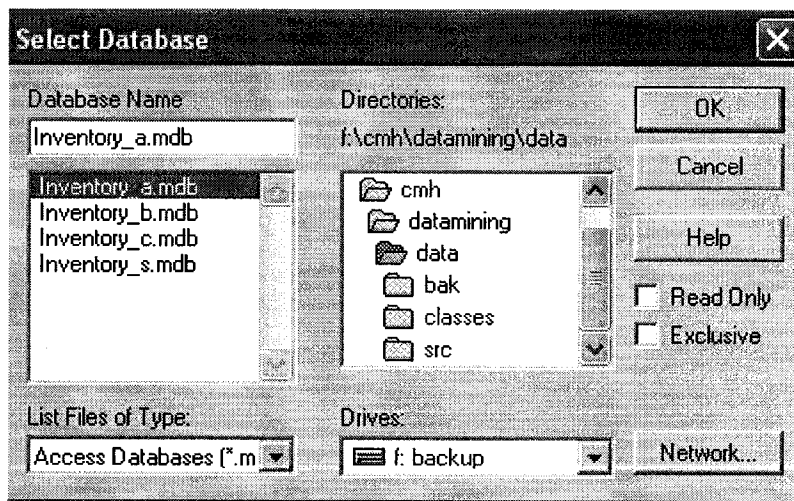

Figure 18 step 4 select the data source

86

In the above steps, we create ODBC data sources for every remote database and give them different names.

A. 3 User Menu of the Local Inventory System

This section tells user how to use the local inventory management system. A user powers on his PC that has installed the local inventory system, double click the short cut that is created in the A1, the system will be launched and the main menu of it will appear in the screen of the terminal like Figure 19.

A.3.1 Main menu

The interface provides the entries to all the functional form, given the picture Figure 19; in the menu, click the first button to add/view/updating a product, click third button to review all kinds of statistical reports, click button "Enter/view a Purchase Order" to create a new purchase order, etc.
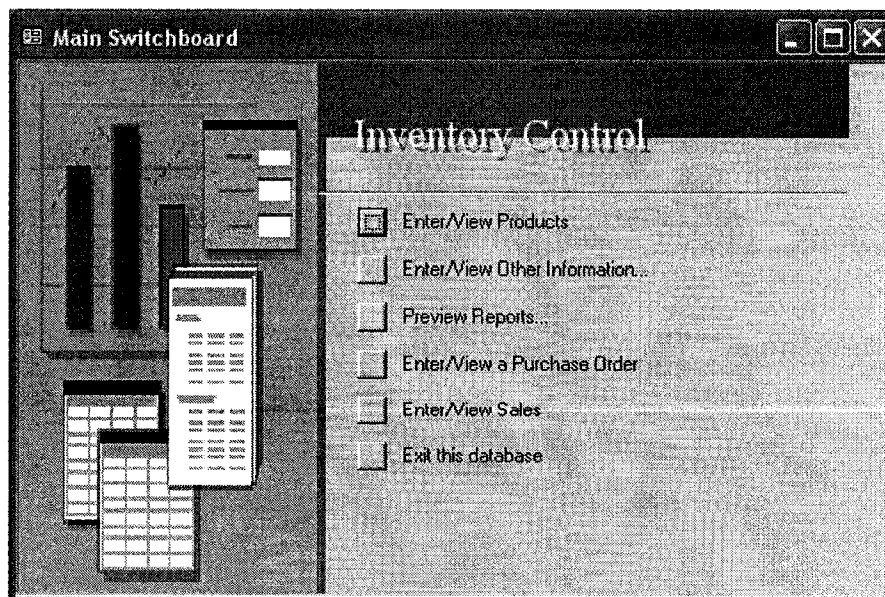

Figure 19 The main interface

A.3.2 Entry/View Product Item Menu

From the menu in the Figure 19, when user clicks the button "Enter/View Product", user enters the form "Enter/View Product" showed in the Figure 20. In the form, user can view or change product item detail, such as name, unit price, quantity per unit, and stock quantity, etc. Further a user can add a new product item. When a user enters an index number in the navigation bar (at the bottom) to view a product, main form will show the product information, and the datasheet will show the inventory quantity; when the user click the "*" button in the navigation bar, he can enter a new product, as a consequence; the inventory area is empty.

Figure 20 The product detail entry form

88

A.3.3 Entry/View Product Purchase Order Menu

From the menu in the Figure 19, when user clicks the button "Enter/View Purchase Order", user enters the form "Enter/View Purchase Order", which is showed in the Figure 21. In the form, user can view or change product purchase order, such as adding/removing product item, changing price or order quantity. Besides these functions, the users can create a new purchase order. If a user wants to find a specific purchase order, such as "PO129", he/she just enters 129 in the text field of the navigation bar (in the bottom of the form) and the returned form will show the purchase order information. If a user wants to enter a new purchase order, he/she just clicks the "*" button to create a purchase order. After he/she enters all the ordering information in the main form and sub-datasheet and return, he/she saves a new purchase order in the system.

Figure 21 The Entry/ view purchase order form

89

A.3.4 Entry/View Product Sales Menu

From the menu in the A.3.1, when user clicks the button "Enter/View Sales Order", user enters the form "Sales Entry Form", which is displayed in the Figure 22. In the form, user can check or change product sales order, such as adding/removing product item, changing price or order quantity. Besides these, he can create a new sales order. If a user wants to find a specific sale contract, such as "S1001", he just enters 1 in the text field of the navigation bar and the returned form will show the sale contract information. If a user wants to create a new sale contract, he just click the "*" button and enter all the ordering information in the main form and sub-datasheet.

Figure 22 The Entry/ view sales contract form

A.3.5 Report View Sub Menu

From the menu in the Figure 19, when user clicks the button "Preview reports",
user enters the sub menu "Inventory switchboard" which is displayed in the
Figure 23. In the menu, users can click the buttons to view inventory, sales, and
order summary reports.

91

Figure 23 The report view menu

A.3.6 Product Inventory Report View

From the menu in the Figure21, when user clicks the button "Preview the Product Inventory Reports", the system generates product inventory summary report. The report is like Figure 24.

92

Inventory Summary

Product	Product Name	Price	Ordered	Sales	#Stock	Inventory Value
Lauan 2.7mm 4' x 8'	Lauan 2.7mm 4' x 8'	$164.12	130389	8529	121860	19999663.2
Lauan 3.6mm 4' x 8'	Lauan 3.6mm 4' x 8'	$324.67	107640	25940	81700	26525539
Lauan 3.6mm 4' x 8' S	Lauan 3.6mm 4' x 8' SSB	$174.64	25200	3010	22190	3875261.6
Lauan 5.2mm 4' x 8'	Lauan 5.2mm 4' x 8' D	$323.99	11040	2600	8440	2734475.6
Lauan 5.2mm 4' x 10	Lauan 5.2mm 4' x 10	$304.06	11500	1850	9650	2934179
		•				
		•				
		•				
paneling mdf Glacier	paneling mdf Glacier Oak	$7.28	12000	2120	9880	71926.4
paneling mdf Pacific	paneling mdf Pacific Maple	$7.28	60000	5280	54720	398361.6
paneling mdf Rustic B	paneling mdf Rustic Birch	$7.28	98280	10119	88161	641812.08
Panelina Paper White	Panelina Paper White Ash	$269.45	24000	2400	21600	5820120

Figure 24 The inventory summary

A.3.7 Product Supplying Report View

From the menu in the Figure21, when user clicks the button "Preview the Product supplying Reports", the system generates product supplying summary report. The report is like Figure 25.

93

Supply Summary Report

Supplier	*Brazil Remittance Import & Export*			*Phone*	*416-588-0749*
Address	*1462 Dundas Street West, ,*			*City*	*Toronto*
Province	*ON*	*Country*	*Canada*	*Postal Code*	*M6J 1Y6*

Product Name	Price	Quantity	Date	Sub Total
Radiata Pine 4' X ** 18mm	$1,015.69	5400	/24/2004	$5,484,726.00
paneling print bleached birch	$9.04	1500	/12/2004	$13,560.00
Total Amount:				$5,498,286.00

Supplier	*ADM Import & Export*			*Phone*	*416-431-0644*
Address	*108 Corporate Drive*			*City*	*Scarboro*
Province	*ON*	*Country*	*Canada*	*Postal Code*	*M1H 3H9*

Product Name	Price	Quantity	Date	Sub Total
Lauan 3.6mm 4' x8'	$324.67	2340	1/1/2004	$759,727.80
Lauan 3.6mm 4' x8' SSB	$174.64	3600	3/3/2004	$628,704.00
1/8 Lauan doorskins 36 1/2X84 1/2	$156.84	600	/12/2004	$94,104.00
Tileboad 4' X 8' Ivory	$13.28	1000	/15/2004	$13,280.00

Figure 25 The product supplying summary

A.3.8 Product Sales Report View

From the menu in the Figure 21, when user clicks the button "Preview the Product Sales Reports", the system generates product Sales summary report. The report is like Figure Figure26.

94

Sales Summary

CompanyNam	Alfreds Futterkiste			PhoneNumber	030-0074321
City	Berlin
Address	Obere Str. 57			City	Berlin
Province		Country	Germany	PostCode	12209

Product Name	Price	Amount	Sales Date	Sub Total
Birth 3mm 4'x 8'	$23.00	125	10/20/2005	$2,875.00
Total Sale Amount:				$2,875.00

CompanyNam	Around the Horn			PhoneNumber	(171) 555-7788
Address	120 Hanover Sq.			City	London
Province		Country	UK	PostCode	WA1 1DP

Product Name	Price	Amount	Sales Date	Sub Total
1/8 Lauan doorakins 36 1/2X84 1/2	$886.70	149	10/22/2005	$132,118.90

Figure 26 The product sales summary

A.4 User Menu of the Inventory Data Warehouse Integrator

This section introduces the warehouse integrator user guide. A user powers on his PC that has installed the inventory data warehouse integrator, double click the short cut that is created in the A1, the system will be launched and the main menu of it will appear in the screen of the terminal like Figure 27.
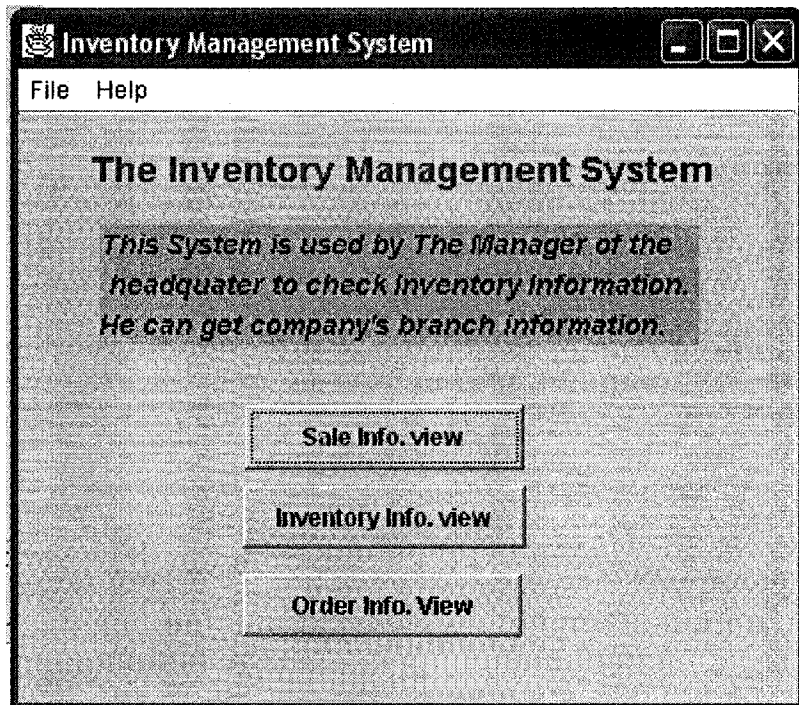
95

Figure 27 The main menu

A.4.1 Product Sales Information Review

From the menu in the Figure 27, when user clicks the button "Sales Information view", the system go to the table display form that display all remote stores product sales summary report. The report is like Figure 28.

96

Sales Information Review

File Look and Feel Help

CategoryID	CategoryName	Sum Of Sales	storeID
1	soft wood for internal	40997588.50	A
2	hard wood floor	2402816.33	A
3	wood for wall	1019255.80	A
4	wood for outside	930378.00	A
5	semi production	43788545.70	A
6	wood application	3203748.45	A
1	soft wood for internal	40997588.50	B
2	hard wood floor	2402816.33	B
3	wood for wall	1019255.80	B
4	wood for outside	930378.00	B
5	semi production	43788545.70	B
6	wood application	3203748.45	B
7	pata	65597522.10	B

Figure 28 The Sales Information view

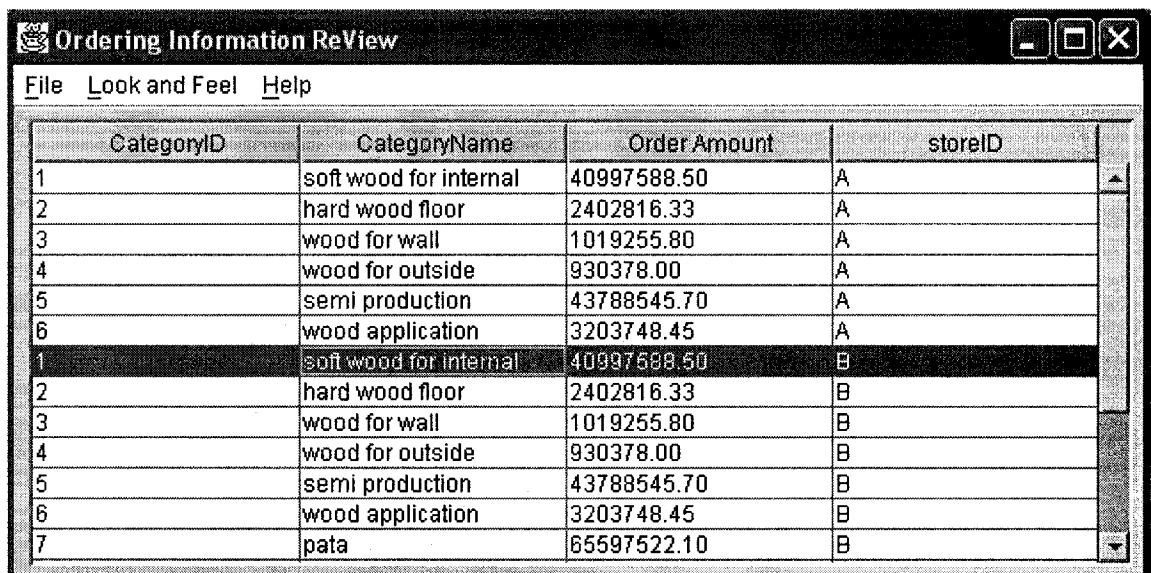## A.4.2 Product Inventory Information Review

From the menu in the Figure27, when user clicks the button "Inventory Information View", the system go to the table display form that display all remote stores product inventory summary information. The report is like Figure 29.

Inventory Information ReView

File Look and Feel Help

CategoryID	CategoryNa...	Price	Order amount	Sales Amou...	Inventory Va...	storeID
1	soft wood fo...	1575.00	59961656.10	14518659.66	715727193...	A
2	hard wood fl...	346.79	3124832.33	762499.13	819233529....	A
3	wood for wall	168.10	1841199.80	448089.36	234181865....	A
4	wood for out...	313.18	1516258.00	373707.73	357823892....	A
5	semi produ...	1250.00	58716025.70	14626357.33	551120854...	A
6	wood applic...	269.45	5639622.05	1376002.03	114883241...	A
1	soft wood fo...	1575.00	59961656.10	14518659.66	715727193...	B
2	hard wood fl...	346.79	3124832.33	762499.13	819233529....	B
3	wood for wall	168.10	1841199.80	448089.36	234181865....	B
4	wood for out...	313.18	1516258.00	373707.73	357823892....	B
5	semi produ...	1250.00	58716025.70	14626357.33	551120854...	B
6	wood applic...	269.45	5639622.05	1376002.03	114883241...	B
7	pata	1016.63	113773612....	27760761.45	874432447...	B

97

A.4.3 Product Order Information Review

From the menu in the Figure 27, when user clicks the button "Order Information View", the system go to the table display form that display all remote stores product order summary information. The report is like Figure 30.

CategoryID	CategoryName	Order Amount	storeID
1	soft wood for internal	40997588.50	A
2	hard wood floor	2402816.33	A
3	wood for wall	1019255.80	A
4	wood for outside	930378.00	A
5	semi production	43788545.70	A
6	wood application	3203748.45	A
1	soft wood for internal	40997588.50	B
2	hard wood floor	2402816.33	B
3	wood for wall	1019255.80	B
4	wood for outside	930378.00	B
5	semi production	43788545.70	B
6	wood application	3203748.45	B
7	pata	65597522.10	B

Figure 30 The Supplying Information view

A.4.4 Product Sales Information Lineage Trace Review

From the table display in the Figure 2.9, when user highlight a record and double clicks, the system run the lineage trace module and go to the table display form that display all information that contribute the record. The result display is like Figure 31.

98

ProductName	Price	Quality	SalesDate	sales Amount
Birch 18mm 4'x 8'	1525.00	280	2004-09-13	427000.00
Paneling monde Frosted Oak	8.15	500	2004-10-20	4074.80
Paneling monde Frosted Oak	8.15	1000	2004-10-14	8149.60
paneling print bleached birch	11.03	400	2004-10-06	4411.52
paneling print bleached birch	11.03	300	2004-10-16	3308.64
paneling print Rocky Pecan 11	9.08	300	2004-09-23	2723.04
snow Flake vinyl 4 x 8	12.22	300	2004-10-20	3667.32
snow Flake vinyl 4 x 8	12.22	400	2004-10-16	4889.76
snow Flake vinyl moulures	2.78	200	2004-10-26	556.32
Honeycomb 1 1/2 x 36	889.16	337	2004-09-13	299647.05
Honeycomb 1 1/8 x 24	538.48	2560	2004-10-12	1378518.02
Honeycomb 1 1/8 x 36	527.67	320	2004-12-11	168855.81
Honeycomb 1 1/8 x 36	727.47	360	2004-12-15	261890.57

Figure 31 The sales query lineage trace view

A.4.5 Product Inventory Information Lineage Trace Review

From the table display in the Figure 30, when user highlights a record and double clicks, the system run the lineage trace module and go to the table display form that display all information that contribute the record. The result displayed is like Figure 32.

99

ProductName	Price	sales	order	#Instock
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	151	2000	1849
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	157	2000	1843
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	163	2000	1837
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	169	2000	1831
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	175	2000	1825
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	181	2000	1819
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	187	2000	1813
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	193	2000	1807
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	199	2000	1801
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	205	2000	1795
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	211	2000	1789
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	217	2000	1783
1/8 Oak doorskins 30 1/2 x 80 1/2	292.94	223	2000	1777

Figure 32 The Inventory query lineage trace view

A.4.6 Product Order Information Lineage Trace Review

From the table display in the Figure 31, when user highlight a record and double clicks, the system run the lineage trace module and go to the table display form that display all information that contribute the record. The result displayed is like Figure 33.

100

Figure 33 The Inventory query lineage trace view

101

References

[1] W. H. Inmon. *Building the Data Warehouse,* Second Edition, John Wiley &
Sons, Inc. 1996.

[2] Rob Mattison. *Data Warehousing: Strategies, Technologies, and Techniques,*
McGraw-Hill. 1996.

[3] Adam N.R, Dogramaci O., Gangopadhyay A., Yesha Y. (1999).
Electronic Commerce Technical, Business, and Legal Issues. Prentice-Hall.

[4] Kelly,Sean. Data Warehouseing - The Key to Mass Customization,
John Wiley & Sons,Inc. New York, 1994.

[5] Data warehouse URL: http://iroi.seu.edu.cn/books/whatis/dataware.htm

[6] Yingwei Cui and Jennifer Widom, Lineage Tracing in a Data Warehousing
System

[7] *Building the Operational Data Store,* John Wiley & Sons, Inc. 1995

[8] Mark Humphries, Michael W. Hawkins, Michelle C. Dy, Data Warehousing
Architecture and Implementation Prentice Hall PTR,1999

[9] Yingwei Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in
a warehousing environment. Technical report, Stanford University Database
Group, November 1997.Available at http://www-
db.stanford.edu/pub/papers/lineage-full.ps.

[10]. Yingwei Cui. *Lineage Tracing in Data Warehouses.* Ph.D. thesis, Stanford
University, December, 2001

102

[11] Felipe Carino. High-performance, parallel warehouse servers and large-scale applications, October 1997. Talk about Teradata given in Stanford Database Seminar

[12]ÖZSU, M. T. AND VALDURIEZ, P. 1991. *Principles of Distributed Database Systems*. Prentice-Hall, Englewood Cliffs, NJ.

[13] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.2, OMG Technical Document formal/98-07-01, February 1998.

[14] http://www.borland.com/jbuilder/

[15] http://java.sun.com/j2se/

[16] http://office.microsoft.com/en-us/FX010857911033.aspx

[17] http://msdn.microsoft.com/

[18] G.J. Myers, *The Art of Software Testing,* John Wiley & Sons, New York, 1976.

[19]. T.J. Ostrand and M.J. Balcer, "The Category-Partition Technique for Specifying and Generating Functional Tests," *Comm. ACM,* Vol. 31, No. 6, June 1988, pp. 676–686.