Lineage Tracing in Data Warehousing Systems:

A Design and Implementation

Jiu Xu

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2003

0-612-77725-1

ABSTRACT

Lineage Tracing in Data Warehousing Systems: A Design and Implementation

Jiu Xu

Data warehouse, as the foundation of decision support system, is critical for the managers to make decisions. It is different with operational database. Data warehouse reads data from multiple operational databases instead of getting the data from the end user transaction input. In a warehousing environment, the data lineage problem is that of tracing warehouse data items back to the original source items from which they were derived. Enabling lineage tracing in a data warehouse environment has several benefits and applications, including in-depth data analysis and data mining, authorization management, efficient warehouse recovery, etc.

In this report, we firstly introduce the basic concept and architecture of data warehouse, as well as the development tools and methods about data warehouse. Secondly, we discuss the lineage tracing problems and challenges in the data warehousing system, and then use an example to present the algorithms and procedure of lineage tracing. As well, we will present our design and implementation of a prototype system called LTI, to demonstrate the lineage tracing procedures using an inventory system as a data warehouse system. We also developed various graphical user interfaces required to facilitate interacting with the system in order to update the source databases in the LTI system. Finally, we will show the experimentation of using our LTI system through tracing inventory and sales order data in the data warehouse system.

Acknowledgments

Thanks to my supervisor, Dr. Nematollaah Shiri, who guided me in this research and gave me lots of valuable advises and instructions. Also, I would like to thank my fellow graduate students, Minghua Chen, Dongmei Liu, Minggang Wu, together with whom we studied this topic as a team project in the course COMP6591, and developed the first version of the ideas of lineage tracing in the data warehouse systems.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction to Data Warehousing

As early as the 1970s the merits of placing specially prepared data on separate platforms for decision support purposes were recognized. This approach provides easy access to the needed data, improves system response time, and enhances data integrity and security [GW1998]. In the 1990s, data warehousing was the subject of numerous developments for data analysis and research. Many organizations developed data warehouse systems to provide the end users with clean, consistent, and relevant data for querying and analysis.

Following the changes in a business situation, the decision-makers in an enterprise hope to get the needed information in short time. They need to look through business data to identify business opportunities that can provide competitive advantages, improve profits, or reduce costs. This review of the enterprise data should be done from different perspectives and at different levels of detail to find and address business problems as the problems arise. The data warehouse in an organization collects the data from a wide variety of data sources, such as the data about customers, products and financial data, for analysis and making decisions based on the integrated information.

Traditional operational systems help storing data into databases quickly, safely, and efficiently. However, they do not support delivering meaningful analysis in return. A data warehouse is an extremely efficient system for gathering data from business transactions or external sources, organizing the information in suitable formats, and providing users with the tools needed to analyze the data, thus supporting better business decisions. The data warehouse makes it possible to analyze business behavior, trends, and changes over time. It offers data timeliness, consistency and comparability, and facilitates the process of turning data into business intelligence information.

In the rest of this introductory chapter, we address the following questions:

- What is a data warehouse?
- Why do we need data warehouses?
- Characteristics of data within a data warehouse system
- Architecture of a data warehousing system
- Data warehousing processes and approaches
- Data warehousing software components
- Methods of extraction and transformation of data
- Data warehousing development life cycles

1.1 What is a Data Warehouse?

Data warehouse was originally envisioned as a separate architectural component that converted and integrated masses of raw data from legacy and other operational systems and from external sources. In data research and industry, the approach for providing integrated access to multiple distributed, heterogeneous databases and other information sources and select data into a single repository is commonly referred to as *data warehousing* [Wid1995]. A large database system storing the selected data for data querying and analysis is called *a data warehouse* [Inm1996] or *a data warehouse system.*

Inmon, known as the father of data warehousing, defines a data warehouse as a "*collection of integrated, subject-oriented databases designed to supply the information required for decision-making*" [HWM1999]. The key characteristics of a data warehouse are listed as:

- **Subject orientation**. Warehoused data are organized by subject area (for example, product, customer, etc), and span organizational and process boundaries. They present a wide business view of information.
- **Integration**. A data warehouse collects data from many operational systems and source databases, and provides an integrated view for the end users.
- **Databases.** The term data warehouse itself refers to a large, read-only repository of data. At the heart of data warehouse lies the large database that stores the integrated data "collected" from different data sources.

A data warehouse is a kind of database in two senses — technical and business [Mat1996]. A large physical database (technical support) must be clearly defined, which holds all information of interest to specific groups of business (business understanding). A data warehouse is nothing really more than a large database that holds copies of data from other systems which is made available for use by other applications [Bra1996]. So a data warehouse is a database that [Mat1996]:

- is organized to serve as a data storage area
- is used by data-mining and other applications
- meets a specific set of business requirements
- uses data that meets a predefined set of business criteria

1.2 Why Do We Need Data Warehouses?

Data warehouse systems are used for managing the time-variant data in order to analyze business trends and make decisions. They can be used to store large quantity of data together with summaries of data, which could be rolled up to higher levels of generalization or drilled down to lower levels of detail. They provide the primary support for on-line analytical processing (OLAP), decision support systems, and executive information systems. An original purpose of a data warehouse system was to support computation intensive queries for data analysis. However, it was then realized, as the so-called the second generation of data warehouse systems, that they could be used for data querying as well.

From a business perspective, a data warehouse provides a single and "consistent" data source. The user can issue different business queries to the data warehouse. Queries can be answered and analyzed quickly and efficiently, since the integrated information is directly availably at the warehouse. The decision makers in the organization can use the results of the queries to make strategic decisions and solve business problems [Mat1996]. More and more companies are using data warehousing as a strategic tool to help them win new customers, develop new products, and lower the costs. The data warehouse becomes the common information resource for making decisions throughout the organization. For the business users, the overall objectives of a data warehousing system as discussed in [MAG1998] are:

- ensure accurate, high-quality data, that are pertinent to the decision-making processes of the enterprise;

- provide a consistent and integrated view of the enterprise data;
- provide easy access to data;
- provide timely access to data through fast, automated data gathering and delivery;
- empower business users through a friendly, effective access interface; and
- provide flexibility to grow with and adapt to changing business requirements.

1.3 Characteristics of Data within a Data Warehouse System

Data in a data warehouse may come from various operational database and other business application systems, for example, accounting information, operation information, inventory information, customer information, etc. Data warehouse builds cross-reference information among these different data sources to enable data analysis and querying capabilities. It can also be used to group data into different subject areas so that users can search and analyze data easily and in a focused manner. Such specialized subject-oriented data warehouses are called data marts.

1.3.1 Contents of a Data Warehouse

A data warehouse is served to store the information produced by other systems, not to produce new information. A data warehouse includes not only copies of the current information, but also historical copies thereof as well. Usually the data in a data warehouse serves more than one application, so the data warehouse must be designed as shared and simplified. As such, the data in warehouse must be organized in a way that makes it easy for users to search and manipulate [Mat1996].

Highlights of the features of a data warehouse are as follows [Mat1996]:

- The tables in a data warehouse are very large
- The number of tables are very large
- The data in tables have a high degree of interdependency
- The data is accessed in a read-only mode by the users
- Changes to the information sources should be reflected to the data warehouse periodically
- Much of the data collected and maintained in the data warehouse will be historical (time-dependent)

1.3.2 Problems of Moving Data into the Warehouse

When building a data warehouse, two major problems need to be solved. First, the data in a data warehouse should be cleaned, validated, and properly aggregated. We cannot simply export disparate data from operational databases to a data warehouse. Second, the data placed in a data warehouse system must be consistent, and be prepared using formal techniques for managing and documenting summary data [Bra1996].

1.4 Architecture of a Data Warehousing System

As defined earlier, a data warehouse is a repository of integrated information from distributed, autonomous, and possibly heterogeneous sources. A data warehousing system collects data from multiple distributed sources and stores the integrated information as materialized views in a local data warehouse, and keeps the view contents "up-to-date" when the sources change. Users can then perform data querying, and analysis on the warehouse views.

In fact, a warehouse stores some materialized views of the source data from different sources. A user can query and analyze the data in the warehouse. Figure 1.1 shows a basic architecture of a data warehousing system [LZW1997]. As we can see, there are three major components: *the data integration component*, *the data warehouse itself*, and *the query and analysis component* [Mat1996].

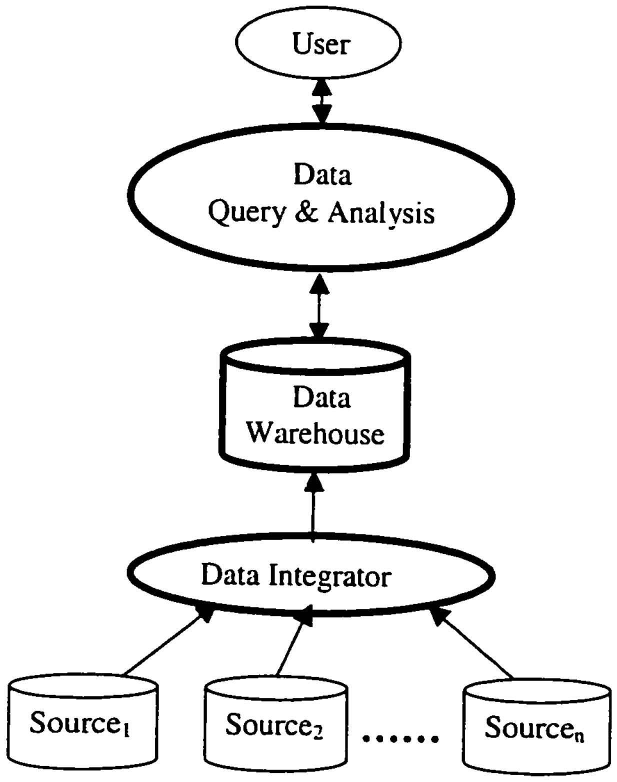


Figure 1.1 A basic architecture of a data warehousing system

The main component of a data warehouse system is the data warehouse itself: a large, physical database that holds a vast amount of information from a wide variety of sources. The data within the warehouse is organized in a way that makes it easy to search and use, and is updated frequently from the sources. The data integration component includes programs that are used for collecting data from the sources and storing them in the warehouse. They are also used to collect and maintain the materialized views. The query and analysis component includes all the data warehouse applications to support specific end user's queries and analyses requirements.

1.5 Data Warehousing Processes and Approaches

1.5.1 Data Warehousing Processes

Commonly, the data integration problem is based on the following two-steps process [Wid1995]:

1. Accept a query, determine the appropriate subset of information sources to answer the query, and generate the appropriate subqueries or commands to each information source.

2. Obtain results form the information sources, perform appropriate translation, filtering, and merging of the information, and return the final answer to the user or the application requested.

1.5.2 Data Warehousing Approaches

There are two alternative and competing approaches for building a data warehouse system — *data-driven* and *application-driven* [Kel1996].

1. Data-driven approach

In this approach, a pool of data is selected to migrate to the data warehouse system which then accumulates and maintains the source data added to the system. For example, we decide to build a data warehouse system to manage the inventory information for the head office to analyze the data obtained from several sources at the branches. This approach can be used to select product inventory and other related information, such as product category, and supplier information, from different sources, and then merge them as an integrated data resource into the data warehouse. An important advantage of this approach is autonomy of the sources, where separate pools of data, each of which is resident in the operational environment with its own technology and platform, send their data to the data warehouse. The mapping of the data from the sources to the target data models is done during migrating the data.

2. Application-driven Approach

In this approach, separate applications are likely to require the data from different operational systems in order to process the queries by the applications. For example, the first application for the inventory information system is to query the products inventory information. Some attributes of data can be used for other application system, such as the sales order information. So the application-driven approach will result in the delivery of a tangible business benefit for the organization.

1.6 Data Warehouse Software Components

In developing a data-warehousing project, a warehousing team will require several different types of software tools. These software products generally fall into one or more of the categories illustrated in Figure 1.2, as described below [HWM1999].

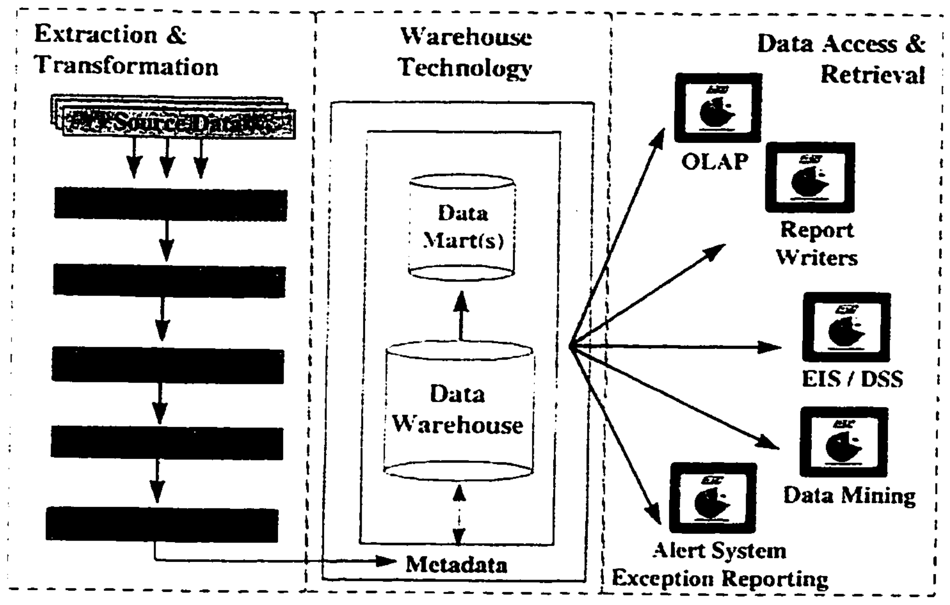


Figure 1.2 Software components of a data warehouse

- **Extraction and transformation.** The warehouse team requires tools that can extract, clean, transform, integrate, and load data from sources to the data warehouse.

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

- **Data access and retrieval.** Different types of software are required to access, retrieve, distribute, and present warehouse data to the end users. Data access and retrieval tools are currently classified into the subcategories such as: Online Analytical Processing (OLAP), Report Writers, Executive Information System (ELS), Data Mining, Exception Reporting and Alert System, etc.

1.7 Methods of Data Extraction and Transformation

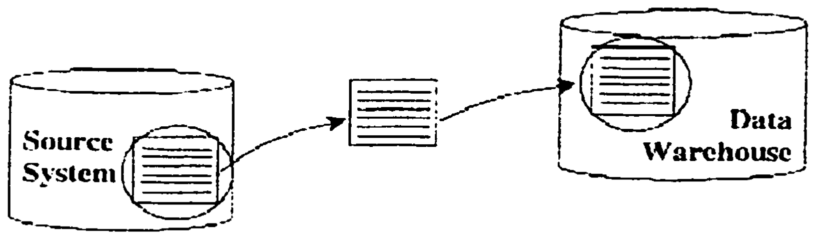
There are several methods used when a warehouse team solves the problem of extraction and transformation data from source systems into the data warehouse. Next, we briefly introduce these methods.

1.7.1 Extraction Methods

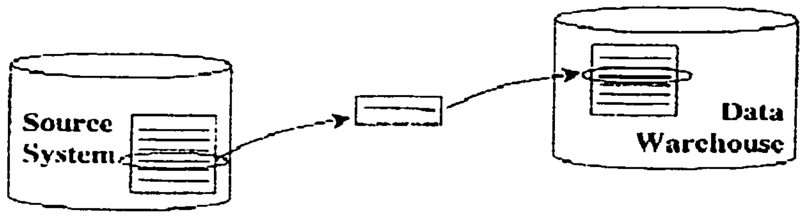
Two primary methods for extracting data from source systems are change-based replication and bulk extractions as illustrated in Figure1.3 [HWM1999].

- **Bulk Extractions.** In this method, the data warehouse is refreshed periodically by extraction data from the source systems. This approach is expensive since it requires the connection between the sources and the data warehouse and transferring huge amount of data from the former to the latter. On the other hand, such warehouses are easier to set up and maintain.

- **Change-Based Replication.** In this methed, only updated data or newly inserted data are extracted and loaded into the data warehouse. This approach places less stress on the network, but requires complex implementation of algorithms to compute, the changes and apply them to the data warehouse.



(a) Bulk Extractions



(b) Change-Based Replication

Figure 1.3 Extraction options

1.7.2 Transformation Methods

Transformation methods transform extracted data from data sources into the appropriate format, data structure, and values, as required by the data warehouse. Most transformation methods provide features illustrated in Figure 1.4 [HWM1999].

SOURCE SYSTEM	TYPE OF TRANSFORMATION	DATA WAREHOUSE
Address Field: #123 ABC Street XYZ City 1000 Republic of MN	**Field Splitting**	**No:** 123 **Street:** ABC **City:** XYZ **Country:** Republic of MN **Postal Code:** 1000
System A: Customer Title: President **System B: Customer Title:** CEO	**Field Consolidation**	**Customer Title:** President or CEO
Order Date: 05/08/1998 **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	**Reduplication**	**Customer Name:** Jone William Smith

Figure 1.4 Data transformations methods

1.8 Data Warehouse Development Life Cycles

As discussed in [Mat1996], the development life cycle of a data warehouse includes three phases, (1) *overall warehouse development*, (2) *infrastructure development*, (3) *applications development*.

1. The overall warehouse development phase

During this phase, all participants must agree on the size, scope, complexity, approach to the construction, and the value of the warehouse project.

2. The infrastructure development phase

During the infrastructure development phase, all of the technical, managerial, and system-support issues should be resolved. The physical infrastructure is built and tested, and is staffed with the appropriate support personnel. After the infrastructure has been stabilized, users will actually begin trying to use the data in the warehouse.

3. The applications development

The last phase in the development of a data warehouse system actually is an iterative one. During this phase, application developers decide on issues such as designing the tables, software modules, and coding of the programs. Each system should meet the specific business needs for the application at hand.

1.9 The Organization of This Report

The rest of this report is organized into four chapters. Chapter 2 discuses the data lineage tracing problem in data warehousing, and introduces the lineage tracing procedure and strategies. We will also illustrate how auxiliary views are useful for data lineage tracing. In Chapter 3, we present our design and implementation of a prototype system for lineage tracing of an inventory application. Chapter 4 will demonstrate the system performance and our experimentation with the system. Finally, concluding remarks and future work are presented in Chapter 5. In Appendix, we will provide some sample codes of the system.

Chapter 2 Lineage Tracing in Data Warehousing

From Chapter 1, we know that a data warehouse is a repository of integrated information from multiple sources. The integrated information is stored as materialized views in the data warehouse. Ideally, the views are kept up-to-date when the sources change. Users can then apply data analysis and mining algorithms based on the data warehouse.

In a warehouse system, the view data come from multiple autonomous sources. In some queries, only the content of warehouse view alone is not sufficient for in-depth analysis of the data information. The collection of data sources that contributed to the view data is also useful for analysis. We can "drill through" from interesting (or potentially erroneous) view data to the original source data that derived the view data for in-depth analysis of 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 as the *data lineage problem* [CW2000].

The primary motivation for supporting data lineage is to enable further analysis of interesting or potentially erroneous view data. This capability is useful in the content of data mining, scientific databases, and network monitoring systems. The lineage tracing algorithms and techniques can also be applied for view update, materialized view schema evolution, and data cleaning [CW2000].

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. However, 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, and how to reduce the cost of data transformation from the distributed sources into the warehouse, etc [CWW1997].

We organize the rest of this chapter as follows. Section 2.1 illustrates the data lineage problem using an example. Section 2.2 explains the lineage tracing procedure using the aggregation (ASPJ views). Section 2.3 discusses using auxiliary materialized views over the source data to help in tracing the lineage of the interesting view data item efficiently. Section 2.4 presents some methods of storing auxiliary views to support tracing the data lineage. Section 2.5 illustrates how auxiliary views can be useful for data lineage tracing in a data warehouse system. Section 2.6 shows some related work about tracing the lineage of view data in a warehousing environment.

2.1 An Illustrative Example

In this section, we use an example to illustrate the idea of data lineage and to show it can be useful. Consider a data warehouse system in a large company, which has three retail branches. Each branch has its own database operating independently, which stores the inventory and sales information. In this example, we use the relational data model to present the tracing procedure, however, the ideas and results could be adapted to object oriented data model as well.

The data warehouse system collects the data from three source databases of retail branches: Source_1, Source_2, and Source_3. These databases have the same structure, for ease of presentation. Each source database consists of the following tables: Products, Categories, Orders, OrderDetails. The structures of these tables are given below, where the underlined attributes in each table denote the key. Tables 2.1 through 2.8 show the database schemas and instances for some of these tables.

Products (<u>ProductID</u>, ProductName, SupplierID, Category, UnitPrice, UnitsInStock)

Categories (<u>CategoryID</u>, CategoryName, Description)

Orders (<u>OrderID</u>, CustomerID, OderDate, RequiredDate, ShippedDate)

OrderDetails (<u>OderID</u>, <u>ProductID</u>, UnitPrice, Quantity, Discount)

Table 2.1 Source_1.Products

Product ID	Product Name	Supplier	Category	Unit Price($)	Units In Stock
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Dairy Products	21.00	11
14	Tofu	Mayumi's	Produce	23.25	35
20	Sir Rodney's Marmalade	Specialty Biscuits, Ltd.	Confections	81.00	40
41	Jack's New England Clam Chowder	New England Seafood Cannery	Seafood	9.65	85
42	Singaporean Hokkien Fried Mee	Leka Trading	Grains/Cereals	14.00	26
51	Manjimup Dried Apples	G'day, Mate	Produce	53.00	20
57	Ravioli Angelo	Pasta Buttini s.r.l.	Grains/Cereals	19.50	36
65	Louisiana Fiery Hot Pepper Sauce	New Orleans Cajun Delights	Condiments	21.05	76
72	Mozzarella di Giovanni	Formaggi Fortini s.r.l.	Dairy Products	34.80	14

Table 2.2 Source_1.Categories

Category ID	Category Name	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

Table 2.3 Source_1.Orders

Order ID	Customer	Order Date	Required Date	Shipped Date
10248	Galería del gastrónomo	04-Jul-2001	01-Aug-2001	16-Jul-2001
10249	Toms Spezialitäten	05-Jul-2001	16-Aug-2001	10-Jul-2001
10250	Hanari Carnes	08-Jul-2001	05-Aug-2001	12-Jul-2001

Table 2.4 Source_1.OrderDetails

Order ID	Product	Unit Price ($)	Quantity	Discount (%)
10248	Mozzarella di Giovanni	34.80	5	0
10248	Queso Cabrales	14.00	12	0
10248	Singaporean Hokkien Fried Mee	9.80	10	0
10249	Manjimup Dried Apples	42.40	40	0
10249	Tofu	18.60	9	0
10250	Jack's New England Clam Chowder	7.70	10	0
10250	Louisiana Fiery Hot Pepper Sauce	16.80	15	15
10250	Sir Rodney's Marmalade	80.00	20	0

Table 2.5 Source_2.Orders

Order ID	Customer	Order Date	Required Date	Shipped Date
10323	Koniglich Essen	07-Oct-2001	04-Nov-2001	14-Oct-2001
10324	Save-a-lot Markets	08-Oct-2001	05-Nov-2001	10-Oct-2001
10325	Koniglich Essen	09-Oct-2001	23-Oct-2001	14-Oct-2001

Table 2.6 Source_2.OrderDetails

Order ID	Product	Unit Price ($)	Quantity	Discount (%)
10323	Tofu	18.60	10	0
10323	Mozzarella di Giovanni	27.80	40	0
10323	Louisiana Fiery Hot Pepper Sauce	16.80	40	0
10324	Jack's New England Clam Chowder	7.70	20	0
10324	Manjimup Dried Apples	42.40	48	20
10324	Singaporean Hokkien Fried Mee	11.20	10	20
10325	Mozzarella di Giovanni	27.80	25	15
10325	Tofu	18.60	9	0
10325	Louisiana Fiery Hot Pepper Sauce	16.80	40	0

Table 2.7 Source_3.Orders

Order ID	Customer	Order Date	Required Date	Shipped Date
10396	Hungry Owl All-Night Grocers	19-Sep-2001	17-Oct-2001	23-Oct-2001
10397	The Big Cheese	20-Sep-2001	18-Oct-2001	27-Sep-2001
10398	Du monde entier	20-Sep-2001	04-Oct-2001	26-Sep-2001

Table 2.8 Source_3.OrderDetails

Order ID	Product	Unit Price ($)	Quantity	Discount (%)
10396	Mozzarella di Giovanni	27.80	21	0
10396	Manjimup Dried Apples	42.40	18	15
10397	Singaporean Hokkien Fried Mee	11.20	40	5
10397	Queso Cabrales	16.80	30	0
10397	Tofu	18.60	12	0
10398	Jack's New England Clam Chowder	7.70	25	20
10398	Tofu	18.60	20	10

Example 1 [Lineage of aggregation view from multiple sources]

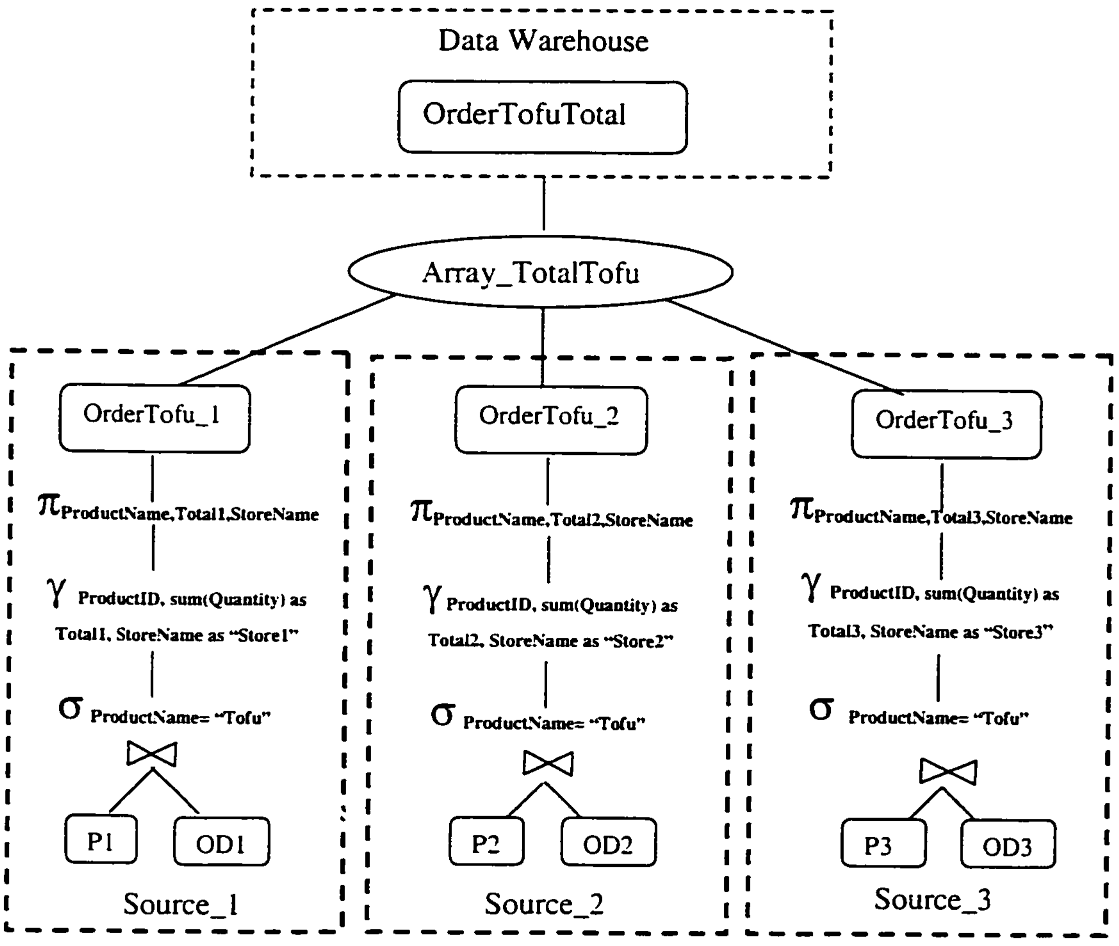
Suppose a user (say an administrator in the head office) needs to check the total order quantity for the product "Tofu" from the three source databases at the branches. Further suppose that the user wants to trace the order information detail for the product "Tofu" from Source_2.

For computing the order quantity for the product "Tofu" from each branch, there are three views: OrderTofu_1, OrderTofu_2, OrderTofu_3, defined over the corresponding source databases. We thus need to formulate and issue the following SQL-like queries to the sources — one to each source, to get the information on the order quantity of the product "Tofu".

```
(Connect to Source_i:  i = 1,2,3)
CREATE VIEW OrderTofu_i AS
SELECT P.ProductName, Sum(OD.Quantity) AS Total, "Store i" AS StoreName
FROM Products P, OrderDetails OD
WHERE P.ProductID = OD.ProductID AND P.ProductName = "Tofu"
GROUP BY P.ProductName;
```

For transferring the source data into the data warehouse, we can use a data array Array_TotalTofu to "collect" the order quantities information of product "Tofu" from sources. In the data warehouse, a table OrderTofuTotal is defined for saving the total order information of the product "Tofu" from the Array_TotalTofu. Figure 2.1 shows the query tree for defining the table OrderTofuTotal, whose intermediate nodes perform relational algebra operations. The root indicates the output and the leaves indicate the input relations. We use the following SQL-like codes to create the table OrderTofuTotal.

```
(Connect to Warehouse)
CREATE TABLE OrderTofuTotal ( ProductName, Total, StoreName)
INSERT INTO OrderTofuTotal VALUES
FROM each line of Array_TotalTofu;
```



Data Warehouse
OrderTofuTotal
Array_TotalTofu
OrderTofu_1
$\pi_{ProductName,Total1,StoreName}$
γ ProductID, sum(Quantity) as Total1, StoreName as "Store1"
σ ProductName= "Tofu"
$\bowtie$
P1
OD1
Source_1
OrderTofu_2
$\pi_{ProductName,Total2,StoreName}$
γ ProductID, sum(Quantity) as Total2, StoreName as "Store2"
σ ProductName= "Tofu"
$\bowtie$
P2
OD2
Source_2
OrderTofu_3
$\pi_{ProductName,Total3,StoreName}$
γ ProductID, sum(Quantity) as Total3, StoreName as "Store3"
σ ProductName= "Tofu"
$\bowtie$
P3
OD3
Source_3

Figure 2.1 Query tree for relational query OrderTofuTotal

Table 2.9 shows OrderTofuTotal, the result of the query: total order information for the product "Tofu" from the branches. If the user wishes to trace the order information detail for the product "Tofu" from Source_2, the data lineage procedure must "connect" to Source_2 again to get the information. From the view OrderTofu_2, the lineage tracing procedure traces back to the source tables, Source_2.Produces and Source_2.OrderDetails. Tables 2.10 and 2.11 show the tracing results about the product "Tofu" in Source_2. Since the data lineage procedure needs to query the source database again to re-compute, it is both time consuming and expensive.

Table 2.9 The table OrderTofuTotal in the data warehouse

ProductName	Total	StoreName
Tofu	9	Store1
Tofu	19	Store2
Tofu	32	Store3

Table 2.10 Source2.Products

Product ID	Product Name	Supplier	Category	Unit Price ($)	Units In Stock
14	Tofu	Mayumi's	Produce	23.25	35

Table 2.11 Source2.OrderDetails

Order ID	Product	Unit Price ($)	Quantity	Discount (%)
10323	Tofu	18.60	10	0
10325	Tofu	18.60	9	0

2.2 The lineage Tracing Procedure

Given a materialized view in a data warehouse, we may want to trace the lineage of selected "interesting" tuples in the view. From Example 1 above, we know that for computing the lineage of a view data item, the view definition, the original source data, and perhaps some auxiliary information are needed. The view definition provides a mapping from the base data to the view data. Given a view data item, the corresponding base data according to the view definition can be traced back. However, determining the inverse mapping—from a view data back to the source data that produced it—is not always as straightforward. To determine the inverse mapping accurately, we not only need the view definition, but also need the base data and some additional information [CWW1997]. In Example 1, the auxiliary views OrderTofu_1, OrderTofu_2, and OrderTofu_3 are used to help in tracing back to the source data.

The lineage of a view tuple is defined as the set of original source tuples that derived the given view tuple. To trace the lineage of a view tuple, we use a predefined sequence of relational queries over the sources, called *tracing queries,* to compute the lineage of the view data [CW1999]. In general, we use relational views with arbitrary use of aggregation, selection, projection, and join operators, called Select-Project-Join views with aggregation (ASPJ views). Firstly, the view definition is transformed into a normal form composed of the sequences aggregate-project-select-join, called ASPJ segments. The lineage of tuples in a view defined by a single ASPJ segment can be computed using 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, an intermediate view for each segment are defined, and recursively traced through the hierarchy of intermediate views in a top-down manner. At each level, the tracing queries are used 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 [CW2000+].

2.3 Auxiliary Materialized Views in Data Warehouse

In the data warehouse, we can store the auxiliary materialized views over the source data to help in tracing the lineage of interesting view data item efficiently. Usually, there are two approaches for tracing the lineage of data in a view. Firstly, re-compute the relevant portion of the aggregation when tracing a tuple's lineage. It needs to access the sources again and need more computation time. This approach, however, requires no extra storage or maintenance cost. In Example 1, we used this approach to trace a view data in a data warehouse system. Secondly, define and maintain some auxiliary materialized views over the data item specifically for lineage tracing in the data warehouse. These auxiliary materialized views contain the intermediate results from source tables about the specific view item. It can reduce the computation cost, and reduce or avoid expensive source accesses, but the auxiliary views must be stored and kept up-to-date.

Another reason for using the auxiliary materialized views for tracing the lineage of data, in a distributed multi-source data warehousing system, is that querying the sources for tracing the lineage information can be difficult or impossible: sources may be inaccessible sometime, expensive to access and transfer data to the warehouse, or inconsistent with respect to the views data in the warehouse. So storing additional auxiliary views in the warehouse can reduce or eliminate source access when tracing the lineage of the specific view data.

Therefore, using auxiliary views in the warehouse to answer queries over the source data and to trace the lineage of the data provides an integrated and efficient mechanism. In general, the more auxiliary views are materialized and stored in the warehouse, the more efficient the tracing and maintenance of the data it would be in the specific view. However, it also increases the warehouse storage and maintenance costs, since auxiliary views themselves also need to be maintained.

2.4 Methods of Using Auxiliary Views for Lineage Tracing

As noted in the above section, it may be advantageous to materialize certain auxiliary views in a data warehouse to improve lineage tracing. Using the auxiliary views for storing the data form the source tables in the warehouse can reduce or avoid computations and expensive source queries, hence resulting in efficient and consistent lineage tracing. There are many possible auxiliary views to materialize for lineage tracing with different performance tradeoffs in different settings. Here we just discuss four methods for storing auxiliary views to support tracing the data lineage. For details, interested readers are referred to [CW2000].

The user views and the auxiliary views need to be maintained in response to changes to data in source base tables. Maintaining a warehouse view requires access to the data that is not available in the view itself [QGMW1996]. If the data in a warehouse view can be traced correctly and can be maintained efficiently without querying the sources, the view is called as *self-traceable* and *self-maintainable*. In cases where the sources are inaccessible or inconsistent with the warehouse views, the user view as well as the auxiliary views must be both self-traceable and self-maintainable [CW2000].

Example 2 [Illustrate the methods of storing auxiliary views for lineage tracing]

For illustrating the methods of using auxiliary views for lineage tracing, we use a simple SPJ view **NotEnoughOrder** defined in Figure 2.2, where Products and OrderDetails are the sample source tables of database Source_1 introduced in Section 2.1. The view **NotEnoughOrder** contains all orders that the stock of product is less than the quantity of sales order for this product, including the OrderID, ProductName, Quantity, UnitsInStock. Table 2.12 shows the view contents over the sample source data.

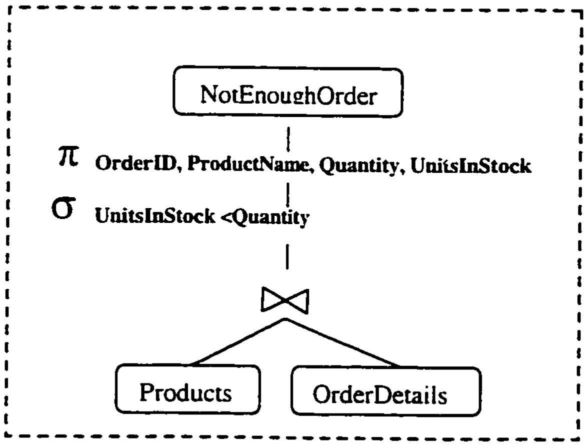


Figure 2.2 View definition for NotEnoughOrder

Table 2.12 Contents of view NotEnoughOrder

Order ID	Product	Quantity	Units In Stock
10248	Queso Cabrales	12	11
10249	Manjimup Dried Apples	40	20

2.4.1 Storing Base Tables (BT)

An easy way to make a view self-traceable is to store in the warehouse a copy of each source table on which the view is defined (after local selection), and issue the tracing queries to the local copies instead of issuing them to the source tables. Storing base tables can improve the lineage tracing, however, base tables could be large, even after applying local selections, and hence much of the source data may be irrelevant to the lineage of any view tuple. For instance, in Example 2, base tables for view NotEnoughOrder are simply copies of the tables Products and OrderDetails tables in the of database Source_1.

2.4.2 Storing Partial Base Tables (PBT)

To reduce the size of the base tables, we can store the semi-join of each source table with the user specified view. For views with selective join conditions, the PBT scheme replicates much less source data than the BT scheme. This 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 smaller table. This scheme is useful for view self-traceability, but not for its self-maintainability. Since the contents of partial base tables are based on the user view's contents and changes, they are not helpful to maintain the user view. So the user view needs to be maintained first in response to the changes of the base data in the sources. For instance, in Example 2, Tables 2.13 and 2.14 show the contents of the partial base table for the view NotEnoughOrder.

Table 2.13 PBT_Products

Product ID	Product Name	Supplier	Category	Unit Price ($)	Units In Stock
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Dairy Products	21.00	11
51	Manjimup Dried Apples	G'day, Mate	Produce	53.00	20

Table 2.14 PBT_OrderDetails

Order ID	Product	Unit Price ($)	Quantity	Discount (%)
10248	Mozzarella di Giovanni	34.80	5	0
10248	Queso Cabrales	14.00	12	0
10248	Singaporean Hokkien Fried Mee	9.80	10	0
10249	Manjimup Dried Apples	42.40	40	0
10249	Tofu	18.60	9	0

2.4.3 Storing Base Table Projections (BTP)

When source tables have known keys, we can store in our auxiliary views key attributes from the source tables together with other necessary attributes, called the *base table projections* (BTP). This scheme improves tracing query performance (over storing nothing) while reducing view maintenance and storage costs (over storing full source replicas). During lineage tracing, since the stored information identifies which source tuples really contribute to a given view tuple, the detailed source information can be fetched from the source, using the key information. The BTP scheme may not satisfy the self-traceability, if we need to trace more detail source data through querying the sources. However, the maintenance of the user view is easy, as we can store the key attributes together with all other necessary attributes from the source tables in our auxiliary views. For instance, in Example 2, Tables 2.15 and 2.16 show the contents of the base table projections for view NotEnoughOrder.

Table 2.15 BTP_Products

Product ID	Product Name	Units In Stock
11	Queso Cabrales	11
14	Tofu	35
20	Sir Rodney's Marmalade	40
41	Jack's New England Clam Chowder	85
42	Singaporean Hokkien Fried Mee	26
51	Manjimup Dried Apples	20
57	Ravioli Angelo	36
65	Louisiana Fiery Hot Pepper Sauce	76
72	Mozzarella di Giovanni	14

Table 2.16 BTP_OrderDetails

Order ID	Product	Quantity
10248	Mozzarella di Giovanni	5
10248	Queso Cabrales	12
10248	Singaporean Hokkien Fried Mee	10
10249	Manjimup Dried Apples	40
10249	Tofu	9
10250	Jack's New England Clam Chowder	10
10250	Louisiana Fiery Hot Pepper Sauce	15
10250	Sir Rodney's Marmalade	20

2.4.4 Storing Lineage View Projections (LVP)

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* (LVP). Compared to the BTP scheme, the LVP scheme further simplifies the tracing query and improves tracing performance. However, the maintenance cost for the lineage view projection is higher than that of the base table projections. Since the LVP view includes only base table keys and user view attributes, at the last step of the tracing process requires a source query for the detail source data. So this scheme does not satisfy the view self-traceability or the view self-maintainability. Table 2.17 shows the content of the lineage view projection for the view NotEnoughOrder.

Table 2.17 Lineage view projection (LVP)

Product ID	Order ID	Product	Quantity	Units In Stock
11	10248	Queso Cabrales	12	11
51	10249	Manjimup Dried Apples	40	20

2.5 Using Auxiliary Views for Lineage Tracing

In this section, we provide another example that shows how auxiliary views can be useful for data lineage tracing in data warehouse systems. In this example, the data warehouse system and source databases are the same as in Example 1.

Example 3 [Using Auxiliary Views for Lineage Tracing in data warehousing]

As in Example 1, suppose a user (say an administrator in the head office) needs to check the total order quantity for the product "Tofu" from the three source databases at its branches, and also trace details of the order information for the product "Tofu" from the database Source_2.

In this example, using auxiliary views over the data item helps tracing the data lineage information, described as follows. First, in our three sources, we define three views: ProductTofu_1, ProductTofu_2, ProductTofu_3 to store details of order information for product "Tofu". Using our SQL-like query language, we formulate and issue the following queries to the source — one to each source, to get the order information of the product "Tofu".

```
(Connect to Source_i:  i = 1,2,3)
CREATE VIEW ProductTofu_i AS
SELECT OD.ProductID, P.ProductName, OD.OrderID, OD.Quantity, "Store i" AS
StoreName
FROM Products P, OrderDetails OD
WHERE P.ProductID = OD.ProductID AND P.ProductName = "Tofu";
```

For transferring the source data into the data warehouse, we use a data array Array_ProductTofu to store the order information of product "Tofu" from our three stores. In the data warehouse, we define the three auxiliary materialized views: ProductTofu1, ProductTofu2, ProductTofu3 to store the order detail information of product "Tofu" from the three sources. We define table OrderTofuTotal for storing the total order information of the product "Tofu". The SQL commands to do this are straightforward and hence omitted.

Figure 2.3 shows the query graphs of lineage tracing using auxiliary views expressed in relational algebra. Table 2.18 shows the auxiliary materialized view ProductTofu2 that stores the order data for the product "Tofu" from Source_2. In this example, we use the method of the base table projections (BTP) to produce auxiliary views ProductTofu2 to support tracing the data lineage.

When tracing the relevant detail information about the product "Tofu" from the Source_2, user can find some detail information in the auxiliary materialized view ProductTofu2. If the information is enough for the user, the data lineage procedure does not need to connect to the source again. If the user needs more details about the product "Tofu", such as supplier name and category, then the data lineage procedure needs to access and use the source again. There is a tradeoff in using the auxiliary materialized views to save some source data for lineage tracing the source data. For example, it needs more space in the warehouse. It also needs to maintain the auxiliary materialized views when the sources are changed.

Table 2.18 The auxiliary materialized view ProductTofu2

Product ID	ProductName	Order ID	Quantity	StoreName
14	Tofu	10323	10	Store 2
14	Tofu	10325	9	Store 2

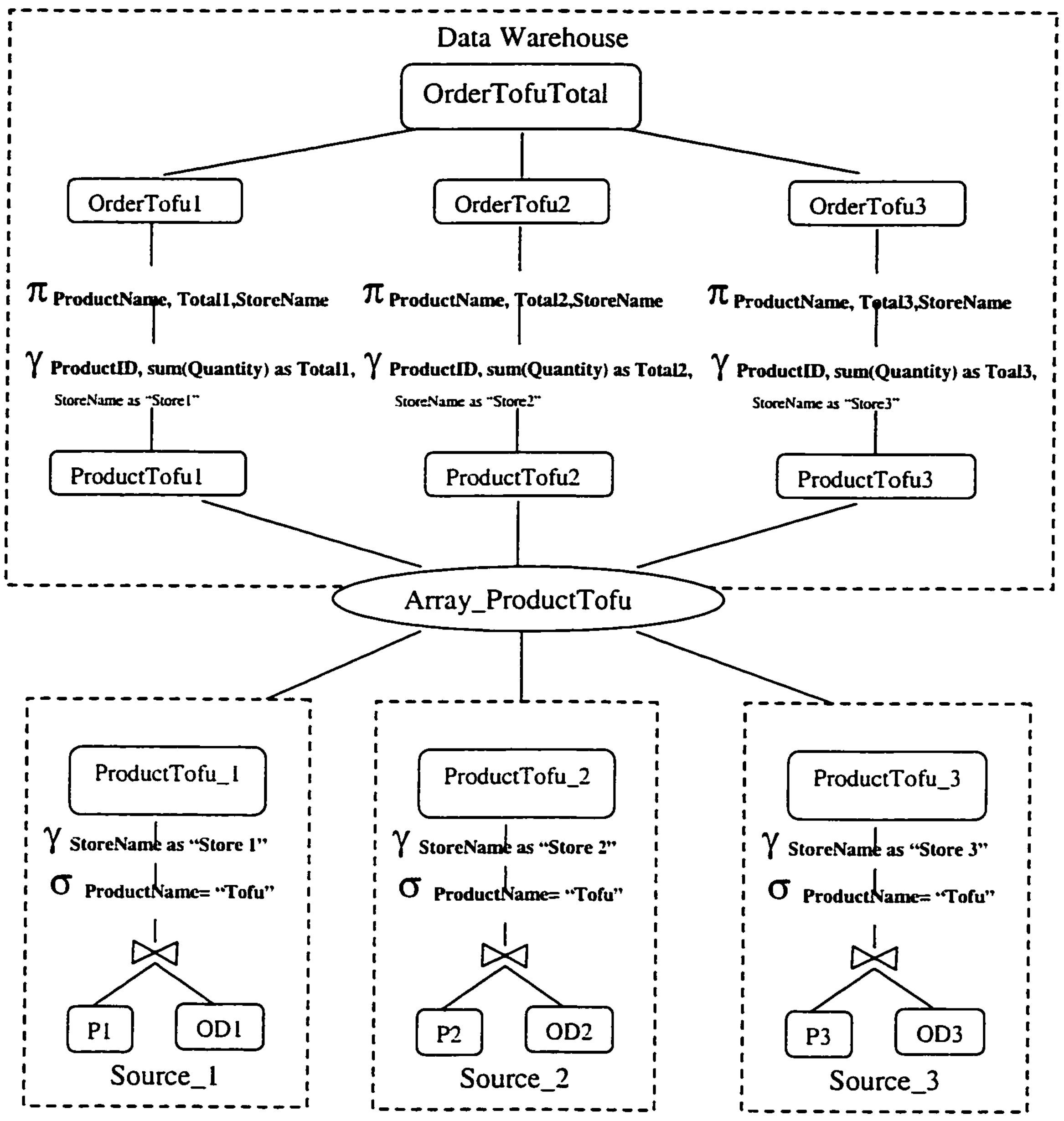


Figure 2.3 Illustration of lineage tracing using auxiliary views

2.6 Related Work

The related works for tracing view data lineage are provided in some papers. [CWW1997] proposes a basic issue about tracing the lineage of view data in a warehousing environment. Storing materialized views over data from one or more sources can provide fast access to the integrated data. However, warehouse views need to be maintained in response to changes to the base data in the sources. Using key and referential integrity constraints to realize self-maintainable —the auxiliary views can be maintained without going to the data sources or replicating all base data [Huy1996,QGMW1996]. In [CWW1999], Y. Cui and J. Widom present an in-depth discussion of storing auxiliary data for efficient maintenance and lineage tracing of complex views, such as investigating the performance of lineage tracing, view maintenance, and storage costs by formulating the view data lineage problem and developing a lineage tracing algorithm for relational ASP views. They have developed a complete prototype that performs efficient and consistent lineage tracing. Their prototype automatically generates lineage tracing procedures and supporting auxiliary views at view definition time, and showing the specific view data derivation process [CWW2000, CWW2000+].

Chapter 3 System Design

To realize and present the lineage tracing method in data warehouse system as introduced in Chapter 2, and display the tracing results, we have designed and implemented a system prototype, called Lineage Tracing Inventory System (LTI), to demonstrate the issues and techniques involved in the development of such system. We developed various graphical user interfaces required to facilitate interacting with the system in order to update the source databases in the LTI system. The design ideas of the LTI system can be used as a basis for a lineage tracing mechanism in a general warehouse setting. It also suggests components that are required in the data warehouse system to support efficient lineage tracing.

This project is an extension of our earlier project in the course COMP 6591 [CLWX2001]. It follows the system architecture in the original version. The current project has more functionalities than the original prototype. The design of the current system is more elaborate, using UML to describe the system and its components, such as the graphic views: Use Case, Class Diagram, and Sequence Diagram. In order to improve the usability and flexibility of the system, we provide more functionalities. For instance, a user can select to trace all the information or part of the information, instructing the system to display the desired information that the user wants. To improve the user friendly aspect of the interface, we provide menu or help button to show the "help" information. In a nutshell, the current version of the system prototype is more flexible, and supports more functionalities, and in more user friendly.

3.1 LTI System Architecture

LTI is a data warehouse system we developed for a typical sales company. The company has three retail stores. Each store has its own database system operating independently, which stores the information about the product name, inventory, sales order, customer name, shipper, and other related information. The data warehouse system collects the data from three source databases at the retail stores: Source 1, Source 2, and Source 3. These databases have the same structure. The user (says an administrator in the head office) can use this system to obtain and verify the total inventory and also order information at branches.

As discussed in Section 1.4, LTI like a basic data warehousing system, includes three components: *the data integration component, the data warehouse, and the query and analysis component.*

1. The data integration component is the back end of the data warehousing system and consists of systems that interface with the operation systems to load data into the data warehouse. As discussed in [MAG1998], there are several steps that need to be done:

- *Data source selection* selects the data from multiple sources that are required for specialized needs.
- *Data extraction* acquires data from the operational systems and stores data in a temporary processing area.
- *Data cleaning* validates and cleans up the extracted data to correct inconsistent. missing, or invalid values.

- *Data transformation* integrates data into standard formats and applies business rules that map data to the warehouse schema.
- *Data loading* loads the cleaned data into the data warehouse.

2. The data warehouse as the storge component of the system, was implemented using a relational database that contains a large number of tables holding the data collected from the data sources.

3. Query and analysis component is the front end of the data warehouse. It contains the access tools and techniques that provide a user with direct, interactive access to the data. The interface developed facilitates interaction with the system and presenting query and lineage tracing results.

3.2 Source Databases and Data Warehouse Design

Data sources include three independent, relational databases with the same schema. Data warehouse in the system is also a relational database; it is set up through creating views and extraction of applicable information from the source databases.

3.2.1 The Design of the Database

Figure 3.1 shows an E/R diagram of a source database.

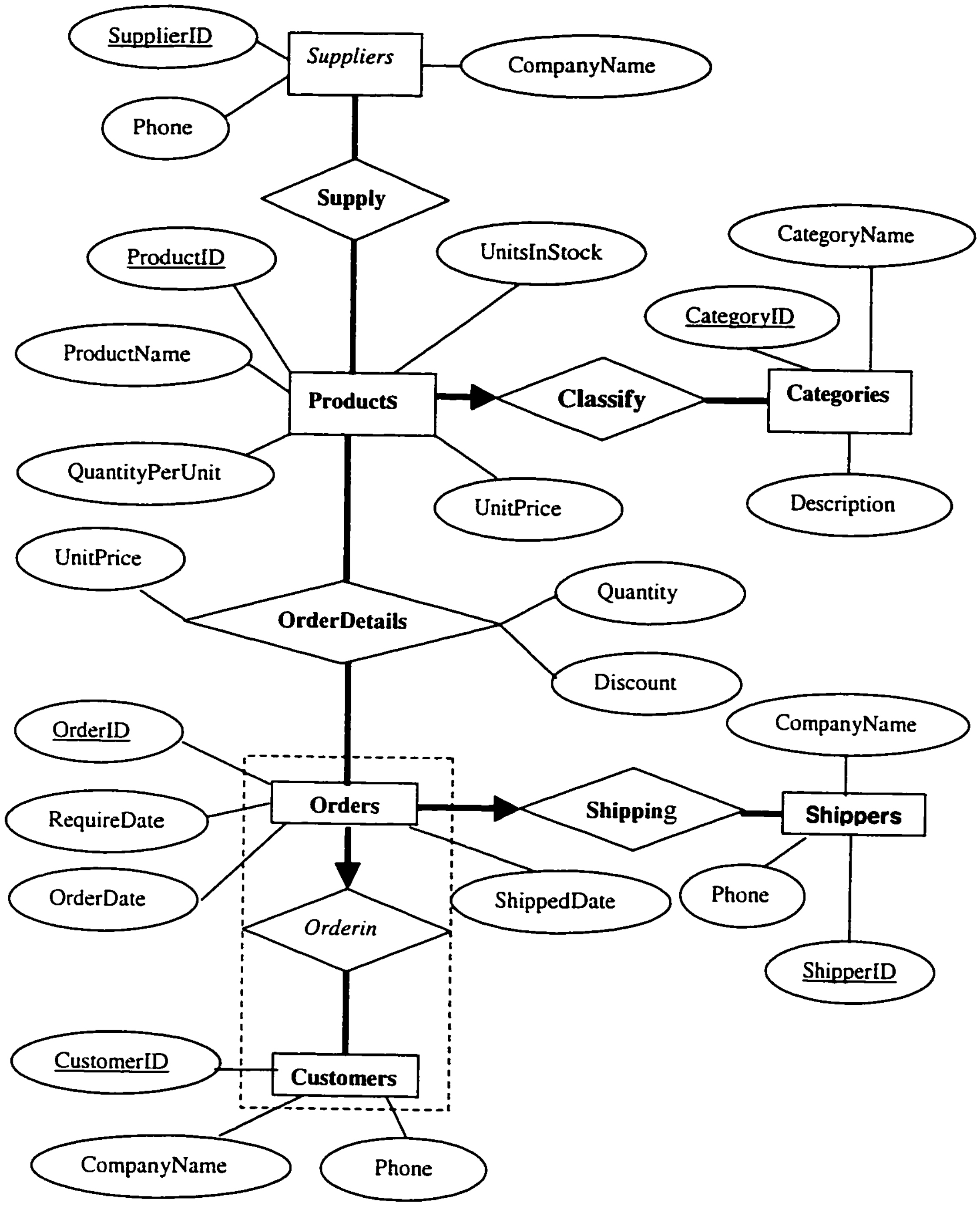


Figure 3.1 The E/R diagram of source databases

3.2.2 Schemas of the Source Databases

The schema of each source database consists of the following tables:

1. Products (ProductID, ProductName, SupplierID, Category, QuantityPerUnit, UnitPrice, UnitsInStock)
2. Categorys (CategoryID, CategoryName, Description)
3. Orders (OrderID, CustomerID, OderDate, RequiredDate, ShippedDate)
4. OrderDetails (OderID, ProductID, UnitPrice, Quantity, Discount)
5. Customers (CustomerID, CompanyName, ContecrName, Address, Phone)
6. Shippers (ShoperID, CompanyName, Phone)
7. Suppliers (SupplierID, CompanyName, ContecrName, Address, Phone)

3.2.3 The Schema of the Data Warehouse

The schema of the data warehouse is as follows:

1. InventoryView (StoreID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder, RecordLevel)
2. OrderView (StoreID, CustomerName, ProductName, UnitsInStock, Quantity, OrderDate, ShippedDate)

4. TotalView(ProductName, TotalInStock, TotalInOrder)

3.3 System Functionalities

The main function of LTI system is tracing the inventory and sales order information from source databases. A subsystem is used to update source databases.

3.3.1 LTI System Functions

- Data warehouse set up and query:
  - ➢ Data warehouse sets up and gets relevant information from source databases.
  - ➢ The user can view total inventory information of three branch's stores.
  - ➢ The user can view total sales order information at branches.

- Data warehouse lineage tracing:
  - ➢ The user can trace the inventory detail information according to the product name, category name, or supplier name.
  - ➢ The user can trace the sales order detail information according to the product name, category name, or customer name.

- Data warehouse maintenance and update:
  - ➢ The system simulates update data process. That is, when source databases are changed, the data in data warehouse are also updated accordingly.
  - ➢ A subsystem is used to update source databases.

3.3.2 Subsystem Functions

- Management and update of source databases:
  - ➢ Update inventory data of the product
  - ➢ Create new sales ordering and order products
  - ➢ Add new customer
  - ➢ Add new product
  - ➢ Add new shipper

3.4 Design of LTI System

To present the design of the LTI system, we used the use case diagrams. It describes the overall functionality of the system — the main functions supported by the system. We then use the class diagrams to show details of the functions and the interrelationships between classes. In order to describe details of the flow of the sequence of events for the use case in the LTI system, we created a sequence diagram. It presents the sequence in the use case that a user selects in tracing the inventory information.

3.4.1 Use Case Diagram

Figure 3.2 illustrates the use case of the LTI system. As show in the figure 3.2, LTI has three main use cases. The user will be able to select (1) tracing inventory information; (2) tracing sales order information; (3) update sources.

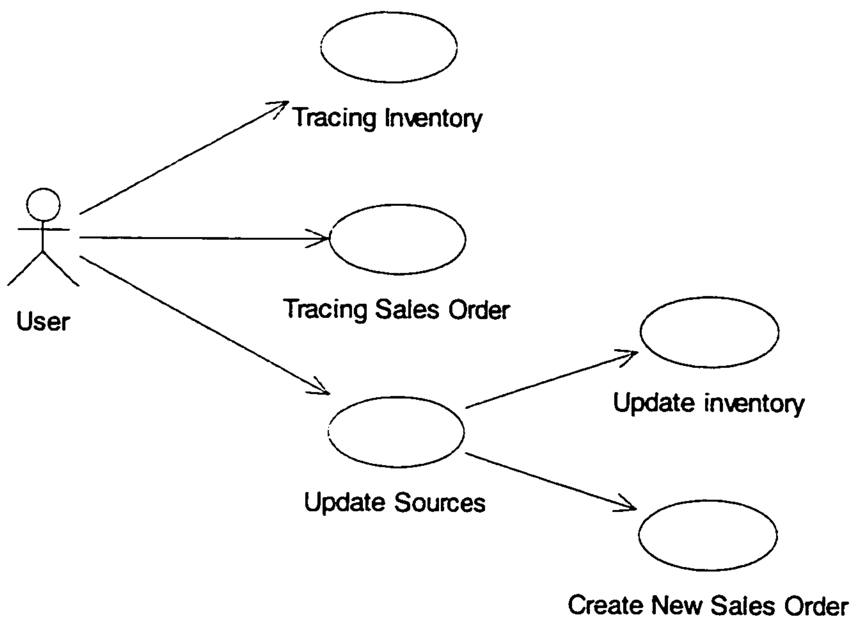


Figure 3.2 LTI system Use Case

As shows in the figure, the subsystem about updating sources has two use cases. The user will be able to select (1) update inventory information and (2) create new sales order.

3.4.2 Class Diagram

Figure 3.3 illustrates the classes used for tracing inventory and sales order information in the LTI system, as well as the relationship between the classes. A brief description of each class is as follows:

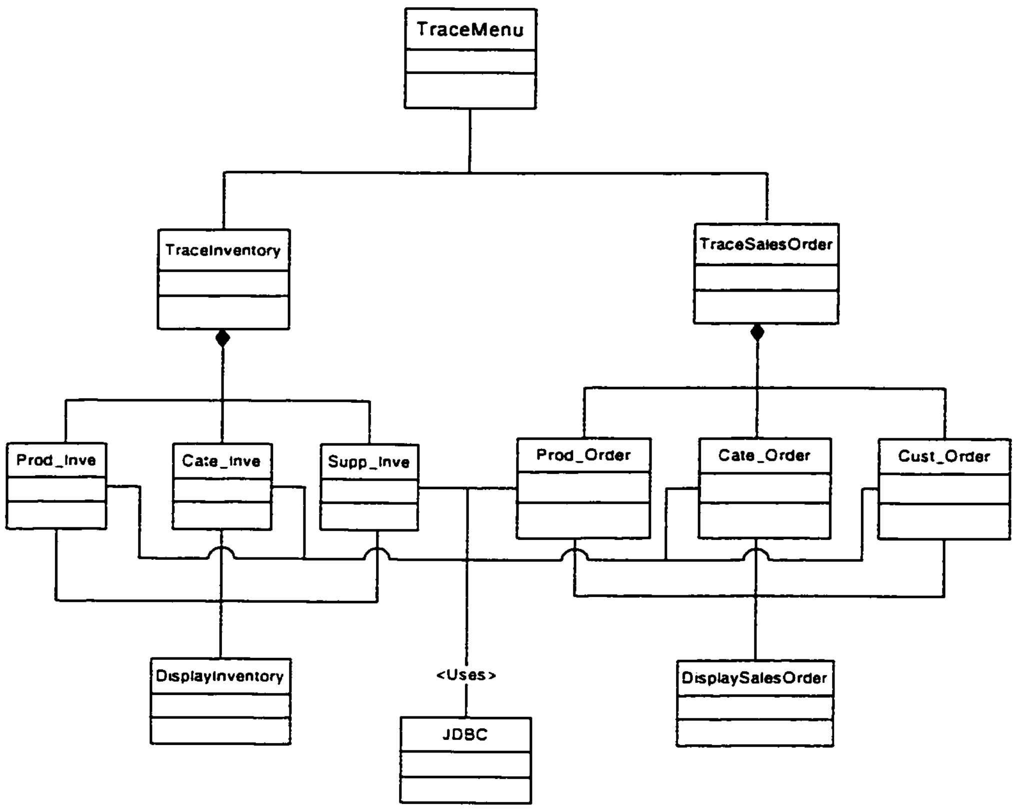


Figure 3.3 LTI system Class Diagram

1. TraceMenu: This class implements the main interface of the LTI system. From the main interface, the user can select one of the two tracing functions to execute: (1) trace inventory information , (2) trace sales order information.
2. TraceInventory: This class implements the interface of tracing inventory information. From the interface, the user can choose to trace the inventory information according to a product name, a category name, or a supplier name.
3. TraceSalesOrder: This class implements the interface of tracing sales order information. From the interface, the user can choose to trace the sales order information according to a product name, or a category name, or a customer name.

4. Prod_Inve, Cate_Inve, Supp_Inve: These classes implemented to realize the functions of tracing the sales order information according to a product name, or a category name, or a supplier name. The user can select to only view the total inventory information or trace the detail inventory information. All three classes need to connect the DB to query data through using the class JDBC.
5. Prod_Order, Cate_Order, Supp_Order: These classes implemented to realize the functions of tracing the inventory information according to a product name, or a category name, or a customer name. The user can select to view only the total sales order information or trace details of sales order information. All these classes need to connect the source databases to the query data.
6. JDBC: This class implements the algorithm to store the data into table.
7. DisplayInventory, DisplaySalesOrder: These two classes are used to display the total information or tracing results about the inventory or sales order.

Figure 3.4 illustrates the classes used for update sources in the subsystem, as well as the relationship between the classes. A brief description of each class is as follows:

1. UpdateMenu: This class implements the main interface of the test subsystem. From the main interface, the user can choose one of source database to update, and then select one of the following functions to execute: Update the inventory information or update the sales order information.

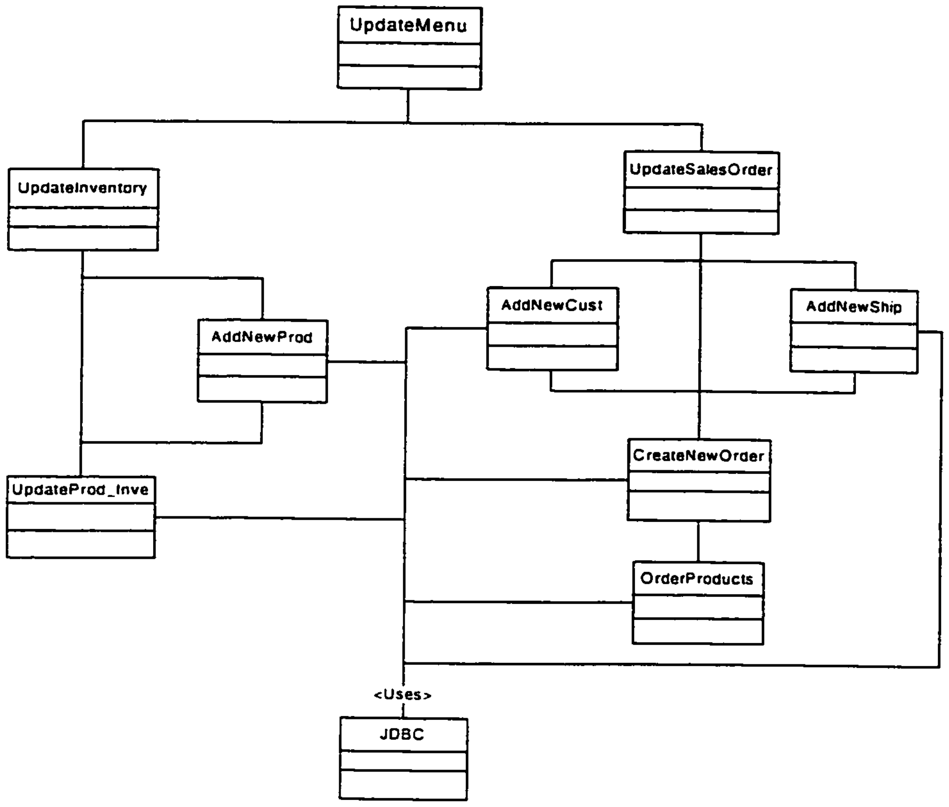


Figure 3.4 Subsystem Class Diagram

2. UpdateInventory: This class implements the interface of update inventory information. From the interface, the user can add a new product in a source database, or update the inventory data for a product that already exists in the source database.
3. UpdateSalesOrder: This class implements the interface of update sales order information. From the interface, the user can create the new sales order using a customer name and shipper name. The user also can add new customer name and new shipper name into source database. Then the user can order the products for this new sales order.

4. AddNewProd, AddNewCust, AddNewShip: These classes implemented to realize to add new product name, customer name, new shipper name into source database. All these classes need to connect to source databases to insert new data into the databases.
5. UpdateInventory: This class is used to update the inventory information through storing the new data entered by the user. It needs to connect to the source to update source databases.
6. CreateNewOrder, OrderProducts: These two classes implements to create the new sales order and order the products for it. They need to connect to the source to update the source database and to insert the new data into source databases.
7. JDBC: This class implements the algorithm to store the data into table.

3.4.3 A Sequence Diagram in LTI System

Figure 3.5 illustrates the flow of the sequence of events that the user selects the use case for tracing product inventory information in LTI.

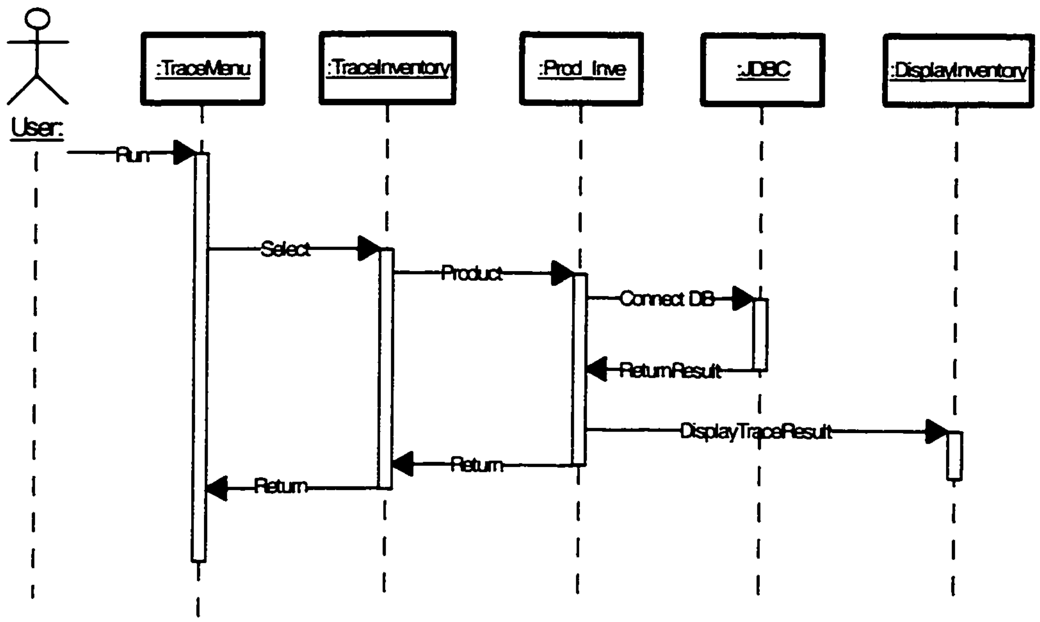


Figure 3.5 A Sequence Diagram in LTI system

The following are details of each of the operation in the sequence diagram:

1. The user executes the LTI system.
2. The user chooses to trace the inventory information from the main interface.
3. The user chooses a product name to trace its inventory information.
4. The class Prod_Inve connects the sources to query details of the inventory data about the product and return the results.
5. Finally, the system displays the tracing results of the information of the product.
6. The user can return to the above interface at step 2 or 3.

Chapter 4 Implementation and Experimentation

In this chapter, we present details of our implementation, including the tools, technologies, and the pseudo code of the algorithms we used to implement the functions step-by-step for tracing inventory and sales order information. We will also present the interaction of the user through the graphical user interface, showing the convenience and ease of interaction with the system developed.

4.1 Implementation Details

4.1.1 Tools

This system includes five main components: user interface, Java data engine, data warehouse, extraction data method, and source databases. The system architecture is shown in Figure 4.1.

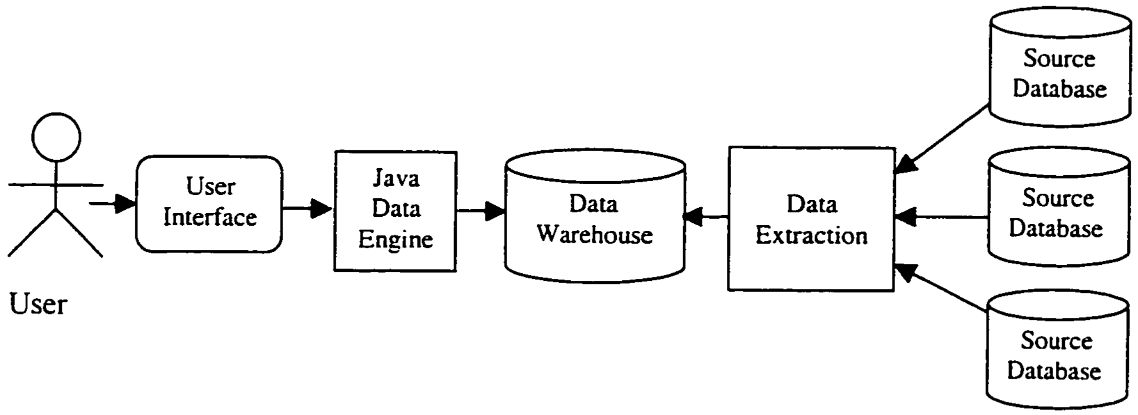


Figure 4.1 The LTI architecture

The user interface of LTI system and test system is implemented by JBuilder 5.0, which provides a Graphic User Interface to query the Data warehouse. JBuilder is a comprehensive group of highly productive tools for creating scalable, high-performance, and platform-independent applications using the Java programming language.

Source databases are developed using Microsoft Access in the Windows 2000 environment. The data warehouse itself is also a relational database, and is implemented in Microsoft Access. The data warehouse is set up through creating views and extracts applicable information from the source databases.

For data extraction and updating the data warehouse, we use the *Bulk extraction method*, described earlier in Section 1.7. The entire data warehouse is refreshed periodically by extracting data from the source. All required data are extracted from the source databases for loading into the warehouse.

JBuilder provides the JDBC API to realize accessing information stored in the databases through using ODBC objects to connect to all source databases. In this implementation, ODBC objects are used to connect to all source databases, and then we use JDBC to connect the ODBC objects and Java Data Engine to transform data from the sources to the data warehouse.

The Java Data Engine serves as a bridge between the user interface and the database. The system also uses JDBC to pose the query against the warehouse. We create the ODBC object to connect the data warehouse, and then connect to the ODBC object and Java data engine using JDBC bridge to transform data from the data warehouse to user interface.

We will next introduce details about our system implementation. There are more than 30 modules used in the implementation. In particular, we will discuss implementation of tracing a product inventory information and sales order information. In Appendix, we will provide some sample codes for the implementation of some modules of LTI system.

4.1.2 Tracing Product Inventory Information

Following is a Java-like pseudo code for implementing the function for tracing product inventory information.

```
Input:  A product name, selected by the user from the list of products and a click to
put a check to show user's intension to trace details of the inventory information
    Var itemname holds the name of the selected product
    Var isTracing is initially set to True

Output: The total inventory information about this product and detailed tracing
result

// Create a jdbc object

jdbc inv = new jdbc();

//Set SQL command to query the selected product inventory information

Query1 = "SELECT P.ProductName, S.CompanyName, P.UnitPrice, "+
          " P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel "+
          " FROM Suppliers S, Products P "+
          " WHERE S.SupplierID = P.SupplierID  AND "+
          " P.ProductName= itemname '";

Call jdbc object to execute Query1.

The jdbc object separately connects to the databases: source1, source2, and
  source3. The data from the source1 marks the store location as "Store 1"; the
  data from the source2 marks the store location as "Store 2; the data from the
  source3 marks the store location as "Store 3".

Connect to the data warehouse, the result of  Query1 is stored as an auxiliary
View1 in the data warehouse.

//Set SQL command to calculate the total inventory information about this
product

Query2 = "SELECT D1.ProductName, "+
          "sum (D1.UnitsInStock) AS TotalStock,"+
          " sum(D1.UnitsOnOrder AS TotalOrder "+
          " FROM "+ View1 +" D1 GROUP BY D1.ProductName; ";

Call jdbc object to execute Query2.

Connect to the data warehouse, the result of Query2 results is stored as an
auxiliary View2 in the data warehouse.
```

```
If (isTracing)
{
        //Set SQL command to search details of the inventory information of this
        product that comes from each store

        Query31= "SELECT D1.ProductName, D1.CompanyName AS" +
                    " SupplierName, D1.UnitPrice, D1.UnitsInStock, " +
                    " D1.UnitsOnOrder " + "FROM " + View1 +" D1 "+
                    "WHERE D1.Store_Location='Store 1';";

        Call jdbc object to execute Query31.

        Connect to the data warehouse, the result of Query31 is stored as an
        auxiliary View31 in the data warehouse.

        Query32= "SELECT D1.ProductName, D1.CompanyName AS" +
                    " SupplierName, D1.UnitPrice, D1.UnitsInStock, " +
                    " D1.UnitsOnOrder " + "FROM " + View1 +" D1 "+
                    "WHERE D1.Store_Location='Store 2';";

        Call jdbc object to execute Query32.

        Connect to the data warehouse, the result of Query32 is stored as an
        auxiliary View32 in the data warehouse.

        Query33= "SELECT D1.ProductName, D1.CompanyName AS" +
                    " SupplierName, D1.UnitPrice, D1.UnitsInStock, " +
                    " D1.UnitsOnOrder " + "FROM " + View1 +" D1 "+
                    "WHERE D1.Store_Location='Store 3';";

        Call jdbc object to execute Query33.

        Connect to the data warehouse, the result of Query33 is stored as an
        auxiliary View33 in the data warehouse.
}

//Declare the object DisplayInventory

DisplayInventory DisplayInve = DisplayInventory(ViewName);
```

4.1.3 Implementation of Tracing Product Sales Order Information

Following is the Java-like pseudo code for implementing the function of tracing product sales order information.

```
Input: A product name, selected by the user from the list of products and a click to
put a check to show user's intension to trace details of the sales order information
    Var itemname holds the name of the selected product
    Var isTracing is initially set to True

Output: The total sales order information about this product and detail tracing result

// Create a jdbc object

jdbc inv = new jdbc();

//Set SQL command to query the selected product sales order information

Query4 = "SELECT P.ProductName, CM.CompanyName, OD.Quantity ,"+
          " P.UnitsInStock, OD.UnitPrice, CG.CategoryName, "+
          "O.OrderDate, O.ShippedDate FROM "+
          "Products P, Customers CM, OrderDetails OD, Order O, Categories CG"+
          "WHERE CM.CustomerID=O.CustomerID AND"+
          " CG.CategoryID=P.CategoryID AND P.ProductID="+
          "OD.ProductID AND O.OrderID=OD.OrderID AND "+
          "P.ProductName='";itemname "';

Call jdbc object to execute Query4.

The jdbc object separately connects to the databases: source1, source2, and source3.
  The data from source1 marks the store location as "Store 1"; the data from the
  source2 marks the store location as "Store 2; the data from the source3 marks the
  store location as "Store 3".

Connect the data warehouse, the result of Query4 is stored as an auxiliary View4 in
the data warehouse.

//Set SQL command to calculate the total sales order information about this product

Query5 = "SELECT D1.ProductName, sum(D1.Quantity) AS TotalSalesOrder, "+
          " avg(D1.UnitPrice) AS AverageSalesPrice" +
          " FROM " +View4 +" D1 GROUP BY D1.ProductName; ";

Call jdbc object to execute Query5.
```

```
Connect to the data warehouse, the result of Query5 is stored as an auxiliary
View5 in the data warehouse.

If (isTracing)
{
        //Set SQL command to search the details of the sales order information of

        this product that comes from each store


        Query61= "select D1.ProductName, D1.CompanyName as " +
                  " CustomerName, D1.Quantity, D1.UnitPrice, " +
                  "D1.OrderDate "+"FROM "+ View4 +" D1" +
                  "WHERE D1.Store_Location='Store 1';";

        Call jdbc object to execute Query61.

        Connect the data warehouse, the result of Query61 is stored as an auxiliary
        View61 in the data warehouse.

        Query62= "select D1.ProductName, D1.CompanyName as " +
                  " CustomerName, D1.Quantity, D1.UnitPrice, " +
                  "D1.OrderDate "+"FROM "+ View4 +" D1" +
                  "WHERE D1.Store_Location='Store 2';";

        Call jdbc object to execute Query62.

        Connect to the data warehouse, the result of Query62 is stored as an
        auxiliary View62 in the data warehouse.

        Query63= "select D1.ProductName, D1.CompanyName as " +
                  " CustomerName, D1.Quantity, D1.UnitPrice, " +
                  "D1.OrderDate "+"FROM "+ View4 +" D1" +
                  "WHERE D1.Store_Location='Store 3';";

        Call jdbc object to execute Query63.

        Connect the data warehouse, the result of Query63 is stored as an auxiliary
        View63 in the data warehouse.
}
//Declare the object DisplaySalesOrder

DisplaySalesOrder DisplayOrder = DisplaySalesOrder(ViewName);

Call object DisplayOrder to display the tracing result.
```

4.2 The User Interface of LTI

4.2.1 Main Interface of our LTI System

Figure 4.2 illustrates the main interface of the LTI system. There are three subtasks supported there: Tracing inventory information, tracing sales order information and updating sources.

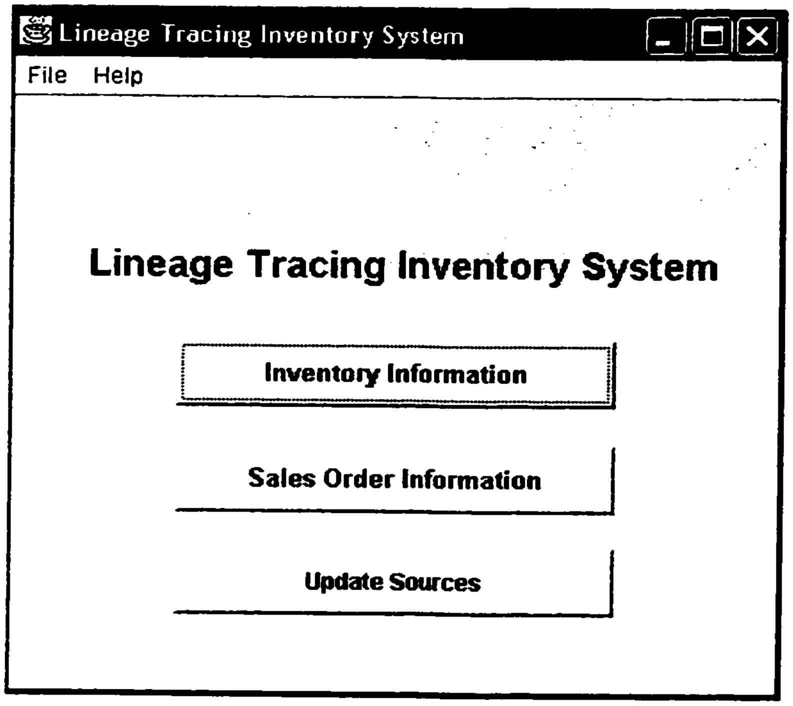


Figure 4.2 Main interface of LTI system

4.2.2 Trace the Inventory Information

1. To trace the inventory information, the user has to click the button "Inventory Information" in the main interface. This pops up the "Tracing Inventory Information" interface, shown in Figure 4.3. The user can choose to trace inventory information

according to product name, category name, or supplier name, and the user can select to trace inventory information of all the stores or each store.

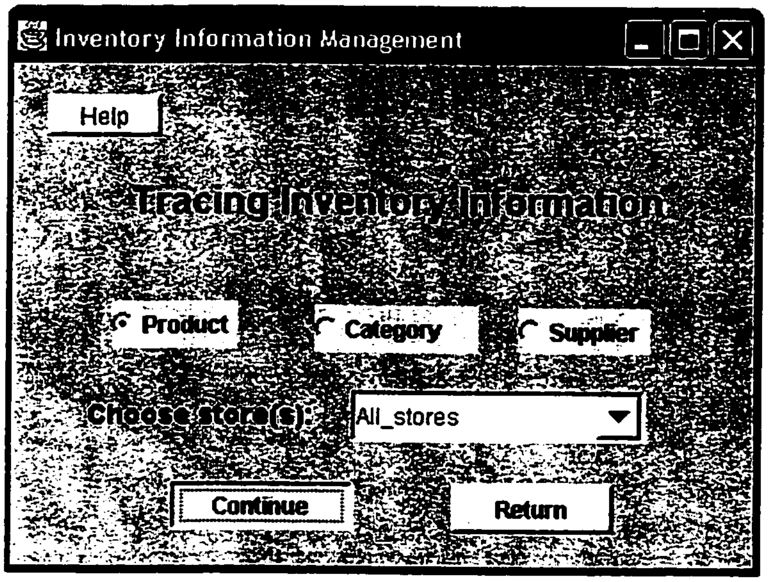


Figure 4.3 Interface of tracing inventory information

2. Choose to trace the inventory information of all the stores according to product name, then click the button "Continue", the "Stock Products information (Products)" interface pops up, shown in Figure 4.4. The user can choose a product name from the combo box, and click the button "Search". It causes a frame to be popped up, displaying only the total inventory of the chosen product from the three branches (Figure 4.5). If the user puts a check on the "tracing" box, and clicks the button "Search", the popped frame displays details of the inventory information of the chosen product from each store (Figure 4.6). Here, the similarity method is used for tracing inventory information according to category name or supplier name.

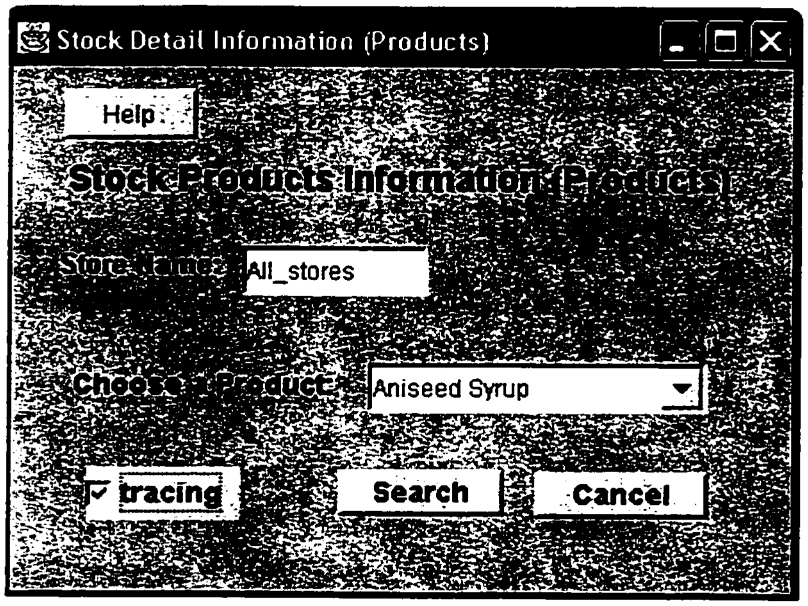
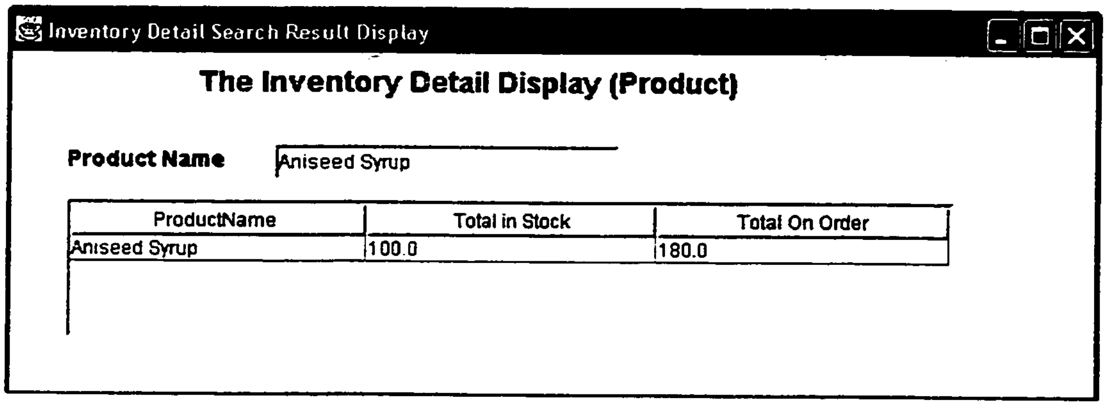


Figure 4.4 Interface of tracing chosen product inventory

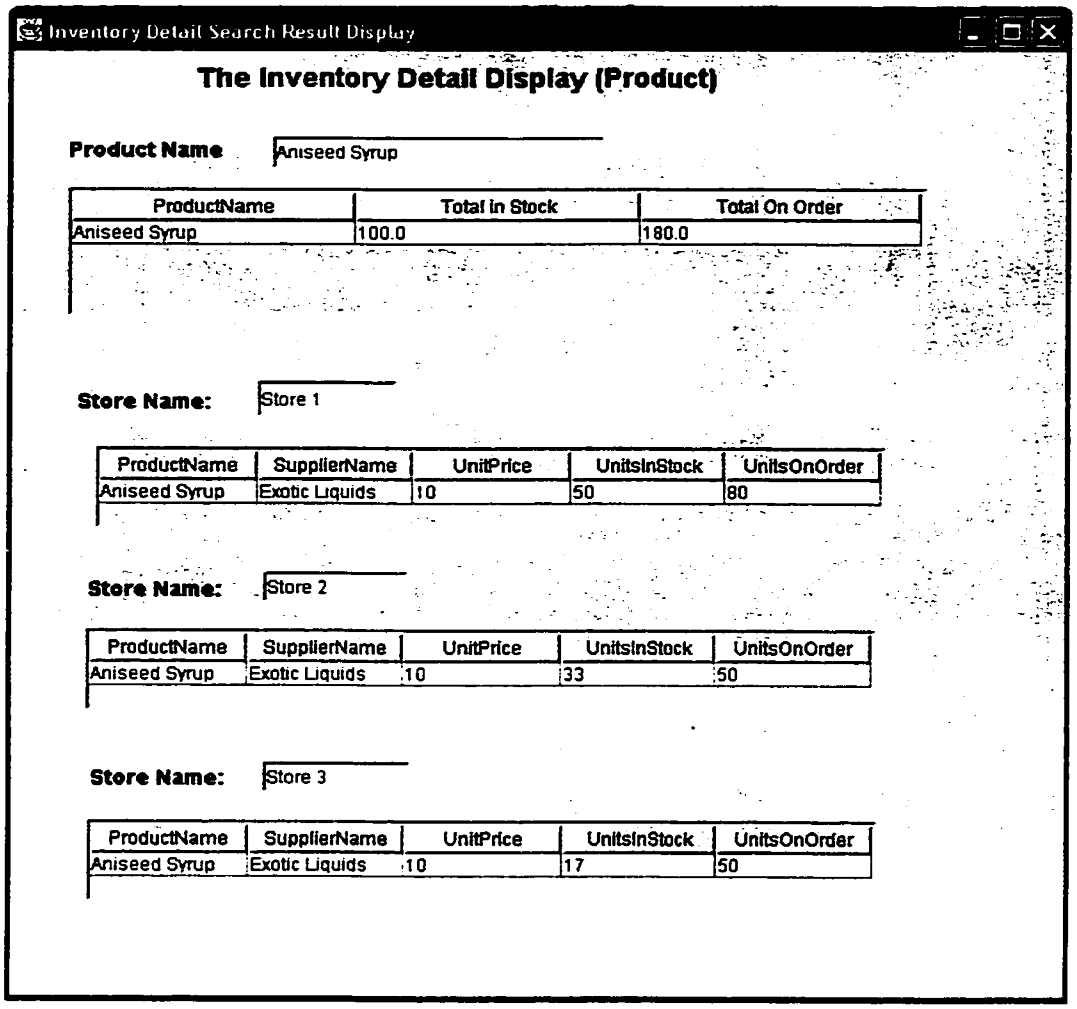


The Inventory Detail Display (Product)

Product Name: Aniseed Syrup

ProductName	Total in Stock	Total On Order
Aniseed Syrup	100.0	180.0

Figure 4.5 Total inventory information of the chosen product in three branches



The Inventory Detail Display (Product)

Product Name: Aniseed Syrup

ProductName	Total In Stock	Total On Order
Aniseed Syrup	100.0	180.0

Store Name: Store 1

ProductName	SupplierName	UnitPrice	UnitsInStock	UnitsOnOrder
Aniseed Syrup	Exotic Liquids	10	50	80

Store Name: Store 2

ProductName	SupplierName	UnitPrice	UnitsInStock	UnitsOnOrder
Aniseed Syrup	Exotic Liquids	10	33	50

Store Name: Store 3

ProductName	SupplierName	UnitPrice	UnitsInStock	UnitsOnOrder
Aniseed Syrup	Exotic Liquids	10	17	50

Figure 4.6 Detail inventory information of the chosen product in three branches

4.2.3 Trace the Sales Order Information

1. In the main interface, clicks the button "Sales order information", the interface "Sales Order Information" pops up (Figure 4.7), the user can choose to trace sales order information according to product name, category name, or customer name, and the user can select to trace inventory information of all the stores or each store.

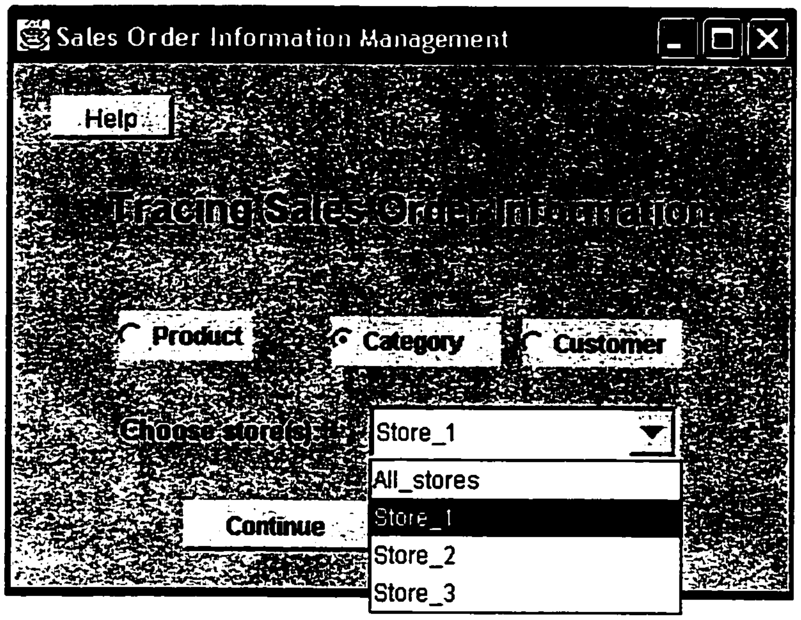
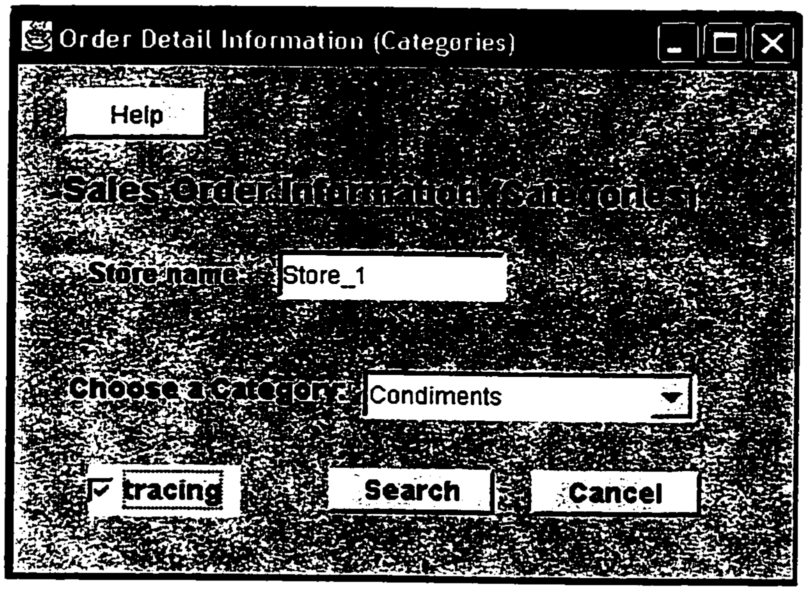


Figure 4.7 Interface of tracing sales order information

2. Choose to trace the sales order information according to a category name, and select to trace sales order information of store 1, then clicking the button "Continue", the interface "Sales Order information (Categories)" pops up (Figure 4.8). The user can choose a category name of products from the combo box, and checks the "tracing" box, and then clicks the button "Search", then pops up a frame which displays the total sales order information and detail tracing information of the chosen category of products in store 1 (Figure 4.9). If the user selects to tracing detailed sales order information of the chosen category of products in all the stores, then pops up a frame which displays the total sales order information and detail tracing information of the chosen category of products in all the stores (Figure 4.10). Here, too, the similarity method is used for tracing sales order information according to product name or customer name.


Order Detail Information (Categories)

Help

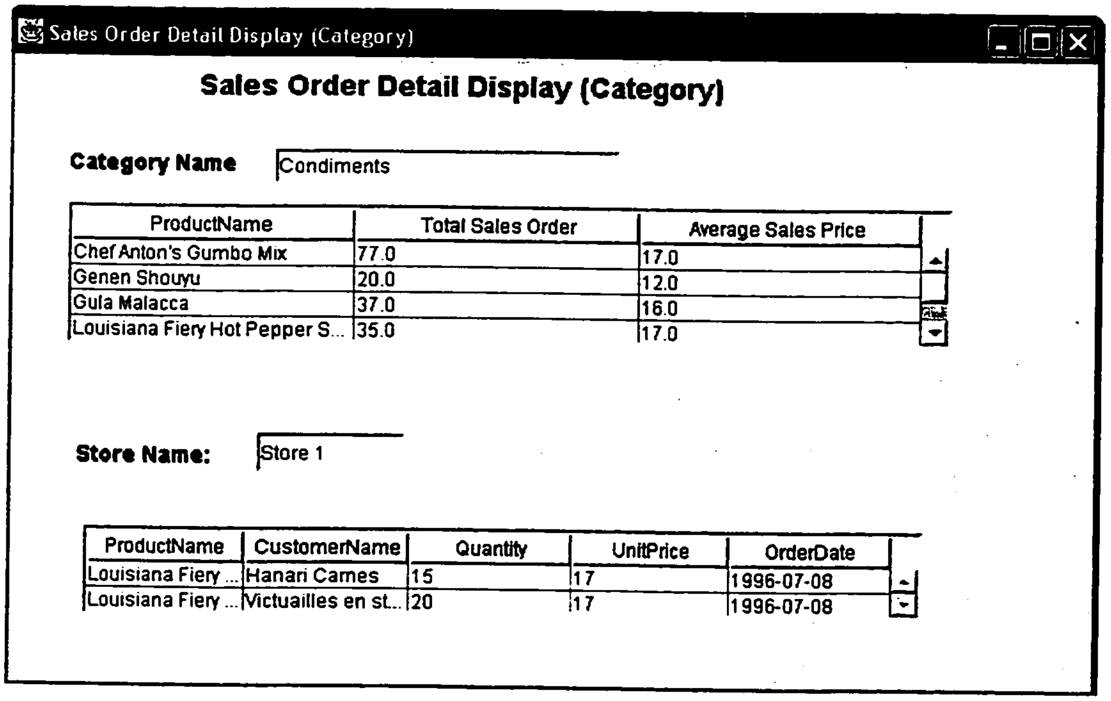
Sales Order Information (Categories)

Store name: Store_1

Choose a Category: Condiments

[✓ tracing] Search Cancel

Figure 4.8 Interface of tracing sales order of the chosen category products


Sales Order Detail Display (Category)

Sales Order Detail Display (Category)

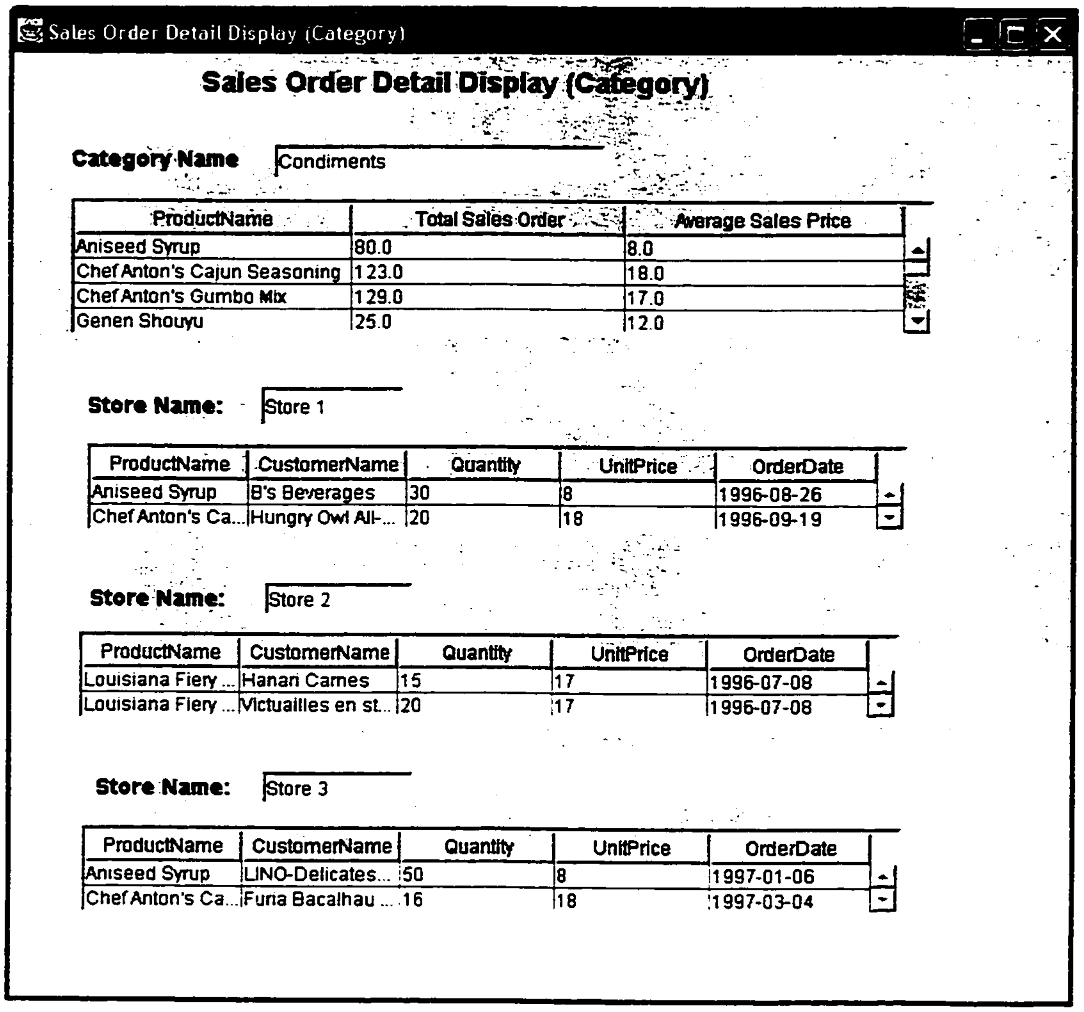
Category Name Condiments

ProductName	Total Sales Order	Average Sales Price
Chef Anton's Gumbo Mix	77.0	17.0
Genen Shouyu	20.0	12.0
Gula Malacca	37.0	16.0
Louisiana Fiery Hot Pepper S...	35.0	17.0

Store Name: Store 1

ProductName	CustomerName	Quantity	UnitPrice	OrderDate
Louisiana Fiery ...	Hanari Carnes	15	17	1996-07-08
Louisiana Fiery ...	Victuailles en st...	20	17	1996-07-08

Figure 4.9 Sales order information of the chosen category products in store 1



Sales Order Detail Display (Category)

Sales Order Detail Display (Category)

Category Name Condiments

ProductName	Total Sales Order	Average Sales Price
Aniseed Syrup	80.0	8.0
Chef Anton's Cajun Seasoning	123.0	18.0
Chef Anton's Gumbo Mix	129.0	17.0
Genen Shouyu	25.0	12.0

Store Name: Store 1

ProductName	CustomerName	Quantity	UnitPrice	OrderDate
Aniseed Syrup	B's Beverages	30	8	1996-08-26
Chef Anton's Ca...	Hungry Owl All-...	20	18	1996-09-19

Store Name: Store 2

ProductName	CustomerName	Quantity	UnitPrice	OrderDate
Louisiana Fiery ...	Hanari Carnes	15	17	1996-07-08
Louisiana Fiery ...	Victuailles en st...	20	17	1996-07-08

Store Name: Store 3

ProductName	CustomerName	Quantity	UnitPrice	OrderDate
Aniseed Syrup	LINO-Delicates...	50	8	1997-01-06
Chef Anton's Ca...	Furia Bacalhau ...	16	18	1997-03-04

Figure 4.10 Sales order information of the chosen category products in all stores

4.3 Interacting with the System

To experiment with LTI, we developed a subsystem to change the underlying source databases. In this section, we demonstrate this capability and the relevant interfaces. In the main interface, if the user clicks the button "Update sources", the interface "Local Inventory Management" pops up (Figure 4.11).

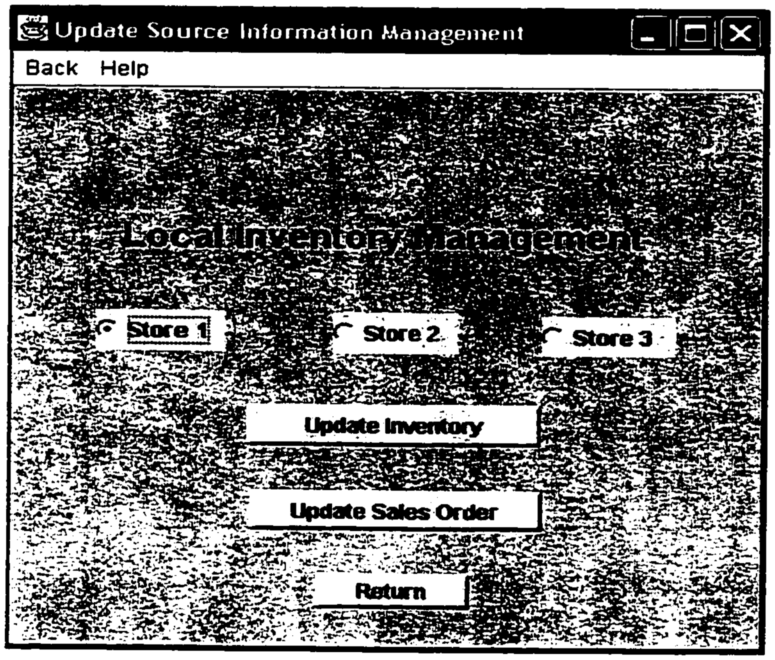


Figure 4.11 Main interface of updating the sources

4.3.1 Update Inventory Information

1. In the main interface of updating the sources, the user can select a branch store to update the database, and then click the button "Update Inventory", then pops up the interface "Local Inventory Information Management" (Figure 4.12). The user chooses a product name, and then clicks the button "Update Stock" to update

inventory data about this product, clicks the button "New Product" to add new product into inventory.

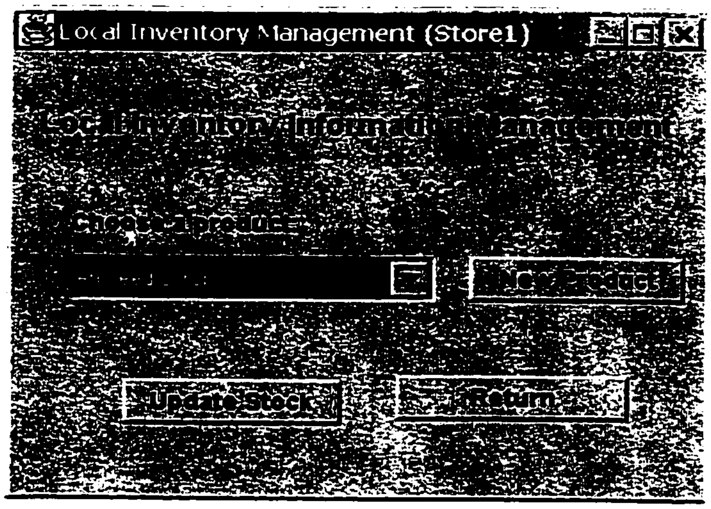
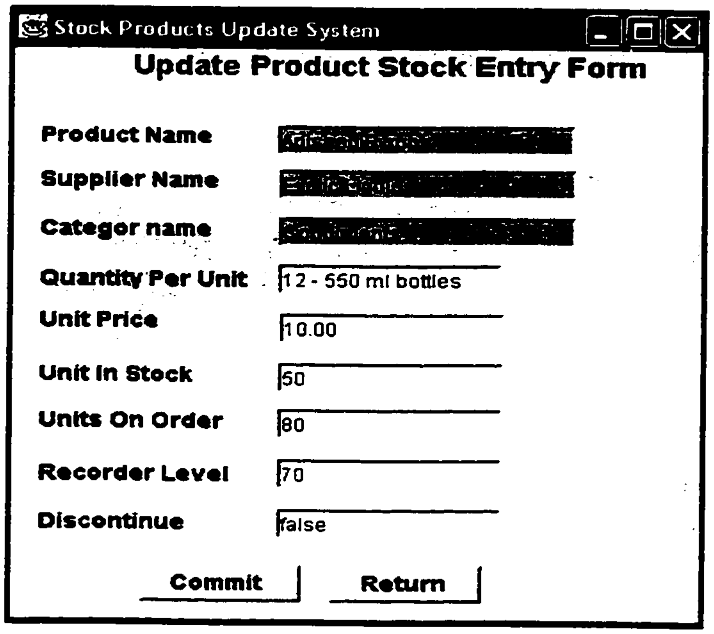


Figure 4.12 Interface of updating inventory information

3. If the user chooses to update the stock information of a product, the interface "Update Product Stock Entry Form" pops up (Figure 4.13). The user can update the inventory data on this product through this interface.

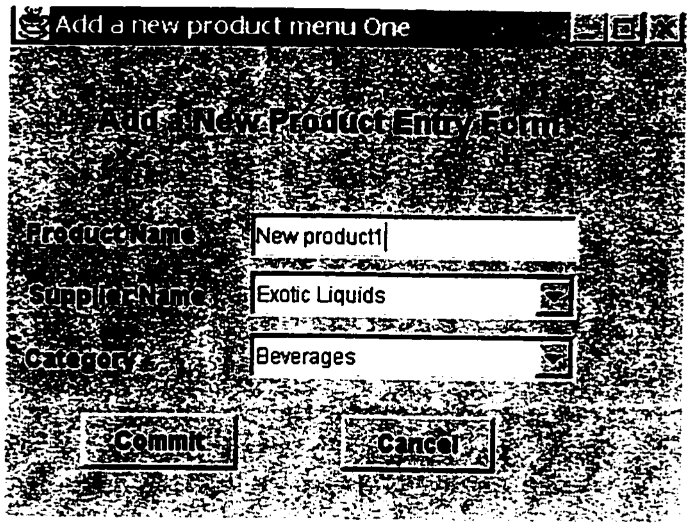


Update Product Stock Entry Form

Product Name	[illegible]
Supplier Name	[illegible]
Categor name	[illegible]
Quantity Per Unit	12 - 550 ml bottles
Unit Price	10.00
Unit In Stock	50
Units On Order	80
Recorder Level	70
Discontinue	false

Figure 4.13 Entry form of updating product stock

3. If the user chooses to add a new product into inventory, then the interface "Update Product Stock Entry Form" pops up (Figure 4.14). The user can add a new product into inventory, then use the update product stock entry form to update the inventory information about the new product.

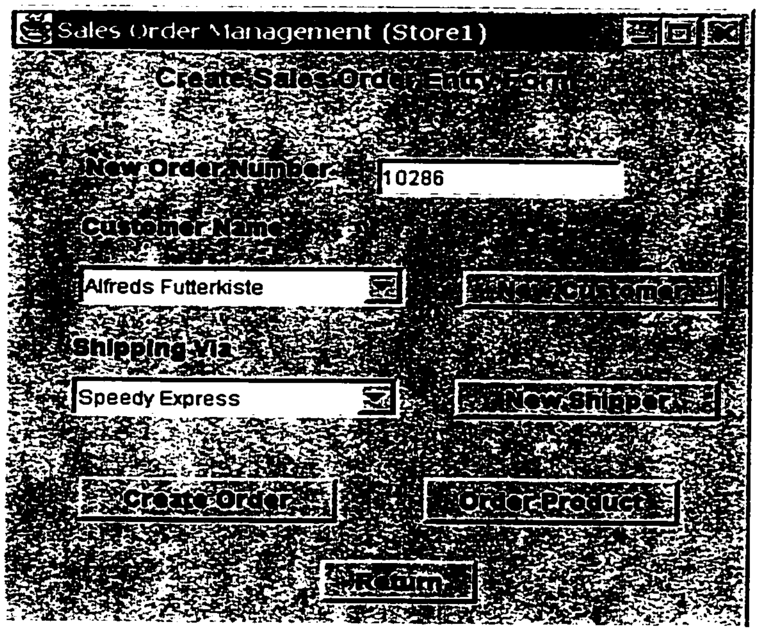


Product Name: New product1
Supplier Name: Exotic Liquids
Category: Beverages

Figure 4.14 Entry form of adding a new product

4.3.3 Update Sales Order Information

In the main interface, if we click the button "New Sales Order", the system pops up the interface "Create Sales Order Entry Form" (Figure 4.15). The user can choose a customer name and shipping company, and then click on the button "Create Order" to create a new sales order. If the user clicks the button "Order Product", system pops up the interface "Entry form of ordering product for the new sales order"(Figure 4.16). The user can click the button "New Customer" to add new customer (Figure 4.17), or click the button "New Shipper" to add new shipping company (Figure 4.18).

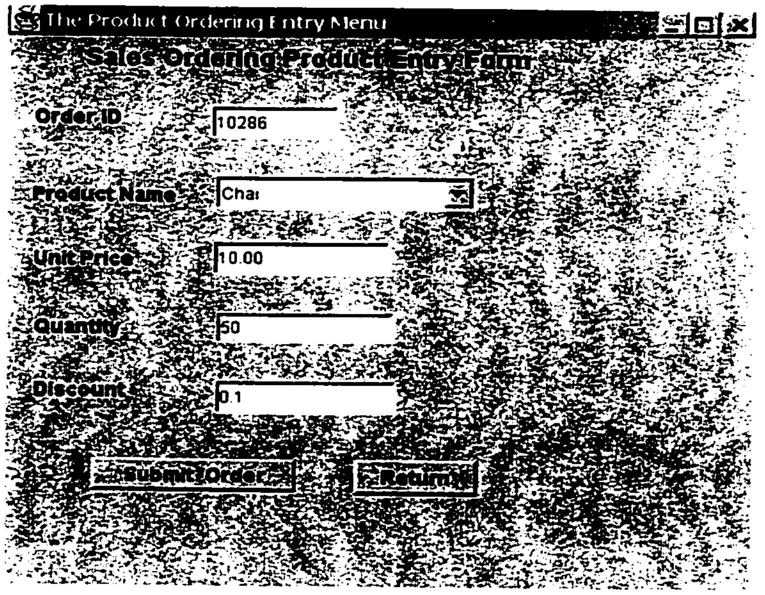


10286

Alfreds Futterkiste

Speedy Express

Figure 4.15 Entry form of creating a new sales order


The Product Ordering Entry Menu
Sales Ordering Product Entry Form
Order ID
Product Name
Unit Price
Quantity
Discount
Submit Order
Return

10286

Chai

10.00

50

0.1

Figure 4.16 Entry form of ordering product for the new sales order

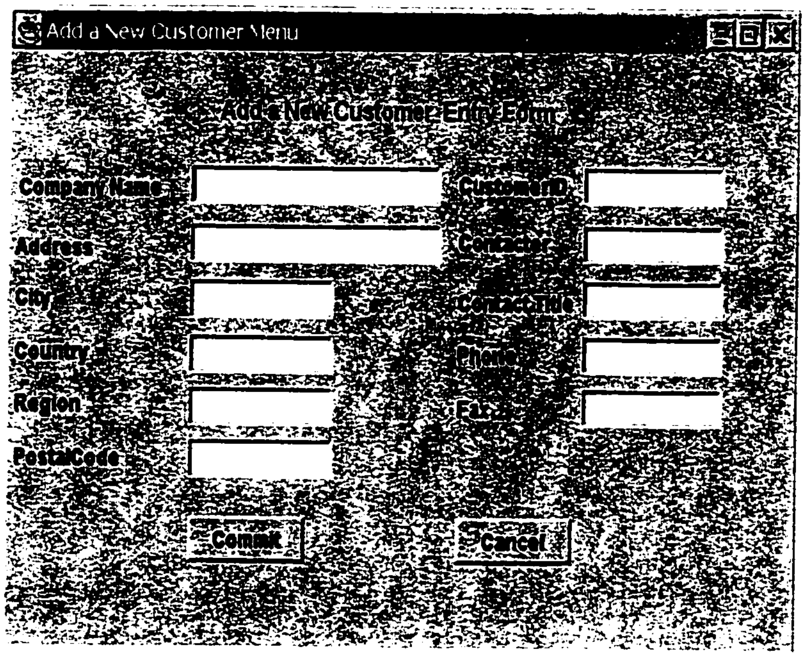


Figure 4.17 Entry form of adding a new customer

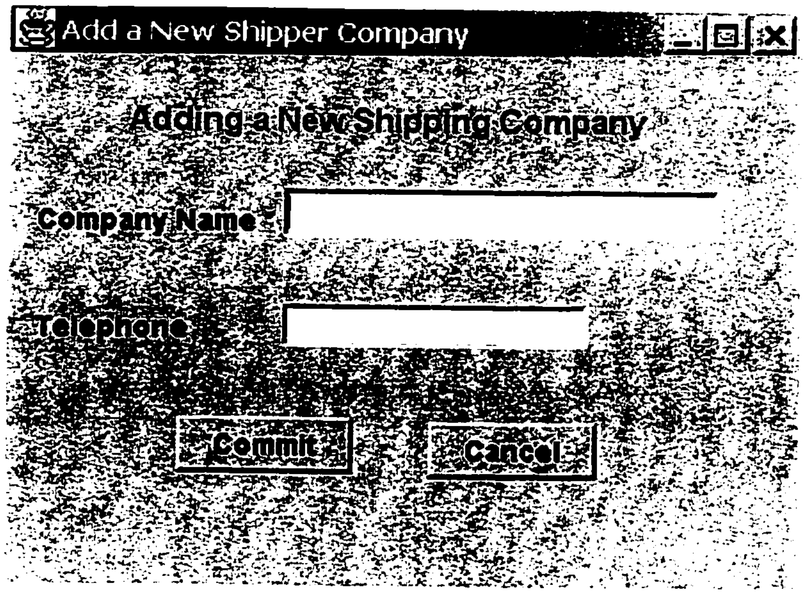


Figure 4.18 Entry form of adding a new shipping company

Chapter 5 Concluding Remarks and Future Work

The LTI system implemented has several limitations, explained below.

5.1 Using Auxiliary View

In the LTI system, we use auxiliary views to help lineage tracing. Once an auxiliary view has been created in the data warehouse, when lineage tracing source data, we just need to trace back to auxiliary views to get enough information. Obviously, it avoids some computations and expensive source queries, thereby reducing maintenance and query costs.

As discussed in section 2.4, there are several approaches for storing auxiliary views to support lineage tracing. For each approach of storing auxiliary view, we specified the lineage tracing procedure, as well as the maintenance procedure. We compared them and chose storing partial base table (PBT) and storing Base table projection (BTP). There are some benefits of our choices, described as follows:

- Our choices reduce the storage requirement as well as the cost of refreshing the auxiliary views. For example, when we create a view, we just store useful information instead of the entire base tables.
- The choices reduce the tracing cost. Just as mentioned above, if there is enough information in auxiliary view, it is not necessary to trace back to source database.

Since accessing source database is very expensive, it greatly reduces the tracing cost.

- Our choices improve tracing query performance (over storing nothing) while reducing view maintenance and storage costs. If we store nothing in the data warehouse, every time for lineage tracing, the system has to trace back to source database, which is very expensive and time consuming. Tracing query operates on a much smaller view table is much faster than querying base table in source databases.

5.2 Data Update in Data Warehouse

As mentioned in Section 1.7, there are two primary methods for extracting data from source systems. In the LTI system, we use the Bulk Extractions method for updating data warehouse when the source data changes. The reason is that such a warehouse is easier to set up and maintain. The data warehouse can be refreshed periodically by extraction of data from the source systems. All applicable data are extracted from the source systems for loading into the warehouse. However, it heavily increases the burden of network connection between source and data warehouse.

In practice, Change-Based replication is often used. Only the data that have been newly inserted or updated in source systems are extracted and loaded into the warehouse. Therefore, it places less stress on the network (due to the smaller volume of data to be transported). However, it requires more complex programming to determine when a new warehouse record must be inserted or when an existing warehouse record must be updated.

5.3 Data Growth in Data Warehouse

In our LTI system, we did not discusses the space or the capacity problems. But as time passes and the warehouse grows in size, with frequent tuple insertions, and changes in the contents and extraction method, the indexes can be built up for the source data and data warehouse to improve efficiency of the data access. Moreover, as discussed in [HWM1999], other ways can be taken to handle data growth, including:

- **Use of aggregates.** The use of stored aggregates significantly reduces the space required by the data, especially if the data are required only at a highly summarized level. The detailed data can be deleted or archived after aggregates have been created. Note however, that the removal of detailed data implies the loss of the ability to drill down for more detail. It has to be balanced.
- **Limiting the time frame.** Although users desire the warehouse to store as much data for as long as possible, there may be a need to compromise by limiting the length of historical data in the warehouse.
- **Removing unused data.** Using query statistics gathered over time, it is possible for the warehouse administrators to identify rarely used data in the data warehouse. These records are ideal candidates for removal since their storage results in costs with very little business value.

5.4 Future Work

5.4.1 Dynamic Tracing

In the LTI system, we implemented two major tracing functionalities: tracing inventory information and tracing sales order information. In the future, we need to implement the dynamic tracing functions in the system, so that user can select any combination of attributes of a tuple to trace back. If for every combination, we implement a class to do the work, the workload will be very huge. For example, if the tuple has 50 attributes, we have to implement 2^{50} classes. We can construct an engine that will dynamically construct SQL according to the user requirements. It is like the CGI (Common Gateway Interface) that connects user interface and the database. That means, for every combination of attributes, the engine creates a SQL statement, in order to query the data warehouse. Then, we can achieve dynamic tracing and can give users more friendly user interface, more flexibility to choose what they want to trace.

5.4.2 Distributed, Heterogeneous Source Databases

In the LTI system, the source databases are developed using Microsoft Access, and the sources databases are stored in one computer. In the future, we can connect this system to distributed, heterogeneous source databases, such as MS SQL Server, Oracle, Access, etc, and that can be distributed in a LAN or WAN network of computers.

5.4.3 More Functionality for Different Users

In the future, we can provide more functionality to improve the usability for different users, for example, checking customer information, supplier information, shipper information, etc. The user interface can be improved to provide and support more flexibility to ender users.

5.4.4 Flexibility and Reusability

LTI system is designed and developed for tracing view data of the inventory information and tracing sales order for a sales company. The design ideas can be used as a basis for a lineage tracing mechanism for the diffident application in a general warehouse setting. In the future, we need to improve the flexibility and reusability of the design and implementation to adapt to different applications.

Our main objective in this project and the prototype developed was to demonstrate what lineage tracing is, how tracing is done, and how data warehouse updates are managed in conjunction with tracing capability. Our prototype system is still a long way to become a real data warehouse system. However, it contains basic, important components of a data warehouse, and it truly demonstrates the lineage tracing procedures.

References

[Bra1996] Michael H. Brackett, *The Data Warehouse Challenge: Taming data Chaos*, Wiley Computer publishing, John Wiley & Sons, Inc., 1996.

[CW1999] Y. Cui and J. Widom, *Storing Auxiliary Data for Efficient View Maintenance and Lineage Tracing*. Techincal Report, Stanford University, 1999.

[CW2000] Yingwei Cui and Jennifer Widom, *Practical Lineage Tracing in a Data Warehousing System*, In Proceedings of the Sixteenth International Conference on Data Engineering, San Diego, California, February 2000.

[CW2000+] Yingwei Cui and Jennifer Widom, *Lineage Tracing in a Data Warehousing System*. In Proceedings of the Sixteenth International Conference on Data Engineering, San Diego, California, February 2000. Demonstration Description.

[CWW1997] Yingwei Cui, J. Widom, and J.L. Wiener, *Tracing the lineage of view data in a data warehousing environment*. Technical report, Stanford University Database Group, November 1997 Technical Report, Stanford University, 1997 (Revised 1999).

[GW1998] Paul Gray, Hugh J.Watson, *Decision Support in the Data Warehouse*, The data Warehousing Institute Series from Pentice Hall PTR, 1998.

[Huy1996] N. Huyn, *Efficient Self-Maintenance of Materialized Views*. Technical Report, Stanford University, 1996.

[HWM1999] Mark Humphries, Michael W. Hawkins, Michelle C. Dy, *Data Warehousing Architecture and Implementation*, Prentice Hall PTR, 1999.

[Inm1996] W. H. Inmon. *Building the Data Warehouse*, Second Edition, John Wiley & Sons, Inc. 1996.

[Kel1996] Sean Kelly, *Data Warehousing: the route to mass customization*, John Wiley & Sons, 1996.

[LZW1997] W.J.Labio, Y. Zhuge, J.Widom. *The WHIPS Prototype for Data Warehouse Create and Maintenance*, Dept. of Computer Science, Stanford Univ. 1997.

[MAG1998] Management Accounting Guideline 48, *Building a Data Warehouse*, Included in Management Accounting Practices handbook, 1998.

[Mat1996] Rob Mattison. *Data Warehousing: Strategies, Technologies, and Techniques*, McGraw-Hill. 1996.

[QGMW1996] D. Quass, A. Gupta, I. Mumick, and J. Widom, *Making Views Self-Maintainable for Data Warehousing*. In Proceedings of the Conference on Parallel and Distributed Information Systems. Miami Beach, FL, December 1996.

[WGLZGW1996] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom, *A System Prototype for Warehouse View Maintenance*. In Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada, June 7, 1996, pp. 26-33.

[Wid1995] J. Widom, *Research Problem in Data Warehousing*, Dept. of Computer Science, Stanford Univ. 1995

[CLWX2001] Minghua Chen, Dongmei Liu, Minggang Wu, Jiu Xu, *Lineage Tracing System in Data Warehouse*. A COMP6591 Project Report, Concordia University, December 2001

Appendix Sample Codes

In Appendix, we present some sample codes for the implementation of the main functions of LTI system. For this we have decided to illustrate the codes for the following modules:

- Module 1. Main Interface of LTI
- Module 2. Interface of Tracing Inventory
- Module 3. Tracing Chosen Product Inventory
- Module 4. Interface of Tracing Sales Order
- Module 5. Tracing Chosen Category Products Sales Order

Module 1. Main Interface of LTI

```
//Implement the main interface of the LTI. There are three subtasks supported: Tracing
inventory information, tracing sales order information and updating sources.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;

public class mainFrame extends JFrame {
 JPanel contentPane;
 JMenuBar jMenuBar1 = new JMenuBar();
 JMenu jMenuFile = new JMenu();
 JMenuItem jMenuFileExit = new JMenuItem();
 JMenu jMenuHelp = new JMenu();
 JMenuItem jMenuHelpTopic = new JMenuItem();
 JMenuItem jMenuHelpAbout = new JMenuItem();
 JToolBar jToolBar = new JToolBar();
 JButton jButton1 = new JButton();
 JButton jButton2 = new JButton();
 JButton jButton3 = new JButton();
 ImageIcon image1;
 ImageIcon image2;
 ImageIcon image3;
```

```
JLabel statusBar = new JLabel();
BorderLayout borderLayout1 = new BorderLayout();
ButtonGroup bGroup1=new ButtonGroup();
Panel dataArea1 = new Panel();
Button button2 = new Button();
ResultSetMetaData rsmd;
JLabel jLabel2 = new JLabel();
Button button1 = new Button();

boolean isSaleTracing=false;
boolean isInvTracing=false;
Panel panel1 = new Panel();
JButton jButton4 = new JButton();
JButton jButton_update = new JButton();
JButton jButton_updateSource = new JButton();

/**Construct the frame*/
public mainFrame() {
 enableEvents(AWTEvent.WINDOW_EVENT_MASK);
 try {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
/**Component initialization*/
private void jbInit() throws Exception  {
 image1 = new ImageIcon(datatrace.mainFrame.class.getResource("openFile.gif"));
 image2 = new ImageIcon(datatrace.mainFrame.class.getResource("closeFile.gif"));
 image3 = new ImageIcon(datatrace.mainFrame.class.getResource("help.gif"));

 contentPane = (JPanel) this.getContentPane();
 contentPane.setLayout(null);
 this.setSize(new Dimension(400, 381));
 this.setTitle("Lineage Tracing Inventory System");
 jMenuFile.setText("File");
 jMenuFileExit.setText("Exit");
 jMenuFileExit.addActionListener(new ActionListener()  {
  public void actionPerformed(ActionEvent e) {
   jMenuFileExit_actionPerformed(e);
  }
 });
 jMenuHelp.setText("Help");
 jMenuHelpAbout.setText("About");
 jMenuHelpAbout.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuHelpAbout_actionPerformed(e);
    }
  });

  jMenuHelpTopic.setText("Topics");
  jMenuHelpTopic.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuHelpTopic_actionPerformed(e);
    }
  });
  dataArea1.setLayout(null);
  dataArea1.setBounds(new Rectangle(23, 118, 356, 26));
  button1.setFont(new java.awt.Font("Dialog", 1, 13));
  button1.setLabel("Inventory Information");
  button1.setBounds(new Rectangle(70, 2, 226, 37));
  button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button1_actionPerformed(e);
    }
  });

  button2.setFont(new java.awt.Font("Dialog", 1, 13));
  button2.setLabel("Sales Order Information");
  button2.setBounds(new Rectangle(70, 60, 226, 37));
  button2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button2_actionPerformed(e);
    }
  });

  jButton1.setIcon(image1);
  jButton1.setToolTipText("Open File");
  jButton2.setIcon(image2);
  jButton2.setToolTipText("Close File");
  jButton3.setIcon(image3);
  jButton3.setToolTipText("Help");
  jButton_update.setText("jButton5");
  jButton_update.setBounds(new Rectangle(77, 139, 226, 30));
  jButton_updateSource.setFont(new java.awt.Font("Dialog", 1, 12));
  jButton_updateSource.setActionCommand("jButton_updateSource");
  jButton_updateSource.setText("Update Sources");
  jButton_updateSource.setBounds(new Rectangle(70, 116, 226, 37));
  jButton_updateSource.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton_updateSource_actionPerformed(e);
      }
    });
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpTopic);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(jToolBar, BorderLayout.NORTH);
    contentPane.add(statusBar, BorderLayout.SOUTH);
    contentPane.setMaximumSize(new Dimension(32767, 33767));

    jLabel2.setFont(new java.awt.Font("Dialog", 1, 20));
    jLabel2.setForeground(Color.red);
    jLabel2.setText("Lineage Tracing Inventory System");
    jLabel2.setBounds(new Rectangle(38, 71, 333, 37));
    panel1.setBounds(new Rectangle(13, 129, 371, 227));
    panel1.setLayout(null);

    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuHelp.add(jMenuHelpTopic);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(panel1, null);
    panel1.add(button1, null);
    panel1.add(button2, null);
    panel1.add(jButton_updateSource, null);
    contentPane.add(dataArea1, null);
    contentPane.add(jLabel2, null);
  }

  /**File | Exit action performed*/
  public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
  /**Help | About action performed*/
  public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
    mainFrame_AboutBox dlg = new mainFrame_AboutBox(this);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
```

```
   dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
       dlgSize.height) / 2 + loc.y);
   dlg.setModal(true);
   dlg.show();
  }
  /**Help | Topic action performed*/
  public void jMenuHelpTopic_actionPerformed(ActionEvent e) {
   mainFrame_TopicBox dlg2 = new mainFrame_TopicBox(this);
   Dimension dlgSize = dlg2.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg2.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height –
      dlgSize.height) / 2 + loc.y);
   dlg2.setModal(true);
   dlg2.show();
  }

/**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e) {
   super.processWindowEvent(e);
   if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
   }
  }

  void jCheckBox1_actionPerformed(ActionEvent e) {
    isSaleTracing=true;
  }

  void jCheckBox2_actionPerformed(ActionEvent e) {
    isInvTracing=true;
  }

  void button1_actionPerformed(ActionEvent e) {
   StockInfo myInventory=new StockInfo();
   myInventory.setTitle("Inventory Information Management");
   myInventory.pack();
   myInventory.setVisible(true);
   myInventory.setSize(400,300);
  }

  void button2_actionPerformed(ActionEvent e)  {
   SalesOrderInfo myOrder=new SalesOrderInfo();
   myOrder.setTitle("Sales Order Information Management");
   myOrder.pack();
   myOrder.setVisible(true);
```

```
    myOrder.setSize(400,300);
  }

  void jButton_updateSource_actionPerformed(ActionEvent e) {
    chooseSource updatesource = new chooseSource();
    updatesource.setTitle("Update Source Information Management");
    updatesource.setVisible(true);
    updatesource.validate();
    updatesource.setSize(400,400);

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = updatesource.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    updatesource.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
  }
}
```

Module 2. Interface of Tracing Inventory

// Implement for tracing the inventory information. The user can choose to trace inventory information according to product name, category name, or supplier name, and the user can select to trace inventory information of all the stores or each store.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import java.net.*;
import jdbc.*;
import viewBean.*;

public class StockInfo extends Frame {
  String StockChoose;
  String storeName;
  Panel panel5 = new Panel();
  Panel panel2 = new Panel();
Panel panel4 = new Panel();
ButtonGroup bGroup=new ButtonGroup();
JRadioButton jRadioButton1 = new JRadioButton("Product", true);
JRadioButton jRadioButton2 = new JRadioButton("Catelogy",false);
JRadioButton jRadioButton3 = new JRadioButton("Supplier",false);
JToggleButton jToggleButton1 = new JToggleButton();
Label label1 = new Label();
JButton jButton_return = new JButton();
String unitOnOrder;
String recordLevel;
String Discontinued;
JButton jButton_help = new JButton();
JComboBox jComboBox_store = new JComboBox();
JLabel jLabel1 = new JLabel();
public StockInfo() {
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
private void jbInit() throws Exception {
  label1.setBounds(new Rectangle(61, 75, 288, 29));
  label1.setText("Tracing Inventory Information");
  label1.setForeground(Color.red);
  label1.setFont(new java.awt.Font("Dialog", 1, 20));
  jToggleButton1.setBounds(new Rectangle(84, 102, 95, 25));
  jToggleButton1.setText("Continue");
  jToggleButton1.setFont(new java.awt.Font("Dialog", 1, 12));
  jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jToggleButton1_actionPerformed(e);
    }
  });

  jRadioButton1.setBounds(new Rectangle(52, 6, 66, 25));
  jRadioButton1.setText("Product ");
  jRadioButton1.setFont(new java.awt.Font("Dialog", 1, 12));
  jRadioButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jRadioButton1_actionPerformed(e);
    }
  });

  jRadioButton2.setBounds(new Rectangle(158, 8, 87, 25));
```

```
jRadioButton2.setText("Category");
jRadioButton2.setFont(new java.awt.Font("Dialog", 1, 12));
jRadioButton2.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jRadioButton2_actionPerformed(e);
  }
});
jRadioButton3.setBounds(new Rectangle(267, 9, 69, 24));
jRadioButton3.setText("Supplier");
jRadioButton3.setFont(new java.awt.Font("Dialog", 1, 12));
jRadioButton3.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jRadioButton3_actionPerformed(e);
  }
});

panel4.setLayout(null);
panel4.setBounds(new Rectangle(5, 150, 388, 147));
jButton_return.setFont(new java.awt.Font("Dialog", 1, 12));
jButton_return.setText("Return");
jButton_return.setBounds(new Rectangle(232, 102, 86, 26));
jButton_return.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jButton_return_actionPerformed(e);
  }
});

jButton_help.setActionCommand("jButton_help");
jButton_help.setText("Help");
jButton_help.setBounds(new Rectangle(19, 33, 60, 22));
jButton_help.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jButton_help_actionPerformed(e);
  }
});

String[] storeList;
final int row=4;
storeList=new String[row];
storeList[0]="All_stores";
storeList[1]="Store_1";
storeList[2]="Store_2";
storeList[3]="Store_3";
jComboBox_store=new JComboBox((Object[])storeList);

jComboBox_store.setBounds(new Rectangle(178, 53, 156, 27));
```

```
    jComboBox_store.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jComboBox_store_actionPerformed(e);
      }
    });

    jLabel1.setFont(new java.awt.Font("Dialog", 1, 13));
    jLabel1.setText("Choose store(s):");
    jLabel1.setBounds(new Rectangle(53, 53, 108, 26));

    panel4.add(jRadioButton3, null);
    panel4.add(jRadioButton2, null);
    panel4.add(jRadioButton1, null);
    panel4.add(jToggleButton1, null);
    panel4.add(jButton_return, null);
    panel4.add(jLabel1, null);
    panel4.add(jComboBox_store, null);
    this.add(panel2, null);
    this.add(panel4, null);
    this.add(panel5, null);
    bGroup.add(jRadioButton1);
    bGroup.add(jRadioButton2);
    bGroup.add(jRadioButton3);
    StockChoose="Product";
    this.setLayout(null);
    panel5.setBounds(new Rectangle(4, 14, 383, 111));
    panel5.setLayout(null);
    this.setForeground(Color.red);
    this.setBackground(SystemColor.activeCaptionBorder);
    this.addWindowListener(new java.awt.event.WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        this_windowClosing(e);
      }
    });
    panel2.setBounds(new Rectangle(41, 128, 332, 20));
    panel2.setLayout(null);
    panel5.add(jButton_help, null);
    panel5.add(label1, null);
    jRadioButton1.isSelected();
  }

  void jRadioButton1_actionPerformed(ActionEvent e) {
    StockChoose="Product";
  }

  void jRadioButton2_actionPerformed(ActionEvent e) {
   StockChoose="Catagory";
  }

  void jRadioButton3_actionPerformed(ActionEvent e) {
   StockChoose="Supplier";
  }

  void jToggleButton1_actionPerformed(ActionEvent e) {
   if (StockChoose=="Product")
   {
      StockProducts f1=new StockProducts(storeName,StockChoose);
      f1.setTitle("Stock Detail Information (Products)");
      f1.pack();
      f1.setSize(400,300);
      f1.setVisible(true);
   }
   else if (StockChoose=="Catagory")
   {
      StockCategories f2=new StockCategories(storeName,StockChoose);
      f2.setTitle("Stock Detail Information (Categories)");
      f2.pack();
      f2.setSize(400,300);
      f2.setVisible(true);
   }
   else if (StockChoose=="Supplier")
   {
      StockSuppliers f3=new StockSuppliers(storeName,StockChoose);
      f3.setTitle("Stock Detail Information (Suppliers)");
      f3.pack();
      f3.setSize(400,300);
      f3.setVisible(true);
   }
 }

 void this_windowClosing(WindowEvent e) {
    setVisible(false);
 }

 void jButton_return_actionPerformed(ActionEvent e) {
    setVisible(false);
 }

 void jButton_help_actionPerformed(ActionEvent e) {
  stockInfo_TopicBox dlg2 = new stockInfo_TopicBox(this);
  Dimension dlgSize = dlg2.getPreferredSize();
  Dimension frmSize = getSize();
```

```
    Point loc = getLocation();
    dlg2.setModal(true);
    dlg2.show();
  }

  void jComboBox_store_actionPerformed(ActionEvent e) {
      String temp=new String();
      temp=((String)jComboBox_store.getSelectedItem());
      storeName=temp.trim();
    }

  public  String checkStr(String str)  {
     char[] ctr=new char[str.length()+10];
     String f="'d";
     int j=0;
     for(int i=0;i<str.length();i++)
     {
          char ch=str.charAt(i);
          if(ch==f.charAt(0)){
            ctr[j]=ch;
            j=j+1;
            ctr[j]=ch;
            j=j+1;
          }else{
        ctr[j]=ch;
            j=j+1;
          }
     }
    return new String(ctr);
  }
}
```

Module 3. Tracing Chosen Product Inventory

```
//Implement for tracing inventory information of the chosen product for all stores or one
store

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import jdbc.*;
import viewBean.*;

public class StockProducts extends Frame {
  String storeName=new String();
  String stockChoose=new String();
  Panel panel1 = new Panel();
  Label label1 = new Label();
  Panel panel2 = new Panel();
  JComboBox jComboBox1;// = new JComboBox();
  JLabel jLabel1 = new JLabel();
  JButton jButton1 = new JButton();
  String itemName=new String();
  final String dbDriver="sun.jdbc.odbc.JdbcOdbcDriver";

  ResultSetMetaData rsmd;
  JCheckBox jCheckBox1 = new JCheckBox();
  boolean isTracing=false;
  JButton jButton_cancel = new JButton();
  JButton jButton_help = new JButton();
  JLabel jLabel2 = new JLabel();
  JTextField jTextField1 = new JTextField();

  public StockProducts(String name,String choose)  {
    storeName=name.trim();
    stockChoose=choose.trim();
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
    this.setLayout(null);
    panel1.setBounds(new Rectangle(2, 15, 395, 143));
    panel1.addHierarchyBoundsListener(new java.awt.event.HierarchyBoundsAdapter() {
      public void ancestorMoved(HierarchyEvent e) {
        panel1_ancestorMoved(e);
      }
    });
    panel1.setLayout(null);
    label1.setFont(new java.awt.Font("Dialog", 1, 18));
    label1.setForeground(Color.red);

    label1.setText("Stock Products Information (Products)");
    label1.setBounds(new Rectangle(28, 58, 342, 27));
```

```
 panel2.setBounds(new Rectangle(9, 167, 379, 130));
 panel2.setLayout(null);

 jLabel1.setFont(new java.awt.Font("Dialog", 1, 14));
 jLabel1.setText("Choose a Product:");
 jLabel1.setBounds(new Rectangle(26, 12, 144, 25));
 jButton1.setFont(new java.awt.Font("Dialog", 1, 14));
 jButton1.setText("Search");
 jButton1.setBounds(new Rectangle(158, 67, 84, 23));
 jButton1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jButton1_actionPerformed(e);
  }
 });
 this.setBackground(SystemColor.activeCaptionBorder);
 this.addWindowListener(new java.awt.event.WindowAdapter() {
  public void windowClosing(WindowEvent e) {
   this_windowClosing(e);
  }
 });

 jCheckBox1.setFont(new java.awt.Font("Dialog", 1, 14));
 jCheckBox1.setText("tracing");
 jCheckBox1.setBounds(new Rectangle(33, 66, 75, 27));
 jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jCheckBox1_actionPerformed(e);
  }
 });
 jButton_cancel.setFont(new java.awt.Font("Dialog", 1, 14));
 jButton_cancel.setActionCommand("Cancel");
 jButton_cancel.setText("Cancel");
 jButton_cancel.setBounds(new Rectangle(258, 68, 86, 23));
 jButton_cancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jButton_cancel_actionPerformed(e);
  }
 });
 jButton_help.setActionCommand("jButton_help");
 jButton_help.setText("Help");
 jButton_help.setBounds(new Rectangle(28, 27, 65, 25));
 jButton_help.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jButton_help_actionPerformed(e);
  }
 });
```

```
  jLabel2.setFont(new java.awt.Font("Dialog", 1, 13));
  jLabel2.setText("Store Name:");
  jLabel2.setBounds(new Rectangle(26, 103, 88, 25));
  jTextField1.setBounds(new Rectangle(116, 106, 94, 25));
  jTextField1.setEditable(false);
  jTextField1.setText(storeName);
  this.add(panel1, null);
  panel1.add(label1, null);
  panel1.add(jLabel2, null);
  panel1.add(jTextField1, null);
  panel1.add(jButton_help, null);
  this.add(panel2, null);

  jdbc items=new jdbc();
  items.setClassname(dbDriver);
  items.setIsQuery(true);

  items.setUrl("jdbc:odbc:source-store1");
  String itemQuery="SELECT [Products].[ProductName] "+
             "FROM Products;";
  items.setQuery(itemQuery);

  int itemNum;
  String[] itemList;
  try{
    items.go();
    Vector itemVector=items.getResult();
    itemNum=items.getRowCount();
    final int row=itemNum;
    itemList=new String[row];
    String[] temp=new String[1];
    for(int i=2 ; i<=row;i++){
     temp=(String[])itemVector.elementAt(i-1);
     itemList[i-2]=temp[0];
    }
    jComboBox1=new JComboBox((Object[])itemList);
    }catch (Exception ex) {
         ex.printStackTrace();
      }

jComboBox1.setBounds(new Rectangle(173, 13, 170, 25));
jComboBox1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jComboBox1_actionPerformed(e);
  }
});
```

```
    panel2.add(jLabel1, null);
    panel2.add(jComboBox1, null);
    panel2.add(jCheckBox1, null);
    panel2.add(jButton1, null);
    panel2.add(jButton_cancel, null);
   }
   void jComboBox1_actionPerformed(ActionEvent e) {
       String temp=new String();
       temp=checkStr((String)jComboBox1.getSelectedItem());
       itemName=temp.trim();
       System.out.println(itemName);
   }
  public  String checkStr(String str)
  {
     char[] ctr=new char[str.length()+10];
     String f="'d";
     int j=0;
     for(int i=0;i<str.length();i++)
     {
          char ch=str.charAt(i);
          if(ch==f.charAt(0)){
            ctr[j]=ch;
            j=j+1;
            ctr[j]=ch;
            j=j+1;
          }else{
        ctr[j]=ch;
            j=j+1;
          }
     }
    return new String(ctr);
  }

 public String getViewName()
 {
   Calendar c = Calendar.getInstance();
   java.util.Date myDate=c.getTime();
   String temp=(new Integer(myDate.getHours())).toString();
   temp=temp+(new Integer(myDate.getMinutes())).toString();
   temp=temp+(new Integer(myDate.getSeconds())).toString();
   return temp;
  }

void jButton1_actionPerformed(ActionEvent e) {
  jdbc inv=new jdbc();
```

```
stockDisplayProduct inventory=new stockDisplayProduct(itemName,storeName);
inv.setClassname("sun.jdbc.odbc.JdbcOdbcDriver");
inv.setIsQuery(true);

String itemQuery;
itemQuery ="SELECT [Products].[ProductName], [Suppliers].[CompanyName], "+
        "[Products].[UnitPrice], [Products].[UnitsInStock], "+
        " [Products].[UnitsOnOrder], [Products].[ReorderLevel] "+
        " FROM Suppliers INNER JOIN Products ON "+
        " [Suppliers].[SupplierID] = [Products].[SupplierID] "+
       " WHERE [Products].[ProductName]='";
     itemQuery=itemQuery+itemName + "';";

Vector resultVector;
viewBean jdbcBean = new viewBean();
jdbcBean.setClassname("sun.jdbc.odbc.JdbcOdbcDriver");
jdbcBean.setViewUrl("jdbc:odbc:trace-store0");
String viewName=new String();
viewName=getViewName();
jdbcBean.setViewName(viewName);
jdbcBean.setQuery(itemQuery);
try {
     jdbcBean.creatViewer();
     String[] metaData;

     if (storeName=="All_stores")  {
         itemQuery = "SELECT [D1].[ProductName], "+
          "sum([D1].[UnitsInStock]) AS [Total in Stock],"+
          " sum([D1].[UnitsOnOrder]) AS [Total On Order] "+
          " FROM "+viewName+" D1 GROUP BY [D1].[ProductName]; ";
     }
     else if (storeName=="Store_1")  {
         itemQuery = "SELECT [D2].[ProductName], "+
          "sum([D2].[UnitsInStock]) AS [Total in Stock],"+
          " sum([D2].[UnitsOnOrder]) AS [Total On Order] "+
          " FROM "+viewName+" D2 WHERE [D2].Store_Location='Store 1'"+
          " GROUP BY [D2].[ProductName]; ";
     }
     else if (storeName=="Store_2")  {
         itemQuery = "SELECT [D2].[ProductName], "+
          "sum([D2].[UnitsInStock]) AS [Total in Stock],"+
          " sum([D2].[UnitsOnOrder]) AS [Total On Order] "+
          " FROM "+viewName+" D2 WHERE [D2].Store_Location='Store 2'"+
          " GROUP BY [D2].[ProductName]; ";
     }
     else if (storeName=="Store_3")  {
            itemQuery = "SELECT [D2].[ProductName], "+
              "sum([D2].[UnitsInStock]) AS [Total in Stock],"+
              " sum([D2].[UnitsOnOrder]) AS [Total On Order] "+
              " FROM "+viewName+" D2 WHERE [D2].Store_Location='Store 3'"+
              " GROUP BY [D2].[ProductName]; ";
        }
        inv.setQuery(itemQuery);
        inv.setUrl("jdbc:odbc:trace-store0");
        inv.go();
        resultVector=inv.getResult();
        final int a=inv.getColumnCount();
        final int b=inv.getRowCount();
        String[][] tableData;
        tableData=new String[b-1][a];
        for(int m=2;m<=b;m++)
        {
         metaData=(String[])resultVector.elementAt(m-1);
         for(int n=1; n<=a;n++)
         {
          tableData[m-2][n-1]=metaData[n-1];
         }
        }
       inventory.setMetaData((String[])resultVector.elementAt(0));
       inventory.setTableData(tableData);
   } catch (Exception ex) {
      ex.printStackTrace();
   }
  if(isTracing)
  {
    if (storeName=="All_stores")
    {
        itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
              "  [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
              "FROM "+ viewName +" D1 "+
              "WHERE D1.Store_Location='Store 1';";
        inv.setQuery(itemQuery);

       try {
              inv.setUrl("jdbc:odbc:trace-store0");
              inv.go();
              resultVector=inv.getResult();
              String[] metaData;
              final int a=inv.getColumnCount();
              final int b=inv.getRowCount();
              String[][] tableData;
              tableData=new String[b-1][a];
      for(int m=2;m<=b;m++)
      {
       metaData=(String[])resultVector.elementAt(m-1);
       for(int n=1; n<=a;n++)
       {
        tableData[m-2][n-1]=metaData[n-1];
       }
      }
      inventory.setMetaData1((String[])resultVector.elementAt(0));
      inventory.setTableData1(tableData);
  } catch (Exception ex) {
  ex.printStackTrace();
 }
   itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
       " [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
         "FROM "+ viewName +" D1 "+
         "WHERE D1.Store_Location='Store 2';";
    inv.setQuery(itemQuery);

 try {
       inv.go();
       resultVector=inv.getResult();
       String[] metaData;
       final int a=inv.getColumnCount();
       final int b=inv.getRowCount();
       String[][] tableData;
       tableData=new String[b-1][a];
       for(int m=2;m<=b;m++)
       {
        metaData=(String[])resultVector.elementAt(m-1);
        for(int n=1; n<=a;n++)
        {
         tableData[m-2][n-1]=metaData[n-1];
        }
       }
      inventory.setMetaData2((String[])resultVector.elementAt(0));
      inventory.setTableData2(tableData);
  } catch (Exception ex) {
   ex.printStackTrace();
 }

 itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
      " [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
       "FROM "+ viewName +" D1 "+
       "WHERE D1.Store_Location='Store 3';";
inv.setQuery(itemQuery);
```

```
      try {
          inv.go();
          resultVector=inv.getResult();
          String[] metaData;
          final int a=inv.getColumnCount();
          final int b=inv.getRowCount();
          String[][] tableData;
          tableData=new String[b-1][a];
          for(int m=2;m<=b;m++)
          {
           metaData=(String[])resultVector.elementAt(m-1);
           for(int n=1; n<=a;n++)
           {
            tableData[m-2][n-1]=metaData[n-1];
           }
          }
          inventory.setMetaData3((String[])resultVector.elementAt(0));
          inventory.setTableData3(tableData);
       } catch (Exception ex) {
       ex.printStackTrace();
      }
    }

  else if (storeName=="Store_1")
  {
      itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
           "  [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
          "FROM "+ viewName +" D1 "+
          "WHERE D1.Store_Location='Store 1';";
      inv.setQuery(itemQuery);

      try {
          inv.setUrl("jdbc:odbc:trace-store0");
          inv.go();
          resultVector=inv.getResult();
          String[] metaData;
          final int a=inv.getColumnCount();
          final int b=inv.getRowCount();
          String[][] tableData;
          tableData=new String[b-1][a];
          for(int m=2;m<=b;m++)
          {
            metaData=(String[])resultVector.elementAt(m-1);
           for(int n=1; n<=a;n++)
           {
```

```
                    tableData[m-2][n-1]=metaData[n-1];
                }
               }
               inventory.setMetaData1((String[])resultVector.elementAt(0));
               inventory.setTableData1(tableData);
          } catch (Exception ex) {
          ex.printStackTrace();
         }
      }
    else if (storeName=="Store_2")
    {
      itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
              "  [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
             "FROM "+ viewName +" D1 "+
             "WHERE D1.Store_Location='Store 2';";
      inv.setQuery(itemQuery);

     try {
         inv.setUrl("jdbc:odbc:trace-store0");
         inv.go();
         resultVector=inv.getResult();
         String[] metaData;
         final int a=inv.getColumnCount();
         final int b=inv.getRowCount();
         String[][] tableData;
         tableData=new String[b-1][a];
         for(int m=2;m<=b;m++)
         {
          metaData=(String[])resultVector.elementAt(m-1);
          for(int n=1; n<=a;n++)
          {
           tableData[m-2][n-1]=metaData[n-1];
          }
         }
         inventory.setMetaData1((String[])resultVector.elementAt(0));
         inventory.setTableData1(tableData);
        } catch (Exception ex) {
        ex.printStackTrace();
       }
    }
    else if (storeName=="Store_3")
    {
      itemQuery= "SELECT D1.ProductName, D1.CompanyName AS "+
             "  [SupplierName], D1.UnitPrice, D1.UnitsInStock, D1.UnitsOnOrder "+
            "FROM "+ viewName +" D1 "+
            "WHERE D1.Store_Location='Store 3';";
          inv.setQuery(itemQuery);
        try {
            inv.setUrl("jdbc:odbc:trace-store0");
            inv.go();
            resultVector=inv.getResult();
            String[] metaData;
            final int a=inv.getColumnCount();
            final int b=inv.getRowCount();
            String[][] tableData;
            tableData=new String[b-1][a];
            for(int m=2;m<=b;m++)
            {
             metaData=(String[])resultVector.elementAt(m-1);
             for(int n=1; n<=a;n++)
             {
              tableData[m-2][n-1]=metaData[n-1];
             }
            }
            inventory.setMetaData1((String[])resultVector.elementAt(0));
            inventory.setTableData1(tableData);
           } catch (Exception ex) {
           ex.printStackTrace();
          }
       }
    }

    inventory.setTitle("Inventory Detail Search Result Display");
    inventory.tableGo();
    inventory.pack();
    inventory.setVisible(true);

    if(isTracing){
     if (storeName=="All_stores"){
        inventory.setSize(680,680);
     }
     else{
        inventory.setSize(680,390);
      }
    }
    else{
     inventory.setSize(680,250);
     }
    setVisible(false);
}

void jCheckBox1_actionPerformed(ActionEvent e)
```

```
 {
    if(jCheckBox1.isSelected()==true)
    {
     isTracing=true;
    }else{
      isTracing=false;
    }
  }

  void this_windowClosing(WindowEvent e) {
    setVisible(false);
  }

  void jButton_cancel_actionPerformed(ActionEvent e) {
    setVisible(false);
  }

  void jButton_help_actionPerformed(ActionEvent e) {
   stockProducts_TopicBox dlg2 = new stockProducts_TopicBox(this);
   Dimension dlgSize = dlg2.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg2.setModal(true);
   dlg2.show();
  }
}
```

Module 4. Interface of Tracing Sales Order

// Implement for tracing the sales order information. The user can choose to trace sales order information according to product name, category name, or customer name, and the user can select to trace sales order information of all the stores or each store.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import java.net.*;
import jdbc.*;
import viewBean.*;

public class SalesOrderInfo extends Frame {
```

```
String OrderChoose;
String storeName;
Panel panel5 = new Panel();
Panel panel2 = new Panel();
Panel panel4 = new Panel();
ButtonGroup bGroup=new ButtonGroup();
JRadioButton jRadioButton1 = new JRadioButton("Product", true);
JRadioButton jRadioButton2 = new JRadioButton("Catelogy",false);
JRadioButton jRadioButton3 = new JRadioButton("Customer",false);
JToggleButton jToggleButton1 = new JToggleButton();
Label label1 = new Label();
JButton jButton_return = new JButton();
String unitOnOrder;
String recordLevel;
String Discontinued;
JButton jButton_help = new JButton();
JComboBox jComboBox_store = new JComboBox();

JLabel jLabel1 = new JLabel();
public SalesOrderInfo() {
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
private void jbInit() throws Exception {
  label1.setBounds(new Rectangle(45, 76, 312, 29));
  label1.setText("Tracing Sales Order Information");
  label1.setForeground(Color.red);
  label1.setFont(new java.awt.Font("Dialog", 1, 20));

  jToggleButton1.setBounds(new Rectangle(84, 102, 95, 25));
  jToggleButton1.setText("Continue");
  jToggleButton1.setFont(new java.awt.Font("Dialog", 1, 12));
  jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jToggleButton1_actionPerformed(e);
    }
  });

  jRadioButton1.setBounds(new Rectangle(52, 6, 66, 25));
  jRadioButton1.setText("Product ");
  jRadioButton1.setFont(new java.awt.Font("Dialog", 1, 12));
  jRadioButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jRadioButton1_actionPerformed(e);
    }
  });

  jRadioButton2.setBounds(new Rectangle(158, 8, 87, 25));
  jRadioButton2.setText("Category");
  jRadioButton2.setFont(new java.awt.Font("Dialog", 1, 12));
  jRadioButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jRadioButton2_actionPerformed(e);
    }
  });

  jRadioButton3.setBounds(new Rectangle(255, 9, 80, 24));
  jRadioButton3.setText("Customer");
  jRadioButton3.setFont(new java.awt.Font("Dialog", 1, 12));
  jRadioButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jRadioButton3_actionPerformed(e);
    }
  });

  panel4.setLayout(null);
  panel4.setBounds(new Rectangle(5, 150, 388, 147));

  jButton_return.setFont(new java.awt.Font("Dialog", 1, 12));
  jButton_return.setText("Return");
  jButton_return.setBounds(new Rectangle(232, 102, 86, 26));
  jButton_return.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton_return_actionPerformed(e);
    }
  });

  jButton_help.setActionCommand("jButton_help");
  jButton_help.setText("Help");
  jButton_help.setBounds(new Rectangle(19, 33, 60, 22));
  jButton_help.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton_help_actionPerformed(e);
    }
  });

  String[] storeList;
  final int row=4;
```

```
 storeList=new String[row];
 storeList[0]="All_stores";

 storeList[1]="Store_1";
 storeList[2]="Store_2";
 storeList[3]="Store_3";
 jComboBox_store=new JComboBox((Object[])storeList);

 jComboBox_store.setBounds(new Rectangle(178, 53, 156, 27));
 jComboBox_store.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jComboBox_store_actionPerformed(e);
  }
 });
 jLabel1.setFont(new java.awt.Font("Dialog", 1, 13));
 jLabel1.setText("Choose store(s):");
 jLabel1.setBounds(new Rectangle(53, 53, 108, 26));

 panel4.add(jRadioButton2, null);
 panel4.add(jRadioButton1, null);
 panel4.add(jToggleButton1, null);
 panel4.add(jButton_return, null);
 panel4.add(jLabel1, null);
 panel4.add(jComboBox_store, null);
 panel4.add(jRadioButton3, null);
 this.add(panel2, null);
 this.add(panel4, null);
 this.add(panel5, null);
 bGroup.add(jRadioButton1);
 bGroup.add(jRadioButton2);
 bGroup.add(jRadioButton3);
 OrderChoose="Product";
 this.setLayout(null);
 panel5.setBounds(new Rectangle(4, 14, 383, 111));
 panel5.setLayout(null);
 this.setForeground(Color.red);
 this.setBackground(SystemColor.activeCaptionBorder);
 this.addWindowListener(new java.awt.event.WindowAdapter() {
  public void windowClosing(WindowEvent e) {
   this_windowClosing(e);
  }
 });

 panel2.setBounds(new Rectangle(41, 128, 332, 20));
 panel2.setLayout(null);
 panel5.add(jButton_help, null);
```

```
    panel5.add(label1, null);
    jRadioButton1.isSelected();
   }

   void jRadioButton1_actionPerformed(ActionEvent e) {
    OrderChoose="Product";
   }

   void jRadioButton2_actionPerformed(ActionEvent e) {
    OrderChoose="Catagory";
   }

   void jRadioButton3_actionPerformed(ActionEvent e) {
    OrderChoose="Customer";
   }

   void jToggleButton1_actionPerformed(ActionEvent e)
 {
    if (OrderChoose=="Product")
    {
     OrderProducts f1=new OrderProducts(storeName,OrderChoose);
     f1.setTitle("Order Detail Information (Products)");
     f1.pack();
     f1.setSize(400,300);
     f1.setVisible(true);
    }
    else if (OrderChoose=="Catagory")
    {
     OrderCategories f2=new OrderCategories(storeName,OrderChoose);
     f2.setTitle("Order Detail Information (Categories)");
     f2.pack();
     f2.setSize(400,300);
     f2.setVisible(true);
    }
    else if (OrderChoose=="Customer")
    {
     OrderCustomers f3=new OrderCustomers(storeName,OrderChoose);
     f3.setTitle("Order Detail Information (Customers)");
     f3.pack();
     f3.setSize(400,300);
     f3.setVisible(true);
    }
  }

 void this_windowClosing(WindowEvent e) {
    setVisible(false);
  }

  void jButton_return_actionPerformed(ActionEvent e) {
    setVisible(false);
  }

  void jButton_help_actionPerformed(ActionEvent e) {
   salesOrderInfo_TopicBox dlg2 = new salesOrderInfo_TopicBox(this);
   Dimension dlgSize = dlg2.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg2.setModal(true);
   dlg2.show();
  }

  void jComboBox_store_actionPerformed(ActionEvent e) {
      String temp=new String();
      temp=((String)jComboBox_store.getSelectedItem());
      storeName=temp.trim();
    }

  public  String checkStr(String str)
  {
    char[] ctr=new char[str.length()+10];
    String f="'d";
    int j=0;
    for(int i=0;i<str.length();i++)
    {
        char ch=str.charAt(i);
        if(ch==f.charAt(0)){
          ctr[j]=ch;
          j=j+1;
          ctr[j]=ch;
          j=j+1;
        }else{
      ctr[j]=ch;
          j=j+1;
        }
    }
   return new String(ctr);
  }
}
```

Module 5. Tracing Chosen Category Products Sales Order

//Implement for tracing sales order information of the chosen category products for all stores or one store

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import jdbc.*;
import viewBean.*;

public class OrderCategories extends Frame {
 String storeName=new String();
 String orderChoose=new String();

 Panel panel1 = new Panel();
 Label label1 = new Label();
 Panel panel2 = new Panel();
 JComboBox jComboBox1;// = new JComboBox();
 JLabel jLabel1 = new JLabel();
 JButton jButton1 = new JButton();
 String itemName=new String();
 final String dbDriver="sun.jdbc.odbc.JdbcOdbcDriver";

 ResultSetMetaData rsmd;
 JCheckBox jCheckBox1 = new JCheckBox();
 boolean isTracing=false;
 JButton jButton_cancel = new JButton();
 JButton jButton_help = new JButton();
 JTextField jTextField1 = new JTextField();
 JLabel jLabel2 = new JLabel();

 public OrderCategories(String name,String choose) {
  storeName=name.trim();
  orderChoose=choose.trim();

  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
```

```
}
private void jbInit() throws Exception {
  this.setLayout(null);
  panel1.setBounds(new Rectangle(7, 74, 395, 89));
  panel1.addHierarchyBoundsListener(new java.awt.event.HierarchyBoundsAdapter() {
    public void ancestorMoved(HierarchyEvent e) {
      panel1_ancestorMoved(e);
    }
  });
  panel1.setLayout(null);
  label1.setFont(new java.awt.Font("Dialog", 1, 18));
  label1.setForeground(Color.red);
  label1.setText("Sales Order Information (Categories)");
  label1.setBounds(new Rectangle(19, 8, 332, 27));
  panel2.setBounds(new Rectangle(5, 173, 379, 116));
  panel2.setLayout(null);

  jLabel1.setFont(new java.awt.Font("Dialog", 1, 14));
  jLabel1.setText("Choose a Category:");
  jLabel1.setBounds(new Rectangle(27, 14, 144, 25));
  jButton1.setFont(new java.awt.Font("Dialog", 1, 14));
  jButton1.setText("Search");
  jButton1.setBounds(new Rectangle(156, 67, 84, 23));
  jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton1_actionPerformed(e);
    }
  });
  this.setBackground(SystemColor.activeCaptionBorder);
  this.addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      this_windowClosing(e);
    }
  });

  jCheckBox1.setFont(new java.awt.Font("Dialog", 1, 14));
  jCheckBox1.setText("tracing");
  jCheckBox1.setBounds(new Rectangle(37, 66, 75, 27));
  jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jCheckBox1_actionPerformed(e);
    }
  });
  jButton_cancel.setFont(new java.awt.Font("Dialog", 1, 14));
  jButton_cancel.setActionCommand("Cancel");
  jButton_cancel.setText("Cancel");
```

```
 jButton_cancel.setBounds(new Rectangle(260, 67, 86, 23));
 jButton_cancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jButton_cancel_actionPerformed(e);
  }
 });
 jButton_help.setActionCommand("jButton_help");
 jButton_help.setText("Help");
 jButton_help.setBounds(new Rectangle(30, 43, 69, 26));
 jButton_help.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jButton_help_actionPerformed(e);
  }
 });

 jdbc items=new jdbc();
 items.setClassname(dbDriver);
 items.setIsQuery(true);

 items.setUrl("jdbc:odbc:source-store1");
 String itemQuery="SELECT [Categories].[CategoryName]"+
 " FROM Categories;";
 items.setQuery(itemQuery);
 int itemNum;
 String[] itemList;
 try{
   items.go();
   Vector itemVector=items.getResult();
   itemNum=items.getRowCount();
   final int row=itemNum;
   itemList=new String[row];
   String[] temp=new String[1];
   for(int i=2 ; i<=row;i++){
    temp=(String[])itemVector.elementAt(i-1);
    itemList[i-2]=temp[0];
   }
   jComboBox1=new JComboBox((Object[])itemList);
   }catch (Exception ex) {
       ex.printStackTrace();
     }

jComboBox1.setBounds(new Rectangle(173, 16, 170, 25));
jComboBox1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jComboBox1_actionPerformed(e);
  }
  });
  jTextField1.setBounds(new Rectangle(128, 53, 115, 25));
  jTextField1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jTextField1_actionPerformed(e);
    }
  });
  jTextField1.setEditable(false);
  jTextField1.setText(storeName);
  jLabel2.setFont(new java.awt.Font("Dialog", 1, 13));
  jLabel2.setText("Store name:");
  jLabel2.setBounds(new Rectangle(34, 51, 89, 29));
  panel2.add(jLabel1, null);
  panel2.add(jComboBox1, null);
  panel2.add(jCheckBox1, null);
  panel2.add(jButton1, null);
  panel2.add(jButton_cancel, null);
  this.add(jButton_help, null);
  this.add(panel1, null);
  panel1.add(jLabel2, null);
  panel1.add(label1, null);
  panel1.add(jTextField1, null);
  this.add(panel2, null);
}
 void jComboBox1_actionPerformed(ActionEvent e) {
     String temp=new String();
     temp=checkStr((String)jComboBox1.getSelectedItem());
     itemName=temp.trim();
     System.out.println(itemName);
}
public  String checkStr(String str)
{
   char[] ctr=new char[str.length()+10];
   String f="'d";
   int j=0;
   for(int i=0;i<str.length();i++)
   {
        char ch=str.charAt(i);
        if(ch==f.charAt(0)){
          ctr[j]=ch;
          j=j+1;
          ctr[j]=ch;
          j=j+1;
        }else{
      ctr[j]=ch;
         j=j+1;
        }
    }
    return new String(ctr);
  }

  public String getViewName()
  {
   Calendar c = Calendar.getInstance();
   java.util.Date myDate=c.getTime();
   String temp=(new Integer(myDate.getHours())).toString();
   temp=temp+(new Integer(myDate.getMinutes())).toString();
   temp=temp+(new Integer(myDate.getSeconds())).toString();
   return temp;
  }

 void jButton1_actionPerformed(ActionEvent e) {

   jdbc inv=new jdbc();

   orderDisplayCategory salesOrder=new orderDisplayCategory(itemName,storeName);
   inv.setClassname("sun.jdbc.odbc.JdbcOdbcDriver");
   inv.setIsQuery(true);
   String itemQuery="select [Products].[ProductName], [Customers].[CompanyName],"+
       "[Order Details].[Quantity], [Products].[UnitsInStock], "+
       "[Order Details].[UnitPrice], [Categories].[CategoryName], "+
       "[Orders].[OrderDate], [Orders].[ShippedDate] FROM "+
       "(Customers INNER JOIN Orders ON [Customers].[CustomerID]"+
       "=[Orders].[CustomerID]) INNER JOIN ((Categories INNER "+
       "JOIN Products ON [Categories].[CategoryID]=[Products]."+
       "[CategoryID]) INNER JOIN [Order Details] ON [Products]."+
       "[ProductID]=[Order Details].[ProductID]) ON [Orders]."+
       "[OrderID]=[Order Details].[OrderID] WHERE [Categories].[CategoryName]='";

   itemQuery=itemQuery+itemName + "';";
   Vector resultVector;
   viewBean jdbcBean = new viewBean();
   jdbcBean.setClassname("sun.jdbc.odbc.JdbcOdbcDriver");
   jdbcBean.setViewUrl("jdbc:odbc:trace-store0");
   String viewName=new String();
   viewName=getViewName();
   jdbcBean.setViewName(viewName);
   jdbcBean.setQuery(itemQuery);
   try {
         jdbcBean.creatViewer();
         String[] metaData;
         if (storeName=="All_stores")
    {
    itemQuery = "SELECT [D1].[ProductName], "+
          "sum([D1].[Quantity]) as [Total Sales Order],"+
          " avg([D1].[UnitPrice]) as [Average Sales Price] FROM "+
          viewName+" D1 GROUP BY [D1].[ProductName]; ";
    }
    else if (storeName=="Store_1")
    {
       itemQuery = "SELECT [D1].[ProductName], "+
          "sum([D1].[Quantity]) as [Total Sales Order],"+
          " avg([D1].[UnitPrice]) as [Average Sales Price] FROM "+
          viewName+" D1 WHERE [D1].Store_Location='Store 1'"+
          " GROUP BY [D1].[ProductName]; ";
     }
    else if (storeName=="Store_2")
    {
          itemQuery = "SELECT [D1].[ProductName], "+
          "sum([D1].[Quantity]) as [Total Sales Order],"+
          " avg([D1].[UnitPrice]) as [Average Sales Price] FROM "+
          viewName+" D1 WHERE [D1].Store_Location='Store 2'"+
          " GROUP BY [D1].[ProductName]; ";
    }
    else if (storeName=="Store_3")
    {
          itemQuery = "SELECT [D1].[ProductName], "+
          "sum([D1].[Quantity]) as [Total Sales Order],"+
          " avg([D1].[UnitPrice]) as [Average Sales Price] FROM "+
          viewName+" D1 WHERE [D1].Store_Location='Store 3'"+
          " GROUP BY [D1].[ProductName]; ";
    }
inv.setQuery(itemQuery);
inv.setUrl("jdbc:odbc:trace-store0");
inv.go();
resultVector=inv.getResult();
final int a=inv.getColumnCount();
final int b=inv.getRowCount();
String[][] tableData;
tableData=new String[b-1][a];
for(int m=2;m<=b;m++)
{
 metaData=(String[])resultVector.elementAt(m-1);
 for(int n=1; n<=a;n++)
 {
 tableData[m-2][n-1]=metaData[n-1];
 }
}
```

```
 salesOrder.setMetaData((String[])resultVector.elementAt(0));
 salesOrder.setTableData(tableData);
 } catch (Exception ex) {
    ex.printStackTrace();
}
if(isTracing)
{
   if (storeName=="All_stores")
   {
     itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName],"+
            " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
            "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 2';";
             inv.setQuery(itemQuery);
     try {
           inv.setUrl("jdbc:odbc:trace-store0");
           inv.go();
           resultVector=inv.getResult();
           String[] metaData;
           final int a=inv.getColumnCount();
           final int b=inv.getRowCount();
           String[][] tableData;
           tableData=new String[b-1][a];
           for(int m=2;m<=b;m++)
           {
            metaData=(String[])resultVector.elementAt(m-1);
            for(int n=1; n<=a;n++)
            {
             tableData[m-2][n-1]=metaData[n-1];
            }
           }
           salesOrder.setMetaData1((String[])resultVector.elementAt(0));
           salesOrder.setTableData1(tableData);
       } catch (Exception ex) {
       ex.printStackTrace();
      }

    itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName],"+
            " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
            "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 1';";
    inv.setQuery(itemQuery);

      try {
            inv.go();
            resultVector=inv.getResult();
            String[] metaData;
            final int a=inv.getColumnCount();
```

```
                final int b=inv.getRowCount();
                String[][] tableData;
                tableData=new String[b-1][a];
                for(int m=2;m<=b;m++)
                {
                  metaData=(String[])resultVector.elementAt(m-1);
                  for(int n=1; n<=a;n++)
                  {
                    tableData[m-2][n-1]=metaData[n-1];
                  }
                 }
                salesOrder.setMetaData2((String[])resultVector.elementAt(0));
                salesOrder.setTableData2(tableData);
            } catch (Exception ex) {
            ex.printStackTrace();
          }

        itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName], "+
                " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
                "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 3';";
                inv.setQuery(itemQuery);

          try {
                inv.go();
                resultVector=inv.getResult();
                String[] metaData;
                final int a=inv.getColumnCount();
                final int b=inv.getRowCount();
                String[][] tableData;
                tableData=new String[b-1][a];
                for(int m=2;m<=b;m++)
                {
                  metaData=(String[])resultVector.elementAt(m-1);
                  for(int n=1; n<=a;n++)
                  {
                    tableData[m-2][n-1]=metaData[n-1];
                  }
                 }
                salesOrder.setMetaData3((String[])resultVector.elementAt(0));
                salesOrder.setTableData3(tableData);
            } catch (Exception ex) {
            ex.printStackTrace();
          }
      }
      else if (storeName=="Store_1")
      {
```

```
itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName], "+
        " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
        "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 1';";
        inv.setQuery(itemQuery);
 try {
      inv.setUrl("jdbc:odbc:trace-store0");
      inv.go();
      resultVector=inv.getResult();
      String[] metaData;
      final int a=inv.getColumnCount();
      final int b=inv.getRowCount();
      String[][] tableData;
      tableData=new String[b-1][a];
      for(int m=2;m<=b;m++)
      {
       metaData=(String[])resultVector.elementAt(m-1);
       for(int n=1; n<=a;n++)
       {
        tableData[m-2][n-1]=metaData[n-1];
       }
      }
      salesOrder.setMetaData1((String[])resultVector.elementAt(0));
      salesOrder.setTableData1(tableData);
   } catch (Exception ex) {
   ex.printStackTrace();
  }
}
else if (storeName=="Store_2")
{
 itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName], "+
        " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
        "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 2';";
        inv.setQuery(itemQuery);
 try {
      inv.setUrl("jdbc:odbc:trace-store0");
      inv.go();
      resultVector=inv.getResult();
      String[] metaData;
      final int a=inv.getColumnCount();
      final int b=inv.getRowCount();
      String[][] tableData;
      tableData=new String[b-1][a];
      for(int m=2;m<=b;m++)
      {
       metaData=(String[])resultVector.elementAt(m-1);
       for(int n=1; n<=a;n++)
```

```
              {
               tableData[m-2][n-1]=metaData[n-1];
              }
             }
             salesOrder.setMetaData1((String[])resultVector.elementAt(0));
             salesOrder.setTableData1(tableData);
         } catch (Exception ex) {
         ex.printStackTrace();
        }
    }
   else if (storeName=="Store_3")
   {
    itemQuery= "select D1.ProductName, D1.CompanyName as [CustomerName], "+
            " D1.Quantity, D1.UnitPrice, D1.OrderDate "+
            "FROM "+ viewName +" D1 WHERE D1.Store_Location='Store 3';";
           inv.setQuery(itemQuery);
    try {
            inv.setUrl("jdbc:odbc:trace-store0");
            inv.go();
            resultVector=inv.getResult();
            String[] metaData;
            final int a=inv.getColumnCount();
            final int b=inv.getRowCount();
            String[][] tableData;
            tableData=new String[b-1][a];
            for(int m=2;m<=b;m++)
            {
             metaData=(String[])resultVector.elementAt(m-1);
             for(int n=1; n<=a;n++)
             {
              tableData[m-2][n-1]=metaData[n-1];
             }
            }
            salesOrder.setMetaData1((String[])resultVector.elementAt(0));
            salesOrder.setTableData1(tableData);
        } catch (Exception ex) {
        ex.printStackTrace();
       }
     }
  }
 salesOrder.setTitle1("Sales Order Detail Display (Category)");
 salesOrder.setTitle("Sales Order Detail Display (Category)");
 salesOrder.tableGo();
 salesOrder.pack();
 salesOrder.setVisible(true);
  if(isTracing){
        if (storeName=="All_stores"){
           salesOrder.setSize(680,680);
        }
        else{
           salesOrder.setSize(680,370);
          }
      }
      else{
        salesOrder.setSize(680,240);
        }
      setVisible(false);
  }

  void jCheckBox1_actionPerformed(ActionEvent e) {
     if(jCheckBox1.isSelected()==true){
      isTracing=true;
     }else{
       isTracing=false;
     }
  }

 void this_windowClosing(WindowEvent e) {
    setVisible(false);
  }

  void jButton_cancel_actionPerformed(ActionEvent e) {
     setVisible(false);
  }

  void jButton_help_actionPerformed(ActionEvent e) {
   orderCategories_TopicBox dlg2 = new orderCategories_TopicBox(this);
   Dimension dlgSize = dlg2.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg2.setModal(true);
   dlg2.show();
  }
```