DISCOVERING DATA LINEAGE IN DATA WAREHOUSE: METHODS AND TECHNIQUES FOR TRACING THE ORIGINS OF DATA IN DATA-WAREHOUSE

By

Roselie B. Webjornsen

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
UNIVERSITY OF OSLO
OSLO, NORWAY
AUGUST 2005

UNIVERSITY OF OSLO

Date: **August 2005**

Author:	**Roselie B. Webjornsen**
Supervisors:	**Naci Akkok and Judith Gregory**
Title:	**Discovering Data Lineage in Data Warehouse: Methods and techniques for Tracing the origins of data in data-warehouse**
Department:	**Informatics**
Degree:	**M.Sc.**

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

A data warehouse enables enterprise-wide analysis and reporting functionality that is usually used to support decision-making. Data warehousing system integrates data from different data sources. Typically, the data are extracted from different data sources, then transformed several times and integrated before they are finally stored in the central repository. The extraction and transformation processes vary widely - both in theory and between solution providers. Some are generic, others are tailored to users' transformation and reporting requirements through hand-coded solutions. Most research related to data integration is focused on this area, i.e., on the transformation of data. Since data in a data warehouse undergo various complex transformation processes, often at many different levels and in many stages, it is very important to be able to ensure the quality of the data that the data warehouse contains. The objective of this thesis is to study and compare existing approaches (methods and techniques) for tracing data lineage, and to propose a data lineage solution specific to a business enterprise data warehouse.

Acknowledgements

First and foremost, I thank God for blessing me with family, friends and people who helped me make this thesis a realization.

I thank my family for their love and support; my parents for teaching me the value of hardwork, my brothers for believing in me, my sons for understanding my busy schedule and most especially my dear husband, Henry, who sacrificed his desires and convenience so that I could write and complete this thesis.

My heartfelt thanks to my supervisors; Naci Akkok who unselfishly shared his knowledge and insights with depth and humor and Judith Gregory for helping and guiding me and teaching me how to write a thesis. If it wasn't for both of you, I would not have come this far.

I thank my leaders and managers, especially Morten Morch who believes in me, motivates me and causes me to perceive things in their proper perspective.

I thank my colleagues and friends, Ragnar, Cathy, Armida, Aurea and many others who have inspired me in many different ways.

Chapter 1

Introduction

A data warehouse enables enterprise-wide analysis and reporting functionality that is usually used to support decision-making. Data warehousing system integrates data from different data sources. Typically, the data are extracted from different data sources, then transformed several times and integrated before they are finally stored in the central repository. The extraction and transformation processes vary widely - both in theory and between solution providers. Some are generic, others are tailored to users' transformation and reporting requirements through hand-coded solutions. Most research related to data integration is focused on this area, i.e., on the transformation of data. Since data in a data warehouse undergo various complex transformation processes, often at many different levels and in many stages, it is very important to be able to ensure the quality of the data that the data warehouse contains. The objective of this thesis is to study and compare existing approaches (methods and techniques) for tracing data lineage, and to propose a data lineage solution specific to a business enterprise data warehouse.

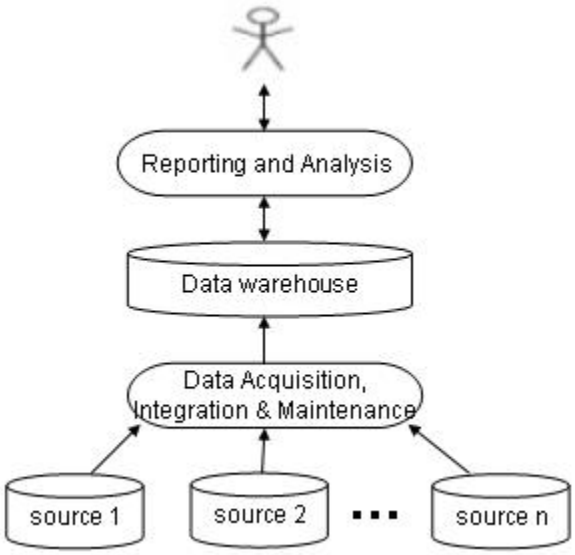

Figure 1.1: Basic data warehousing architecture - adapted from [7], page 2

1.1 Architecture of Data Warehousing System

Data warehousing is the process of extracting, cleaning, transforming, and loading data into a data warehouse for reporting and analysis [6]. Figure 1.1 (improvised diagram from Cui's thesis [7]), illustrates a basic architecture of data warehousing system consisting of two major components, namely; the Data Acquisition, Integration and Maintenance, and the Reporting and Analysis. The data Acquisition Integration and Maintenance component encompasses complex and numerous processes which mainly involve data extraction, cleaning, transformation and loading processes. The data in the data warehouse act as the central point of data integration (data are gathered and integrated from different sources) as well as the point of distribution (data are delivered to consumers of information for reporting, analysis and data mining) [26].

1.1.1 Data Acquisition, Integration and Maintenance

Data acquisition and integration is an important process in data warehousing. The data in the data warehouse passes through cleansing and integration processes before it enters the data warehouse [27]. Melding data from heterogenous and disparate sources is a major challenge (e.g. given the differences in naming, domain definitions, identification numbers, and the like) [18]. Data integration processes are indeed complex and prone to error. Since data in a data warehouse is often integrated from a variety of sources and transformed by complex processes, the original source is often obscured.

Data Transformation

Data Acquisition and Integration is generally part of Extraction Transformation and Loading, ETL. ETL is generally referred to as **Data Transformation** and is a well known process cycle inherent in warehousing environment [13]. The typical end result of an ETL process in a warehousing environment are data stored in multidimensional schema. The ETL typically contains Transformation programs which perform data cleansing, integration, and summarization tasks over the source data before loading them into the warehouse [7]. In this thesis, we evaluated existing data transformation solutions and studied their approaches to data lineage. The result of the evaluation forms part of the basis of the proposed data lineage solution.

Maintenance

Maintaining the objects, processes and data is a major administrative task in data warehousing systems. Data lineage reporting plays a very important role in data

warehouse maintenance tasks. Being able to view the origins of data is very important from design to implementation. A common warehouse administrative task for example is changing, updating or just plain analyzing a specific transformation task (e.g. analysis of sources and targets mapping). Another common example for administrative task is analysing the data quality uploaded to the warehouse. This thesis, presents the different approaches of data lineage maintenance implemented in existing solutions. These existing data lineage solutions are considered to form part of our proposed data lineage solution.

1.1.2 Reporting and Analysis

Providing integrated access to multiple, distributed, heterogenous databases and other information sources has become one of the leading issues in database research and industry [54]. The main objective of the data warehouse is to provide "integrated" enterprise-wide reporting and analysis. A data warehouse does not just facilitate data-intensive and frequent ad hoc queries [18], it is also used for further data exploration and data mining. In a business enterprise context, a typical example of a data mining application is enabling statistical analysis (on integrated and mined data) of the past behavior and actions of the enterprise in order to understand where the organization has been, determine its current situation, and predict or change what will happen in the future [26].

The mission of a data warehouse is to provide information to most effectively support decision-making [6, 28, 18]. One of the important factors to achieve this mission is to ensure data quality. Having the ability to navigate and "drill-down" reports to certain levels of detail supports efficient data analysis, but being able to

"drill-through" to data origin provides for a deeper level of understanding of the data in question. A question may be raised about the origin of data for validation or for legal purposes. This thesis considers Reporting and Analysis which answers questions related to data origin and data transformation history, the **data lineage**. Illustrations of existing lineage tracing mechanisms are presented in this thesis and the same mechanisms are considered to form part of our proposed data lineage system components.

1.2 Data Lineage

Discovering the origin of a specific warehouse data element is known as the *data lineage problem* [12]. Basically **data lineage** is the answer to the question, "where did this data come from?" [17]. The answer to this question is often not straightforward because the data in question can be either a derivation from a complex view[1] or a result of complex transformation [7]. Transformed data usually pass through different stages and at some point of each stage, they may undergo calculation based on complex procedural code. The "where" question does not just refer to the sources of data, but it also describes how the data was derived or transformed from its source [3, 55, 16]. In general, the data lineage of a datum consists of its source and its entire processing history [56]. Knowledge about the data sources and subsequent processing history of a piece of data is fundamental to assess the data quality and reliability [23]. For some applications, data lineage traceability is imperative (e.g. for data warehouses that produce official reports where legal dispute over data is a possibility).

[1] Often several layers of simple to complex views.

1.2.1 Data Lineage Tracing

The ability to visualize data in a reverse order, following its transformation step by step, is called lineage data process [12]. Different approaches in lineage tracing are proposed in different studies [52, 25, 4, 2, 19, 56, 7, 9, 11, 13, 55]. Each of these approaches are evaluated to form part of the basis for the data lineage solution proposed in this thesis.

1.2.2 Data Lineage Benefits and Application

Supporting lineage tracing in data warehousing environments brings several benefits and applications. Some of these benefits are enumerated below:

- **In-depth data analysis** - Being able to trace back to the origins of data is useful and sometimes even necessary for in-depth data analysis. To some application where data produced in the data warehouse is used for legal purposes, one important requirement would be data traceability.

- **Investigation of anomalies** - Sometimes data in data warehouse may seem anomalous. The analyst may pose a question, "where did this data come from?" and may require an explanation of how it is derived. In the absence of data lineage, it would be difficult and costly if not impossible to answer this question.

- **Debugging** - Data Lineage can be used to investigate the source data programs that produced anomalous data [56]. Having the possibility to report on the source-to-target mappings and seeing the transformation description (i.e. derivation process, operators) is an important aspect for data warehouse maintenance.

- **Impact Analysis** - Different forces or influences may impose changes to transformation processes or maybe even to data warehouse schemas. Businesses evolve and internal and external requirement may change over time. The impact of these changes to the data warehouse can be analyzed prior to implementation in the existence of data lineage. Impact Analysis determines which tables, columns and processes are affected by changes. In addition, lineage information allows the user to trace the impact of faulty source data or buggy programs on derived sets.

- **Facilitating data mining and data discovery** - Knowing about the sources of data can enhance data mining and data discovery processes. It also improves confidence on data used for data mining.

- **Correct Data in the Operational System** - Data Lineage can trace back to cleansed data and can be used as a feedback loop to correct data back in the operational system.

1.3 Research Problem and Contribution to Knowledge

This section discusses the data lineage problem and the overview of our approach in solving the problem.

1.3.1 Data Lineage Problem

In a real data warehouse implementation, data are not only derived by fixed sets of operators or algebraic properties. Most often data are transformed by a complex programming procedure. In practice, these mean sometimes as many as 60 or more

transformations [7]. Transformation can often be so complex that implementing data lineage tracing is considerably difficult and open-ended [8, 7]. In addition, some of the transformation logic is customized and hand-coded and may not adhere to any defined formal transformation. As the data traverse through different transformations, the traces to the data origin become obscure (e.g. the format is changed) and somewhere in between the complexity of transformation process, an anomaly may occur. Without data lineage traces, discovering this anomaly would be difficult if not impossible. These and many other aspects in data lineage make implementing a formal approach to lineage tracing a big challenge. This challenge stirs the interest of the research community resulting in a number of studies presenting formal calculation (i.e. algorithms and standard procedures) for tracing data lineage [7, 56, 55, 21, 2, 25, 4, 19, 8, 11, 10]. Existing solutions in the industry provide lineage solutions at higher granularity level [34, 14, 53, 44, 39, 24, 17, 38, 28, 49], but lack the effectiveness of the finest level of lineage granularity that academe presents. In addition, most business enterprise data warehouse implementation blueprints do not include a clear strategy for tracing data lineage [51]. This thesis combines and distills the different data lineage existing solutions both from university research and industry and formulates a conceptual framework for a data lineage system.

1.3.2 Academic Research

In academic research, different approaches in lineage tracing are introduced, from coarse-grained [4, 2, 25] to fine-grained [56, 45, 7] data lineage. Most existing works on data lineage in the academe focus on providing algorithms and procedures to trace back data from sources. Although, this thesis does not go into detail on algorithms and procedures, we provide illustrations on the existing tracing mechanisms

introduced in different data lineage studies. The purpose of these illustrations is to present some details of pertinent concepts behind the data lineage solution introduced in this thesis. Data lineage problem is a challenge that needs more than just lineage tracing mechanisms; it requires strategic initiatives [51]. This thesis provides a higher level of data lineage solution by introducing a conceptual framework for data lineage systems components.

1.3.3 Industry Implementation

This thesis presents different data lineage solutions implemented by some prominent players in industry. The intention is to provide an overview of existing features that we combine to our data lineage solution - a solution which attempts to cover the different aspects of the data lineage problem at a strategic level. The prominent solution providers[2] of data integration which we included in our evaluation show data lineage features. However, to the best of my knowledge, these lineage features do not offer the finest level of granularity, which the academic works introduce. For example, the industry solutions provide data lineage reporting functionality that reports only against the metadata [34, 14, 53, 44, 39, 24, 17, 38, 28, 49]. These features can provide reporting about sources and targets (i.e. sources-to-target mapping) and may allow "drill-down" to transformation descriptions. However having the ability to "drill-through", or to trace back the data lineage in step-wise fashion, offers an avenue of improvement. This is one of the important considerations in data lineage solution introduced in this thesis, although this thesis does not detail data lineage reporting

[2]Data integration solution providers included in this thesis are SAP Business Information Warehouse, Oracle, Teradata, Microsoft and Ascential Software.

mechanisms. Metadata[3] plays an important role in data lineage solutions. For this reason we also evaluated the metamodel[4] standardization efforts provided by OMG[5], the Common Warehouse MetaModel (CWM)[6] [40]. CWM provides constructs that support data lineage solutions. This thesis considered the metamodel constructs as a basis for the part of the data lineage solution covering transformation metadata design.

1.3.4 Data Lineage Solution

The data lineage problem is seen in different angles and perspective in the academe and industry. Different data lineage approaches show partial solutions addressing some aspects of the data lineage problem. Most of the academic works consider the data lineage on a data value granularity level, while the industry focuses more on schema mappings. In industry, the schema mapping report is considered as a data lineage report, but in the academe, the data lineage report not only produce schema, schema elements and transformation descriptions, it also produces lineage data values themselves. The academe details formal approaches for data lineage computation (algorithms and procedures), while the industry provides features and tools that support data lineage. Academic research may seem to provide better solutions in terms of data lineage solution concepts; however industry provides tangible solutions as evidenced by real data warehouse implementations. In this thesis, we combine and

[3]Metadata is generally known as the data about the data. A typical example is the system catalog in relational DBMS. The system catalog describes data definition (i.e. it describes the tables and columns that contain your data.) Some tools can manipulate system catalog as you would manipulate any other data. Examples of manipulating metadata include viewing data lineage or table information. In this case, metadata are treated as ordinary data.

[4]Metadata model

[5]Object Management Group.

[6]Metamodel Standard specification mainly focusing on interchanging metadata.

distill different approaches to data lineage problem. First, we describe data lineage by presenting the different aspects of the data lineage problem and classifying different types of data lineage information. This thesis presents a higher level but holistic data lineage solution by providing a conceptual framework for data lineage systems components relevant to business enterprise contexts. This work can be used as a reference or a basis for designing and integrating data lineage solution to the business enterprise data warehouse.

1.4 Related Works

The data lineage problem attracts interest in the research community. In this section, we present the different works related to data lineage. Most of these works generally provide data lineage tracing approach which help constructs the solution proposed in this thesis.

Fan and Poulovassilis [21], introduced lineage tracing approach based on schema transformation pathways. In this work, they show how individual transformation steps is used to trace the derivation of the integrated data in a step-wise fashion. However their work focuses on a Hypergraph Data Model, (HDM) which is designed to suit automed integration system [20, 22], this thesis focuses more on Relational Data Models.

Other earlier works offered coarse-grained or schema-level lineage tracing. The approaches propose lineage tracing based on annotations or attributions [25, 4, 2]. [25, 2] use *data derivation information* which is stored in metadata. Faloutsos, Jagadish and Sidiropoulos in [19] also use metadata in addition to condensed information (e.g. summary data) to trace data lineage. Statistical calculation is performed against

the metadata and condensed information to reconstruct the estimated value of data origin.

The approach proposed by Woodruff and Stonebraker does not use metadata. Rather than relying on metadata, their approach computes lineage using a limited amount of information about the processing operators and about the base data [56]. [56] introduced lineage tracing based on weak inversion. Weak inversion does not perfectly invert the data, it uses weak inversion and verification to provide a number of guarantees about the lineage it generates. The *weak inversion* and *verification* functions are to be registered by the user to the DBMS. When tracing a data lineage, an inversion planner determines which weak inversion and verification function to invoke, constructs a plan and then executes the plan by calling the corresponding sequence of functions within the DBMS.

All of the data lineage approaches above do not guarantee accurate and exact tuple or tuples of the origin, but these approaches are potentially feasible in certain situations and can be practical solutions to some business enterprise-specific requirements.

Cui and Widom offer lineage tracing algorithms for relational views with aggregation [12]. These algorithms provide a means for users to select a data warehouse view tuple and "drill-through" to examine the exact source tuple or tuples that produced the view. Lineage tracing involves other aspects also, such as performance and maintenance. In [10], Cui and Widom provide a way of optimizing lineage tracing by introducing auxiliary views. Auxiliary views are parts of data lineage which are physically stored in the data warehouse. Cui and Widom, in [8] further introduced

algorithms based on general transformation properties[7]. For each transformation property, an equivalent tracing procedure is used to enable lineage tracing. Cui's and Widom's works gained significant contribution in data lineage problem and are cited in many data lineage related works.

Bose does not introduce a data lineage tracing mechanism but rather presents a framework for composing and managing data lineage specific to scientific data [3]. This thesis provides lineage solution for business enterprise data warehouse by characterizing data lineage and providing framework for data lineage system components.

1.5 Thesis Scope and Delimitation

This thesis evaluates and describes different lineage tracing mechanisms in theory as well as the existing implementations in the industry. Existing theories which form part of the data lineage solution presented in this thesis are illustrated. Additionally, existing data lineage implementations in the industry are presented. Based on the evaluations, (i.e. both in theory and industry), a data lineage solution which is specific to business enterprise context, is introduced. This thesis attempts to provide a holistic view of the different aspects of the data lineage problem and aims to introduce a solution which covers these different aspects. However this thesis does not go into detail on algorithms and procedures, although illustrations on some algorithms and procedures extracted from different academic research are provided.

[7] Transformation property tells the type of transformation applied to the data.

1.6 Thesis Outline

The thesis is organized as follows. Chapter 2 elaborates different tracing mechanism based on different data lineage related studies. Chapter 3 discusses the Standard metamodel specification which provides metamodel constructs that support data lineage. Additionally, the overview of data lineage implementation for each solution providers included in the evaluation in the industry is presented. Base on the evaluations, Chapter 4 characterizes data lineage by describing the different aspects of the data lineage problem and classifying the different types of data lineage. Chapter 5 proposes a data lineage solution covering the aspects of the data lineage problem discussed in chapter 4. In Chapter 5 we introduce the conceptual framework for lineage system components and discuss data lineage design and implementation. In our discussion we emphasize that lineage system Components must be seamlessly integrable with the data warehousing system. Finally, Chapter 6 discusses the conclusion and the possible future works related to the proposed solution.

Chapter 2

Existing Lineage Tracing Mechanisms

This chapter presents different tracing mechanisms which are gathered from previous works related to data lineage. Section 2.1 provides a sample case which shall be used as reference for illustrating the different data lineage approach. Section 2.2 discusses the basic lineage granularity levels as commonly described in academic works related to data lineage. In addition, section 2.2 also illustrates examples for each granularity level. Section 2.3 highlights different approaches in storing data lineage physically and section 2.4 presents the different ways for physically retrieving the data lineage for reporting purposes.

The lineage tracing mechanisms presented in this chapter are mostly based on the previous works of [25, 4, 2, 19, 56, 12, 10, 8, 52]. The main intention of this chapter is to describe and highlight the different concepts of existing tracing mechanisms. However this only focuses on the overview of each approaches and does not detail formal calculations (i.e. algorithms and procedures) presented in the referred previous works. Each or a combination of each tracing mechanisms may be applicable to certain enterprise data warehouse tracing requirements. Therefore, knowing the overview of

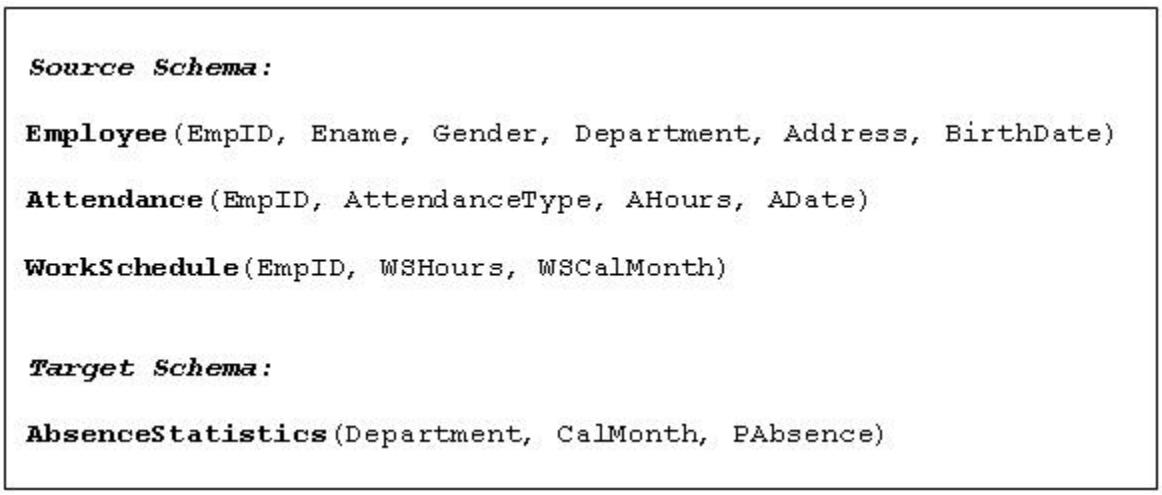

Figure 2.1: Source and Target Tables

these approaches is useful to determine which specific solution is applicable to which problem in a data warehouse environment. For details on algorithms and procedure, refer to [25, 4, 2, 19, 56, 12, 10, 8, 52].

2.1 Sample Case

This section is prepared for illustration purposes. This illustration shall be used in different tracing mechanisms presented in this chapter.

Source Data. Warehouse data *AbsenceStatistics* are produced based on three source tables: *Employee*, *WorkShedule* and *Attendance*. See to figure 2.1 which presents the source tables and target tables. Employee table is mostly self-explanatory. WorkSchedule table stores the number of hours that the employees are supposed to be working each month. The Attendance Table is also self-explanatory except for the AttendanceType which indicates if hours are entered as ordinary working hours, sick leaves or overtime. Figure 2.2 shows the content of the source tables.

Employee

EmpID	Name	Gender	Department	Address	Birthday
E1	Rose	F	D70	Oslo	11.01.1975
E2	Henry	M	D20	Oslo	31.03.1968
E3	Joshua	M	D20	Stavanger	10.01.1985
E4	Helen	F	D20	Stavanger	04.03.1972
E5	Beth	F	D40	Stavanger	30.05.1974

WorkSchedule

EmpID	WSHours	WSCalMonth
E5	157.50	1.2005
E4	157.50	1.2005
E3	157.50	1.2005
E2	127.50	1.2005
E1	157.50	1.2005
E5	150.00	2.2005
E4	150.00	2.2005
E3	150.00	2.2005
E2	120.00	2.2005
E1	150.00	2.2005
E5	172.50	3.2005
E4	172.50	3.2005
E3	172.50	3.2005
E2	135.00	3.2005
E1	172.50	3.2005

Attendance

EmpID	TimeType	Ahours	Date
E1	Ordinary	7.50	01.02.2005
E3	Ordinary	7.50	01.02.2005
E5	Ordinary	7.50	31.01.2005
E3	Overtime	2.50	12.02.2005
E2	SickLeave	7.50	01.02.2005
E3	SickLeave	7.50	03.01.2005
E2	SickLeave	7.50	03.02.2005
E3	SickLeave	7.50	04.01.2005
E2	SickLeave	7.50	04.02.2005
E3	SickLeave	7.50	05.01.2005
E3	SickLeave	7.50	06.01.2005
E3	SickLeave	7.50	07.01.2005
E1	SickLeave	7.50	07.03.2005
E1	SickLeave	7.50	08.03.2005
E1	SickLeave	7.50	09.03.2005
E1	SickLeave	7.50	10.03.2005
E1	SickLeave	7.50	11.03.2005
E1	SickLeave	7.50	14.03.2005
E1	SickLeave	7.50	15.03.2005
E1	SickLeave	7.50	16.03.2005
E1	SickLeave	7.50	17.03.2005
E1	SickLeave	7.50	18.03.2005
E2	SickLeave	7.50	31.01.2005

Figure 2.2: Source Tables Content

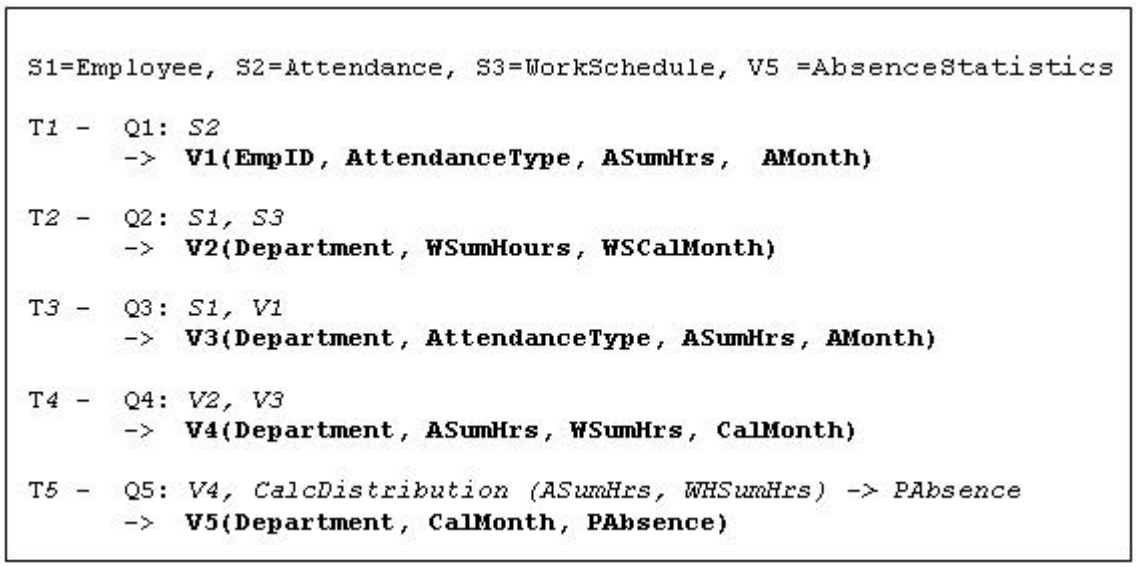

Figure 2.3: Transformation Steps - adapted from Cui [7]

Warehouse Data. Suppose that the Head of Health and Safety Environment, (HSE) Division would like to analyze healthy working environment for each department. Sickness Absence may not always be directly related to healthy working environment but may also, for example, indicate employees' work satisfaction. Therefore this warehouse view may also serve the department head to analyze the absences of his or her employees. To materialize a view in the data warehouse that has this information, the transformation steps shown in figure 2.3 are implemented.

Target Views represented by Vi where i is the sequence number of Views in each transformation steps, are views that store transformed data which may function as an intermediate view for further transformation or the ultimate data target. The transformation steps traversed by the warehouse data result in the intermediate views *V1,V2,V3,V4*, refer to figure 2.3. To simplify the transformation illustration, we use the transformation graph shown in figure 2.4, similar to Cui's illustration in [7].

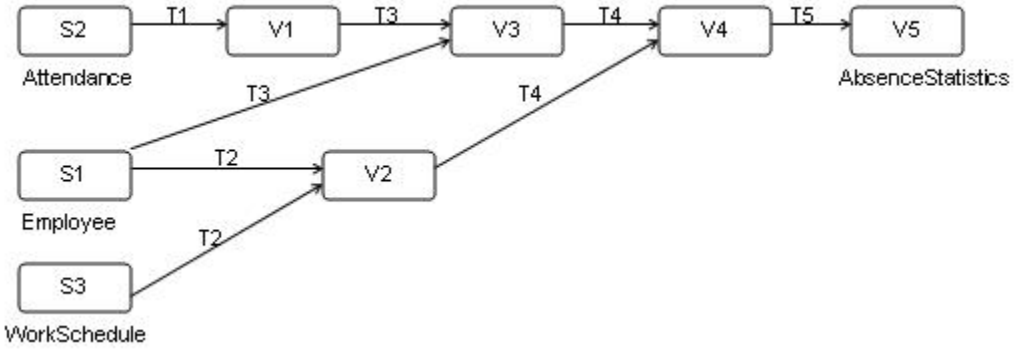

Figure 2.4: Transformation Graph - adapted from Cui [7]

AbsenceStatistic

Department	CalMonth	Pabsence
D20	1.2005	10.2
D20	2.2005	5.4
D20	3.2005	0.0
D40	1.2005	0.0
D40	2.2005	0.0
D40	3.2005	0.0
D70	1.2005	0.0
D70	2.2005	0.0
D70	**3.2005**	**43.5**

Figure 2.5: Target Content

The data shown in figure 2.5 are loaded to the warehouse after a series of transformation. Refer to figures 2.3 and 2.4.

2.2 Data Lineage Granularity Level

Lineage granularity pertains to the level of details of the data lineage when tracing back to its origin. Different data lineage related works in the academe describe data lineage granularity in two main categories; Schema level and Instance level. The first refers to the origin of data in terms of data structures (e.g. column to column mapping), the latter refers to the original values themselves from which the data are

derived. In this section, we look in to the different academic works and describe each of their data lineage tracing approaches. Section 2.2.1. presents the Schema level way of tracing the data lineage and section 2.2.2. presents the Instance level approach.

2.2.1 Schema-level approach

Schema-level or coarse-grained tracing approach refer to the origin of data not in terms of data values but in terms of schema elements and transformations from which they are derived. Schema-level approaches utilize metadata repository to store lineage information (e.g. transformation or derivation set) involving a warehouse data item [2, 52]. [2, 52] introduce a possibility to tag each warehouse instance with identifier of the derivation set that produced it. The identifier stored in the warehouse instance and the derivation set stored in the Meta data repository are then used for query computation during lineage reporting.

In [2], the equivalent of a derivation set is named a *package*. Each *package* is a workflow that defines a transformation. To enable lineage traceability, a package identifier is stored in a warehouse item. When a user asks about the data lineage of a warehouse item, the package identifier is used to view the package transformation description, (e.g. the sources and targets of the transformations and the computation applied to the data in transformation process).

Applying the method of [2, 52] in our example, we then assume that the transformation information such as transformation steps, intermediate view definitions etc., are already stored in Meta data repository . Referring to our sample case in section 2.1, the transformation Steps T*1* to T*5* and View Definitions V*1* to V*5* are assumed to be already stored in the Meta data repository.

Schema Level Tracing Scenarios

The transformation process shown in figure 2.2 and figure 2.3 can be seen as the transformation package as described in [2]. Additionally to complete the Target Schema, according to [2], we add the ***PId*** attribute which stands for Package identifier. The resulting table would then be ***AbsenceStatistics*** (*Department, CalMonth, PAbsence, PId*). To simplify our example, we refer only to the warehouse target as a one dimensional schema (typically, data in the data warehouse for OLAP are loaded in a Multidimensional Schema). Assuming that our data target is a multidimensional schema, when we add package identifier, ***PId***, we are actually adding a dimension to the schema. We name it *package* dimension for illustration purposes. Dimensions are simply tables linked to the main table which is known in data warehouse as the fact table. The package dimension that we add may contains pertinent information about the data, for example package descriptions among others. Figure 2.6 illustrates an example of the Multidimensional Schema that has a package identifier, PId. Package dimension is similar to the idea of audit dimension mentioned in [6]. Although the intention of audit dimension in [6] is to indicate data quality, it does capture important ETL processing and contains a description of the fixes and changes that have been applied to a specific row in a fact table. In addition, it may also contain more attributes that may describe the data lineage.

Reporting data lineage using the package dimension may initially use the data in the Package dimension if the user would only require lineage overview (e.g. data source, timestamps). From here, the user may trace back the transformations and queries regarding the source (including intermediate views) to target mappings. In

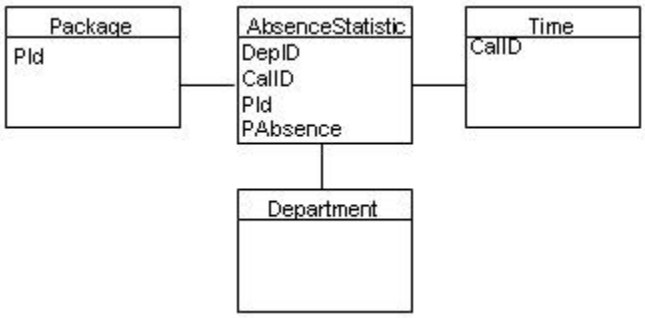

Figure 2.6: Multi-dimensional Schema

each transformation step, the user may also view information that describes the transformation step (e.g. calculation applied to the source or intermediate views resulting to intermediate or target views).

A schema-level related work in [52] details on the mapping mechanisms utilizing the Meta data repository. [52] introduced mappings in the form of **foreach** $\boldsymbol{Qs}$ and **exist** $\boldsymbol{Qt}$. $\boldsymbol{Qs}$ and $\boldsymbol{Qt}$ stands for source and target queries respectively. These Queries are used to describe the mappings. The source query describes what to retrieve from the source and the target query describes how the retrieved data will be structured to the destination schema. For each mapping, schema or schemas may be transformed in terms of element name or element types and the data may be aggregated or calculated. Figure 2.7 shows an example of this mapping to populate the target schema using our sample case in section 2.1. Consider the mapping $\boldsymbol{m1}$ in our example shown in figure 2.7. The source column *Ahours* of this example is transformed to *AsumHours* in the target schema while the data value is aggregated to summarize the "SickLeave" hours for each employee in each month.

In addition to this mapping mechanism, [52] introduced a Meta data storage schema which includes seven tables shown in figure 2.8. ***Mapping*** table is the main

```
m1:
foreach
  select a.EmpID, a.Attendancetype, sum(a.Ahours)->Asumhours, Month(Adate)->Amonth
  from   Attendace a
  where  a.Attendancetype = 'SickLeave'
exists
  select V1.empid, V1.Attendancetype, V1.Asumhours, V1.Amonth
  from   V1

m2:
foreach
  select w.Department, sum(WSHours) -> WSumHours, WSCalMonth
  from   Employee e, WorkSchedule w.
  where  e.EmpID =  w.EmpID
exists
  select V2.Department,  V2.WSumHours, WSCalMonth
  from   V2

m3:
foreach
  select e.Department, V1.  Sum(ASumHrs) -> V1.ASumHrs, V1.CalMonth
  from   Employee e, V1
  where  e.EmpID = V1.EmpID
exists
  select V3.Department, V3.ASumHrs,  V3.CalMonth
  from   V3

m4
foreach
  select V2.Department, ASumHrs 0, V2.WSumHours -> WSumHrs,
           V2.WSCalmonth ->CalMonth
  from   V2
       union
  select V3.Department, V3.ASumHrs -> WSumHrs 0,  V3.ACalmonth -> CalMonth
  from   V3
exists
  select V4.Department, V4.ASumHrs, V4.WSumHrs,  V4.CalMonth
  from   V4

m5:
foreach
  select V4.Department, Calc(V4.ASumHrs,V4.WSumHrs)->Pabsence, V.CalMonth
  from   V4
exists
  select V5.Department, V5.PAbsence, V5.CalMonth
  from   V5
```

Figure 2.7: Mapping Scenario - adapted from [52]

```
DB(name)
Element(eid,name,type,parent,db)
Query(qid)
Binding(bid, qid, eid, prev)
Condition(qid, bid, eid, op, bid2, eid2)
Mapping(mid, forQ, conQ)
Correspondece(mid, forBid, ForEid, conBid, conEid)
```

Figure 2.8: Meta Data Storage Schema - Source from [52] page 9

responsible table for linking the source-to-target schemas. A mapping table consists of *mid*, *forQ* and *conQ* columns. The *mid* is the unique key identifying a mapping instance. The two columns, *forQ* and *conQ* contain the source and target **select** clause respectively. The source query entered in **foreach** describes what to retrieve from the source and the target query entered in **exists** describes how the retrieved data will be structured to conform to target schema.

In addition to the mapping table, the source-to-target schema elements are stored in the ***Element*** relation which consists of element ID, name, type (e.g. string), parent (i.e. the origin of the element) and db (i.e. the data source). Three tables are used to contain the schema-to-target schema definitions: ***Query***, ***Binding*** and ***Condition***. Each query has an indentification which is stored in a Query table. The ***Binding*** table stores the **from** clause and ***Condition*** table stores the **where** clause of the source-to-target queries.

Looking back at our sample case in section 2.1., what if, for example, the employees within the same department serve different projects and each project has its own time-sheet application database. This scenario extends our sample case because the data source now originates from different databases. Our mapping example in figure 2.7 is extended to record the source databases of the source items.

Suppose the head of the department investigates the source of the specific tuple in the data warehouse. The information stored in Meta data repository will make it possible to query on schema elements and transformations which will show the manager how the data are transformed and which source databases the values in the warehouse came from.

2.2.2 Instance-level approach

Instance-level or fine-grained tracing attempts to find the specific values in the base tables that justify the appearance of data in a warehouse view. While schema-level tracing refers to the origin of data in terms of schema elements and transformations description, instance-level tracing refers to the data value itself. The following subsections describe each different method of tracing data lineage and provide examples. The examples given are based on the works [19, 7, 55, 9, 11, 12, 8, 7].

Views and Transformations

Widom and Cui in their works [9, 11, 12, 8, 7] describe tracing mechanisms by which the exact or nearly exact data lineage for a given warehouse data item can be produced. In their works, they provide algorithms for lineage tracing for data warehouse views and lineage tracing for general data warehouse transformations. When a warehouse materialized view is populated using standard relational operators, then it is possible to trace the exact lineage tuples for a given warehouse item using the algorithms Widom and Cui provided in [12, 9, 7]. However, in practice, a data warehouse is often populated through a complex transformation process. In this case, Widom and Cui introduced lineage tracing for general warehouse transformations. In their

works, they identified eleven transformation properties which become the basis for selecting a tracing procedure to be used in lineage tracing.

Lineage tracing for Views. This lineage tracing mechanism considers a relational scenario. The data warehouse in this scenario is populated through simple to complex relational views which are specified using select, project join, aggregation, union, intersection and difference operators. [12, 9, 7] develop tracing algorithms for this scenario. The tracing algorithms use tracing procedure that is automatically created based on the view definition.

Computing the lineage of data warehouse view generally requires not only the view definition but also the original data and sometimes even additional information. In data warehousing implementation, this requirement is not an appealing solution because data may come from different sources. For example, data may come from different systems (e.g. legacy systems) or databases, or may reside in an inaccessible logical location (e.g. source data maybe archived) among many other practical reasons.

[7, 12] introduced view lineage tracing without requiring the data source with certain practical trade-offs (e.g. loading against tracing performance). To achieve this goal, [7, 10] proposed the idea of auxiliary views. These views reside within the data warehouse environment and contains some part of lineage information (see to section 2.3.2). In addition to this, [7, 10] introduced different options to store auxiliary views by providing algorithms for their implementation and maintenance.

To illustrate view lineage tracing introduced in [7, 12], we consider our sample case in section 2.1. To simplify the illustration we consider only the transformation $\mathbf{T2}$ shown in figure 2.3 and 2.4 which produce the View $\mathbf{V2}$. Figure 2.9 a) shows

the source data which produced the View **V2**. If we apply the *Tracing Query* as introduced in [7, 12] to compute the lineage of **V2** tuple(D70, 172.5, 3.2005), then we will get the result shown in figure 2.9b. Refer to section 2.4.2 for the resulting query based on algorithms in [7, 12].

Lineage tracing for general data warehouse transformations. In real data warehouse implementations, data in the data warehouse are often populated through a series of complex transformation. In this case, tracing the data lineage cannot be just calculated using the standard relational operators. Additional logic needs to be incorporated in order to produce the data lineage for a particular data warehouse item. This problem is identified in [7, 8]. [7, 8] introduced lineage tracing basing on general transformation. The intention of this mechanism is to produce the exact or nearly exact tuples that are responsible for the appearance of the warehouse item produced by data warehouse general transformation[1].

To be able to trace back to the exact origin of data requires knowledge on the transformation path, the data traversed as well as the computation applied to the data within each transformation. Producing a data lineage as accurate as possible, requires knowledge of the transformation description. It may not be emphasized in [2, 52], but their work clearly shows, that transformation information are used to enable lineage tracing. To emphasize the notion that data lineage tracing is dependent on Transformation description, a simple example is given, shown in figure 2.10 (see [7, 8]). This figure shows a source table I that has tuples (a,-1), (a,2), and (b,0) and Transformed data having the tuples (a,2),(b,0). In this example, the data in question are the tuple (a,2). Consider two transformation scenarios 1 and 2 below:

[1]These are the common transformations that occurs in data warehouse environment.

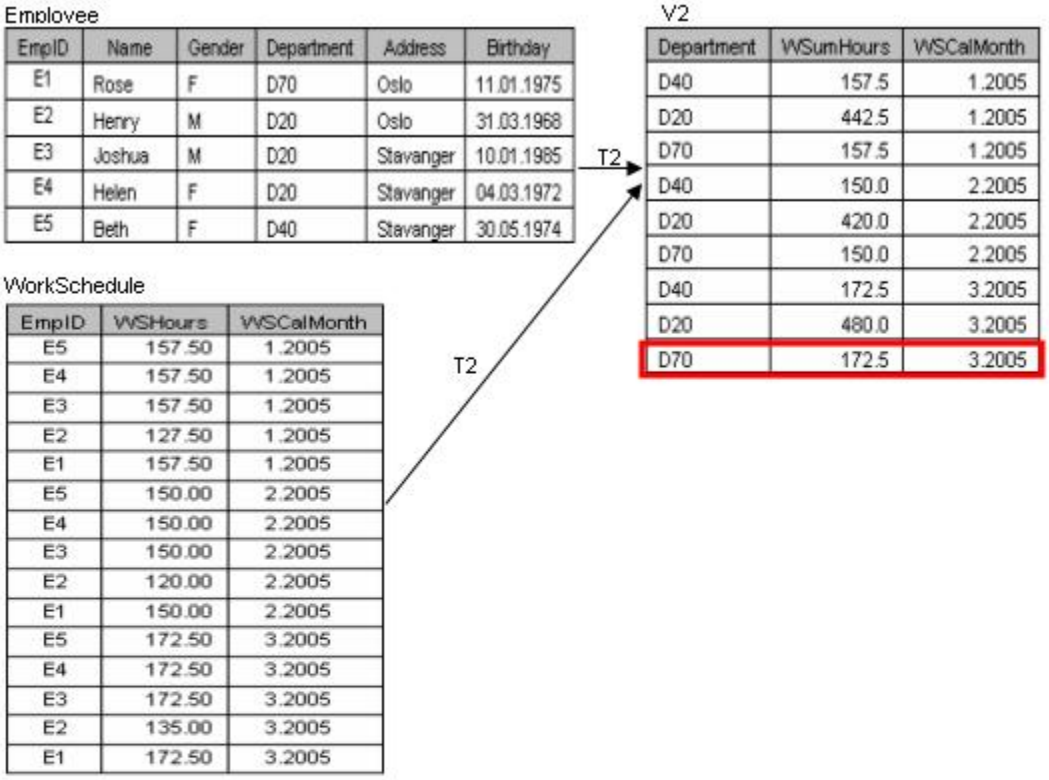

Figure 2.9: Lineage for View **V2**

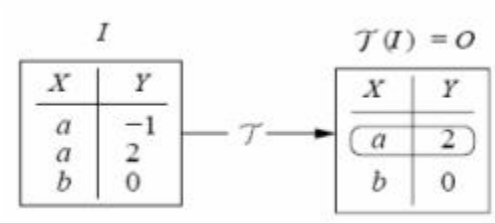

Figure 2.10: Transformation Instance - source from [7]

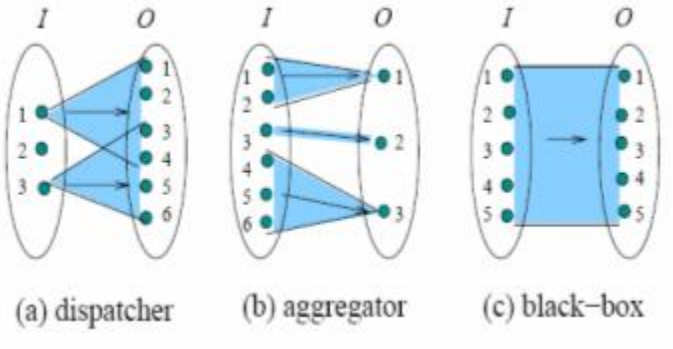

Figure 2.11: Transformation Classes - source from [7]

Scenario 1: Transformation that filters out input with negative Y value

Scenario 2: Transformation that groups input data based on X values and computes the sum of their Y values multiplied by 2

Tracing the data lineage for this data in question depends on how the data is transformed. If data are transformed as described in Scenario 1, then the data lineage for this transformation would only be the tuples (a,2) and (b,0). However, if the data are transformed as described in Scenario 2, the data lineage would be the entire table, because all tuples in the entire table participates in the transformation computation.

Transformation Properties. In reality data transformations often involve

more than just standard relational operators. Tracing the data lineage for such transformations requires some known transformation structure or properties that can be used to determine and trace the data lineage [8, 7]. [8, 7] develop lineage tracing mechanisms using the transformation properties. First, they identified transformation properties for general data warehouse transformations.

Transformation Classes. [7, 8] classifies General Transformation into three namely, the *Dispatcher, Aggregator and Black-box*, refer to figure 2.11. These three classes are based on how they map input data items to output items.

A ***Dispatcher*** produces zero or more output data items independently. Figure 2.11(a) illustrates a dispatcher, in which an input items ***I*** consisting of 1, 2 and 3 produces output items ***O*** consisting of 1, 2, 3, 4, 5 and 6 independent to each other. An input item 2 produces nothing and an input items 1 and 3 produces more outputs. A dispatcher can be identified as ***filter*** if each input item produces either itself or nothing. The data lineage for any output through transformation with ***filter*** property is the same item as in the input.

An ***Aggregator*** transforms one or more input data items and produces one output data item. Figure 2.11(b) illustrates an aggregator in which output data items 1 and 2 and 3 are transformed using one or more input data items. Output data items 1 and 3 are the transformed from multiple input items while output item 2 is a result of single input item. A more efficient tracing pocedure for aggregator is develop by subclassifying it in to ***context-free aggregator***and ***key-preserving aggregator***.

A ***Black-box*** illustrated in figure 2.11c) uses all combination of data set to transform one or more data sets, therefore tracing procedure for this type of transformation simply returns the entire input.

Table 2.1: Summary of Cui's Transformation Properties - source from [7] page 160

Property	**Tracing Procedure**
dispatcher	TraceDS
filter	return O
aggregator	TraceAG
context-free aggregator	TraceCF
key-preserving aggregator	TraceKP
black-box	return I
forward key-map	TraceFM
backward key-map	TraceBM
backward total-map	TraceTM
tracing procedure requiring input	TP
tracing procedure not requiring input	TP

Schema Mappings. Cui and Windom [8, 7] consider schema information to improve lineage tracing but do not entirely depend on it as Velegrakis, Miller and Mylopoulos [52] do. Schema mappings specify the link of input attributes to the output attributes. [52] details this by introducing a Meta data storage schema that contains this information. Our sample case shows schema mapping refer to figure 2.3. This is further demonstrated in figure 2.7. [8, 7] develop a lineage tracing procedure using schema mapping to improve the tracing procedure for dispatcher and aggregator. However, [8, 7] procedure does not report on mappings as [52] does. [8, 7] uses schema mappings to identify specific transformation properties based on how the source tuples attribute values is mapped to the tuples of target attributes. Three schema mapping properties are classified, namely ***forward key map***, ***backward key-map*** and ***backward total-map*** (see [8, 7] for details). A lineage tracing procedure is developed for each of these properties which allows tracing the lineage values of the source, rather than to the lineage schema of the source.

Table 2.2: Transformation Summary for *AbsenceStatistics* Warehouse

Name	Description
T*1*	Select on Attendance type
T*2*	Aggregate Employee and WorkSchedule
T*3*	Join and aggregate Employee and V1
T*4*	Union views V2 and V3
T*5*	calculate percentage of absence and add column PAbsence

A Transformation may exhibit more than one properties. Table 2.1 lists the transformation properties. Some properties are better than the others. This means that the tracing procedure formulated for better properties may give a more efficient tracing or may give a more accurate lineage result. In which case, [8, 7] determines the best one to exploit for lineage tracing and show the hierarchy according to importance.

Lineage Tracing Procedure. Tracing procedure take specified output items and may take Input items as parameters to produce the data lineage. The tracing procedures formulated in [8, 7] introduce a mechanism in which a data lineage is traced by identifying which transformation property or properties have transformed a specific data in the data warehouse. The lineage tracing procedure is composed for transformation sequence and transformation graphs.

Widom [8] formulated a tracing procedure for each of the transformation properties. Consider figure 2.4, the transformation graph in our sample case. Each transformation step is represented as Ti where $\boldsymbol{i}$ refers to the sequence number of transformation. If we consider the work of [8], each of the transformation in our sample case T*1* to T*5* has a transformation property. The summary of transformations in our sample case in section 2.1 is presented in table 2.2 based on the work of [8].

Consider the warehouse data in our sample case in figure 2.5. Suppose the Head of HSE Division would like to know who are the employees in a particular department were having sick leaves for a specific month. Specifically the HSE Head may wish to list the employees that produce the warehouse data tuple(D70, 3.2005, 43.5). Tracing back the data lineage is like traversing the path in the transformation graph shown in figure 2.4 and reconstructing the views in the relations shown in figure 2.3. In our sample case, HSE head may "drill-through" V*4* to V*1* then finally to the source. [7, 8] proposed lineage tracing mechanism that allows the user to do exactly this. Lineage tracing will traverse back the transformation path to produce the exact tuples of the origin of the data in question.

Statistical Method

Faloutsos, Jagadish and Sidiropoulos in [19] introduce a statistical method to reconstruct as good as an estimate of the original base data. [19] describes a formal approach to the recovery of information from a summary data in form of constraints and utilizes the well-developed "inverse problem" theory.

We include this method not to detail a statistical calculation approach but to recognize the relevance of this approach for practical application on data warehousing environment. Ralph Kimball [32] describes a practical example for using statistical method to answer the question about the the correctness of data loaded in a warehouse . If there is a need to respond to queries that can be answered accurately only from the base data, the views and transformation approach of lineage tracing of [8, 10, 7] are applicable. This type of question requires to "drill-through" to the exact tuple of the data origin. In some cases it may be too expensive if not impossible to trace

the exact or almost exact tuples of the origin of data. The reason may be that the source data reside on an unreachable location (e.g. off-line, archive etc.) However, some data origin related questions can be answered quickly from the summarized data. [19] proposed a mechanism to meet this requirement.

Inverse Method

Some transformation may be accompanied by a tracing procedure or inverse transformation, which is the best case for lineage tracing according to Widom [8]. Because it is the inverse of the transformation it could be assumed that it basically inverts data perfectly. Intuitively, if inverse procedure or function inverts the data perfectly, (assuming the data data is transformed in several layers and levels), then a certain amount of lineage information should have been stored within the data warehouse. In practice, inversion function is seldom available [8, 56]. ETL development is complex and time consuming that preparing an inverse function for each transformation is often not considered during the development.

Woodruff and Stonebraker [56] introduced a general framework for computing the data lineage using weak inversion and verification. This approach is to compute lineage on-demand using a limited amount of information about the processing operators and the base data. Weak inversion does not perfectly invert the data, it uses weak inversion and verification to provide a number of guarantees about the lineage it generates. The *weak inversion* and *verification* functions is to be registered by the user to the DBMS. When tracing a data lineage, an inversion planner determines which weak inversion and verification function to invoke, constructs a plan and then executes the plan by calling the corresponding sequence of functions within the DBMS.

2.3 Recording the Data Lineage

How and where to record data lineage is essentially part of the lineage problem solution. If data lineage is to be traceable, then information about data lineage or part of data lineage itself needs to be stored where lineage tracing process can reach it. Intuitively, the problem of recording data lineage involves balancing trade-offs (e.g. performance). Recording data lineage or part of data lineage after every transformation is sure to penalize the loading process in data warehouse. Thus, deciding whether to store data lineage or to reconstruct it on demand requires extreme caution. Different lineage tracing mechanism views lineage recording differently. Cui and Widom [7, 12, 10, 8] for example describe algorithms to determine which part of data lineage information to store or store nothing at all. Woodruff and Stonebraker [56] propose to register weak inversion function for each transformation and use it to reconstruct data lineage. Faloutsos, Jagadish and Sidiropoulos [19] uses limited knowledge about the data to reconstruct the data from their base origin. Whether the option is to store or reconstruct data, all tracing mechanism still use some form of information about the data lineage. This section discusses the different approaches to recording data lineage based on my analytical survey made on the existing research [25, 4, 2, 19, 56, 12, 10, 8, 52].

2.3.1 Identifying Lineage information

Based on the survey of this study, there are different types of lineage information necessary to enable lineage tracing. Lineage information can be data about the source systems, source and intermediate schemas, functions or procedures used for calculations, or source data values themselves. This section classifies some of these types

and describe each type.

Schema Information

Any information describing the composition of the schema that participates on data transformation. This information may be in the form of the following:

- **View definition** - Views that participates in transformation and may form part of lineage tracing queries used [52, 7].
- **Schema Name** - Names of views, tables, or record or attributes. relation [52, 56, 55].
- **Schema Elements** - Attributes or column names [52, 56, 55]
- **Element or Attribute Types** - the source element or attribute types and the intermediate target element or attribute type [52, 56, 55].

It may also be pieces and parts of the schema definition such as: ***select*** clause, ***from*** clause or ***where*** clause of the statement, refer to the work of [52] describe in section 2.2.1. which describes how to map source-to-target using this information.

Functional and Execution Information

Any information pertaining to the tracing program, procedure or functions that will be used in lineage tracing specific to a particular warehouse data. This may consist of the name and the description of the program. This may also consist of information that describes the transformation name and process and links to one procedural code to another that comprise the lineage tracing instruction invoked during lineage query.

Woodroof and Stonebraker [56, 55] propose the idea of function registration coherent to its proposal to trace data lineage basing on weak inversion, refer to section 2.2.2. Several pieces of information about weak inversion and verification function is to be registered to trace data lineage using the weak inversion. During lineage tracing, an inversion planner, (i.e. a program which generate an execution plan to trace the data lineage), infers which functions are to be used for weak inversion verification. [56, 55] describe the information which is to be registered; the *Inverting Function* (i.e. which perform the weak inversion or verification) and the *Function to be inverted*. Cui and Widom [8, 7] introduce a tracing procedure which is invoked during the lineage tracing. The properties of the procedure which dictate which programming procedure to execute during lineage tracing and the reference to the procedure itself are stored within the data warehouse environment. Bernstein and Bergstraesser [2] introduce the transformation package which works as a workflow; starting from extraction up to loading process. The identifier that references this package is stored within the data warehouse.

Lineage Intermediate Values

It is not usual but it may be necessary to store lineage intermediate values to enhance lineage tracing performance. Storing these values in data warehouse is resource intensive and requires serious consideration. The works of Cui and Widom [8, 7], describe different algorithms to store intermediate lineage values. During lineage tracing, lineage intermediate values is accessed to produced a lineage view instead of reconstructing from nothing. The presence of the lineage view in the data warehouse for tracing, intuitively improves performance during lineage queries but penalizes the

data warehouse loading process. Additionally, data in the warehouse grows in volumes of magnitude and storing data lineage alongside it poses a serious storage and administration challenge.

Other Data Lineage Related Information

Other lineage related information is stored in a dimension table. [1, 2, 6] introduced similar ideas about an audit dimension. The audit dimension is a table that contains pertinent information about the row of the fact table. For example it may contain the identifier of derivations applied to the data before the data arrives to the fact table, the source systems, the date and time it was loaded.

2.3.2 Storing Lineage information

In this section we look into the mechanisms for storing lineage information. We pick one example for each of the two main lineage granularity levels for illustration. Specifically we look on the work of [52] for storing schema information, and on the works of [7, 10] for storing lineage intermediate data and the works of [55, 7, 8] for storing function or procedural code.

Schema Information

Schema information is stored to support tracing the data origin or mapping target data to its source. A good illustration for storing schema information is the ones introduced by [52]. [52] presented Meta data storage schema which includes seven tables, refer to figure 2.8. In this section we present how these Meta data storage schema are used.

Looking back to our sample case in section 2.1. Three source tables; `Employee`,

`Attendance` and `WorkSchedule`, produce the warehouse data `AbsenceStatistics` through transformation steps `T1` to `T5`. The transformation steps are shown in figure 2.3. In section 2.2.1, we presented schema level approach of data lineage as introduced in [52]. Section 2.2.1 presented transformation steps in the form of mappings. Figure 2.7 shows the mappings scenario using our sample case in section 2.1. To illustrate how the Meta data storage schema is used, refer to figure 2.7, particularly the ***m1*** mapping. Mapping ***m1*** involves the `Attendance` table expressed in ***foreach*** query and the intermediate view ***V1*** expressed in ***exist*** query. We assume that the source database for `Attendance` table is `sdb` and ***V1*** is stored in a transient database `tdb`. The schema elements which are involved in mapping ***m1*** is stored in the Meta data storage as shown in figure 2.12. ***Element*** table stores the schema name, its type (e.g. string) its parent element, and the databases where it resides. ***Query*** maintains the query identifiers, in our example, q0 and q1 for the ***foreach*** and ***exist*** queries respectively. The ***Binding*** table records the ***from*** clause of each query as a list of bindings. In our example the information recorded are the binding identifier `a` within the query `q0` , the schema element where it starts (entered as `r1`) and the schema which it refers (entered as `e3`). The ***Condition*** table records the elements in the ***where*** clause. In our example shown in figure 2.12, the recorded information are the query id `q0`, the binding id `a` and element id, `e3` which participate in the expression with operator being "=" to a constant value "Sickleave.". The column **eid2** would have been an element id if the value in **bid2** is not a constant value. The **Mapping** table is used to encode mappings. It records the mapping id ***m1*** the query identifiers `q0` and `q1` for ***foreach*** and ***exist*** queries respectively. The expression in the ***select*** clause is recorded in the **Correspondence** table. The binding id and the element

id for the ***foreach*** and ***exist*** queries are encoded in **forbid**, **forEid**, **conBid** and **conEid** columns respectively.

Data Sources and Lineage Intermediate Values

Figure 2.3 in our sample case, illustrated transformation steps. Each transformation step produces an intermediate view before the final transformation step which populates the data warehouse. The intermediate views[2] in this transformation steps may be created and populated dynamically during execution time, or may be stored physically in a disk. Storing the intermediate views can be very expensive as the data in the data warehouse is usually huge. There are different options for storing lineage data values in the data warehouse. The maintenance aspect of this option (storing lineage data values) can also be complex and this topic does not escape the curiosity of the research community [22, 20, 9, 10, 45, 12, 7, 5].

Cui [7] details on different mechanisms of storing intermediate lineage values by presenting different algorithms of storing auxiliary information. The auxiliary view which stores the intermediate lineage values is generated during view specification. When the data from source are loaded to the warehouse through data integrator, the data integrator also populates the auxiliary view if view is tagged to be traceable.

Cui and Widom [7, 10] describe different options to store the lineage auxiliary views. First is ***Store Nothing*** option. This option, as its name implies, stores nothing in the auxiliary view. This scheme retrieves all information from the data source tables during lineage tracing. This saves storage and storage maintenance but provides poor lineage tracing performance. Another option is ***Store Base Tables***.

[2]Intermediate or Auxiliary views contain transformed data that will be used for further transformation.

Element

eid	named	type	parent	db
e0	Attendance	Set	-	sdb
e1	*	Rcd	e0	sdb
e2	EmpId	string	e1	sdb
e3	AttendanceType	string	e1	sdb
e4	Asumhours	numeric	e1	sdb
e5	Amonth	date	e1	sdb
e10	V1	Set	-	tdb
e11	*	Rcd	e10	tdb
e12	EmpId	string	e10	tdb
e13	AttendanceType	string	e10	tdb
e14	Asumhours	numeric	e10	tdb
e15	Amonth	date	e10	tdb

Query

qid
q0
q1

Condition

qid	bid	eid	op	bid2	eid2
q0	a	e3	=	"Sickleave"	-

Binding

bid	qid	eid	prev
a	q0	E0	-
V1	q1	e10	-

Mapping

mid	forQ	conQ
m1	q0	q1

Correspondences

mid	forBid	forEid	conBid	conEid
m1	a	e2	v1	e12
m1	a	e3	v1	e13
m1	a	e4	v1	e14
m1	a	e5	v1	e15

Figure 2.12: Metadata Storage Implementation from **m1** mapping - adapted from [52]

Within the data warehouse, create a copy of each source table that the view is defined. View maintenance in this scheme is uncomplicated, but the base table can be large and may contain data that are irrelevant to warehouse data, and thus are not usable for lineage tracing. Another method is to create and implement ***Store Lineage View.*** This scheme stores all lineage information for all tuples in the primary view. This schema significantly simplifies the tracing query and potentially reduce the query cost. But lineage views can be large and costly to maintain. Another option is ***Store Split Lineage***. This option split the lineage view and store a set of tables instead. Split lineage tables contain no irrelevant source data, since every row in this table contributes to some data warehouse row. Furthermore, the size of the split lineage tables can be much smaller than lineage view. The rest of the options are ***Store partial base table*** and ***Store Base Projection***. These options store specific portions of base tables only which reduces the size of Base Table method. See [7, 10] for details about the different options in storing the lineage auxiliary views.

2.4 Reporting the Data Lineage

In the previous section, we presented the data lineage granularity levels, we classify different types of data lineage basing on the different data lineage related studies, and we look on different approaches in storing and preparing the data lineage information for possible queries about the origin of data in the data warehouse. No matter how brilliant the algorithms and procedures for storing data lineage and preparing it for tracing, they amount to nothing if the user is unable to view and use the data lineage itself. In this section, the process for tracing or drilling-through the data lineage is presented. We look to specific approaches in the different works on how the lineage

is constructed. The tangible part of all the mechanisms discussed above, is the data lineage produced and made visible for the user who ask questions about the data origin of a warehouse data item.

2.4.1 Metadata Query

In section **Schema Information**, under section 2.3.2, we illustrated an example on how schema elements are stored basing on [52]. [52] proposes Meta data storage implementation consisting of seven tables which stores schema information to allow data lineage tracing. Furthermore, we also presented how the schema is stored by using our sample case in section 2.1. (see to figure 2.12).

Velegrakis, Miller and Mylopoulos [52] provide a formal basis for realizing schema mapping which is a form of tracing data lineage. This formal basis builds on the Meta data storage implementation where schema elements and transformations are available for querying. To realize this mapping, [52] extend standard query language with special operators to utilize the Meta data. This extended language is referred to as MXQL which stands for *meta-data extended query language*. Schemas and mappings are to be stored as described in section 2.3.2 and illustrated in figure 2.12, in order to be queried and returned in answer sets as regular data. MXQL queries can be executed to exploit the Meta data storage schema while hiding the details of the meta-data storage implementation. Tools for invoking the queries and presenting information in the report can then be used to show the result of the queries to the users.

2.4.2 View Tracing Query

A section **Source and Lineage Intermediate Values** under section 2.3.2, describes the overview of different approaches in storing the values of data lineage within the data warehouse. The primary objective of this approach is to optimize "drill-through" or step-wise query for the data lineage of the data warehouse items that are transformed via SQL views or different levels of views. Furthermore, this approach ensures to return accurate lineage values when queried in step-wise fashion.

[7, 10] formulate tracing algorithms which enable tracing the origin of data for a specific data warehouse item. A ***tracing query*** is built based on the target view query. To illustrate an example, let us go back to our sample case in section 2.1 which presents a data warehouse view ***AbsenceStatistics***, refer to figure 2.5. ***AbsenceStatistics*** view is populated from data coming from the three source tables which are transformed via five transformations (see figures 2.2, 2.3 and 2.4). Prior to the transformation process, each target view in the five transformations is assumed to be already defined as part of the of the transformation description. Assuming that the data in each of the intermediate views are permanently stored for lineage traceability, then drilling-through to the data origin is possible using the ***tracing query*** described in [7, 10]. To simplify our illustration, we utilize the example shown in figure 2.9. Figure 2.9a) presents the source tables and the intermediate target view ***V2***. Furthermore, this figure also highlights the view item in question, row (D70, 172.5, 3.2005). Figure 2.9b) illustrates the data lineage for row (D70, 172.5, 3.2005). The data lineage is derived using the ***tracing query*** created based on [7, 10]. The tracing query is built using the view definition. View ***V1*** definition is shown below.

```
CREATE   VIEW AS V1,
SELECT   e.Department,
         sum(w.WSHours) as WSHours,
         w.WSCalMonth
FROM     Employee e
         WorkSchedule w
WHERE    e.EmpID = w.EmpID
```

To trace the data lineage of row (D70, 172.5, 3.2005), the ***tracing query*** splits the Relations that are involved in the view definition, in this particular View. In our view example the relations are the **Employee** and **WorkSchedule**. After the split, a query is made against each relation basing on the view definition condition and the row in question. Implementing the tracing query produces the exact data lineage for the row in question as shown in figure 2.9b). The tracing queries created for the row in question (D70, 172.5, 3.2005) appears as follows:

Split 1

```
SELECT   e.EmpID,
         e.Name,
         e.Gender
         e.Department
         e.Address
         e.Birthday
FROM     Employee e
         WorkSchedule w
         V2Tuple v2
WHERE    e.EmpID = w.EmpID
         w.WSCalmonth = v2.WSCalMonth
         e.Department = v2.Department
```

The result of this query is the data lineage from **Employee** source table shown in figure 29b)

Split 2

```
SELECT   w.EmpID,
         w.WSHours
         w.WSCalmnth
FROM     Employee e
         WorkSchedule w
         V2Tuple v2
WHERE    e.EmpID = w.EmpID
         w.WSCalmonth = v2.WSCalMonth
         e.Department = v2.Department
```

The result of this query is the data lineage from **WorkSchedule** source table

shown in figure 29b)

2.4.3 Tracing Procedure

Tracing procedure based its calculation on the properties of the transformation which transforms and produce the data warehouse items. Cui and Widom [8, 7] formulated a tracing procedure for each of the transformation properties which allows the user to trace and view the data lineage for a particular items in the data warehouse. Subsection **Lineage tracing for general data warehouse transformations** under section 2.2.2, describes the overview of the lineage tracing procedure.

2.4.4 Inverse functions

Subsection **Inverse method** under section 2.2.2, describes the overview of the inverse method in lineage tracing. Furthermore, section 2.3.2 discusses how the lineage related information is stored in preparation for possible inquiry regarding the origin of data in the data warehouse.

This method invokes the ***inversion*** and ***verification*** functions when lineage report is needed. An inversion planner determines which ***inversion*** and ***verification*** function to invoke for a given attribute.

Chapter 3

Industry Implementations

In this chapter, we explore how the data lineage problem is being addressed in data integration industry, specifically in the arena of data warehousing technology. First, we look into the efforts put into standardization in connection with the data lineage problem, then we evaluate existing solutions provided by industry.

Based on my informal survey, standardization efforts related to data lineage are directly connected to metadata model and transformation systems. Metadata is essential in the process of data integration in a warehousing environment. The metadata standard was conceptualized to provide a generic metadata model[1]. The model represents metadata independent of platform and ensure metadata semantic equivalence among all the warehousing components. In the metadata standard specification, we delve into the metamodels that represent data transformation.

We also investigated specific prominent data integration solutions in the industry such as Oracle, Microsoft, NCR, SAP BW and Ascential Software. Evidently, the data lineage solutions offered by these providers are describe in their metadata and

[1]The term **model** is generally used to denote a description of something from the real world. A metadata model or Metamodel is the physical database model that is used to store all of the metadata [35]

transformation systems. This chapter provides the overview on how each of these industry solutions addresses data lineage problem.

3.1 Standardization Efforts

The globalization and internationalization of organizations increase the requirement of integrating disparate data to effectively provide meaningful information for decision-making [36, 40]. Most organization have multiple data formats and locations in which data are stored [33]. An important requirement to make this data useable across the enterprise is to integrate it into one location and representation. Data warehousing is a common enterprise solution for data integration [27, 18, 26, 6]. One of the most essential aspects of data warehousing is metadata [40]. In this section we look in to the existing metadata standard and how it supports data lineage.

OIM and CWM Overview. Two main metadata standardization works were created, namely the Open Information Model, (OIM) and the Common Warehouse Model (CWM). The first was from the metadata Coalition (MDC) and the latter was created by Object Management Group (OMG). In the year 2000, MDC merged with OMG [29, 41]. The two separate specifications, [36] and [40], are now owned by OMG. OIM Data transformation model is specific to described relational-to-relational transformations only and has dependencies on the Database Schema package while CWM transformation package is more generalized and is not tied to any particular schema [40]. However, OIM and CWM provide similar data lineage solution. Both standards provide facilities whereby data lineage can be tracked across a series of transformations. Because both provide similar data lineage solution, this thesis refer to them as the **Standard Metamodel** for data transformation.

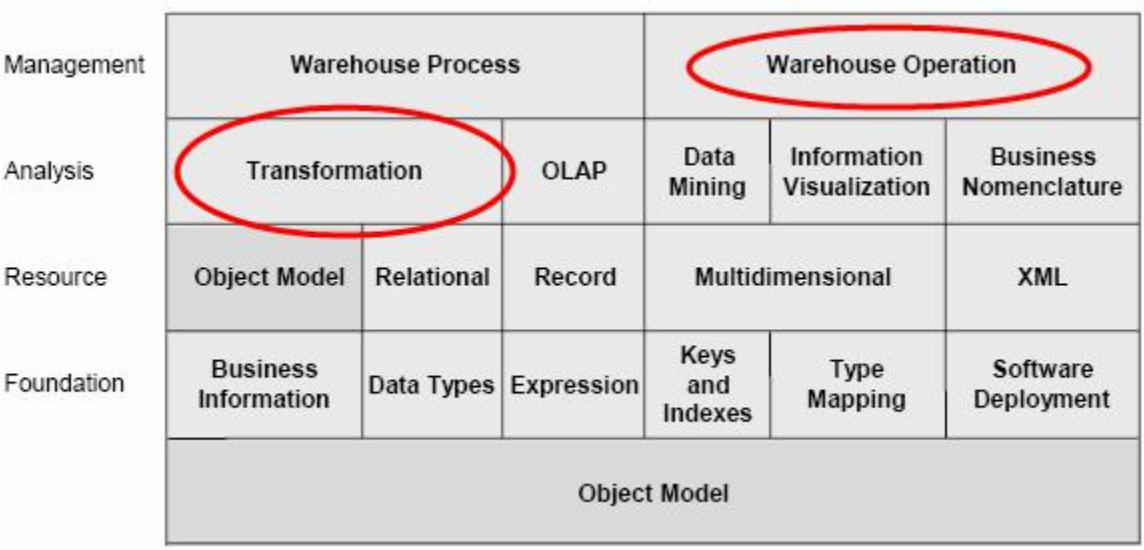

Figure 3.1: The Metadata Standard Metamodel - source from [40]

3.1.1 Organization of Standard Metamodel

Standard metadata Model has three main metadata modeling constructs; Classes[2], Associations[3] and Packages[4]. The Standard metamodel is split up into sets of packages and sub-packages. But this work will only focus on Management and Analysis, specifically the **Transformation** and the **Warehouse Operation** packages, refer to figure 3.1. However, since these packages have dependence on other packages not within the scope of our study, we make mention of them in some areas of our discussion. For this purpose, we list up all of the packages here below for easy reference, for details refer to [40]:

[2]Classes are specifications of a set of objects that have common structural and behavioral features. Classes can have Attributes and Operations. Attributes is a representation of metadata. Operations are provided to support metamodel specific functions on the metadata.

[3]Associations are descriptions of relationships between classes. An association has two Association Ends, which connect the association to two classes.

[4]Packages are collections of related Classes and Associations. A ***Package*** is a mechanism for organizing the elements of a model or metamodel into groups.

- **ObjectModel package -** is a collection of packages that provide basic metamodel constructs to other packages.
  - *Core package.* Contains classes and associations that form the core of the object model, which are used by all other packages including other ObjectModel packages.
  - *Behavioral package.* Contains classes and associations that describe the behavior of objects and provide a foundation for describing the invocations of defined behaviors.
  - *Relationships package.* Contains classes and associations that describe the relationships between objects.
  - *Instance package.* Contains classes and associations that represents instances of classifiers.
- **Foundation package -** is a collection of metamodel packages that contain model elements representing concepts and structures that are shared by other packages. Foundation model elements are meant to provide a common foundation which other packages can extend as necessary to meet their specific needs.
  - *Business Information package.* Contains classes and associations that represent business information about model elements.
  - *Data Types package.* Contains classes and associations that represent constructs that modelers can use to create the specific data types they need.
  - *Expressions package.* Contains classes and associations that represent expression trees.

  - *Keys and Indexes package.* Contains classes and associations that represent keys and indexes.
  - *Software Deployment package.* Contains classes and associations that represent how software is deployed in a data warehouse.
  - *Type Mapping package.* Contains classes and associations that represent mapping of data types between different systems.

- **Resource package**
  - *Relational package.* Contains classes and associations that represent metadata of relational data resources.
  - *Record package.* Contains classes and associations that represent metadata of record data resources.
  - *Multidimensional package.* Contains classes and associations that represent metadata of multidimensional data resources.
  - *XML package.* Contains classes and associations that represent metadata of XML data resources.

- **Analysis package**
  - *Transformation package.* Contains classes and associations that represent metadata of data transformation tools.
  - *OLAP package.* Contains classes and associations that represent metadata of online analytical processing tools.
  - *Data Mining package.* Contains classes and associations that represent metadata of data mining tools.

- *Information Visualization package.* Contains classes and associations that representing metadata of information visualization tools.
- *Business Nomenclature package.* Contains classes and associations that represent metadata on business taxonomy and glossary.

- **Management package**
  - *Warehouse Process package.* Contains classes and associations that represent metadata of warehouse processes.
  - *Warehouse Operation package.* Contains classes and associations that represent metadata of results of warehouse operations.

3.1.2 Metamodels for Data Lineage Support

The relevance of the Standard Metamodel in this work is that it provides a framework for representing metadata about data sources, data targets and transformations which allow encoding of data lineage information. The **Warehouse Operation** and the **Transformation**, are the packages which together provide metadata model constructs enabling ***data lineage*** reportng in data warehouse. The **Transformation** and **Warehouse Operation** packages are designed to allow navigation of metadata correlated to schemata, which tells about the source-to-target mapping.

Transformation package

The Transformation package contains classes and associations which represent common transformation metadata used in a data warehouse. It is designed specifically to:

- Relate a transformation with its data sources and targets. The relationship of data source and targets can be in any granularity level and can be of any type. The data source and target can also be persistent, (i.e. physically stored in database), or transient, (i.e. exist only during transformation execution).

- Accommodate both "black box" and "white box" transformations. The black box transformation here is similar to the black box as defined by Cui [7] in figure. 2.11 in section 2.2.2. In "Black box" transformations, the data sources and targets are associated to each other but association of specific piece of data source and target are not known. In the case of "white box" transformations, the relationship of the specific piece of data sources and targets is known.

- Allow grouping of transformations into logical units. At the functional level, a logical unit defines a single unit of work, within which all transformations must be executed and completed together.

Transformation can be packaged into groups. These groups can represent the transformations performed in a single program or in a logically related set of transformations. Figure 3.2 shows a sample transformation package. Transformations can be grouped into logical units. At the functional level, they are grouped into transformation tasks, each of which defines a set of transformations that must be executed and completed together - a logical unit of work (e.g. TransformationTask A and TransformationTask B in figure 3.2).

Transformation task is modeled in the ***TransformationTask*** class. ***TransformationTask*** class represents a set of Transformations that must be executed together as a single task (logical unit). A TransformationTask may have an inverse

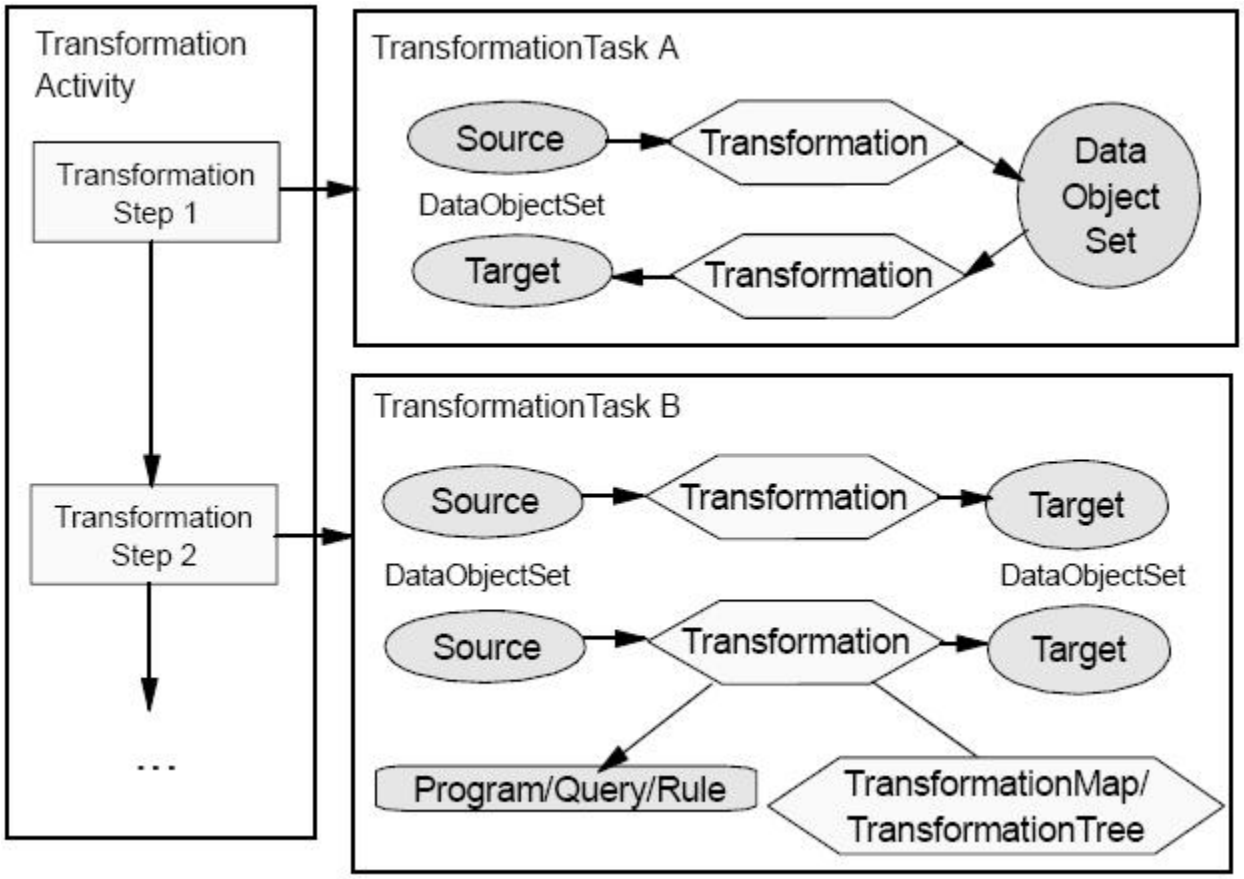

Figure 3.2: Sample Transformation Package - source from [40]

task. A transformation task that maps a source set A into a target set B can be reversed by the inverse transformation task that maps B into A. At the execution level, transformation steps are used to coordinate the flow of control between transformation tasks, with each transformation step executing a single transformation task. This model potentially signifies a lineage tracing possibility.

The transformation steps are modeled in ***TransformationSteps*** class. ***TransformationSteps*** are used to coordinate the flow of control between TransformationTasks. The transformation steps are further grouped into transformation activities. Within each transformation activity, the execution sequence of its transformation steps are defined. Transformation activity data is modeled in ***TransformationActivity*** class.

Transformation Classes and Associations supporting Data Lineage. The Standard Metamodel transformation package contains metamodel elements that support Transformation and **data lineage**. These classes and associations deal with transformations and their sources, targets, constraints, and operations. The transformation classes and associations that directly (i.e. encircled items in figure3.3) or indirectly supports the data lineage are shown in figure 3.3. Within the transformation package several groupings of classes and associations form related sets of functionality.

Functional Area	Classes	Associations
Transformation and data lineage	Transformation DataObjectSet TransformationUse	TransformationSource TransformationTarget DataObjectSetElement
Transformation grouping and execution	TransformationTask TransformationStep TransformationActivity PrecedenceConstraint StepPrecedence	TransformationTaskElement InverseTransformationTask TransformationStepTask
Specialized transformations	TransformationMap ClassifierMap FeatureMap ClassifierFeatureMap TransformationTree	ClassifierMapSource ClassifierMapTarget FeatureMapSource FeatureMapTarget CFMapClassifier CFMapFeature

Figure 3.3: Transformation Classes and Associations - source from [40]

Warehouse Operation

The Warehouse Operation package contains classes which record the warehouse processes daily operations. This package includes *Transformation Executions* and *Measurements* which have classes that allow the lineage of data in a warehouse to be preserved. *Transformation execution* records when and how the warehouse data were derived, and where they came from and the *Measurement* allows metrics to be stored (e.g. actual, estimated, or planned values for the size of a table).

3.2 Industry Solutions

In this section, we evaluate prominent industry solutions and look into the solution they offer for data lineage problem. First, we consider industry's different perspective

on the data lineage. Then we look into each solution package which addresses the data lineage issues. Moreover, we describe the products components or tools which record the data lineage and prepare the lineage information for reporting. The solution providers included in this evaluation are the following; Oracle, Microsoft, NCR, SAP BW and Ascential Software. In this section, we describe how each of these commercial products addresses the problem of data lineage.

3.2.1 Industry's view of Data Lineage

Data warehouse industry uses the term **lineage** to describe the traceable origins of data and ownership of something [31]. Data **lineage** is directly considered as part of industry's effort in striving towards a measurable quality of data and closely link it to audit [53, 1, 49]. SAP [1] considers building an audit dimension similar with the idea describe by Kimaball [32]. Teradata [49] defines **lineage** as a collection of details about data's journey from source to target and considers **lineage** as an audit trail of the data warehouse ETL processes. Ascential Software [17] describes data lineage as the answer to the question, "Where did this data come from?" and reports on processes that create and update warehouse data.

3.2.2 ETL and Metadata Services

Data warehousing is a process consisting of several components. Two essential components of Data warehouse are ETL and metadata. This thesis evaluates the ETL and metadata Services[5] components of the following products: Oracle, Microsoft, NCR, SAP BW and Ascential Software. ETL process in itself is consist of a wide and complex subprocess (e.g. data cleaning, conforming, loading) that are beyond

[5]Generally, a service is something that can be called and that provides a visual representation.

the scope of this thesis. However, it is in between the steps of these processes that **data lineage** information is recorded and possibly prepared for **data origin** related inquiries. Metadata is central to all data integration processes. Effort to standardize metadata model is a proof for its significance. For the purpose of this research, we only consider the metadata role in ETL processes and its relevance in **lineage** reporting. The processes for extracting, transforming and loading data are documented in metadata repository. All of the products and tools evaluated in this thesis utilizes the metadata repository to record the history of ETL processes, albeit data lineage related information is encoded in different manners and levels of lineage granularity.

SAP Business Warehouse

SAP Business Warehouse (BW) is the Data warehouse solution offered by SAP. SAP BW architecture is consist of several layers of services which include **Metadata** and **ETL** services among others.

SAP BW Metadata Services. SAP BW is completely based on metadata managed by the metadata services. SAP metadata Services component provides integrated metadata repository and metadata manager. The metadata repository is where the data is stored and the metadata manager handles requests for retrieving, adding, changing, or deleting metadata. Metadata objects are centrally controlled by the Administrator Workbench, AWB. The AWB is a GUI that aids in utilizing the administration services which allows for managing and maintaining the metadata data repository (see figure 3.4).

SAP BW Metadata stores information about data and processes. InfoObjects are

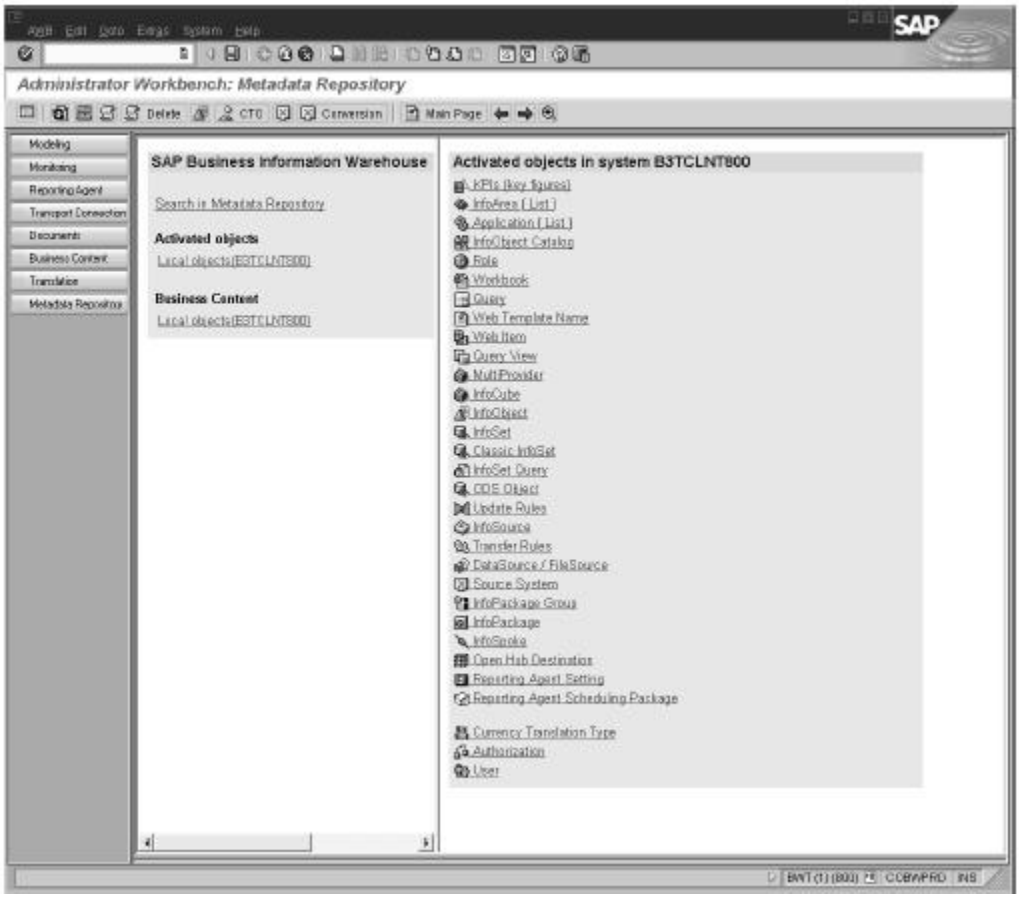

Figure 3.4: SAP BW MetaData Repository - source from [28]

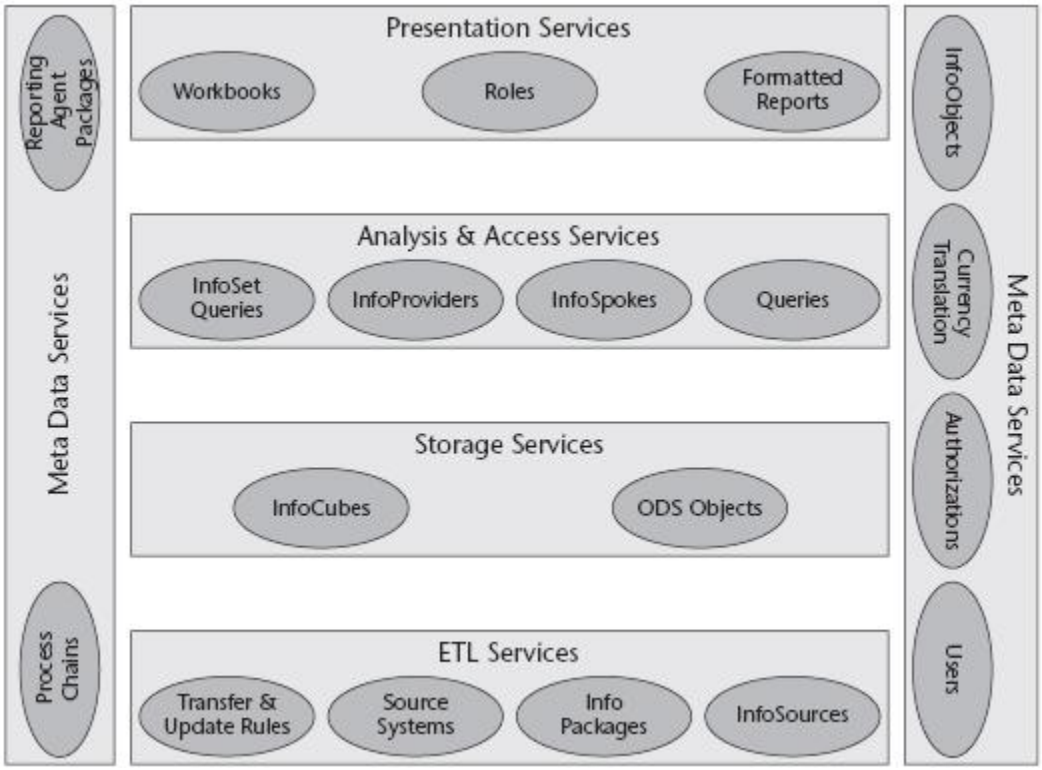

Figure 3.5: SAP BW Metadata Objects in context - source from [28]

the core building blocks of SAP metadata which are commonly termed as Dimensions and Facts in data warehouse domain. InfoObjects are the main SAP BW data targets among others, although some dimensions may not be data target which functions similar to the idea of degenerate dimension describe by Kimball [6]. Objects for extracting, transforming and loading (source system, datasource, infosource, infopackage, transfer and update rules) being part of metadata repository, can also be navigated, managed and maintained within AWB among many other objects which pertain to catalogs, queries and security. For details refer to [28].

SAP BW ETL. Besides extraction, transformation and loading of data, ETL services layer also serves as staging area for intermediate data storage for quality assurance purposes [28], refer to figure 3.6. Staging engine is the core part of SAP ETL service layer. Staging engine manages the staging process for all data received from different source systems. It is this engine that generates and executes transformation programs and performs the transfer and update rules. As far as this research work has gathered, there is no direct mention of data **lineage** found in SAP BW documents referred in this study. However, based on the analyzes made on SAP ETL processes, some data lineage related information are actually recorded and assembled (in higher level of granularity) basically for the consumption of business warehouse systems administrator. The DataSource manager in SAP ETL manages the definitions of different sources of data. DataSource manager supports the interfaces between SAP BW and its sources. SAP has their own proprietary transaction systems producing transaction data owned by different application component. The interface between SAP's BW to its own application components is tightly integrated via SAP BW

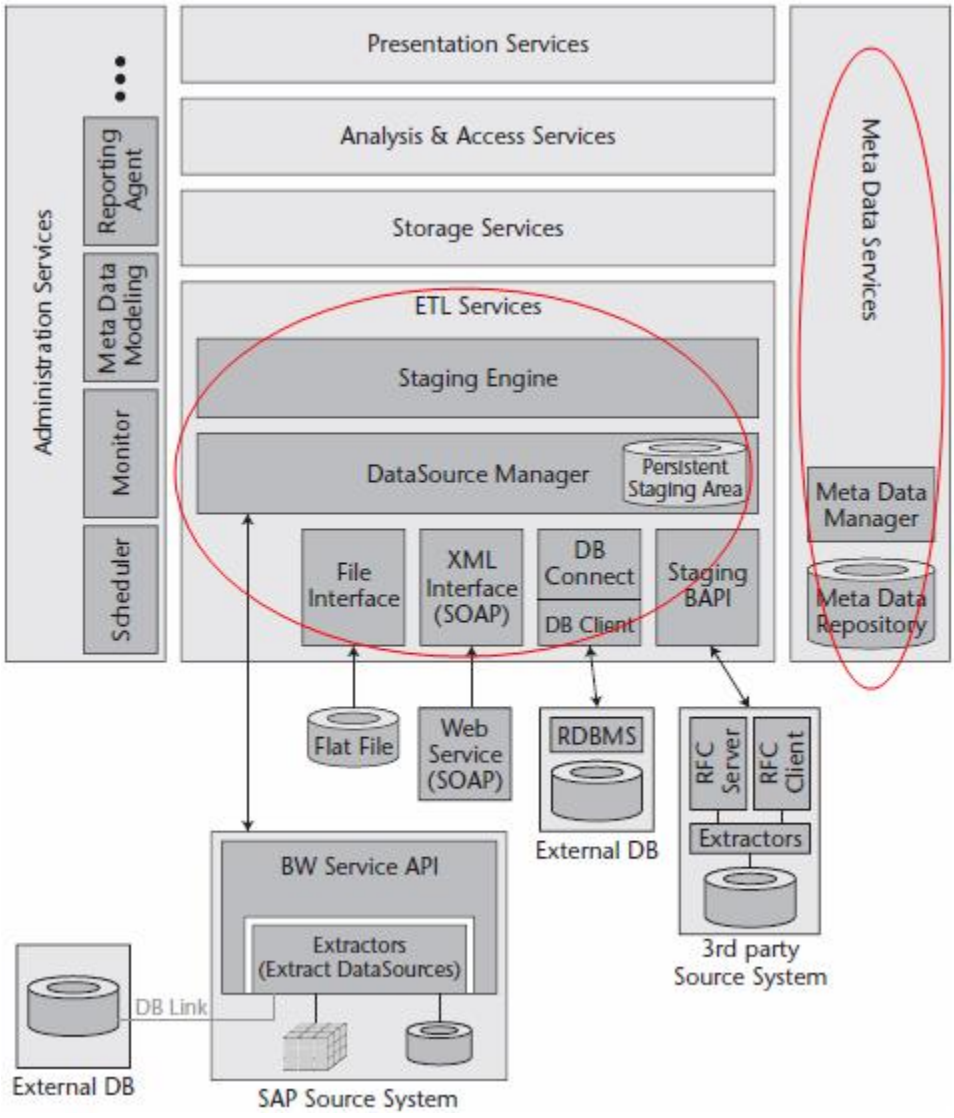

Figure 3.6: SAP BW ETL Services architecture - source from [28]

Service API. For these application components, data warehouse schemas and source-to-target mappings are ready made, available for activation and enhancement. SAP BW is also open for Flat File, XML and DB Connect interfaces and an Interface called Staging BAPI which provides an open interface for exchanging metadata. Prominent integration tools such as Ascential, ETI and Informatica have adopted this interface.

Oracle Warehouse Builder, OWB

OWB is a design tool for building oracle data warehouse. The OWB basic architecture is comprise of two components, namely; the design and runtime, which handle the design and runtime environment respectively. Refer to figure 3.7. The objects in this figure is coded to distinguish between the components. Figure 3.7 shows the basic OWB product Architecture. This thesis only focus on the aspects of OWB which has relevance to data lineage. Referring to figure 3.7, we basically focus on the design schema(B), repository schema(2) and target schema(3) and the audit browser(D1, D2). For details about OWB, refer to [34, 14, 44, 53, 43, 47]. Specifically, for details about the OWB product Architecture figure, refer to [42].

OWB provides graphical tool for designing ETL system at a logical level, i.e. creating source-to-target mapping (refer to figure 3.8). These mappings are fully documented with metadata stored in Warehouse Builder design repository (3.8-B). After the ETL system is designed at a logical level and the database information logical design added, the complete design (ETL and database) needs to be moved to the runtime environment (i.e. Database y in 3.8) for implementation. After the configuration is complete, code or script is generated automatically basing on the logical design. The generated code is deployed ready for the real runtime "extraction,

transformation and loading activities" to populate the data warehouse. Warehouse builder enables to perform **lineage** analysis on all ETL operations.

OWB Metadata. Metadata is created and managed and queried in the *design environment*. All activities in the design environment works against the design metadata. These activities involve specification of warehouse objects abstract representations and business rules. Warehouse objects, which are specified logically using a graphical tool, may include database schemas, multi-dimensional schemas and ETL processes.

The design metadata is stored in to the OWB repository at a granular level. For example, the design of data and process elements in the source-to-target mappings is stored in OWB metadata repository. When the design pertaining to ETL processes and all data elements involved in the processes are translated into physical implementation, it can be used later for **data lineage** queries. Since metadata reporting is provided, important information in the repository can be accessed in reporting environment. OWB reporting allows browsing capabilities.

OWB ETL. The *runtime environment*, takes this design metadata and turns it into physical database objects and process flows. The logical design defined in design environment is translated for physical implementation. For example, the data and process elements of source-to-target mappings defined in the design environment, are now translated in to real mappings in the form of physical tables and program code which is generated during the translation. ETL processes are executed by packages contained in the Target Schema (see figure 3.7-3). The Target Schema represents

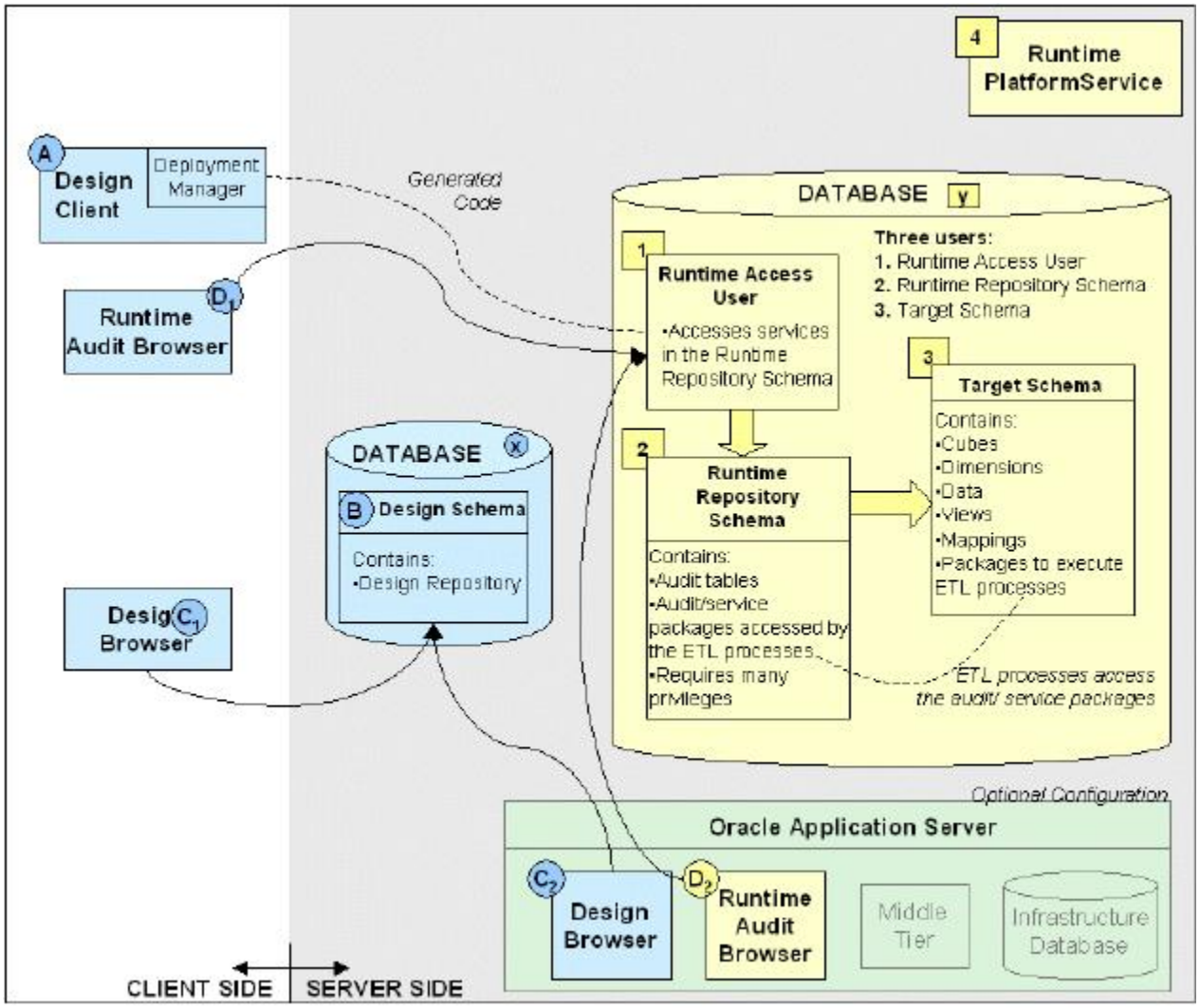

Figure 3.7: Oracle Warehouse Builder Basic Architecture - source from [42]

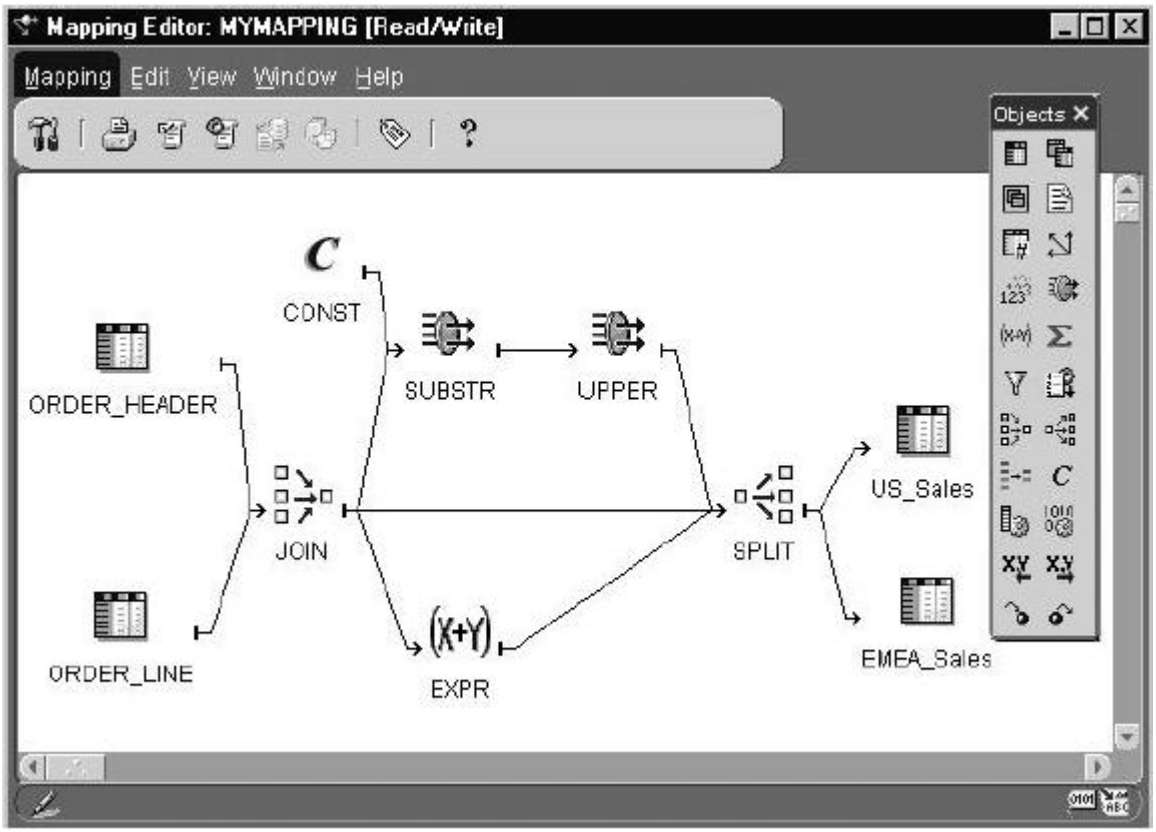

Figure 3.8: OWB Mapping Editor - source from [34]

the the actual warehouse data target, and contains the target data and data objects (e.g. cubes, dimensions, views, and mappings). The Target Schema stores the implementation of the design, and is where the warehouse jobs run.

OWB is designed to allow extraction from heterogeneous platforms (e.g. file, relational). OWB generates PL/SQL to handle the Oracle based transformations, which provide operators to do SQL like activities. Additionally OWB also provides user defined transformation. Complex processes can be created in graphical representations in the design environment and generate either row based or set based code. Row-based code gives a high degree of control over what happens in the load as well as extensive error handling capabilities. Row based code allow row level data ***lineage***. However row-based is typically slow, although OWB provides an alternate way to improve ETL performance. But in most situations, set based processing will still be faster.

Microsoft - Data Transformation Services

Data Transformation Services, DTS performs an essential role in Microsoft Data Warehousing and Data Integration solution. It provides a set of tools for extracting, transforming, and consolidating data from disparate sources and loading these data to the defined destination. DTS Architecture is shown in figure 3.9. In DTS architecture, relational and non-relational data sources are pulled using data pump and sent to destinations after required transformation. Complex transformation and data validation logic can be implemented using scripting. These scripts can invoke program codes to modify or validate the value of a column. Advanced developers may create reusable transformation objects that provide data scrubbing capabilities.

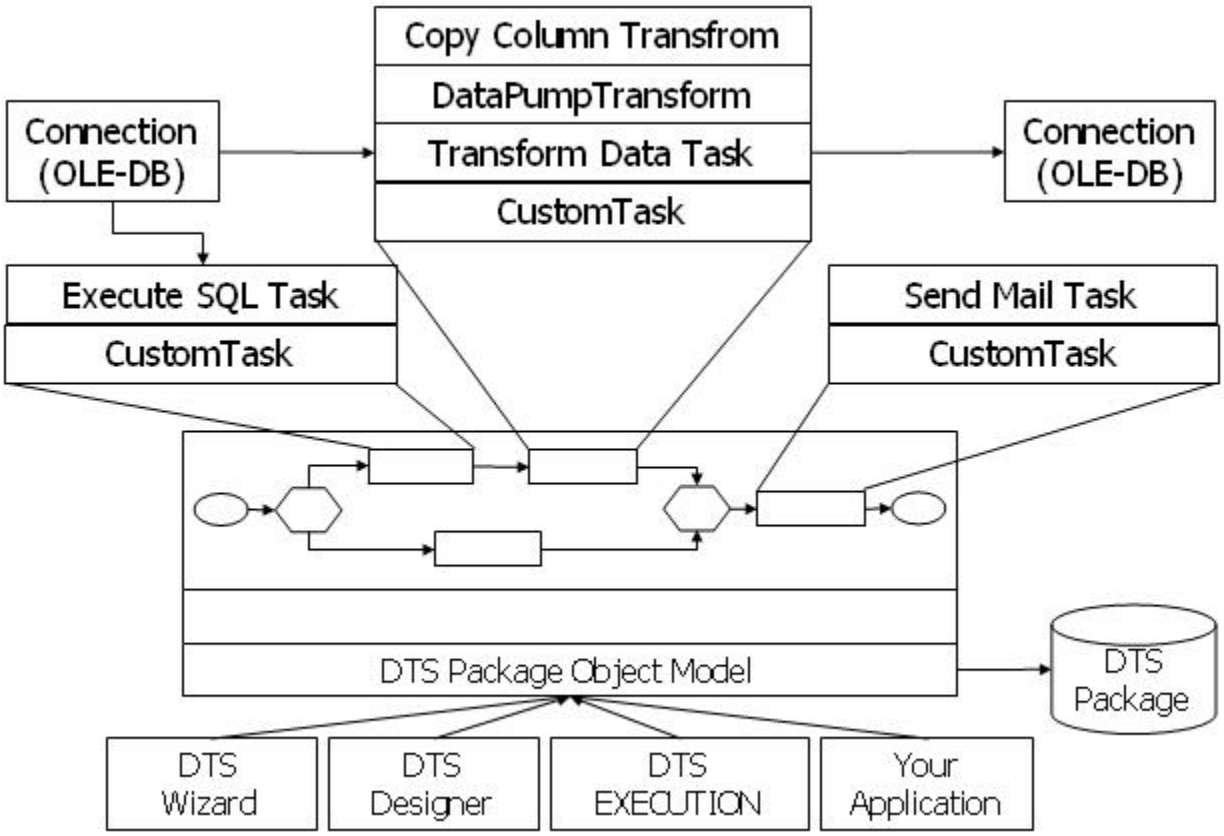

Figure 3.9: DTS Architecture Overview- source from [15]

Custom tasks may also be created to launch external processes. The data pump opens a rowset from the data source and pulls each row from the data source. The data pump executes functions to copy, validate, or transform data from the data source to the destination. Transformed values are returned to the data pump and sent to the destination. This thesis considers only the parts of the DTS architecture that describe the data lineage solution. For details, refer to [37, 33, 30, 38].

DTS ETL. DTS solution is created as package or packages. Each transformation package contains an organized set of tasks which describes a complete ETL process

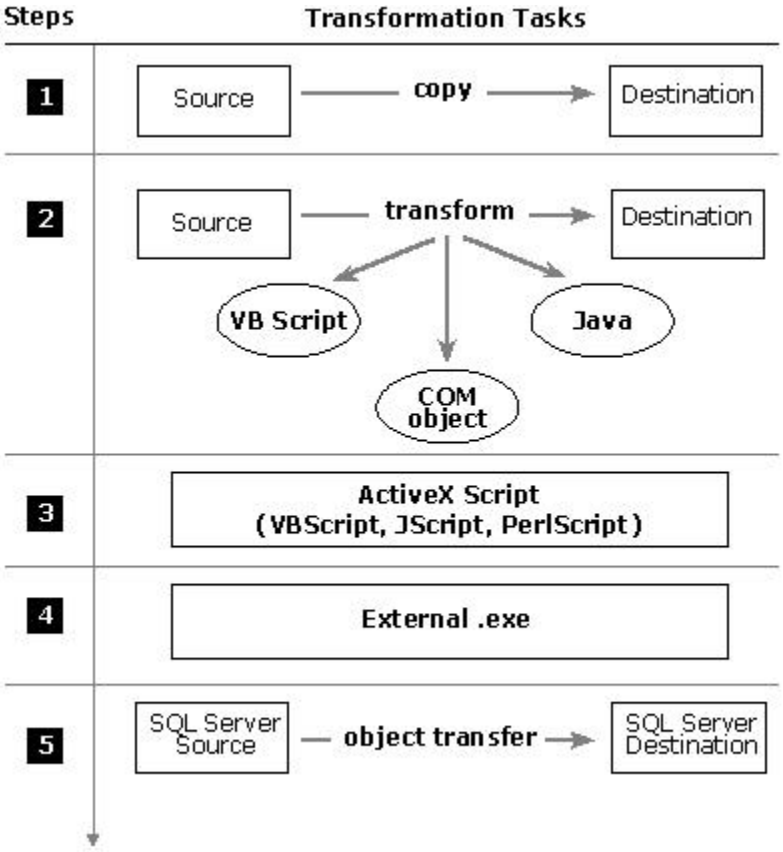

Figure 3.10: DTS Package illustration - source from [37]

(see figure 3.10). The set of tasks is executed in a coordinated sequence as specified in the package. The relevance of DTS architecture in this thesis is that, it allows data **lineage** recording and reporting. DTS records and documents the lineage of each transformation in the repository. Depending on the specification, data **lineage** may be tracked at table row and column levels. This provides audit trail for the information in the data warehouse. DTS package can be created using DTS Wizards, DTS Designer, or DTS programming interfaces and can be executed using DTS graphical interface or command line interface.

Microsoft Metadata Services. Microsoft, being the main contributor of OIM

has data lineage implementation based on the standard metamodel. To enable the data lineage traceability, DTS package must be saved to Microsoft metadata Services. Microsoft metadata Services is an object-oriented repository technology that can be integrated with the enterprise information systems or with applications that process metadata. In addition, metadata Services support browsing repository databases through metadata Browser. Figure 3.11 shows an illustration of the interfaces between Microsoft metadata services and other tools. The illustration shows the DTS packages which are developed at Development time and implemented at run time. At run time, browser tools, which work directly with repository contents, can be used to query about the DTS package information stored in the metadata services. One important information about DTS package is the lineage information.

NCR Teradata

Teradata is the data warehousing solution offered by NCR. Teradata data warehousing solution consists of a number of utilities and tools required to build the data warehouse. Teradata provides stand-alone extract and load utilities: FastLoad, MultiLoad, TPump and FastExport. Each of these tools has specific loading features that are more applicable for specific loading cases. Fastload for example provides high performance data loading from client files in to empty tables. Another Teradata tool that provides similar architecture to its stand-alone extract load utilities is the Teradata Warehouse Builder, WB. This thesis is not going to detail on this tool, but will only describe the overview of its data lineage solution. Teradata lineage solution is emphasized in Teradata metadata Services. To make data lineage information available for inquiries, each source-to-target mappings details should be stored. Refer

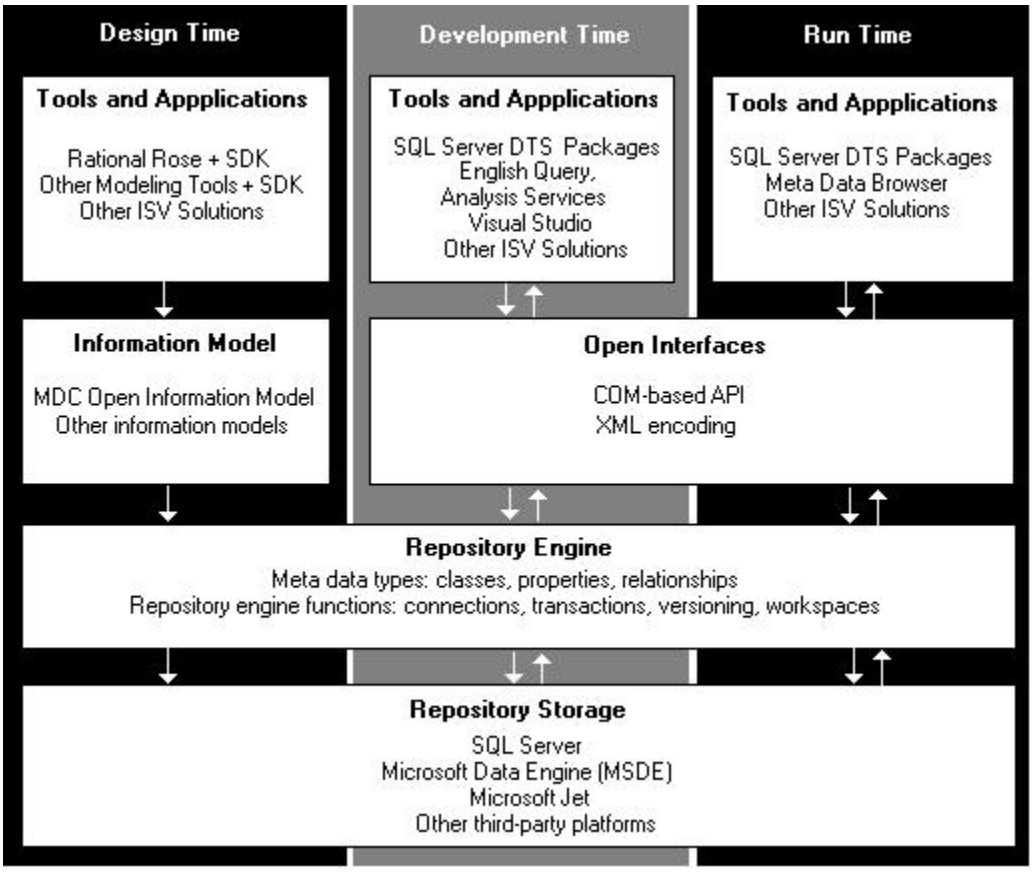

Figure 3.11: Microsoft Metadata Services interfaces - source from [37]

to [50, 46, 49] for details.

Teradata ETL. The Teradata WB architecture provides a parallel extraction, transformation, and loading (ETL) environment. It helps build and maintain the warehouse using SQL-like scripting language that performs all aspects of ETL processes using custom transformations. Figure 3.12 illustrates the typical load architecture using Teradata WB. The source environment is where the data sources reside. Data may come from disparate sources, such as disk files, queues, and database management systems (DBMSs). Teradata WB resides in the load environment where transformation processes are performed. Teradata WB pulls data from the source environment, transforms them, then pushes them into the target environment, which in this case is the Teradata Database. In between the steps of ETL processes, lineage data are stored in metadata repository to enable lineage traceability.

Teradata Metadata Services. Teradata metadata Services is a software that creates a repository in the Teradata Warehouse in which metadata is stored.

Teradata provides extension Lineage Model for holding lineage data. Figure 3.13 shows Teradata LineageModel. LineageModel provides classes for documenting the data's journey from source to target. All data receptacles are defined in the DataDepot. Data receptacles are the intermediate, transient or transfer point of data during their journey from source to target. Transform is the class that holds rules which are applied to data sources in the Depot to produce targets in the Depot. Some examples of transform objects are steps in Teradata's load utilities (FastLoad,MultiLoad, TPump or FastExport), specifications of modifications made to an extract, transform and load (ETL), and CreateView statements. Mapping is a class that holds what are often referred to as job streams or scripts. The relationships between these Lineage

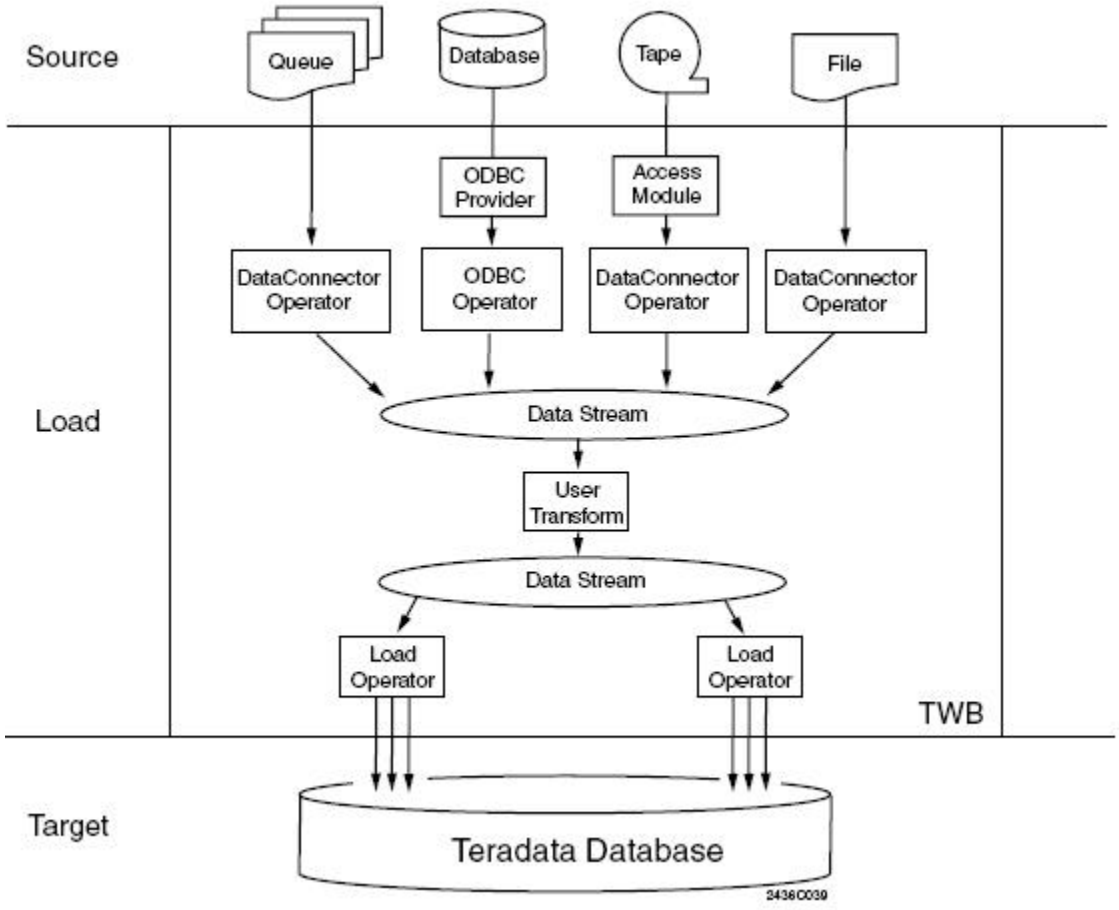

Figure 3.12: Typical TBW Load Architecture - source from [49]

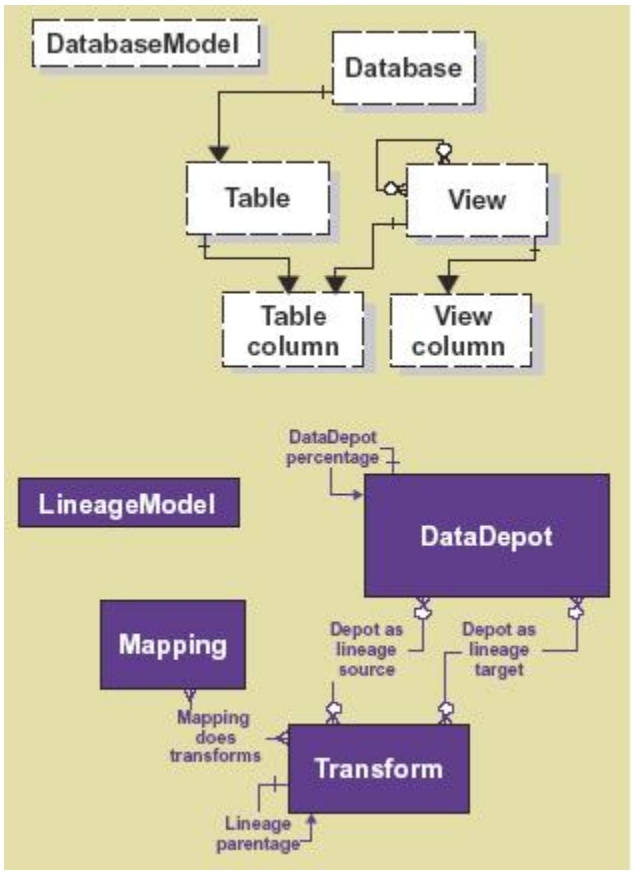

Figure 3.13: LineageModel within Models of Teradata MDS - source from [46]

classes are to have names that reflect their purpose.

Ascential Software

Ascential Software is a solution provider that offers enterprise integration suite consisting of components relevant to this research. The Ascential DataStage and MetaStage components are ETL and metadata tools respectively. Ascential Datastage provides access to a wide variety of disparate systems, enabling data extraction from different sources, transforms data and load it to the data targets. Ascential offers MetaStage

product which helps integrate data from DataStage ETL jobs and other data movement processes. MetaStage is responsible for management of metadata, which is categorized into four; Business, Technical, Operational, and Project metadata. The Technical metadata repository stores data that define source and target systems, and their table and fields structures and attributes, derivations and dependencies. Typically the metadata that describes the transformation processes can provide data for queries regarding the data lineage. Ascential software specifically considers storing lineage information in order to enable data lineage reporting. For details about DataStage and MetaStage components of Ascential software, refer to [39, 17, 48, 51]

DataStage ETL. The DataStage ETL component is consist of four design and administration modules, namely; *Manager*, *Designer*, *Director*, and *Administrator*. The *Manager* module is the basic metadata management tool used to create and organize metadata. Organizing metadata includes source and target data definitions and ETL job components. DataStage designer is a graphical interface that can be used to define ETL jobs in the form of visual source-to-target mapping. DataStage support complex data transformation and provides pre-defined functions, including boolean logic, string manipulation, arithmetic function, and higher level functions such as measurement conversions. The visual design of source-to-target mapping and the transformations defined between source and target is translated in to physical model and transformation programs which will perform the actual ETL process. Figure 3.14 is a screen shot of a DataStage designer that shows an example of a datagestage job named `examplejob1`. This example job shows the data being extracted from three different sources; file, oracle database, and from a ODBC connection. The extracted data are then transformed and loaded in data targets. The first contains data that is

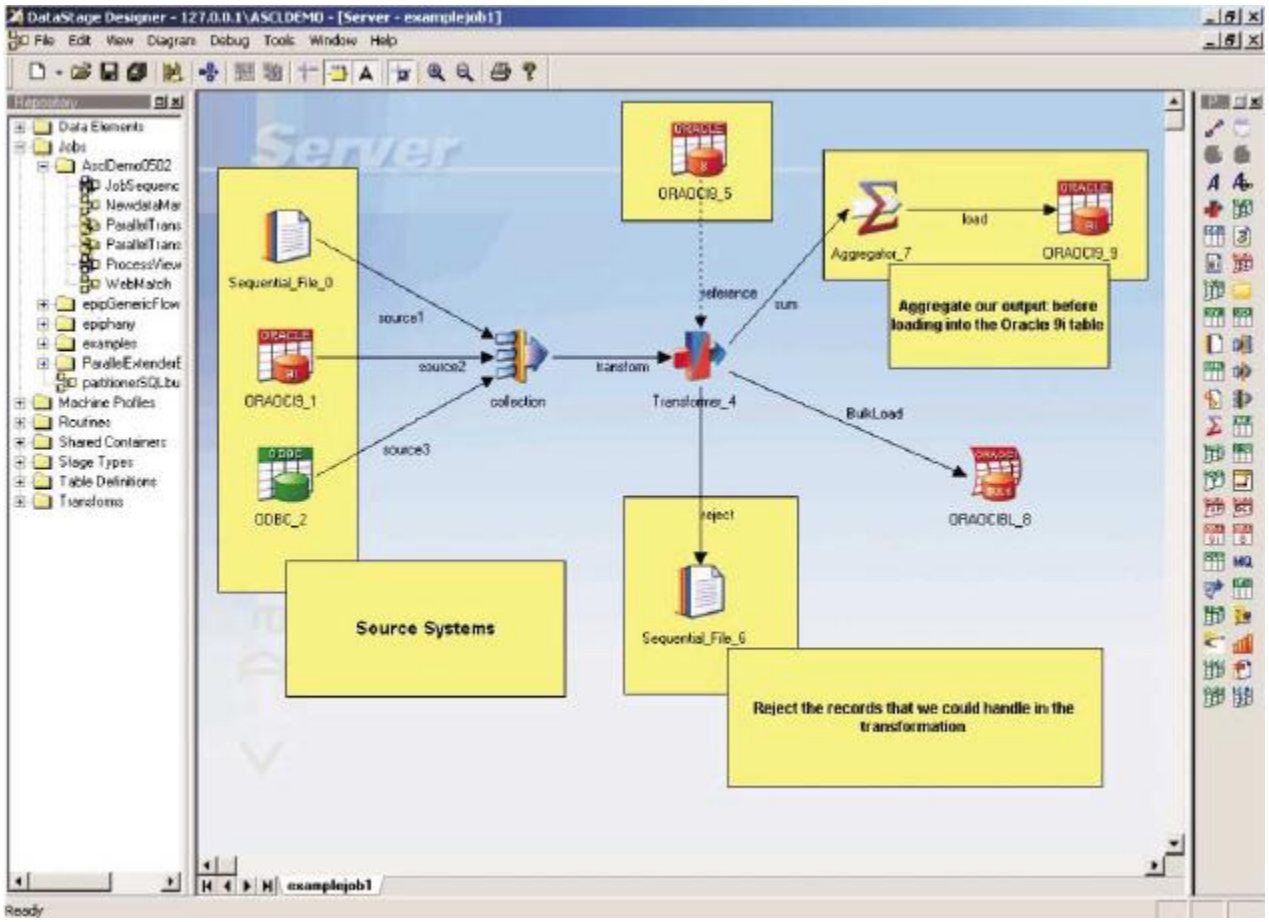

Figure 3.14: Ascential Data Stage Designer - source from [39]

aggregated prior to final loading while the second target stores the data immediately after the first transformation. Creating DataStage jobs will produce process metadata. The process metadata will consist of the source-to-target mappings and the transformation description which are useful for data **lineage** reporting. The ETL job is scheduled and executed in the DataStage *Director*. The DataStage Administrator supports housekeeping and security tasks. It is use for defining authorizations and tuning.

MetaStage. MetaStage is a tool that captures, integrates and shares metadata.

It helps manage and administer data integration and coordinates data across the entire enterprise. The most relevant aspect of MetaStage in this thesis is its feature to handle metadata related to data movements, which is basically the information about the path the data traversed from its source up to its target. MetaStage is consist of software components which support data lineage. MetaStage Directory component is capable of sourcing, sharing, storing, and reconciling a comprehensive spectrum of metadata which is classified into four; business, technical, operational and project metadata. Technical and Operational metadata are the two metadata categories which allow data lineage reporting. Technical metadata defines source and target systems, and their table and fields structures and attributes, derivations and dependencies and ETL. Operational metadata are data about process execution (e.g. process frequency, record counts, analysis, and statistics). MetaStage's process MetaBroker gathers the operational metadata to answer questions about Data Lineage. Information regarding data resources affected by, for example, a job run is recorded. MetaStage may also import operational metadata from DataStage ETL jobs, which mainly describes data lineage. This gives possibility to track data lineage of the released DataStage jobs. Contents of the set of transforms, routines, and any other non-job objects which were invoked by the run can be investigated as well. You can then output a report that shows the steps a job goes through to generate each column of an output table. Data lineage information can be inferred combining the operational metadata and process metadata which is produced by creating a Datastage job. Before the DataStage job is run, the data model and the job design metamodel have to be imported in MetaStage.

3.2.3 Data Lineage Reporting

The previous sections discussed the overview of industry product components that directly or indirectly provide the solution for recording or physically storing the data lineage. These components are generally referred to as ETL and metadata services. We presented the overview of the process through which the data lineage information were recorded, so they can be used later for lineage reporting. In this section we consider to look at the tools the industry offers for data lineage reporting. The four solution providers that we investigated show similar approach to data lineage support. They provide a tool for creating the metadata and ETL logical design and translate this design to a physical design. Translating the ETL logical design creates metadata physical model which is constructed to support lineage reporting. Reporting data lineage requires structural constructs that support lineage related queries.

SAP BW does not deliberately mention lineage reporting, although it is capable of showing a data lineage report at a higher level of granularity. In SAP, BW Administrator Workbench allows to browse through the flow of data from their source to their target. Figure 3.15 shows an example of Data Load Dataflow Report. This report presents the objects involved in the ETL process, such as source system (in this case, SAP's own system), the data source, persistent staging object, communication structures, the transfer and update rules and finally the data target. The transfer and update rules basically contain the data transformation information which may simply be a column to column mapping, or may include arithmetic formula or routines that transform data via simple to complex calculation. Clicking on any of the objects in the report will display some details. For example, data transformation information can be viewed by double clicking on the transfer rules or update rules. Target Column

value assignment specification such as column-to-column mapping and calculations applied to data are shown in Update Rules. However the column-to-column mapping in Transfer Rule is based on a pre-created "structure" from the SAP's source transaction system. The pre-calculated structure is maybe created involving complex views combined with simple to complex programming code. This makes it difficult if not impossible to trace data up to the original tables and columns of the sources. Albeit not straightforward, there are different ways for validating the data in SAP BW. Besides the report about the ETL process, SAP BW also provides a possibility to do report to report jump. This enables the users to view source data presented as a report from the source system. Although this information is not the data lineage itself as described in [19, 56, 7, 12, 10, 8, 52], this information say something about data origin. A report has to be created in the SAP source system. The report ID of the source system should be encoded in SAP BW and associate it to the SAP BW query. When a user navigates through SAP BW report (using the query with link to the source system report) and wishes to invoke the report from the source system, the user will be able to view the original data. This is with the assumption that the right combination of the arguments are passed to the source system.

Oracle Warehouse. When defining the warehouse objects by either using the **OWB** client graphical or user interface or scripting interface, **Oracle** captures the metadata as part of the normal process. It includes the warehouse data models and data models involving ETL processes. For example the models involved in Source-to-Target mappings shown in figure 3.8 are actually stored in the a metadata repository. Oracle Warehouse provides reporting environment which allows developers and business users to browse and investigate system elements. A very important component

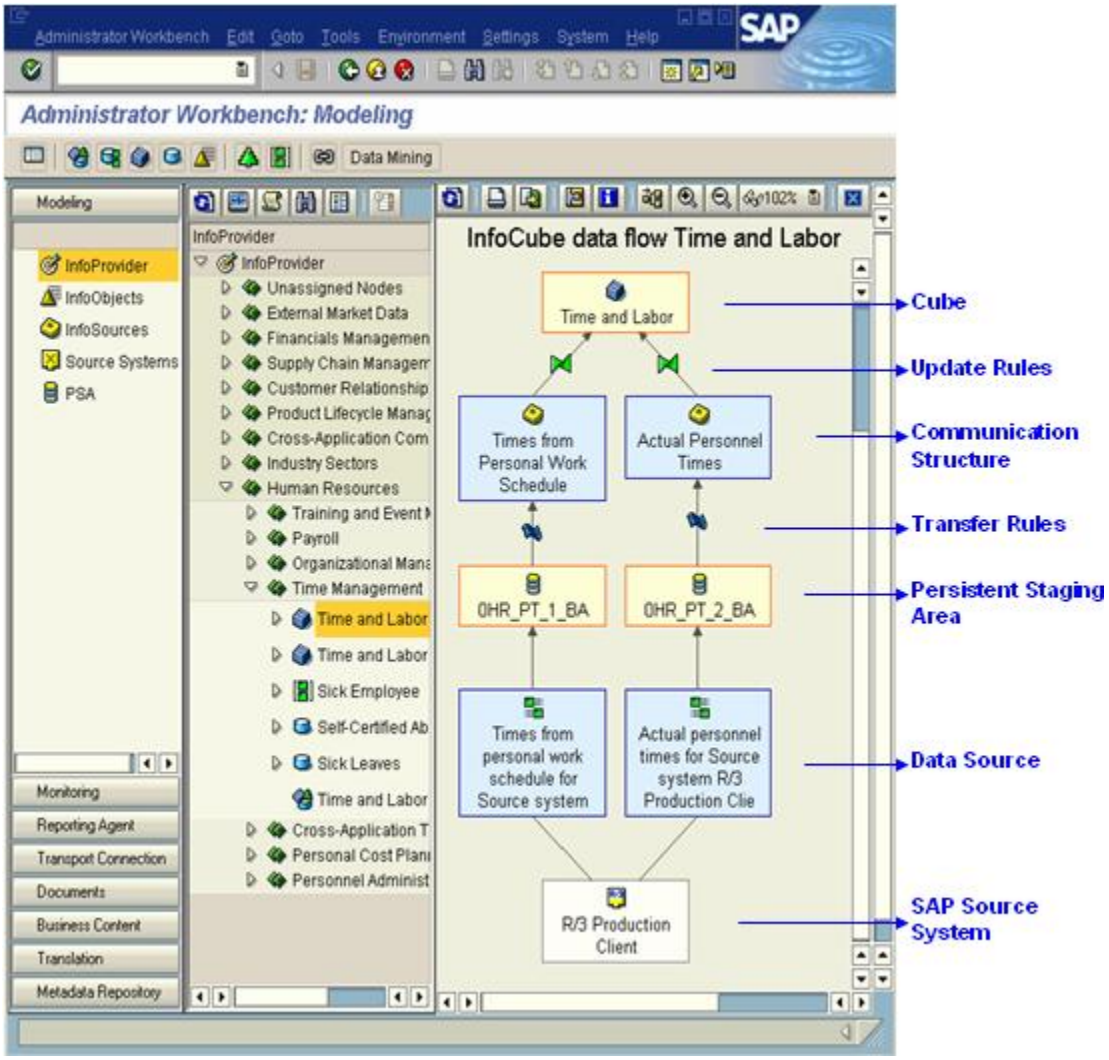

Figure 3.15: SAP BW Example of Data Load Dataflow Report

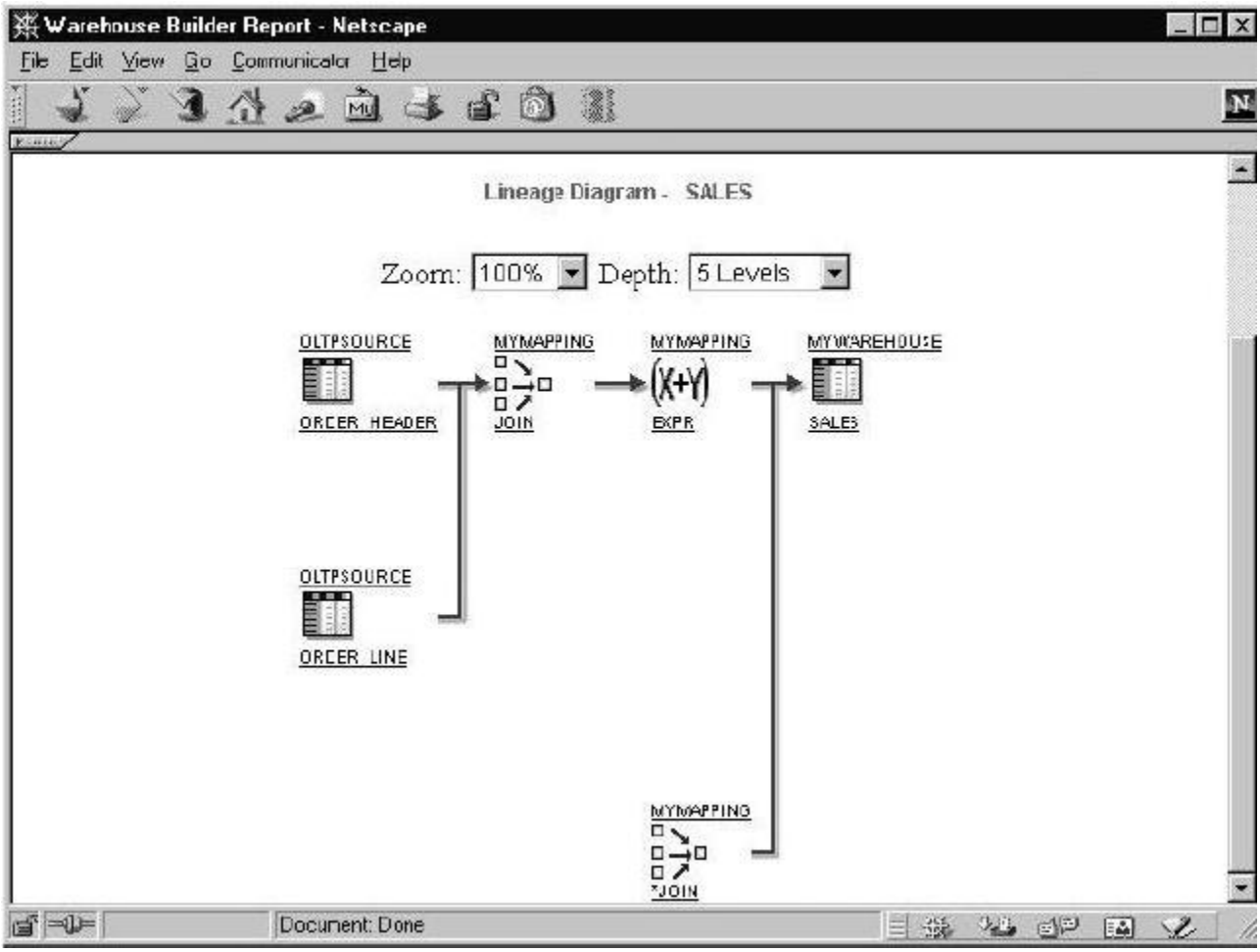

Figure 3.16: An example of OBW Lineage Report - - source from [34]

of this reporting environment in relation to this thesis, is the feature that allows tracing back where the data is originated. Oracle Lineage uses the design metadata for lineage reporting. An example of Oracle lineage report is shown in figure 3.16.

Microsoft DTS. Lineage takes on two distinctly different forms in DTS; the column level and the row level lineage. DTS implements column level lineage at the time of transformation. The information is used for reporting using the metadata services. The package execution is the second form of lineage in DTS. When a package

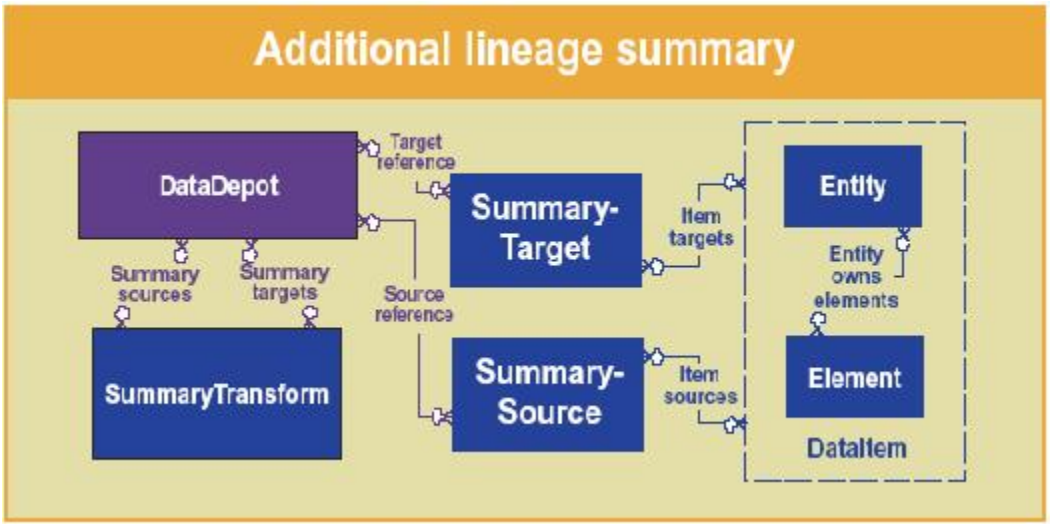

Figure 3.17: Teradata Additional Lineage Summary - source from [49]

executes, (assuming that lineage option is selected), DTS creates an identifiers which can be used for row level lineage tracing. Lineage reporting is basically queries made against the metadata.

NCR Teradata. Similar to the first three data warehousing solution providers (SAP, Oracle, and Microsoft), Teradata also uses the metadata for storing and calculating data lineage queries. In addition, Teradata provides classes in Metdata Models which simplify the data lineage report navigation. Figure 3.17 shows the model for data lineage summary classes; Summarysource and SummaryTarget.

Ascential Software. *DataStage run* process populates the metadata repository. Pertinent information about the process run, (e.g. affected sources and targets, derivations) are recorded in the metadata repository. Ascential MetaStage is used to run data lineage queries. Figure 3.18 shows the way to run the data lineage queries in the MetaStage Explorer. This illustration shows that the lineage query looks at the CONSUMER dimension table and what happened to it during DataStage job

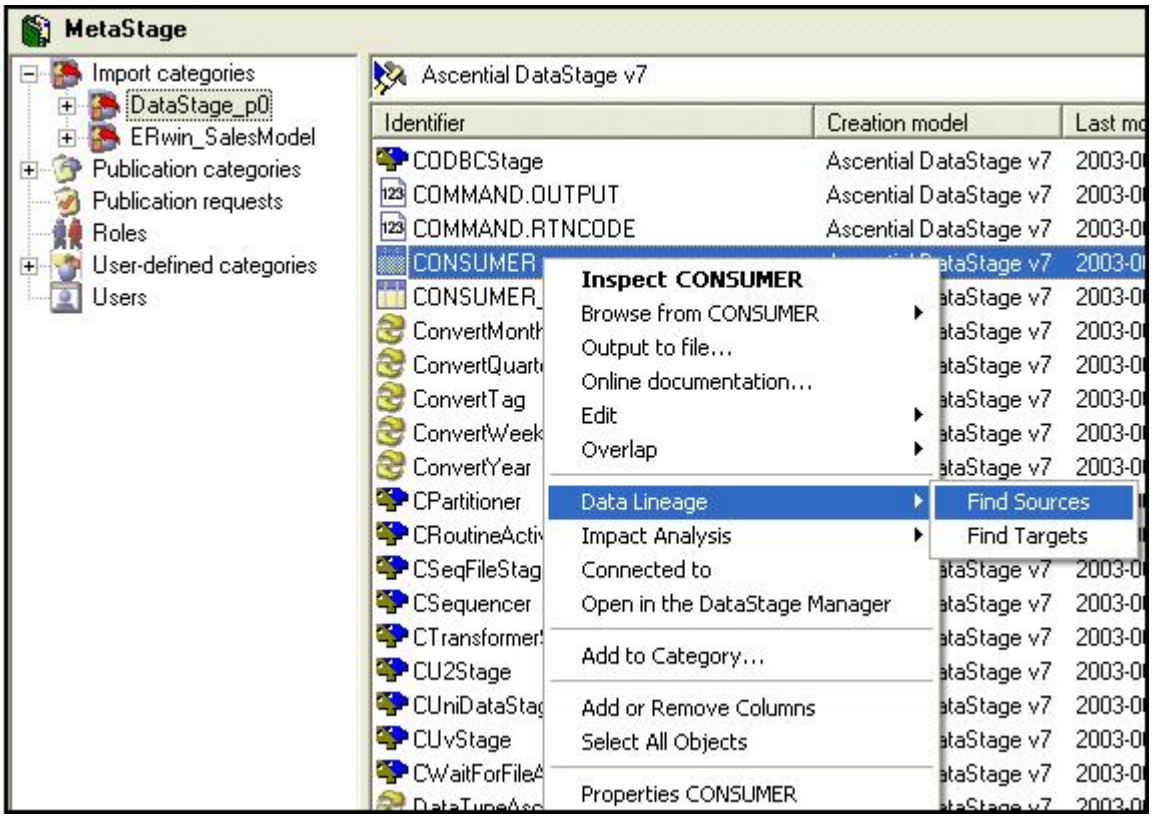

Figure 3.18: Data Lineage Menu - source [39]

runs. Figure 3.19 shows the data lineage path of the CONSUMER dimension. This report presents lineage for a particular DataStage job run showing the data source (consumer.txt), the transformations objects, (fromConsumerFile, ToConsumerTable) and the target. Additionally, it also shows the number of rows inserted to the CONSUMER table. Each object on the data lineage can be inspected in detail to find out more information about the particular object. For example the object toConsumerTable can be opened to investigate the transformations that occurred to each column.

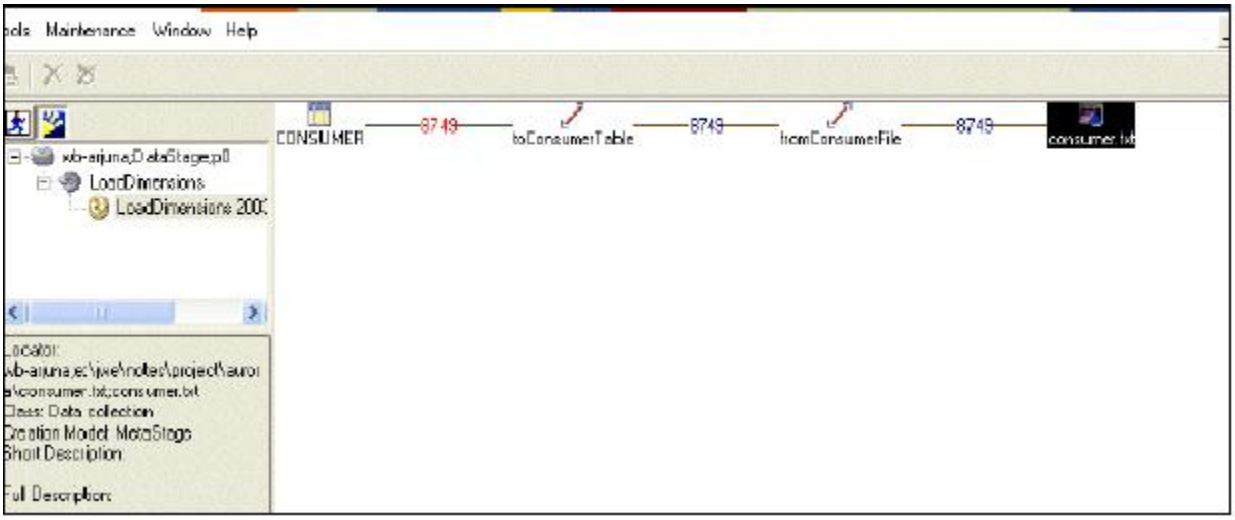

Figure 3.19: Data Lineage Path - source [39]

Chapter 4

Characterizing Data Lineage

The problem of Data lineage Tracing in data warehouses is seen from different perspectives and in different aspects by academic research and industry solution providers. Each academic research included in this study reveals similarities and differences in the way the researchers address the data lineage problem. The industry solutions provide greater similar solutions which are mostly based on the Metamodel standards offered by OMG. From the research I gathered, both from the academe and industry solutions, I noted the differences on how they characterize the data lineage. Existing solutions ultimately produce different types of data lineage information. In this chapter, we characterize data lineage. In section 4.1, we present the different aspects of data lineage problems, then in section 4.2, we describe the composition of data lineage. In section 4.3, we describe the types of data lineage information which appears when tracing the data lineage of a warehouse data item.

4.1 Aspects of Data Lineage Problem

To recommend an alternative solution to the data lineage problem, we first describe the different aspects of data lineage problem that we consider in this study. We

combine the different notions of the data lineage problems from different related works presented in the previous chapters. In this section, we describe the different aspects of data lineage problem. The intention is to concretize the scope of data lineage problems, which this thesis attempts to solve.

4.1.1 Source Systems

The enterprise-wide data warehousing requirement may need to integrate specific related data from different sources to a single data target. For example, customer data may be built from different source databases on different platforms (e.g. OS DBMS) and its data structure may be different (e.g. a customer field length may be 10 alphanumeric in one and 12 in the other).

Different challenges are faced if we attempt to trace back data origin and "drill-through" the source system itself. Besides the technicalities about establishing connection from the data warehouse system to the source system, calculating lineage all the way back to the source system can be difficult if not impossible. Some of the key problems are discussed below.

- **Connectivity** - Establishing connection to a source system must consider hardware and software components. The problem may include consideration of physical network connections, configurations, and the programming required to code the connection and selection of data from the source system.

- **Availability** - The source system may not be always available. It may be down or disconnected or physically located where online connection is not possible. Even if the data warehouse system is able to connect to the source system, the data may already be archived and will not be available anymore during the time

of a data lineage query.

- **Compatibility** - There might be system incompatibility that may mean that establishing connection to the source system directly is not possible. Even if it is possible, establishing connections may not be an option as the cost is much too high for its benefits.

The problem is not limited to the ones described above. In a real data warehouse implementation, performance is an important consideration in querying. Drilling-through the source system to find the data origin can be quite expensive in terms of performance and resources.

4.1.2 Sources of data

It is common in an enterprise to have varying structures of the same data from different places and different systems. The problem of determining **source data**, (i.e. data from sources where the point of data extraction happens) needs to be explicitly described. In this section, we emphasize the importance of explicitly defining the **source data** and present the most common structures of source data in many data warehousing implementations in business enterprise today.

System-of-Record

Identifying the **originating source data** is important in data lineage solution because it helps determine the point at which the lineage tracing ends. We define system-of-record as the **originating source data** as Kimball defines it [6]. We consider the System-of-Record as the point of extraction of data populating the Data warehouse. It is common in an enterprise to have varying versions of the same data.

For example customer Juan dela Cruz is recognized as Juan Cruz in one department and J. dela Cruz in another. Typically, the individual varying version of the same data is extracted, integrated or conformed by the data warehouse ETL process. In this case, the real original data is the system-of-record. However, the System-of-Record may not always resemble the real original data. Original data may have already been altered, manipulated and cleansed and may be transformed within and across heterogenous sources themselves. Some business enterprises establish procedures to conform varying versions of the same data outside the data warehousing activity.

Source Data Structures

Data in the data warehouse often comes from heterogeneous disparate data sources. Data sources may come from a range of old systems to modern systems. Some common examples of sources are the following:

- **Flat Files** - Flat files are common data sources for data warehouse uploads. Data from source system that cannot be reached by or directly connected to the data warehouse extraction module are often prepared in flat files for extraction. The most common example for this, are the **Mainframe sources**. Many large business enterprises use mainframes and are still using this technology up to this time. Therefore, the flat file data source is common in many data warehousing business enterprise implementations.

- **XML Data Source** - An XML data source is a common emerging data source for data warehouses today. Some existing solutions in the industry already have functionality that allows extracting from XML data source.

- **Relational(DB) Data Sources** - Many large and medium enterprises use relational databases for storing the day-to-day business transactions.

- **Non-relational (DB) Data Sources** - Some large and medium enterprises use other DBMS platforms for storing and managing their daily transactions (e.g. Object Relational, Object Oriented).

Data source information may include not only the name of source systems and data sources such as those that are common today. It may include connection information and pointers to connection routines. This information can be used to establish connection to the source system. When a query is performed that requires "drill-through" up to the source system, assuming that this type of solution is prepared in the data warehouse, then the connection information can be used as a parameter to the connection routines to automatically establish connection to the source systems.

To the best of my knowledge, there are no existing solutions offered in the industry yet that offers a real "drill-through" in data lineage giving the lineage values themselves. I know, however, from the research I gathered that the WHIPS project (WareHouse Information Prototype at Stanford) in Standford university, attempted this type of solution. The solution is specific to relational databases only and the data warehouse item in question is stored in a non multi-dimensional schema [7].

4.1.3 Data Transformation

Extraction Transformation and Loading is commonly referred to as data transformation and is an important part of data warehousing. Solving data lineage problem requires knowledge of the data transformation logic from the point of extraction to

the point of loading the data to the warehouse. In between the process of data transformation, data lineage information can be recorded and prepared for lineage queries. The complexity of the transformation process contributes to the challenge in addressing the data lineage problem. The main data lineage information that is part of the transformation are the answer to the questions:

- **When were the data transformed?** Derivation programs may change and evolve as required by business or outside influences. The *When* basically requires to answer the question about which version of the transformation programs was used to derived the data in question (in the data warehouse). It may serve another purposes as well, for example if query is made about which data are transformed when.

- **How were the data transformed?** Basically this should explain the calculation applied to the data when they were derived. This may involved the operators, the logical flow, or data mapping. This information can be used when a query regarding the data derivation is required. Or this can be used as a reference, for basing the calculation of queries about the data lineage value itself.

- **What were the data that participate on the transformation?** Data derivation usually uses data reference for calculation (e.g. translation tables, intermediate views). If data lineage values are required to answer a data warehouse question, the data (combined with the derivation version and description) can be used as a reference for calculating the data lineage queries.

4.1.4 Performance

Performance can not be overlooked when addressing the data lineage problem. Loading data to the warehouse is one of the resource intensive processes of data warehousing. The ETL data model and programs have to be designed in a way that makes the entire process as optimal as possible. Incorporating data lineage to the ETL and program design can make the design more challenging. Saving the data lineage or part of it while populating the data warehouse is sure to penalize the loading process. Therefore careful analysis and preparation is required to create an optimal ETL process design, when the data lineage is to be a part of the ETL process. When a significant part of lineage information is saved, lineage query performance is greatly enhanced.

Data Loading

Deciding whether to save the data lineage or not in the data warehouse environment requires extreme caution because of its potential effect on resources and performance. Moreover, if saving the data lineage is an imperative requirement, the main problem that must be considered is at which point of the process we need to save the data lineage. In data warehouse implementation, the data may be transformed a number of times and saving data after each transformation can be highly impractical.

Lineage Reporting

Required data lineage report may not only be about source-to-target schema or may not only be about the description of how the data are derived similar to the industry solutions. There may sometimes be inquiries regarding the data values themselves. If the original data values of the warehouse item are required, then lineage information

that is saved within the data warehouse environment can be used. It may be too difficult, expensive or even impossible to query data lineage from the source system (refer to section 4.4.1). But when data are within the data warehouse environment, it is not only possible to "drill-through" exact or almost exact data values of data lineage, it can also improve lineage query performance (see discussion in chapter 3).

4.1.5 Maintenance

This section discusses maintenance considerations when incorporating data lineage in the ETL process. Besides the data and execution model maintenance, data update maintenance is an important aspect of data warehousing and incorporating data lineage update increases data update challenges.

Data Transformation Design Maintenance

The Data Transformation Design may evolve and change with time as imposed by business requirements or outside influences (e.g. market forces, legal requirements). When data lineage is incorporated to the Data Transformation Design, maintenance must be advertently considered. Maintenance involves not only allowing changes to the data and execution design, but also involves handling the effects of these changes. For example, if transformation calculation is required to be modified, the lineage tracing query should be able to pick up which version of transformation logic is applied to the data in question. It is evident that changes to the transformation design can directly or indirectly affect the data lineage computation logic.

Warehouse Updates Maintenance

In real data warehouse implementation, there are basically two types of data uploads. We can refer to them as Full upload and Incremental upload. Full upload, refers to complete data uploads and incremental upload, refers to uploads that only include new or modified data from the source. Incremental data upload is common in real data warehouse implementations. Many research works have been done for this type of maintenance and industry solutions have already stabilize this requirement. However, if we consider saving data lineage values within the data warehouse environment, Incremental updates to auxiliary or intermediate lineage views presents a real challenge.

Tools

Maintenance tools are essential for data warehousing systems. A data lineage related maintenance tool also has to be incorporated if we consider data lineage solutions. Many solutions in the industry already provide tools for data lineage related maintenance. Oracle provides Oracle Warehouse Builder, which allows the user to create ETL physical data base models which include the data lineage. Acential software, Microsoft and Teradata provides similar features. However as the data lineage solution matures, the tools also need to be enhanced to be able to adapt to the data lineage maintenance requirements.

4.2 Composition of Data Lineage

In this section, we describe the data that make up or create the data lineage information to be used for data lineage tracing. First, we describe the data lineage

information containers, then we discuss the metadata that are used and referred during data lineage tracing.

4.2.1 Data Lineage Containers

Based on the study we conducted both in the academe and industry solutions, warehouse data lineage information can basically be derived from Metadata Repository and from Data Receptacles. Although, from the technical point of view, Metadata Repositories and Data Receptacles overlap, this section presents them to separately classify data lineage information in data lineage tracing point of view.

Metadata Repository

Metadata repository houses the data about all data elements and descriptions of how these data elements work together. In chapter 3 section 3.1.2, we present the overview of the Standard Metadata Model Specification and how it addresses the data lineage problem. Metadata indeed plays an indispensable role in data lineage tracing. At some point of any data lineage calculation, access to metadata repository is necessary (e.g. getting the source and target schema definition or accessing information about the load processes). metadata is a wide and broad topic in data warehousing world because its about everything except the data itself. This work only refers to metadata that participates in data lineage tracing computation.

Data Receptacles

Data receptacles are data containers that store intermediate and persistent data during Warehouse Transformation Process. During the data's journey from source to target, it typically passes through a transfer point or a container where it is staged

and prepared for further transformation. In chapter 2, we presented a simple data transformation by illustrating a transformation graph, refer to figure 2.4. The intermediate views in the transformation graph can be seen as data receptacles. Moreover, in section 2.4.2, we illustrate how these views are used to trace the data lineage.

4.2.2 Metadata for Lineage Tracing

Basically, we can categorize the data lineage basing on it's container (i.e. metadata repository and data receptacles). In section 2, we describe tracing procedure based on lineage granularity levels as described by academic research; Schema Level and Instance level. Schema level refers to tracing mechanisms which rely on metadata - this produce source-to-target mapping reports, while instance level tracing refers to lineage tracing mechanisms which produced the data lineage values itself. However there are some overlaps when we categorize data lineage in this manner as data receptacles themselves are part of metadata. Additionally some instance level tracing described in chapter 2 actually uses some part of data lineage information from the metadata to reconstruct the data lineage values.

In this section, we expands a little bit more on metadata and its important role in data lineage tracing. Metadata encompassing the following are of interest to data lineage tracing:

Technical MetaData

Technical Definition of data is primarily the main participants in data lineage tracing because it defines the container and structure of the data. This part includes the definition of data receptacles, and the ultimate warehouse data targets.

- **Data Definitions** - Include table names, column names data types (e.g. numeric, character, date), domain (i.e. set of values allowed to be entered into a column), relationships etc.

- **Business Rules** - Can include allowed values to default values which may be embedded in data definitions or may be part of procedures, functions or routines which transforms data.

ETL Metadata

In chapter 3, we briefly presented the overview of the Transformation and Warehouse Operataions Metadata from the Standard Metadata Specification. Additionally, we described the ETL and Metadata services provided by some software vendors or industry solution providers. Some ETL tools exhibits ability to generate metadata automatically from ETL logical design which is created using a Graphical Tool. Here, we discussed different categories of ETL-generated metadata similar to categories described in [6, 35, 36, 40].

- **ETL job metadata** - ETL job [1] metadata determines the path of the data's journey from source up to its final resting place in the warehouse (i.e. from extraction to load and all the transformation in between). ETL job can provide Source-to-target mapping information which is an important data lineage information. Mapping gives the basic data lineage questions: "*where the data comes from?*". ETL job is like the transformation package referred in the Standard Metadata Specification.

[1] An ETL job is a collection of transformations that performs the physical extraction, transformation and load routines. The metadata for a job is the source-to-target mapping.

- **Transformation Metadata** - Transformations are typically composed of custom functions, procedures, and routines which may involve simple to complex data manipulation and calculation. This is a part of data's journey that is most difficult to document and be offered as part of metadata. Some ways to make data lineage tracing possible using this metadata is to introduce custom inverse functions, procedures and routines that may calculate the data origin for the a specific transformation. This information is part of ETL metadata but can be used as a reference for calculating the data lineage values in instance level granularity.

- **Process Execution Metadata** - Information about the execution process and about the actual load-process results. This metadata is like the warehouse operation metadata described in Standard Metadata Model Specification. The information in this metadata can also be a source for upload history and measurement values which can be used for other data lineage related reports (i.e. used as a reference for calculating data lineage approximate values)

4.3 Data lineage Information

This section describes the different classification of data lineage information that can be made available in the report to answer the questions about the origin of data in the data warehouse. In section on 2.3.1, we identified the different lineage information as presented from different academic works. In this section, we combine and distill the identified data lineage information from academic works and and solutions offered in the industry.

4.3.1 Mapping Information

The most common data lineage solution offered by the academe and the industry is the schema mapping. Source-to-target lineage path reports are constructed from ETL job metadata combined with the technical metadata described in previous section. Example for this type of lineage reporting is offered by Oracle and Ascential Software in previous chapter. This section discusses a data lineage classification that pertains to the objects which participate in source-to-target mapping.

Schema Information

Schema Information in data lineage report is basically data coming from the technical metadata described in previous section. The Transformation Package in the Standard Metamodel specification provides constructs that support source-to-target mapping, refer to section 3.1. It provides specification that introduced classes and associations which deals with sources and targets. The sources and targets can be any object sets and the elements of a data object are typically tables, columns, or model elements that represent transient or persistent objects. Data objects sets can be both sources and targets for different transformation. In section 2.3.2, we also presented a specific example that showed different schema information being stored in the metadata storage. This metadata storage implementation is further illustrated in figure 2.12. Section 2.3.1 describes the different data lineage information in terms of schemas and schema elements. **Schema** pertains to table or view names, table or view definitions, or parts of the schema definition. **Schema elements** pertains to the columns or attribute names and their types (e.g. string, integer etc.)

Execution flow

Standard metamodel provides constructs that determines the flow of the transformation execution. This information is tightly connected with the schema mapping. While schema mapping determines the sources and targets, execution flow determines the sequence of transformation for a transformation package. The transformation flow metadata model is prepared to contain the Transformation tasks, Transformation Steps, Transformation activity and other models related to transformation execution. The transformation task may contain inverse task which become a reference for doing inverse calculation to get the source value for a transformed item (i.e. intermediate data or the data in the warehouse data target). Most data transformations do not only involve source-to-target mapping, but may also involve custom function, procedures or routines which perform simple to complex calculation of data before delivering them to next data transfer points. Some ETL software vendors provide data lineage reports by allowing users to navigate through the sources and targets and "drill-down" to look-up the functions, routines or procedures that are responsible for the data manipulation specific to an ETL job. In this respect, we consider function, procedure and routines as data lineage information, although it does not actually produce the data lineage but rather it produce information about the data lineage.

4.3.2 Lineage Data Values

The finest granularity level of data lineage is the lineage data values themselves. This level of granularity is the most desirable, however the most expensive in terms of resources and performance. In section 2.2.2, we presented an instance level lineage

tracing approaches extracted from different academic research. Some mechanism offered to re-construct the data lineage information, as needed, using minimum amount of lineage data. Another approach is to store base data or parts of it within the data warehouse environment, or store each transformed data in auxiliary views within the data warehouse environment. All of the approaches attempts to provide data lineage information, not in terms of mappings but in terms of data lineage values themselves [22, 20, 9, 10, 45, 12, 7, 5].

Base data

Any data that basically is raw extracted from the ultimate source or origin from the source system is considered the base data. This may be a table or tables, view or views from the source system which is the point of extraction of our ETL process. This base data can be complete, partial, or projection from the base tables, refer to section 2.3.2. Algorithms for storing base data or parts of it within the data warehouse is introduced in [7, 10]. Storing base data within the data warehouse is one option to address lineage problem describe in section 4.1.1. about source systems and data connectivity, availability and compatibility.

Intermediate Data

Intermediate data is data being staged for further transformation. In real data warehouse implementation, transformation often happens in many stages. In each stage, the data are stored (either temporarily or permanently) for further calculation until they finally reach the warehouse data target. Storing intermediate data after each or series of transformation or after major transformation can also be an option that

addresses data lineage problem describe in section 4.1.1. When Base data and Intermediate data are stored within the data warehouse environment, it can be possible to visualize data in reverse order, following its transformation, step by step, up to the point of extraction. A simple illustration of lineage tracing mechanism that allows visualizing data in reverse order, in step-wise fashion, is shown in section 2.4.2.

Other data lineage related Data

Some data lineage solution rely on limited amount of data to reconstruct data lineage information [19, 56]. [56] uses inverse function and some portion of base data and metadata to calculate the approximate data lineage value. [19] described a solution that reconstruct the original data from the summary data (e.g. warehouse data) combined with limited amount of information. This mechanism uses statistical information about the data upload history and other pertinent information which are recorded during data loads. Some solutions record information about the loading process and store it as part of data target [2, 6, 1].

Chapter 5

Data Lineage Solution

In this chapter, we combine and distill the existing solutions both in the academe and industry discussed in previous chapters.

In section 5.1, we summarize the tracing mechanisms which form part of our data lineage system conceptual framework. In section 5.2, we discuss the conceptual framework, comprising of Lineage Model and Lineage Functions. Finally we discuss the Lineage Design and Implementation in sections 5.3 and 5.4

5.1 Data Lineage Tracing Mechanism

Based on the data we gathered from our research, we summarize the data lineage tracing mechanisms into three main categories: Map, Reconstruct and Step-by-step tracing mechanisms (see figure 5.1). In this section, we discuss each tracing mechanism category.

Tracing Mechanism	Data Used in Tracing	Information in Lineage Reports
Mapping	Metadata	• Relationships of sources and targets • Fucnction, Procedures, Routines, • Algebraic and Mathematical Operators
Reconstructing	Metada, Other related data	• Approximate data lineage values
Step-by-Step	Intermedieate Lineage Data	• Actual Data lineage values

Figure 5.1: Summary of Data Lineage Tracing Mechanism

5.1.1 Map

The source-to-target mapping mechanism provides information about the path the data traversed as they move from the point of origin until reaching the final destination in the data warehouse. The source-to-target tracing mechanism is a solution already provided in the industry. Each of the software vendors evaluated in this work, provides this tracing mechanism at different granularity levels. To simplify the granularity levels, we categorized them into three: Column-to-Column, Table-to-Table and Object-to-Object. In a real data warehouse implementation, however, a single ETL job may comprise different combinations of these mappings.

- **Column-to-Column mapping** - This mapping mechanism provides information about the source and destination column of the data. This mapping may be straightforward, may use transformation standard routines, or may involve hand-coded calculation.

- **Table-to-table mapping** - This mapping mechanism provides information about source and destination tables or views. This mapping may be straightforward, may use transformation standard routines, or may involve hand-coded

calculation.

- **Object-to-Object** - An object may consist of a combination of different structures or data stores (e.g. file to view). This mapping may use transformation standard routines or may involve hand-coded calculation.

We can categorize all of the mappings above and all the other source-to-target mappings under Object-to-Object mapping which can be arbitrarily any type of schema mapping.

5.1.2 Reconstruct

The Reconstruct tracing mechanism recreates a good estimate of the data origin based on data stored within the data warehouse environment and inverse functions. The items which can be needed for approximate data origin reconstruction are the following:

- **Metadata** - The type of metadata that can be needed for data lineage recreation are process execution metadata discussed in section 4.2.2.
- **Data Lineage related information** - This may or may not be required depending on the data lineage calculation requirement. The data is to be created during transformation or to be registered as look-up data which can be referred during lineage tracing.
- **Warehouse data item** - The warehouse data item in question.
- **Inverse function** - A function that calculates the approximate value of data origin.

5.1.3 Step-by-step

This tracing mechanism allows for tracing the data lineage in step-wise fashion following the transformation, step by step. This type of tracing mechanism is still an active research topic in the data integration arena in relation to many conflicting issues which need to be resolved. In section 2, we presented an existing tracing mechanism that offers this type of lineage tracing mechanism. Basically this mechanism requires the following:

- **Lineage Data Values** - These can be the intermediate values or base values described in section 4.3.2. or these can be data that are stored after major data transformation.
- **Tracing Query** - Tracing query calculates the data lineage based on the transformation query that produces the warehouse data item in question.
- **Tracing Procedure** - This tracing mechanism determines the tracing procedure to be used based on the transformation property of the transformation that produces the warehouse data item.

Step-wise fashion tracing mechanism may use a combination of the items mentioned above and may also use metadata. In a real data warehouse implementation, a warehouse data item may undergo many transformations. Storing intermediate data values after each transformation may be too expensive in terms of storage and loading performance. An optimal and practical solution is still an area for improvement in this type of tracing mechanism.

5.2 Conceptual Framework for Lineage System

Conceptual Framework helps organize, plan, manage and set things in context. In this section, we present the components of the conceptual framework for a data lineage system. The framework consists of two main components: Lineage Physical Storage and the Lineage functions. Lineage Physical Storage comprises the Metadata Repository and two categories of Data Receptacles and the lineage functions subcomponents are Storing, Reporting (i.e. Query, Navigate) and Lineage Adminstration and Maintenance components. Figure 5.2 shows the summary of our lineage system conceptual framework component.

5.2.1 Lineage Physical Storage

The Lineage Physical Storage consists of two main categories; Metadata Repository and Data Receptacles discussed in section 4.2.1. This section discusses the Lineage models comprising of Metadata Data receptacle model.

Lineage MetaModel

Our data lineage solution considers the Standard Metamodel Specification. The data transformation and warehouse operations Standard Metamodel, which is backed up by industry-leading database software vendors, provides a framework that addresses the basic data lineage problem and can be extended to provide data lineage solution that provides finer level of granularity. Section 3.1.3 discusses the overview of Standard metamodels for Data Lineage support. For details of the data transformation standard metamodel specification, refer to [40].

The data transformation and warehouse operation Metamodel, models the transformation process of the data from the time the data are extracted from the source

Lineage Physical Storage	Lineage Function			
	Recording	Reporting		Administration and Maitenance
		Query	*Navigate*	
Metadata	Populate Metadata	Query generator for Lineage Metadata	Metadata Query Generator	Automatic generation of lineage metamodel from ETL design Modifying models or transformation definition Specifier for tracing query, tracing procedure, inverse function
Data Receptacle: Other data lineage related storage (*this may form part of the data target or a separate view or table*)	Populating storage for other lineage related data	Query for recontructing approximate value of data orgin	Inverse Function generator	Specifier for data receptacle Modifying programs for populating data receptacles Specifier for inverse function
Data Receptacle: Base, Intermediate Lineage Storage	Populate Base and Intermmediate Lineage and/or tables	Query generator for Base and Intermediate Lineage data (*may be combined with Metadata*)	Tracing query generator Tracing procedure generator	Specifier for Base or Intermediate Lineage data receptacle Modifying programs for populating base and intermediate storage Specifer for tracing query and tracing procedure.

Figure 5.2: Lineage System Conceptual Framework Component

system up to the time they are loaded to the data targets. The standard metamodel provides constructs to document the data lineage and accommodates "black box" and "white box" transformation. In "black box" transformation, data sources and targets are related through the transformation, but how the source is related to a specific piece of a data target is not known. White box transformations, however, can show how the data sources and targets are related to a transformation and exactly how a specific piece of a data source is related to a specific piece of a data target through a specific part of the transformation.

Transformations can be grouped into logical units as Functional and Execution logical units. At the functional level, a logical unit defines a single unit of work, within which all transformations must be executed and completed together. At the execution level, logical units can be used to define the execution grouping and sequencing. An important consideration is that both parallel and sequential executions (or a combination of both) can be accommodated.

Standard Metamodel specification addresses the Lineage metamodel requirement and is ready to be used and exploited. Complying with the Standard Metamodel ensures greater possibility of compatibility with other platforms and products and wider possibility of being implemented.

Another data lineage solution that we consider in this study is the one discussed in sections 2.2.1 and 2.3.2. These sections provide a good illustration of the data lineage solution that utilizes the metadata storage. Velegrakis, Miller and Mylopoulos [52] provide a metadata storage schema and develop an extended query language that allows metadata to be queried as a regular data. This enables to query about transformations involving source-to-target mapping report.

Data Model

From the academic point of view, the granularity of data lineage information within metadata repository is at course-grain level. Data Lineage information provided by a metadata repository is about data source-to-target mapping and tansformation description (i.e. lineage information discussed in sections 4.2.1, 4.2.2. and 4.2.3). The data transformation standard metamodel can be extended by providing a way to trace data lineage at finer levels of granularity. That is, besides including the the schema definitions of auxiliary data containers (i.e. intermediate data targets) it can also include data lineage tracing mechanisms for each transformation activity. Transformation involves data sources and targets. In a Transformation process, a source may function as a target of a certain transformation and a data target may function as a data source for yet another transformation. A Standard Metamodel provides an option to make these data sources and targets persistent or transient. Making the source and target objects persistent is a way to prepare for finer levels of data lineage tracing. Source and target objects which function as transfer points during transformation are data receptacles. The data receptacles model models the storage of lineage data values, not the mappings as it is with the transformation metamodel. In this section, we discuss data receptacle the data model and the other lineage-related data models.

Base and Intermediate Lineage data model. Base and Intermediate data are described in chapter 4 section 4.2.4. Figure 5.2, shows the Base and Intermediate storage as part of the components in our proposed lineage system. The Base and Intermediate Lineage storage records base and intermediate data lineage values, which will support tracing the data origin in as fine a level of granularity as possible. Our

proposed solution considers the idea for storing auxiliary views introduced by Cui and Widom in [7, 10]. A part in Section 2.3.2 discussed the overview of Storing auxiliary views. For details, refer to [7, 10].

Other data lineage related data model. Models the other data lineage values that can be needed for recreating data lineage. The storage for other data lineage related information can be an independent storage which contains pertinent information about the transformed data, or may be part of the warehouse data target. Base and intermediate storage stores data lineage values itself while this data storage records pertinent information related to data lineage.

5.2.2 Lineage functions

Figure 5.2 shows the Lineage function components in our Lineage System Conceptual Framework. Lineage function components includes: Storing, Query, Navigation, and Maintenance and Administration. In this section we discuss these different components.

Storing functions

The Storing function component in our proposed lineage system must be integrable with the ETL functions. During the ETL design the lineage storing function must also be defined with it. This is necessary because the Lineage Storing function is executed during the ETL process (see figure 5.5). Furthermore, if lineage navigation is enabled, ETL design must include the *tracing query* or *tracing procedures* or combination of both and therefore they should also be stored as part of the data transformation definition.

Populating Metadata. Transformation metadata storage is prepared and is

populated when the ETL design specification is translated into a physical implementation (recall OWB and Ascential Software tools in chapter 3). The types of lineage information stored in the metadata repository are Sources of Data (e.g. Source systems, source data structures), mapping information (e.g. data sources and targets and the transformation functional and Execution information, i.e. transformation logical units, and sequence of execution), and the scripts and procedural code (or pointers to scripts and procedural code). For details on different classification of data lineage information, refer to chapter 4, sections 4.2.1 to 4.2.3. This information serves two main purposes: for transformation processes and for lineage query and navigation (recall figures 2.7, 2.8 and 2.12). These figures provide a good illustration for storing the mapping information or data transformation elements to metadata storage. A storing function is used to store data transformation elements into the metadata storage. The illustration in figure 2.8 shows the metadata storage which is populated by mapping **m1** in figure (2.7).

Populating Base and Intermediate Lineage Storage. Base and intermediate lineage storage are populated during data transformation. Section 4.2.4 describes base and intermediate data values. The option to permanently store the intermediate data values must be deliberately specified during the design process. If the problem we describe in 4.1.1 is an actual problem (e.g. that connecting to the source system from data warehouse is not feasible), then we may decide to store the base data within the data warehouse system. In this case, the option to store base and lineage data within the warehouse environment needs to be deliberately specified so that it can be implemented in the actual ETL process.

Populating Other Lineage Related storage. This storage can be populated

automatically during the ETL process or it may be registered manually in the Warehousing Adminstration component.

Lineage Query

The intention of storing data lineage information is to make it available when lineage question is asked. The Query component assists in creating or generating queries imposed by the user. In our lineage system, we recommend that the Query component must provide a feature that generates queries via a user friendly interface similar to the solutions we presented in chapter 3.

Querying metadata. Most common existing data lineage solutions rely on querying the metadata to provide data lineage report. The five industry solutions we evaluated in chapter 3 already implemented a data lineage solution that queries against a metadata repository. Most common reports they provide are about the source-to-target mapping. Our proposed lineage system includes a metadata query generator. Essentially, this component should comprise of routines that dynamically prepare scripts for querying against the metadata based at the user's request.

Querying Base, Intermediate and other data lineage related storage . A question may require lineage data values as an answer not only the source or target information. The query generator for Base and Intermediate Lineage values should provide the ability to dynamically generate a query that answer questions regarding a source for transformed data. This component assists in generating queries imposed on Base and intermediate data lineage during navigation.

On the other hand, a query may be imposed to provide an approximate value about the source of the data. This query is typically imposed on the upload history and measurement data.

Navigate

The **Navigate** component provides a feature that allows a user to browse (backward or forward) through the lineage. The **Navigate** component combines *tracing query generator* and tracing procedure, *inverse function* or routines to navigate through data lineage reports.

Chapter 2 discussed the overview of data lineage tracing mechanisms. The concepts being presented in chapter 2 can be referred to when deciding which type of tracing mechanism is most applicable for a certain situation. The **Navigate** component is different from an ordinary query but it may invoke the functions of the **Query** component. Browsing through the lineage requires a user friendly interface that allows the user to go backward from the lineage or forward to the ancestor of the lineage. The **Navigate** component may involve data from metadata storage and combine it with lineage data storage.

Metadata Query. Metadata query provides a feature for tools to query against metadata.

Tracing Query. The **Tracing query** introduced in [7, 10] is considered in our lineage system although the algorithms need to be extended to handle multi-dimensional schema. Section 2.4.2 discusses view tracing query and provides example application for the tracing query introduced in [7, 10].

Tracing Procedure. We consider the tracing procedure introduced in [7, 10]. Using this tracing mechanism requires information about transformation properties for each transformation applied to the data. This information needs to be registered in the Adminstration function component. In section 2.2.2, concerning subsection *lineage tracing for general data warehouse transformation*, we discussed the Tracing

procedure approach to data the lineage problem.

Inverse function. Our discussion in section 2.2.2 concerning subsection *Inverse method*, describes an approach for lineage tracing functionality. We consider this solution as an option for lineage tracing in our lineage system. The existing research which are evaluated in this thesis can be exploited and translated into functions that can be useful for enterprise data warehouse lineage requirement.

Administration and Maintenance

The Lineage Administration and maintenance component is essential to our lineage system. This component is responsible for designing, modifying and updating lineage models and functions. This component must consist of objects (programs routines and interfaces) that can be plugged into ETL and Data Design Tools. Administration and Maintenance components involving data lineage can include but are not limited to the following functions:

- **Automatic generation of lineage metamodel from ETL design** - Components (or functions) that perform automatic generation of lineage metamodel and codes from ETL design. Generating metadata from ETL design is already an existing solution offered in the industry (refer to the industry solutions we evaluated and presented in chapter 3, e.g. Oracle). Components (or functions) involving data lineage design must be plugged into the ETL designer. Each transformation may produce intermediate lineage values which will be useful for lineage tracing; therefore, it is during ETL design that data lineage design is best integrated.

- **Modifying models or transformation definition** - Components (or functions) that facilitate and control the editing and updating of lineage design. When an ETL design is altered, it potentially affects data lineage design. Therefore, the changes to lineage design must also be documented alongside the ETL design changes. This is important because a specific version of transformation needs to be tracked and known during lineage tracing.

- **Specifier for tracing query, tracing procedure, inverse function** - Components (or functions) that assist on registering the tracing query, tracing procedure or inverse functions to the data warehouse environment.

- **Specifier for Data Lineage Storage** - Components (or functions) that assist on specifying the Base, Intermediate and Data Lineage related Storage.

- **Modifying programs for populating data lineage related storage** - Components (or functions) that assist on modifying the scripts, routines or procedural codes that populate data lineage related storage within the data warehouse environment.

Some of the enumerated functions for Administration and maintenance presented above are already existing features in the industry solutions. Oracle, Microsoft and Ascential software provide a tool that allows for designing ETL intuitively via a graphical interface. We recommend similar features and extend this tool to include a lineage storage specifier that can be used for navigation. As it is now, at the time of this writing, the solutions we evaluated offered lineage reporting against metadata in terms of mapping diagram (refer to section 3.2.3). These reports are often used for ETL maintenance and administration.

5.3 Design

Data lineage design consists of the physical data models and the programs that manipulate the data. The metadata model can be extended to include separate storage model which include base, intermediate and other related data lineage values. Most ETL tools in the industry already offer solution in handling transformation metamodel design. These tools may be extended to handle lineage design in some granularity level not currently offered in the industry (i.e to extend for lineage navigation in terms of data values not only schema mapping and transformation description).

5.3.1 Lineage Components

The components of the data lineage system must be integrable to the ETL design and to data warehouse reporting system. For the components to be useful to existing solutions, they must be seamlessly integrated with data warehouse systems. Figure 5.3 shows lineage data residing within the warehouse system environment. This figure illustrates a seamlessly integrated lineage system to the data warehouse system. The lineage system components are shown in figure 5.2. **Physical storage** and **Lineage functions** become part of the warehouse systems and are established during the data warehouse ETL design.

Physical Components

Column one in figure 5.2, shows the Lineage physical component comprising Lineage metadata, data receptacles and other data lineage related storage. These subcomponents become part of the data warehouse environment (see figure 5.3).

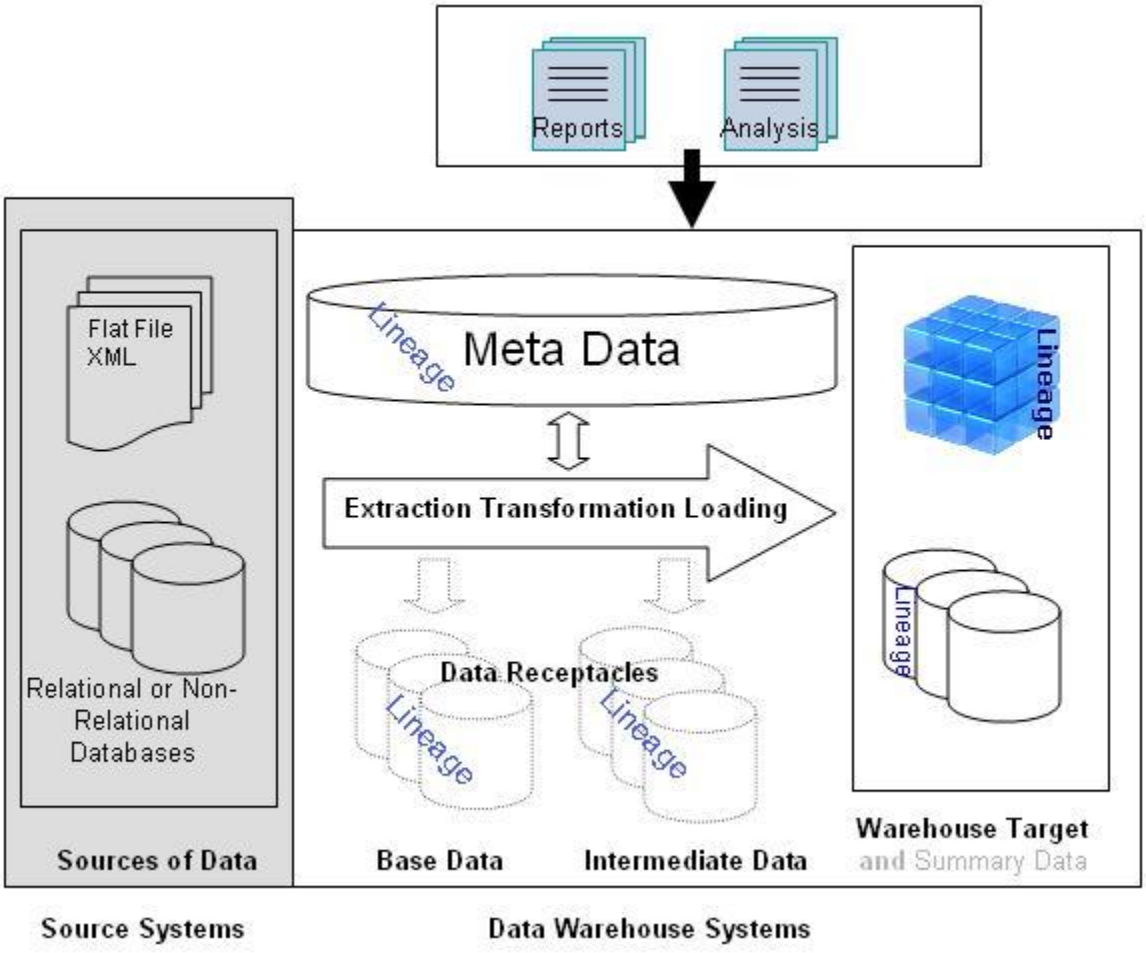

Figure 5.3: Data lineage within the warehouse environment

Procedural Components

Lineage functions are the procedural components in our data lineage system framework. Lineage functions are comprised of three subcomponents: Recording, Reporting and Administration and Maintenance components (refer to figure 5.2). The Recording component records the data lineage or data lineage related information within the data warehouse. Reporting components produce the lineage information for the data warehouse information recipients and the Administration and Maintenance is the component which allows for data lineage solutions enhancement, modification, as well as handling day-to-day data lineage system maintenance.

5.3.2 Design considerations

In designing data lineage, the aspects of the data lineage problem must be given careful consideration (recall section 4.1). Section 4.1 discussed the different aspects of data lineage problem that may affect decisions on data lineage design. Design requirements are dictated by business and sometimes by legal requirements. Balancing the contradicting and competing requirements are important consideration in lineage design and the following are common areas:

- **Granularity against Complexity** - Tracing data lineage at the finest granularity level requires permanently storing and saving the data lineage within the data warehouse environment and possibly being able to establish connection to the source system to find the ultimate data origin. This requirement raises consideration for detailed analysis on types of data sources, e.g., source system and source data structure (refer to section 4.1.1). The complexity of creating a path (a combination of base, intermediate or other data lineage related values)

and tracing procedures and functions can also be an important consideration that should be balanced with the level of granularity the lineage system should provide. Another very important consideration is the complexity of maintaining lineage data values that are stored within the data warehouse environment (e.g. handling incremental upload).

- **Loading against Reporting** - Tracing the data lineage of a specific warehouse item value can be very expensive in terms of performance. One obvious solution to the problem is to store base and intermediate lineage values within the data warehouse. Storing data lineage values within the data warehouse environment has to consider the best possible optimal design to minimize loading performance. However, being able to find the optimal loading design does not ensure being able to completely minimize the loading performance if lineage value is to be stored within the data warehouse environment.

5.3.3 Design Components

The main data lineage design components include, ETL design, Specification of Lineage Physical Storage and Registration of the tracing queries, procedures or inverse functions. Figure 5.4 illustrates the main data lineage design components. Although designing the data lineage storage is oftentimes included in ETL design, we specify Design ETL in a separate box to emphasize its main function.

Design ETL

A component that handles the data lineage design in ETL. Designing Lineage involves storage and lineage functions. The design can be made part of the ETL tool standard design but can allow manual and hand-coded design (i.e allows the user to create a

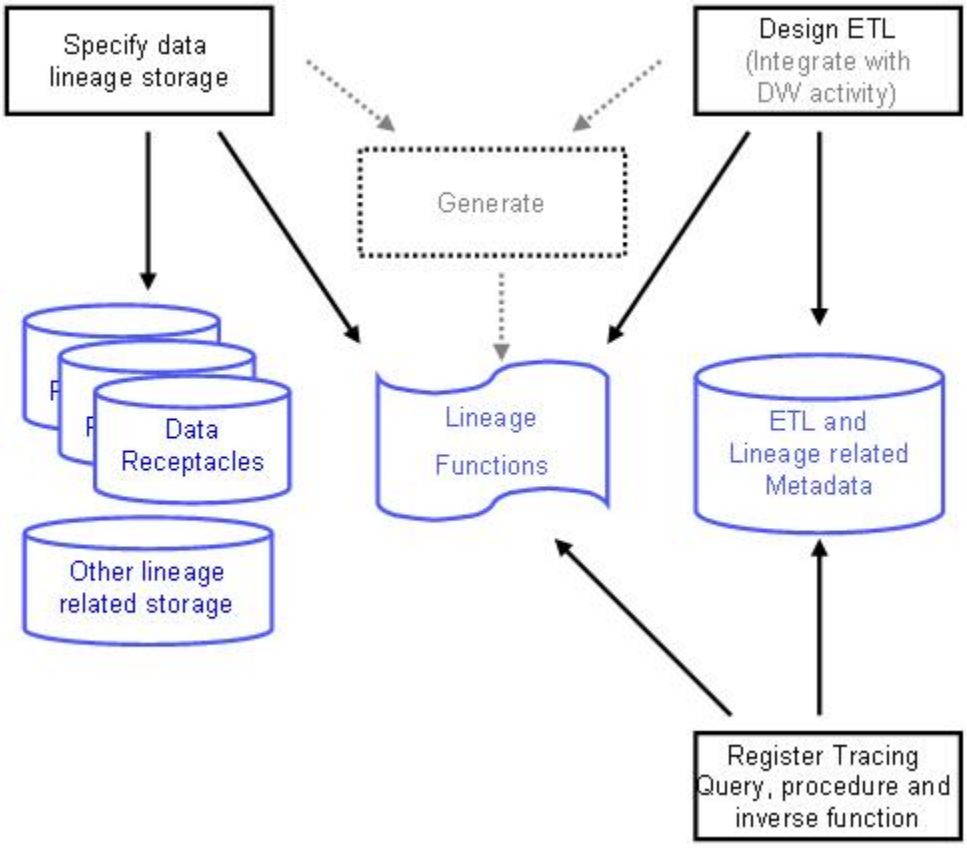

Figure 5.4: Designing Data lineage

customized design which can often be the case in real data warehouse implementation). These tools provide an interface that allows for graphically specifying ETL and automatically generating the transformation metadata and functions. Existing solutions may be extended to generate lineage functions as well (lineage functions that are described in different part of this thesis).

Specify Base, Intermediate and other lineage related storage

A component that will handle the design and modification of the base, lineage views or tables. Design options to activate or deactivate data lineage are recommended as part of our lineage solutions in the data warehouse. If navigation is activated to be up to the finest granularity level, it is necessary to specify the base or lineage views or tables.

Register Tracing Query, procedure or inverse functions

Activating the data lineage traceability may require the designer to register a *tracing query*, *procedure* or *inverse function* for certain transformation. This component allows for registering the Tracing Query, procedure or inverse functions in the data warehouse systems.

5.4 Implementation

Data lineage implementation begins from the point of data extraction from source system, loading the data to the warehouse and Reporting. Lineage system components, which participate in the implementation, handle two main roles: **Recording** and **Reporting** the data lineage. Data lineage implementation is based on the different lineage components which are established during the design.

5.4.1 Recording

When the data warehouse ETL design is translated into a physical design the lineage metadata will be automatically generated and populated and form part of the metadata repository. The ETL design tool records the Transformation information mapping to the metadata. Basically this information tells something about the data lineage which includes the source-to-target schema and schema elements, the procedure, inverse function or routines (or pointer to them) which will execute the transformation and the sequence of transformation (refer to figure 5.4). Recording to the base, intermediate and other related data lineage information happens during actual ETL job runs, although in some cases, other related data lineage information may be registered and recorded independent of ETL job run. Figure 5.5 illustrates the data lineage which is recorded during the Extraction, Transformation and Loading process. The base and Lineage data may not be recorded if not deliberately specified during the design.

5.4.2 Reporting

We categorize the main purposes of data lineage reports: For Administration and Maintenance and for Data Analysis and Validation. Our lineage solution includes components that will handle lineage queries. Both of the lineage reports' purposes may use query or navigate or combination of both.

Administration and Maintenance

Most existing solutions provide lineage reports in terms of schema and schema element mappings. This kind of report is basically used for administration and maintenance purposes. Some solutions are beginning to consider this information as a foundation

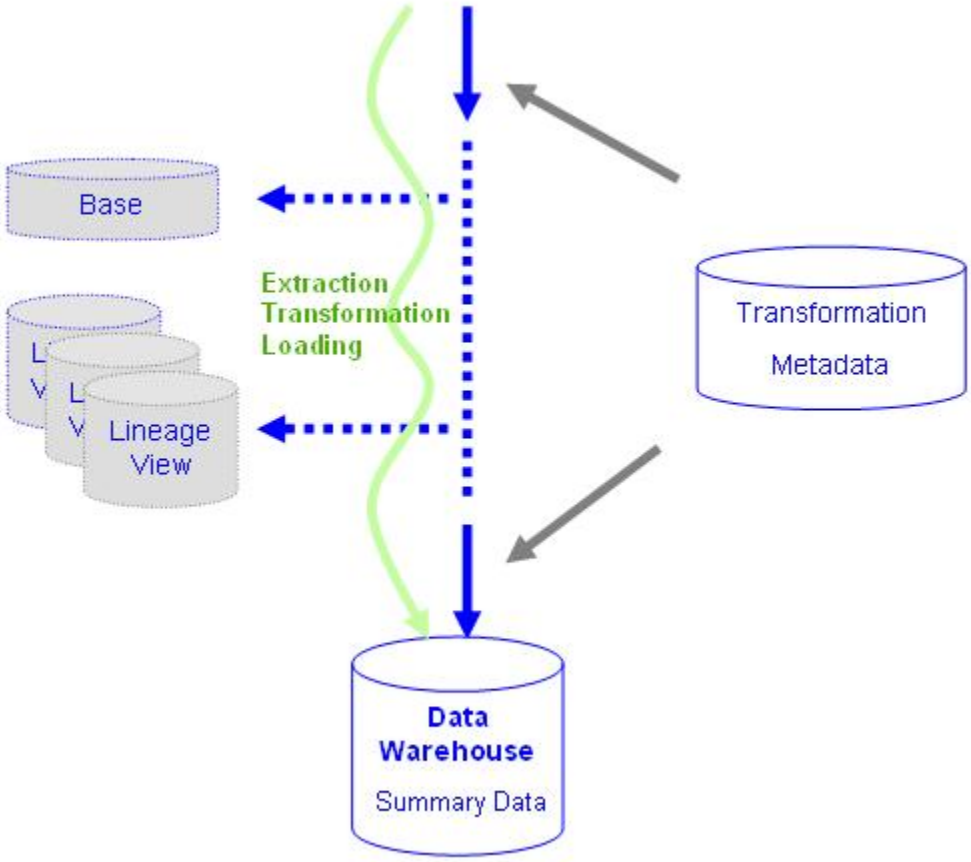

Figure 5.5: Lineage Data flow

for more detailed lineage reporting. The schema mapping report can provide the user with the answer to the basic question, *where did the data come from?*. The report may enable the users to "drill-down" to know the derivation description and answer the question, *what are the calculations being applied to the data from one transformation to the other?*. But this information can oftentimes be too technical for the users and therefore less helpful for them. However, for the systems administrator and other technical persons, this can be an important information when source data analysis is required. Some common daily Administration and Maintenance activities are to check the load status and check the data quality.

Data Analysis and Validation

Data in the data warehouse undergoes several transformations that may obscure the traces of the the origin of data. Reporting the data lineage at the lineage data values level can be necessary and thus is an important part of a data warehouse solution. Data lineage reporting may use the following approaches to reporting:

Step-wise fashion Lineage tracing. The sources of data or the system-of-record described in section 4.1.2 are the ultimate or the last points of data lineage tracing if tracing is allowed up to the source systems. This kind of tracing (going back up to the source system) is potentially a rare choice for data lineage tracing because of the problems we discussed in section 4.1.1, i.e., problems with source system connectivity and compatibility, and source systems data availability. However, in some cases, this may be feasible and a requirement to be able trace lineage up to the source system may arise. There is a possibility that the source data already consists of processed data within the heterogenous source system. But we describe our system-of-record data as our ultimate data origin (refer to section 4.1.1). The

reason for this is because we can only integrate our lineage system in ETL and Data warehouse reporting systems. Therefore data outside the ETL reach cannot be part of our data lineage tracing, unless these data become part of the system-of-record.

Step-wise fashion lineage tracing aims to provide exact or almost exact data linage values as the user navigates and analyzes the data origin by following the reverse order of the transformation steps. Step-wise fashion may not always give exact data lineage values (i.e. a tracing query or procedure may not invert a transformation perfectly), but being able to analyze the lineage in this manner provides a deeper understanding of the data origin. Step-wise fashion linage tracing may invoke tracing queries, or procedures (refer to our discussion in chapter 2 section 2.2.2 and in this chapter section 5.2.3).

Approximate Answers. Analyzing and validating the data origin may not always require a step-wise data lineage tracing and may not require the exact data lineage values. A good estimate or approximate data lineage values may be enough. While *step-wise navigation data lineage* analysis provides a deeper understanding of the data origin, *approximate answer* may be preferable in some data lineage analysis requirement because it provides immediate answer to question about the data origin.

The component used in this reporting is the reconstruction of the origin using a limited amount of information (refer to section 5.1.2 in this chapter). This may use statistical information and may invoke inverse functions to calculate the *approximate answers* (refer to our discussion in chapter 2 section 2.2.2 concerning *statistical* and *inverse* method of lineage tracing).

Chapter 6

Conclusion and Future Work

Data lineage is a broad and complex problem surrounding data integration implementations. It is still complex even within the confinement of the data warehouse environment. Collecting different approaches does not guarantee a complete data lineage solution because the solution requirement varies widely both in technical (physical and procedural design) and business enterprise implementation.

One common notion in the industry is that a data lineage is an answer to the question, "Where did this data comes from". The question though is too simplistic for an intricate answer. The academe has delved more deeply into this question, attempting to provide answers that complete the "where how and why" questions of data lineage in the data warehouse. However, the best solution introduced in theory, may not be feasible for real data warehouse implementation. A number of considerations are essential to implement a successful lineage solution. This work attempts to present these considerations in relation to our proposed solution.

The conceptual framework for data lineage systems components described in this thesis can provide a basis for further research. The focus may be more on how to implement the existing theory and integrate it with the data warehouse solutions. A

coarse-grained or schema level approach is the most common approach implemented in industry today, and this solution have already been successfully implemented by some of the prominent solution providers in the industry. However, a fine-grained approach is still an area for improvement. Different data lineage related studies attempt to solve this problem by providing data lineage tracing algorithms.

Data lineage problem is a challenge that needs more than just lineage tracing mechanisms; it requires initiative at a strategic level. While other solutions detail on addressing specific problems, this thesis provides a holistic view of the different aspects of the data lineage problem and provides a framework that helps organize, plan, manage and set the data lineage solution in context. First, we identify the different aspects of data lineage problem, classify the different items that are needed for data lineage solution and describe the main tracing mechanisms that form part of our data lineage solution. Then we build a conceptual framework for data lineage system which provide a basis for designing and implementing data lineage solution and seamlessly integrating this solution to a data warehousing system.

Bibliography

[1] SAP AG, *Data validation and error handling*, www.sap.com (2004).

[2] P. Bernstein and T. Bergstraesser, *Metadata support for data transformations using microsoft repository*, IEEE Data Engineering Bulletin (March 1999).

[3] R. Bose, *A conceptual framework for composing and managing scientific data lineage*, 14th International Conference on Scientific and Statistical Database Management (SSDBM'02) (Edinburgh, Scotland), July 2002, pp. 15–19.

[4] S. Bressan, T. Lee, , and S. Madnick., *Source attibution for querying against semi-structured documents*, Proceedings of the Workshop on Web Information and Data Management (Washington, DC, United States), November 1998, pp. 33–39.

[5] P. Buneman, S. Khanna, and W. Tan, *On propagation of deletions and annotations through views*, Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (Wisconsin, USA), June 2002, pp. 150–158.

[6] J. Caserta and R. Kimball, *The data warehouse etl toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data*, Wiley Publishing, Inc., USA, January 2003.

[7] Y. Cui, *Lineage tracing in data warehouses*, Ph.D. thesis, Stanford University, USA, December 2001.

[8] Y. Cui and J. Widom, *Lineage tracing for general data warehouse transformations*, The International Journal on Very Large Data Bases **12** (2003), 41–58.

[9] ———, *Practical lineage tracing in data warehouses*, Proceedings of the 16th International Conference on Data Engineering Page: 367 (Washington DC, USA), February 2000, p. 367.

[10] ———, *Storing auxiliary data for efficient maintenance and lineage tracing of complex views*, 2nd International Workshop on Design and Management of Data Warehouses (DMDW'00) (Stockholm, Sweden), June 2002.

[11] ———, *Lineage tracing in a data warehousing system*, Proceedings of the 16th International Conference on Data Engineering (San Diego, California), March 2000, pp. 683–684.

[12] Y. Cui, J. Widom, and J. Wiener, *Tracing the lineage of view data in warehousing environment*, ACM Transactions on Database Systems **Volume 25 , Issue 2** (June 2000), 179–227.

[13] A. de and A. de Carvalho Moura, *Metadata to support transformations and data and metadata lineage in a warehousing environment*, Data Warehousing and Knowledge Discovery **3181** (September 2004), 249–258.

[14] M. de Wiel, *Oracle warehouse builder 10g: How to implement your enterprise metadata repository for business intelligence solutions*, www.oracle.com (November 2003).

[15] G. Drapers, *Programming microsoft sql server dts 2000 using .net*, msdn.microsoft.com (2002).

[16] P. Eagan and S. Ventura, *Enhancing value of environmental data: Data lineage reporting*, Journal of Environmental Engineering **Volume 119 , Issue 1** (February 1993), 5–16.

[17] W. Eckerson, *Meta data management in the data warehouse environment*, www.ascentialsoftware.com (2002).

[18] R. Elmasri and S. Navathe, *Fundamentals of database systems*, Person Education International, USA, 2003.

[19] C. Faloutsos, H. Jagadish, and N. Sidiropoulos, *Recovering information from summary data*, Proceedings of the Twenty-Third International Conference on Very Large Data Bases (Athens, Greece), August 1997, pp. 36–45.

[20] H. Fan, *Incremental view maintenance and data lineage tracing in heterogeneous database environments*, Nineteenth British National Conference on Databases (University of Sheffield), July 2002.

[21] H. Fan and A. Poulovassilis, *Tracing data lineage using schema transformation pathways*, Knowledge Transformation for the Semantic Web (July 2003), 6479.

[22] ———, *Using automed metadata in data warehousing environments*, Proceedings of the 6th ACM international workshop on Data warehousing and OLAP (Louisiana, USA), November 2003, pp. 86–93.

[23] R. Fileto, C. Medeiros, L. Liu, C. Pu, and E. Assad, *Using domain ontologies to help track data provenance*, XVIII Simpsio Brasileiro de Banco de Dados - Manaus (Brazil), 2003.

[24] P. Kamalapur G. Wadhwa, *Customized metadata solution for a data warehouse - a success story*, www.wipro.com (2003).

[25] N. Hachem, K. Qiu, M. Gennert, and M. Ward, *Managing derived data in the gaea scientific dbms*, Proceedings of the 19th International Conference on Very Large Data Bases, August 1993, pp. 1–12.

[26] C. Imhoff, N. Galemmo, and J.G. Geiger, *Mastering data warehouse design: Relational and dimensional techniques*, Wiley Publishing, Inc., USA, August 2004.

[27] B. Inmon, *Information management: World-class business intelligence*, www.dmreview.com (February 2005).

[28] W. Inmon, K. McDonald, A. Wilmsmeier, and D. Dixon, *Mastering the sap business information warehouse*, Wiley Publishing, Inc., USA, 2002.

[29] IntelligentEAI, *Omg and metadata coalition merge*, www.intelligenteai.com/news (December 2000).

[30] T. Johnson and M. Chaffin, *Best practices for using dts for business intelligence solutions*, msdn.microsoft.com (June 2004).

[31] R. Kimball, *Data quality indicators*, www.intelligententerprise.com (April 2000).

[32] ———, *Is your data correct? simple statistical techniques can help you ensure that users have accurate information at their fingertips*, www.intelligententerprise.com (December 2000).

[33] D. Larsen, *Data transformation services (dts)*, msdn.microsoft.com (September 2000).

[34] J. Leung, *Oracle data warehousing guided tour*, www.oracle.com (2001).

[35] D. Marco, *Building and managing the meta data repository*, Wiley Publishing, Inc., USA, August 2000.

[36] MDC, *Meta data coalition, (mdc) open information model*, www.omg.org/docs/ad/99-04-06.pdf, April 1999.

[37] Microsoft, *Microsoft sql server 7.0 data warehousing framework*, msdn.microsoft.com (1998).

[38] ———, *Recording data lineage in dts*, msdn.microsoft.com (2005).

[39] A. Neroda, C. Baragoin, and J. Ellis and, *Db2 cube views: Getting started with ascential metastage*, www.ibm.com (September 2003).

[40] OMG, *Common warehouse metamodel, (cwm) specification*, www.omg.org/technology/cwm, March 2003.

[41] ———, *Competing data warehousing standards to merge in the omg*, www.omg.org/news (September 2000).

[42] Oracle, *Oracle warehouse builder installation and configuration guide*, www.oracle.com (April 2004).

[43] ———, *Oracle warehouse builder 10g, architectural white paper*, www.oracle.com (February 2004).

[44] ———, *Oracle warehouse builder 10g reviewers guide*, www.oracle.com (May 2004).

[45] S. Patnaik, M. Meier, B. Henderson, J. Hickman, and B. Panda, *Improving the performance of lineage tracing in data warehouse*, Proceedings of the 1999 ACM symposium on Applied computing (Texas, United States), February 1999, pp. 210–215.

[46] D. Riehle, *Drawing a line: Lineage collects details along data's perfect path*, www.teradata.com (June 2004).

[47] M. Rittman, *An introduction to oracle warehouse builder 10g*, www.dbazine.com.

[48] Ascential Software, *Metastage overview*, Ascential Software Corporation (2003).

[49] Teradata, *Teradata warehouse builder reference*, www.teradataforum.com (April 2004).

[50] _____, *Introduction to teradata warehouse*, www.teradataforum.com (November 2003).

[51] G. Variar, *The origin of data*, www.intelligententerprise.com (February 2002).

[52] Y. Velegrakis, R. Miller, and J. Mylopoulos, *Representing and querying data transformations*, Proceedings of the Twenty-first International Conference on Data Engineering (Tokyo, Japan), April 2005.

[53] B. Westman and N . Rochnik, *Oracle9i warehouse builder, integrated data quality*, www.oracle.com (July 2003).

[54] J. Widom, *Research problems in data warehousing*, Proceedings of the 4th international conference on Information and knowledge management (Maryland, United States), December 1995, pp. 25–30.

[55] A. Woodruff, *Data lineage and information density in database visualization*, Ph.D. thesis, California State University, USA, Fall 1989.

[56] A. Woodruff and M. Stonebraker, *Supporting fine-grained data lineage in a database visualization environment*, Proceedings of the Thirteenth International Conference on Data Engineering (Birmingham, UK), April 1997, pp. 91–102.