

Data warehouse with XML Technology

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

By

Supriya Dwivedi

December 2012

© 2012

Supriya Dwivedi

ALL RIGHTS RESERVED

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

Andrzej Bieszczad 12/20/12

Advisor: Dr Andrzej Bieszczad

Date

Peter D. Smith

12/20/12

Dr Peter Smith

Date

Richard Wasniowski

12/20/12

Dr Richard Wasniowski

Date

APPROVED FOR THE UNIVERSITY

Gary A. Berg

12-20-12

Dr Gary A. Berg

Date

Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Data warehouse with XML Technology
Title of Item

Data warehouse, XML, Business Intelligence
3 to 5 keywords or phrases to describe the item

SUPRIYA DWIVEDI
Author(s) Name (Print)

Supriya 01/07/2013
Author(s) Signature Date

Data warehouse with XML Technology

By

Supriya Dwivedi

Computer Science Program
California State University Channel Islands

Abstract

Data warehousing is not a trivial task, when dealing with vast amounts of distributed and heterogeneous data. Traditional Data warehouses are not well equipped to deal with the heterogeneous data. To meet the increasing business demands and to overcome the challenges faced by the traditional Data warehousing, the Extraction Transformation and Loading (ETL) technology needs to be extended. We exploit XML as a pivot language in order to unify, model and store heterogeneous data. We will describe an architecture for an XML based Data warehouse that is capable of integrating heterogeneous data into a unified repository.

In this thesis, we will show how XML technologies can be effective in improving the Data warehousing process. The basic idea behind this thesis is to associate XML with Data warehousing. We will focus on the integration process and describe the architecture to integrate heterogeneous data into an XML Data warehouse.

Later in this thesis, we will implement an XML Data warehouse and then validate its performance against traditional Data warehouse in terms of data load time, disk space utilization, data retrieval time and speed of operation.

Acknowledgement

I would like to express my greatest regards and gratitude to my advisor Dr. Andrzej Bieszczad for his continuous support, patience, motivation, enthusiasm, and immense knowledge. His guidance helped me all through the course of my research, from the selection of the topic to the entire write up. I could not have imagined having a better advisor and mentor for my research study.

Besides my advisor, I would like to thank Dr. Peter Smith and Dr. Richard Wasniowski for their encouragement and insightful feedback.

Also, I thank my friends both at Computer Science department at CSU Channel Islands and outside, and the family members for their invaluable feedback and assistance, especially during the course of implementation of this thesis.

Table of Content

CHAPTER 1 : Introduction	9
1.1 XML Data warehousing	9
1.2 Objective of the Thesis.....	11
CHAPTER 2 : Architecture and Components of XML Data warehouse.....	13
2.1 Comparison of the XML model and the relational model	13
2.2 How can XML Improve Data warehousing.....	14
2.2.1 Source Data Integration	15
2.2.2 Native XML Data Storage	15
2.2.3 Front-end Information Delivery	15
2.2.4 Efficiency	16
2.2.5 Maintenance	16
2.2.6 Scalability	16
2.2.7 Interoperability	17
2.3 XML based Integration Architecture.....	17
2.4 Architecture Components.....	18
2.4.1 Data Sources	18
2.4.2 ETL (Extraction, Transformation and Loading) Services:	19
2.4.3 XML Data warehouse.....	20
2.4.4 Event Triggering Engine	21
2.4.5 Event Log.....	21
2.4.6 Metadata.....	22
2.5 Active ETL Dataflow	23
CHAPTER 3 : Steps to Build XML Data warehouses	25
3.1 Methods for cleaning and transforming data.....	25
3.2 Selecting significant data and multiple level of summarization	26

3.3 Building necessary dimensions depending on multiple level summarization	27
3.4 Building middleware XML documents	27
3.5 Building facts.....	28
CHAPTER 4 : Implementation	30
4.1 Introduction to Power Center Client.....	30
4.1.1 Repository Manager.....	30
4.1.2 Designer	31
4.1.3 Workflow Manager	33
4.1.4 Workflow Monitor	33
4.2 Extraction	34
4.3 Importing XML definition	35
4.4 Implementation	39
4.5 Performance comparison.....	45
4.5.1 Loading data in Data warehouse	45
4.5.2 Front-end Information Delivery.....	47
4.5.3 Disk space utilization.....	48
4.5.4 Data Retrieval in Reports	48
4.5.5 Speed of operations.....	52
CHAPTER 5 : Conclusion and Future Work	55
5.1 Conclusion.....	55
5.2 Future Work.....	55
References	57

Table of Figures

1. Figure 1.1: Data-centric XML document	10
2. Figure 1.2: Document-centric XML document	11
3. Figure 2.1: Architecture of XML Data warehouse	18
4. Figure 2.2: Heterogeneous data integration workflow	24
5. Figure 3.1: Building dimensions depending on multiple levels summarization	27
6. Figure 3.2: Building middleware or intermediate XML documents	28
7. Figure 3.3: Linking fact and dimension XML documents	29
8. Figure 4.1: Overview of Power Center client tools	31
9. Figure 4.2: Designer	33
10. Figure 4.3: Data Flow diagram from source to target	34
11. Figure 4.4: Sample Customer XML file with multiple-occurring elements	36
12. Figure 4.5: Customer XML source definition	37
13. Figure 4.6: Sample DTD file	38
14. Figure 4.7: Sample DTD Source Definition	38
15. Figure 4.8: Sample XML Schema	39
16. Figure 4.9: XML Schema Definition (XSD) of the SalesTerritory Table	42
17. Figure 4.10: ER diagram of Sales department	45
18. Figure 4.11: ER diagram of HR department	46
19. Figure 4.12: Comparative graph for Load time	47
20. Figure 4.13: Graph for time taken to retrieve data	49
21. Figure 4.14: Sample Traditional Data warehouse table	50
22. Figure 4.15: Sample Customer XML Document	51
23. Figure 4.16: Execution time comparison for Insert Statement	52
24. Figure 4.17: Execution time comparison for Select Statement	53
25. Figure 4.18: Execution time comparison for Delete Statement	53
26. Figure 4.19: Relative Execution time graph	54

CHAPTER 1

Introduction

Recently, there has been a lot of interest in Analytics which can be accomplished with the help of Data warehousing. Data warehouse stores huge amounts of historical data used by Decision Support Systems. According to Bill Inmon; “Data warehouse is Subject Oriented, Integrated, Time Variant, Non-Volatile collection of data in support of management decision making process” [4]. Many companies use Data warehousing to help their business make fast decisions and manage risk better.

Today, large amounts of corporate data are on the web. These corporate data are present in different formats (for example, relational database, text, flat files, multimedia, html, etc.). These types of data are known as complex or heterogeneous data. According to Tseng and Chou; “Only 20% of corporate information system data are transactional, i.e., numeric, this numeric data can be easily processed because multidimensional analysis is robust and it is a mastered technique on numeric centric Data warehouses” [6]. The remaining data is semi-structured or unstructured that stay out of reach of On-Line Analytical Processing (OLAP) technology, because of the lack of tools to integrate and analyze the complex data. With the limitations of traditional ETL (Extraction, Transformation, and Loading) technology prevalent in current Data warehouses, it is not possible to augment the analysis with complex data. To meet the increasing business demands and to overcome the challenges faced by the traditional data warehousing, the ETL technology needs to be extended. Moreover, traditional Data warehouse are limited in their ability to provide the real time data. There is a time gap between source refresh and target refresh.

Recently, XML technologies have provided ample support for sharing, storing, spreading and working with the semi-structured data. XML Data warehouse can integrate complex data, provide near real time data, support computerized decision support system, improve data coming from heterogeneous sources, and provide online analysis of complex data. XML Data warehouse will be capable of providing the most refreshed data every single time the user requests for it thus helping to increase business competitiveness.

1.1 XML Data warehousing

In order to store heterogeneous data sources into a unified repository, an XML based Data warehouse model is required. XML is used for managing, modeling or representing facts and

dimensions. Facts are the metrics to be analyzed (e.g., sales) and dimensions are those attributes that describe facts (e.g., product, time, region or salesperson). The advantage of using XML is that it seamlessly integrates data and structure into the same document and also it does not require any predefined schema. Another feature of XML is its simplicity and flexibility.

XML documents are of two types depending on its content: Document-centric XML document and Data-centric XML document. Data centric XML document are fairly regular structure document, as shown in Figure 1.1. These XML document contain structured data such as the textual representation of relational data, for example financial data, invoice data, sales data etc. These XML document can originate from relational databases.

```
<order id='X123'>
  <customer account="10">
    <name>Thomas Edison</name>
    <telephone>123456123</telephone>
    ...
  </customer>
  <items>
    <item id='CD1'>
      <name>cd player</name>
      <unitprice>60.6</unitprice>
      <quantity>1</quantity>
    </item>
    ...
  </items>
</order>
```

Figure 1.1: Data-centric XML document

Document-centric XML documents are semi-structured or unstructured document and contain large amount textual data, as shown in Figure 1.2. These documents are normally written in XML or in another format, such as RTF (Rich Text Format), PDF, or SGML (Standard Generalized Markup Language), and then converted to XML format.

```
<business.newspaper date= 'Dec.1,1998' >
<economy>
<article>
<headline>Financial Crisis Hits Southeast Asian
Market</headline>
<paragraph>
The financial crisis in Southeast Asian countries,
has mainly affected companies in the food
market sector. Particularly, Chicken SPC Inc. has
reduced total exports to $1.3 million during this
half of the year from $10.1 million in 1997.
</paragraph>
<paragraph> ...
</article> ...
</economy> ...
</business.newspaper>
```

Figure 1.2: Document-centric XML document

In XML Data warehouse, we can use web services to solve the problem of transferring heterogeneous data among various systems. When web services are embedded into the XML documents, it is known as Active XML document (AXML). An Active XML document contains some data available explicitly while the other data is called through the web services. When these web services are triggered, it will update the document with the fresh data.

1.2 Objective of the Thesis

The main objective of this thesis is to associate XML with Data warehousing. Several heterogeneous data sources can be easily converted to the XML format as it is a neutral format. Other features of XML are its simplicity and flexibility which makes it useful for every application that requires translation and exchange. In this thesis, we will prove that XML is in right place inside the architecture of Data warehouse to integrate different data sources. In this thesis first, we will propose conceptual architecture of the XML Data warehouse to integrate heterogeneous data sources. This architecture will demonstrate the use of web services in XML Data warehouse.

Later, we will implement an XML Data warehouse. We will validate the performance of our XML Data warehouse with the traditional Data warehouses. In practical implementation, we will use all the concepts of XML Data warehouse which we have defined earlier except the use of web service as it will be beyond the scope of this thesis.

In chapter two, we will initially draw comparisons between XML and the relational model, and then discuss how XML technology can improve the Data warehousing process. Later, we will focus on the architecture of XML Data warehouse using web services. We will define the use and functionality of all the components of XML Data warehouse.

In chapter three, we will propose a logical approach to load data from multiple heterogeneous sources into an XML Data warehouse, so that queries can be optimized. Furthermore, a procedure of building XML Data warehouse from an XML document is demonstrated by creating required fact and dimension tables.

In chapter four, first we will discuss the tools used to implement an XML Data warehouse. Later in this chapter, we will show the practical implementation of the XML Data warehouse using sample data from heterogeneous data sources. We will integrate data from disparate sources into XML Data warehouse and then validate the performance of the XML Data warehouse. We will validate the performance of XML Data warehouse against traditional Data warehouse in terms of time taken to load the data in Data warehouse, time taken to retrieve data from data warehouse, disk space utilization and the speed of operation.

CHAPTER 2

Architecture and Components of XML Data warehouse

Web is a vast source of information and is growing at a fast rate. A large amount of corporate information is generated from a variety of sources like emails, HTML files, web data, unstructured text as well as structured databases. This unstructured and semi-structured data are known as complex or heterogeneous data. A lot of research has been done to integrate this heterogeneous data into a unified repository. However, the integration process involves identification, querying and merging of data from heterogeneous sources which is difficult.

A major amount of information available on the web today is semi-structured. Semi-structured data has an irregular structure and does not contain a fixed schema. Many researches have been done in the past in developing data models, query languages and systems to manage semi-structured data. Object Exchange Model (OEM) is one such model that was explicitly defined to represent semi-structured data in heterogeneous systems. A modified version of this data model has been used in the development of the Extensible Markup Language (XML) and has kindled a lot of interest in modeling semi-structured data. XML is well suited to model semi-structured data because it makes no restrictions on the tags and relationships used to represent the data. XML also provides advanced features to model constraints on the data, using an XML schema or a Document Type Definition (DTD). However, XML does have some differences with the other semi-structured data models: For example XML is ordered collections while semi-structured data is unordered, and in XML references can be used for unique identifiers for the elements; this is absent in most other data models. Regardless of these differences, XML is a popular data model is most commonly used to represent semi-structured data. So we are using XML database instead of relational database in our Data warehouse. Following are the points which are taken into consideration to decide between XML database and relational database:

2.1 Comparison of the XML model and the relational model

The main differences between relational database and XML database are:

- **Hierarchical relationships**

In relational databases, hierarchies are maintained in the form of logical relations. The only way to define hierarchy is by using more than one table where one table acts as parent and

other as a child. An XML document itself can be hierarchical. It contains information about the relationship of data items to one another as a hierarchy.

- **Self-describing nature**

An XML document is self-describing in nature. We can easily understand XML data just by looking at it as XML document contains tags that describe the data. It may contain different types of data. However, in relational databases the column definitions are used to describe the data content.

- **Flexibility of the data**

Relational databases are fairly rigid. For example, normalizing and de-normalizing the tables is generally difficult once the database is in use. In contrast, XML models are relatively flexible and serve as a better option when database design is changed frequently.

- **When highly structured data is present in small quantity**

Sometimes highly structured data are present in very small quantities. In relational database these data are represented using complex star schemas. This star schema contains many joins between fact and dimension table, with most of the table containing very few rows. This type of data can be better represented using single XML document as XML is hierarchical by nature.

So due to the above discussed reasons, we will integrate all heterogeneous data into XML database instead of relational database. XML Database will improve the performance of our Data warehouse in many ways.

2.2 How can XML Improve Data warehousing

For information systems, XML have the capability to exchange data between browsers and application programs, between application programs and other application programs, etc.

XML can contribute in the warehousing process: data integration, cleansing procedures, data storage, and front-end information delivery. Following are the reasons which show that XML will improve the performance of Data warehouse:

2.2.1 Source Data Integration

Source data may come from different heterogeneous sources and integrating these sources is the most important step. We can have sources which have data created by legacy systems based on mainframe application. Apart from this, we can have multiple web applications providing data for real time processing or data sharing. All this information is used to create the Data warehouse for the company.

If there are systems which generate data in XML, the communication among these will be much easier. The systems which have a common XML schema can exchange information with each other. Although in some old applications XML writers need to be embedded, several recent systems are already equipped to create output data in XML format. Existing relational databases already support query output directly in XML form

2.2.2 Native XML Data Storage

Native XML databases store the core of XML based Data warehouse. They provide the possibility to integrate semi-structured data inside the Relational Data warehouse. This direct XML storage can fulfill the needs of storage support for Data Web-House.

In this category, we can identify Lore, Natix, and Tamino. In Hybrid scenarios, XML data can be saved directly inside the database in relational database tables. Indeed, market DBMS (e.g., DB2, Oracle, Sql Server etc.,) does offer the capability to bulk load data inside a relational table from XML sources. This step is done with the help of XML schema.

2.2.3 Front-end Information Delivery

More and more sophisticated tools are used to deliver information outside the Data warehouse. The extensive utilization of XML will be helpful in eliminating reporting and query tools from end user's terminals. An XML document along with an XSLT transform can generate user friendly reports. The resulting business information could be published with HTML pages in the company's intranet website. XSLT is utilized to change an XML document into one another XML document, or any other kind of document that is understood by a browser, like HTML and XHTML.

2.2.4 Efficiency

XML is a neutral format, which enables communication among different Data warehousing tools. No owner format is needed to link and merge different warehousing tools, which helps to divide the process such that different tools can be used for extraction and transformation purposes. The efficiency of the global process will be higher, as tools have a good performance within different steps. We can chain a high extraction tool with XML generation capability and a high transformation tool with XML input, etc. Since XML is used in all the tools, no extra transformation among different owner tool's format is required.

2.2.5 Maintenance

The business rules of the organizations are never the same, due to changes in the real world. The changes may happen due to regulatory changes, or other changes in business practices or as a result of corporate reorganizations.

Due to the evolution of management rules, many programs need to be modified to enable new decision elements or new formulae for restitution. These rules define the decisional elements, equations, and parameters. For most of the existing tools, to realize this operation, a query has to be formulated into the metadata. Then, the user has to update all the concerning programs. Data warehousing process should be updated automatically to make the maintenance process work efficiently. XML presents the ability to specify the transformations using XSLT (XSL Transformations). If the system is able to generate a new transformation automatically, no extra update or change will be needed to enable the evolution.

2.2.6 Scalability

An important aspect to achieve the scalability is to ensure that changes in the implementation on either the application system side or the data side do not change the transported data format between the two applications. This means that the client application or the server application does not need to be aware of that change.

One way of achieving this is to move the data in the standardized format between these components. Currently, the main format that is used in all the scenarios is XML. The idea is to take the data in the data system or in the application system, move it into the standardized format of XML, both on the wire as well as using standardized grammar, which we can express in XML, to warp the data and transport it between the different systems.

2.2.7 Interoperability

These days organizations are using web for interaction, blending partners, suppliers, and customers to form a virtual enterprise that functions as the superset of the physical organizations. This e-business is performed through multiple real-time information exchange technologies such as an electronic document interchange (EDI), business-to-business exchange, and e-business server application.

Establishing interoperability between these different sources is needed to integrate data inside Data warehouse. XML can play a major role in the integration of web sources inside the business Data warehouse, from corporate internal information portals, parts databases, product catalogs, and business-to-business document exchange, etc.

2.3 XML based Integration Architecture

In this section we will show an architecture which will integrate the data coming from heterogeneous systems and then getting loaded into a uniform warehouse. ETL services are used to integrate these heterogeneous data into a unified XML repository. This architecture is based on components like metadata, event log, and event triggering engine. The architecture has a common language as XML for all the components. We can categorize XML documents created in this process in two ways. Fact and dimension are created in XML or we can say that warehouse data will be in XML format. All the metadata, rules, logs are also created in XML which can be the second type of XML used in the architecture. Thus XML Data warehouse architecture can look like as below:

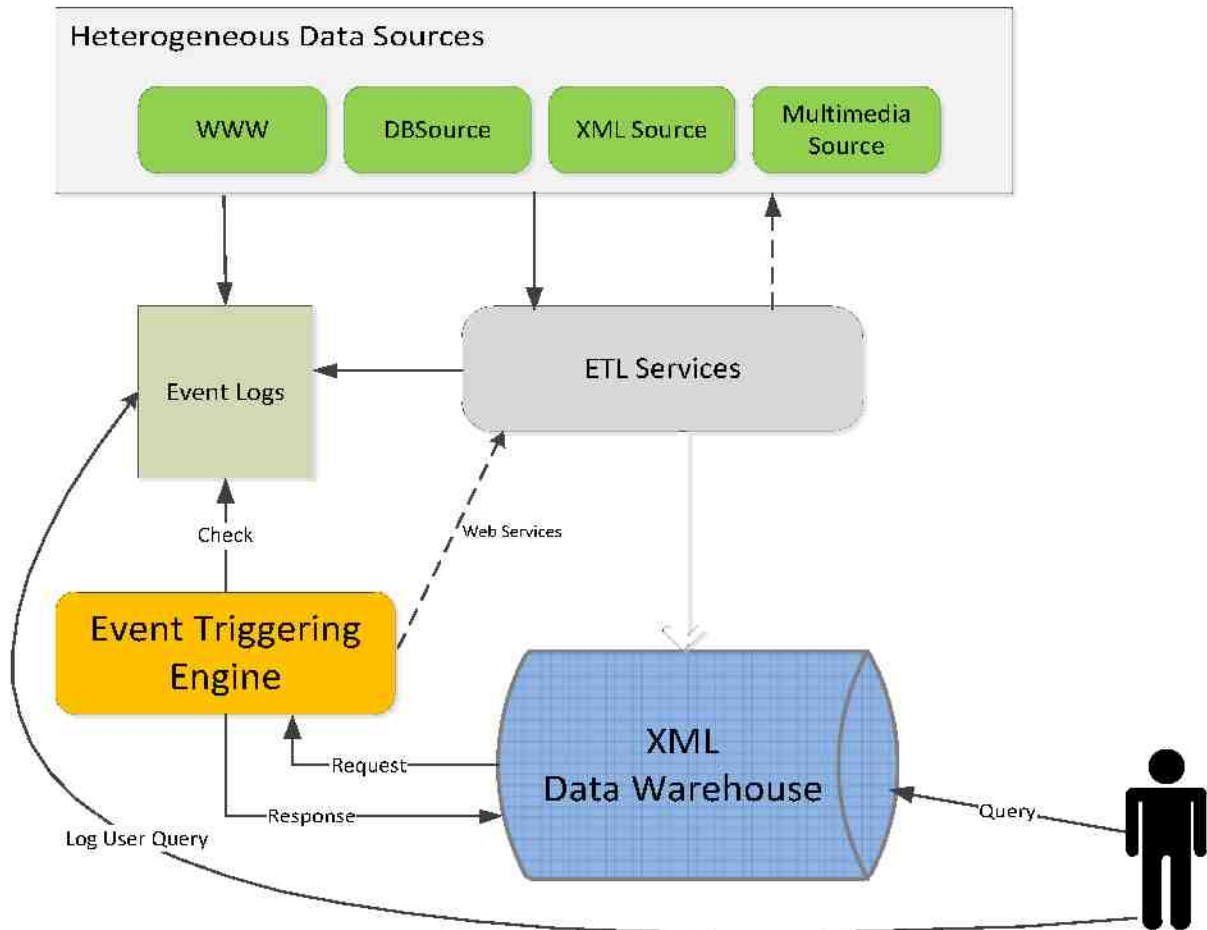


Figure 2.1: Architecture of XML Data warehouse

2.4 Architecture Components

Following are the components of XML Data warehouse:

2.4.1 Data Sources

Traditional Data warehouses integrate data from transactional and structured data sources. One objective of our architecture is to integrate heterogeneous data which can be translated into XML format automatically. It's worth being noted that input schema can be updated at any time either by adding, altering or dropping data sources. These input schemas describe complex data and the relevant entities, elements and/or attributes for each data source.

Technically, when a developer accesses a database via the web-based GUI, he can browse through the database structure and select interesting tables. He can select or filter out records based on graphs or he may query the tables. The similar method is followed for other data source systems. For an XML data source too, the developers can go through the element hierarchy and then prefer specific paths.

2.4.2 ETL (Extraction, Transformation and Loading) Services:

ETL processes are used for fetching data from source systems, applying business rules to transform data, changing it to XML format and loading it into XML Data warehouse. This process fetches data from different sources via multiple interfaces. The ETL process not only performs integration of low level characteristics like image color, image texture, image shape, file location, file size, update date, creation date, etc., but also integrates other characteristics of complex data like relationship between entities, image content, etc. Integrating these characteristics from complex data, however require semantic data extraction and mining strategies.

We have used web services on developing ETL tasks, which results in multiple advantages. Web Services can resolve issues related to interoperability and distribution of data. Web services can also be embedded into XML documents, which transform them into an Active XML (AXML) document. These AXML documents can then be refreshed by triggering these web services.

As we deal with heterogeneous source systems, the data can be in different structure or even with similar systems also the structure may differ. Usually we face challenges in accessing sources or converting them into single format and structure. We also face issues in integrating those sources and schemas. And at the end we need verify data quality and accuracy. We need to make sure data is consistence and complete.

There are also other challenges related to security, gaining access to web services, licensing and developing integration services. To overcome these issues, we can develop several source handler services, each of which deals with a specific type of data source.

Transformation and cleaning tasks face some conflicts related to both structure and data levels as listed below:

Structure level conflicts:

- We might get issues for Domain of different source systems. Similar Domain from multiple systems may differ in the structure.
- Difference in names for similar objects is also possible. Same elements may carry different names in the source systems.
- Schema model may also have a difference for similar objects.

Data level conflicts can arise when similar data are represented using different ways:

- Data value conflicts where we use different ways to instantiate a certain element through different data sources (e.g., France vs. FR)
- Data unit conflicts where the same element can be instantiated using different measuring units, originating from heterogeneous data sources (e.g., Dollar vs. Euro)

Data representation conflicts where different representations are used for the same element through different sources of complex data (e.g., date format) require accurate mapping and matching between contents and structure of heterogeneous data sources. Moreover, developing transformation services that are based on metadata can avoid such conflicts. Loading transformed data, and writing them into XML documents is not simple. We cannot load the entire data each time. Elements already present in xml should get updated and new elements should get inserted. They should also identify their target document.

There are various ETL tools available where the latest versions have features to extract data from XML or even from a website itself. For example, the upgraded version of Informatica (Power Center) has brought few new transformations. These transformations are specially used for extracting XML data or using an http website to extract.

2.4.3 XML Data warehouse

XML Data warehouse is the repository where data are loaded through ETL services. The outputs of the ETL process are XML documents. Some of the important questions taken into consideration are which data should be given explicitly in XML document? Which data should be given implicitly? The explicit parts of Active XML documents are similar to the starting and ending tags in XML documents. But the dynamic parts are shown as a method to call web services.

XML schema supports complex data types and is considered as a grammar for the warehoused Active XML documents. AXML documents induce multiple benefits. First of all, XML can be used as uniform language and can store complex data in it. Secondly, web services are the best solution to overcome the problem of non-interoperability and distribution of data over distributed and heterogeneous data sources. Thirdly, when querying Active XML documents, their embedded web services are invoked to refresh the document with most updated data. Lastly, Active XML documents can be stored in XML format. On contrary, no browser can read AXML documents yet. Also, the end-user applications do not fully support AXML documents. To parse AXML documents, we propose scanning requested AXML document to find the embedded web services, then invoking these web services, and subsequently enriching the current documents with the result returned from the web services.

2.4.4 Event Triggering Engine

The event triggering engine manages metadata and event logs, to ensure smooth processing of data. It handles multiple processes which can be described as below:

- It verifies that the source systems are available.
- It verifies the data elements and makes sure that sources-to-targets mapping is correct.
- It takes care of event triggering and checks their conditions throughout the process.
- It monitors event logs for all processes and their impacts on XML Data warehouse.
- It has some rules defined and triggers tasks based on conditions.
- It manages XML web services activation.
- When web services are called it refresh the XML documents with most recent data.

The timing of triggering web services is also an important factor. It can be scheduled on weekly, daily, hourly or based on some events. It can also be triggered on demand.

2.4.5 Event Log

The purpose of the event log is to capture information about all events detected throughout the architecture, either by data sources, ETL service modules, or XML Data warehouse. Some events can be triggered based on the information available in event log. Queries used by the users are also important information that is maintained by the event log. Information about the events is logged in the event log into XML format. It mainly stores event type, event description, date, time, status, error message. Thus, the event log not only handles exceptions, but also provides history of integration tasks and describes user's events. This log is checked by the Event Triggering Engine at regular interval. Some examples of events that result in logging

are: changing the list of data sources specified in the input schema, changing attributes of a specific data source, integrating data from their sources (extracting, transforming or loading), and so on.

2.4.6 Metadata

We believe that if the metadata are defined correctly upfront, the automation of various processes in the architecture becomes very easy. Metadata can be of various natures: data source descriptions, ETL services, storage structures of the XML Data warehouse, and data refreshing policies. Hence, metadata can be considered as a grammar for XML documents and as configuration files for data sources and ETL services.

Metadata may contain the following information:

- **Metadata of data sources:** This includes data types, descriptive information such as structure, ownership, interfaces, format, update frequency over a certain period, access methods, refresh time, and so on.
- **Metadata of ETL services:** This includes ETL service interfaces, transformation and cleaning rules given by business, targeted XML documents, data elements mapping names, updating rules, and so on.
- **Metadata of XML Data warehouse:** This includes Data warehouse structure, warehouse size, refreshing plan, physical location of XML documents, indexing, and so on.

Active Rules

In the architecture, rules are created based on different steps like data extraction from complex sources, transformation or XML warehouse data load. We need to make rules to trigger processes based on variety of events. For example any change in data source schema will result in an event to add or alter a source. So these events continuously monitor source system schema.

The architecture can allow some temporal events. Such events are used in our integrating architecture, to detect changes in the structure of data sources. For example, modifications in source system can be checked every hour. Source system availability can be checked every time before the extraction. This can also be scheduled weekly or daily for maintenance purpose.

Weekly or monthly backup of the XML warehouse can be taken. After any insert or update data quality can be checked.

Eventually, there are many rules which can trigger the actions in the architecture. These rules can send a notification to a server for all the changes happening in the source system and trigger the ETL to extract it. They may affect the XML Data warehouse itself by invoking embedded web services, when querying Active XML documents in order to refresh them with up-to-date results.

2.5 Active ETL Dataflow

The ETL Dataflow is bi-directional in nature. In "forward direction," complex data is pulled from heterogeneous sources, converted into the desired formats, and then loaded into the XML Data warehouse through the ETL services. In "backward direction," the XML Data warehouse is refreshed by calling web services. The evaluations of web services are controlled by the Event Triggering Engine, to call and perform particular integration tasks or other third-party Web service.

Integrating semi-structured or unstructured data into an XML Data warehouse involves all the above defined components. The design below (Figure 2.2) shows how we can integrate heterogeneous data sources in the XML Data warehouse.

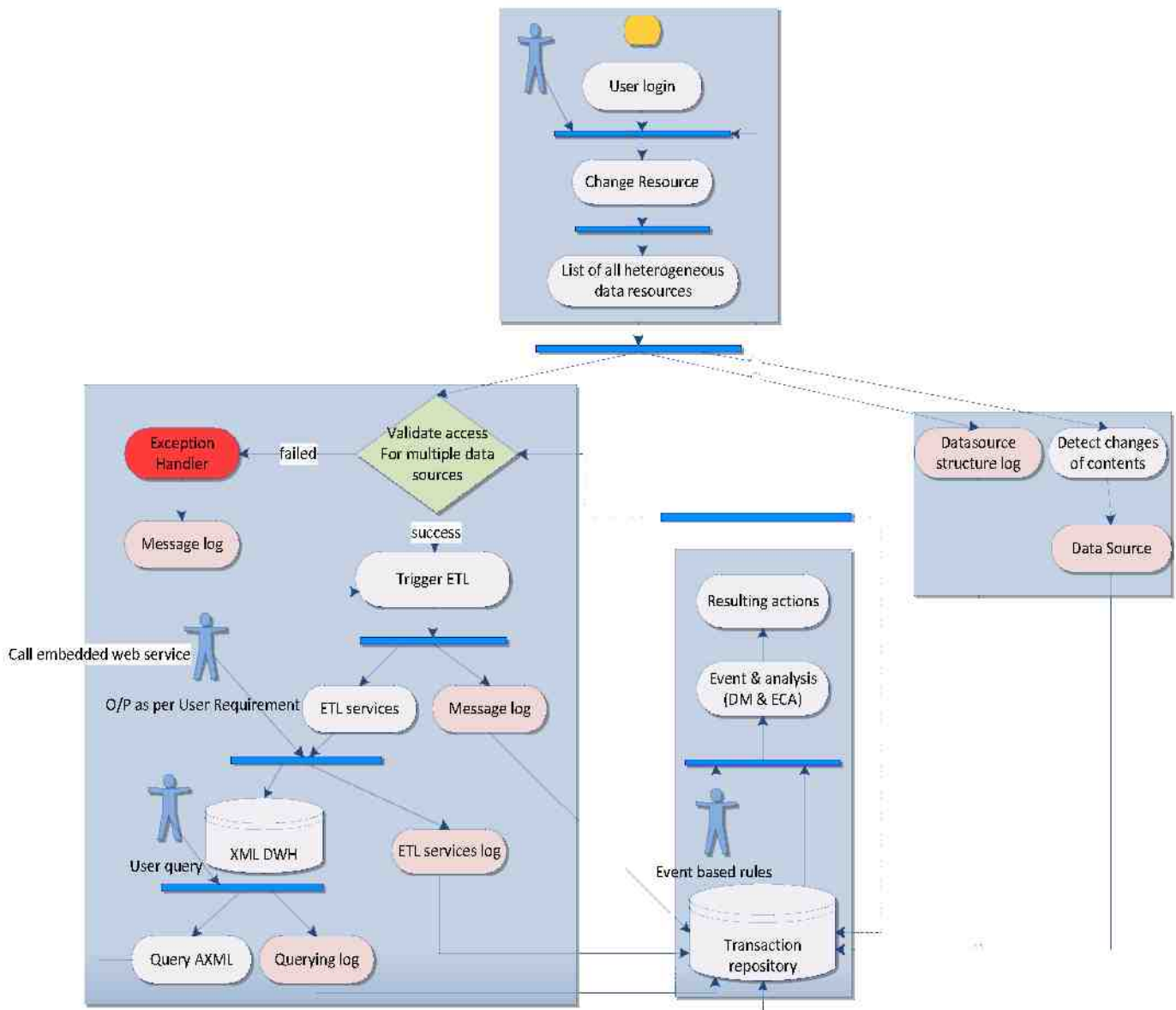


Figure 2.2: Heterogeneous data integration workflow

CHAPTER 3

Steps to Build XML Data warehouses

This chapter we will propose a logical approach to load data from heterogeneous sources into an XML Data warehouse, so that queries can be optimized. Furthermore, a procedure for building XML Data warehouse from an XML document is demonstrated by creating required fact and dimension tables.

3.1 Methods for cleaning and transforming data

The presence of an XML schema is very essential, mainly for evaluating data accuracy. XML schema defines a valid method to specify the content and structure of XML documents that are utilized for analysis. If XML schema is correctly established and all documents are compliant with it, there is a lower probability to have inconsistent data. We have defined some rules for carrying out data cleaning and transformation process.

1. If a XML schema is present, we have to validate the correctness of all schema conditions.

- Validating if accurate usage of the attributes and name of entities in the document: For example, <CUSTOMERTYPE>, <customertype>, <CustomerType> will be considered as three different fields
- Verifying if data type and natural logic are followed. For example, a zip code might have string data type, but it can have only digits, not letters, etc.
- Validate the hierarchy is followed for all elements and attributes.
- In parent-child relationship observe if order indicators are followed. For example, the order indicators can be All, Choice and Sequence. Indicator “All” signifies that there is no order defined for child elements, and they can be included in any sequence. Indicator “Choice” signifies that any one child can be included. “Sequence” means that there is an order defined for child, and they can be included in defined order.
- Define null values. In a database, null values can be defined in two ways: (1) Null value is coming from source (2) Invalid values coming from source is treated as null. For example, characters coming from source for field telephone number will be treated as null.

- In XML schema, occurrences of any element might be restricted. For example, for an element minimum occurrence is two and the maximum is 10. This tells that the element should appear at least two times and not more than 10.
- Verify other schema restrictions also. Other restrictions on elements might be present in the schema. For example, "Status Field" can have only three values: Open, In-progress, and Closed.

2. Removing duplicate data, for e.g., a customer name was entered in the different manner, by multiple order entry systems placed in different locations. Possible ways can be: surname & first name, surname & first name & the father's initial, first name & surname etc.

3. Remove conflicts from entities & attributes values, for e.g., two different blood groups existing for a person. As a person will have only have one blood group, so the correct, one must be captured.

4. Removing erroneous records, occurred during the process of entering data, for e.g., mistyping. Manual cleaning for some of the data may be required because only the domain experts can understand the correctness of data. Moreover, we can also implement some of the rules by understanding the business logic. This will reduce the manual intervention in the cleaning process.

3.2 Selecting significant data and multiple level of summarization

Following steps should be considered for selecting significant data and multiple level of summarization:

- Not all data in the heterogeneous databases is integrated into the XML Data warehouse. We should be very careful in the process of selecting significant data and data summarization level. We must extract only valuable and significant information from all the data sources. The extracted information should also be considerably diversified, so that in the future it is capable enough to handle multiple queries.
- We decide the multiple level of summarization based on many factors, e.g., how much disk-space is available, how frequently a user request for a specific query, or if historical data were necessary.

- Furthermore, when multiple levels of summarization exist, it will considerably increase the performance of the query. In most of the Data warehouse projects, the speed of querying data is more important than the disk-space. So we should consider detailed levels of summarization as it increases the performance of XML Data warehouse.

3.3 Building necessary dimensions depending on multiple level summarization

We have decided the multiple level of summarization for a specific dimension in the previous step. Depending on the levels of summarization for specific dimensions are required in reports, we would create and populate new XML document. The content of new XML will be extracted from heterogeneous data source. For example, we need to display “country” or “region” as a level of summarization to see the forecasted sale in reports. We can find this information directly from the original document by fetching distinct values of “country” and/or “region” element and storing it in a new document.

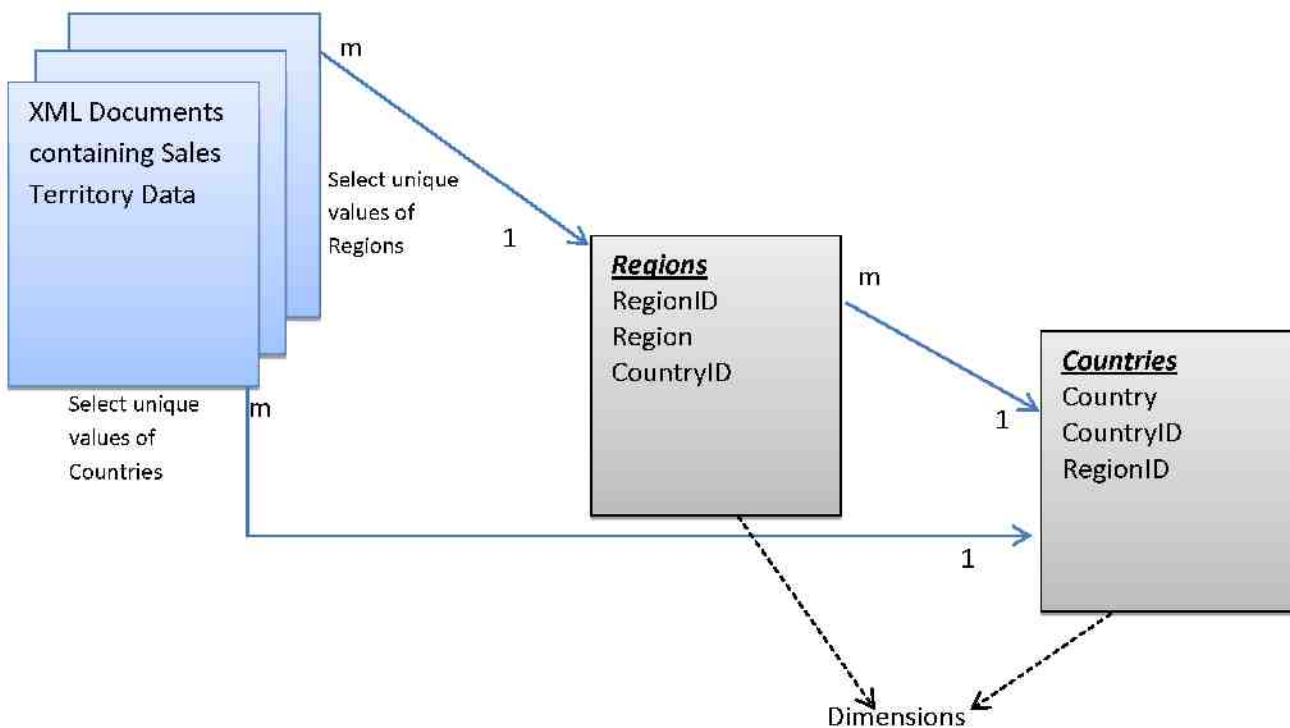


Figure 3.1: Building dimensions depending on multiple levels summarization

3.4 Building middleware XML documents

In the course of building an XML Data warehouse from data residing in heterogeneous sources, building middleware documents are a common method to store useful & significant

information. We will select only significant information and will reject insignificant information. Now which information in the heterogeneous data sources are most significant and should be integrated in the XML Data warehouse is a good question. Researchers have answered it by defining different techniques, e.g., mining and analyzing transactional repository for detecting different user's queries, or detecting shared hierarchies and merging of dependencies etc. However, for general users analyzing the user's queries in each data mart (or domain) is a common way to do it.

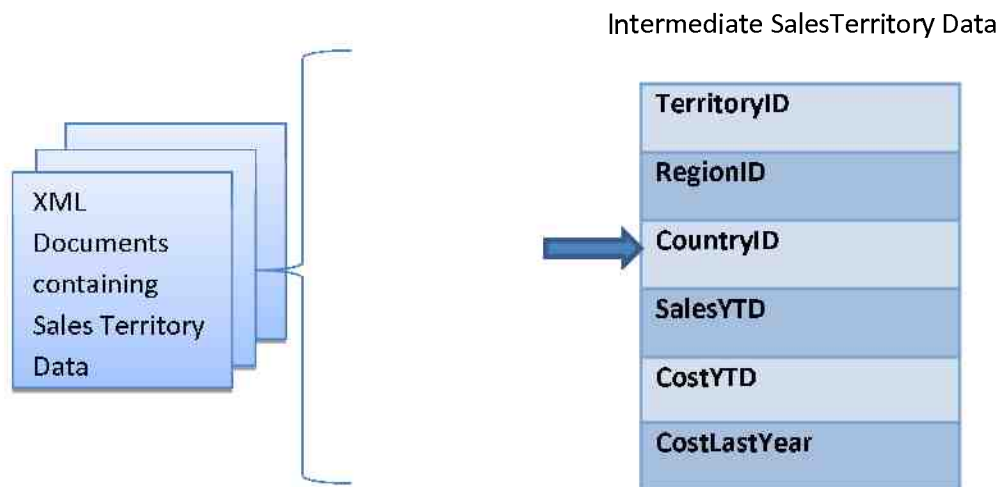


Figure 3.2: Building middleware or intermediate XML documents

Here, we are interested in data representing activity. We will select the data for middleware document based on data used in different queries, calculations, etc. For this, we will be extracting data from original document and storing it into middleware XML document. The facts in the XML Data warehouse are this middleware document, which are connected to the dimensions.

3.5 Building facts

In this step, we will identify the relationships between keys of different middleware documents. We need to make sure that the link between these documents is not lost, and they maintain fact and dimension relationship. These documents should maintain primary-foreign key relationship for facts and dimensions.

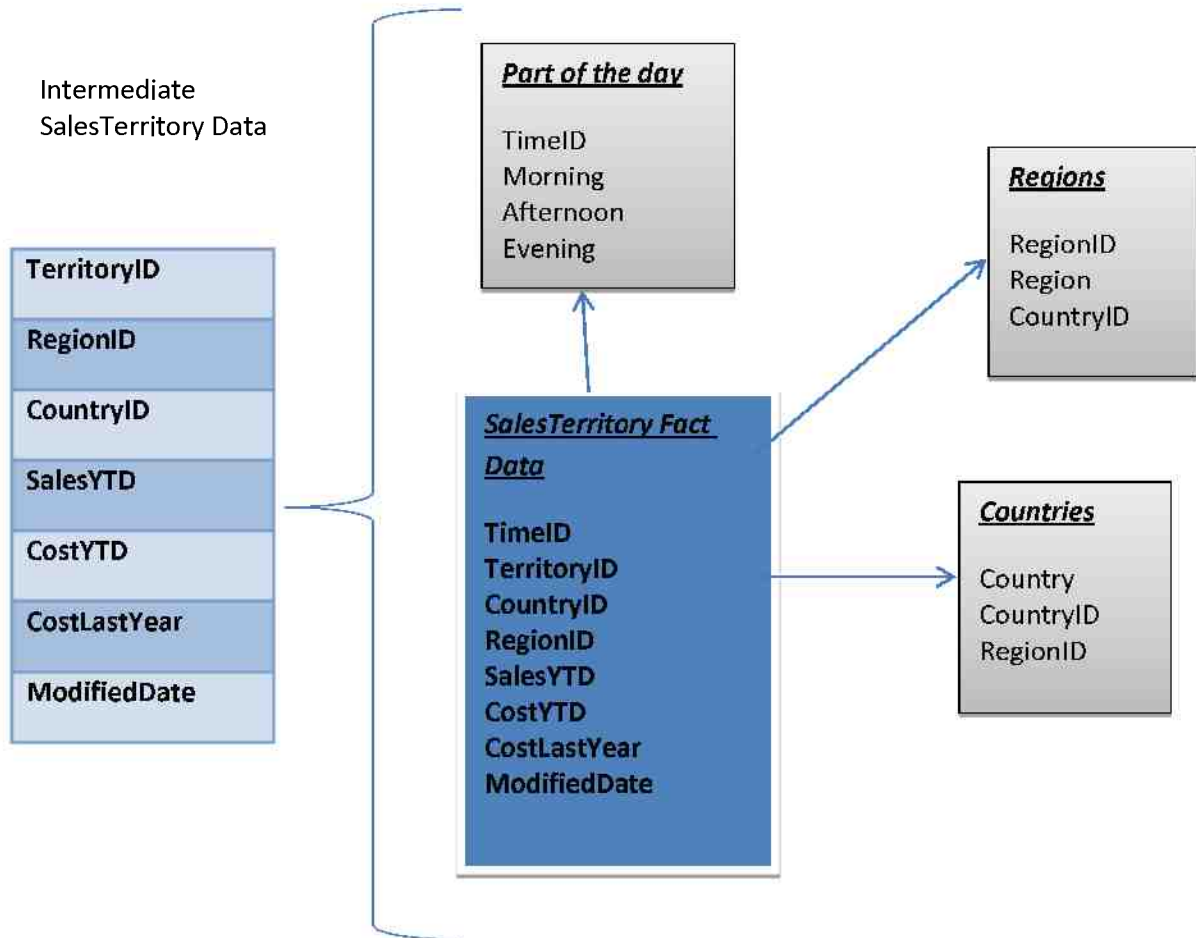


Figure 3.3: Linking fact and dimension XML documents

Following all the above mentioned steps, we can build the schema to load the data into an XML Data warehouse.

CHAPTER 4

Implementation

In this chapter, first I will discuss the tools which are used to implement an XML Data warehouse. I will also demonstrate how we can import or create source and target XML definition.

Later in this chapter, I will integrate the data coming from different sources into an XML Data warehouse. These heterogeneous data will include flat file data, relational database, and XML data. I will integrate all these data into an XML Data warehouse and then validate that it's performance is better than the traditional Data warehouses. I have built an XML Data warehouse using Informatica Power Center and Oracle database. As mentioned earlier, I will not implement the web services part in this XML Data warehouse.

4.1 Introduction to Power Center Client

Informatica Power Center Client tool allows us to specify how to transfer the data from source to target through the transformations. In the backend, the Power Center Server will fetch data from the source and deliver it to the target database based on the instructions given from the Repository Service. To design the Data warehouse we will work with different types of client tools as listed below:

- Repository Manager
- Designer
- Workflow Manager
- Workflow Monitor

Below is the brief description of all the tools and their functionality:

4.1.1 Repository Manager

Repository Manager is a database managed by the Repository Service. It is used to store the metadata created by Informatica Client tools. The Integration Service uses repository objects to extract, transform, and load data based on the instructions given in the repository objects. The repository also stores administrative information such as user names, passwords, permissions, and privileges. In the Repository Manager, we create folders to work in the designer window.

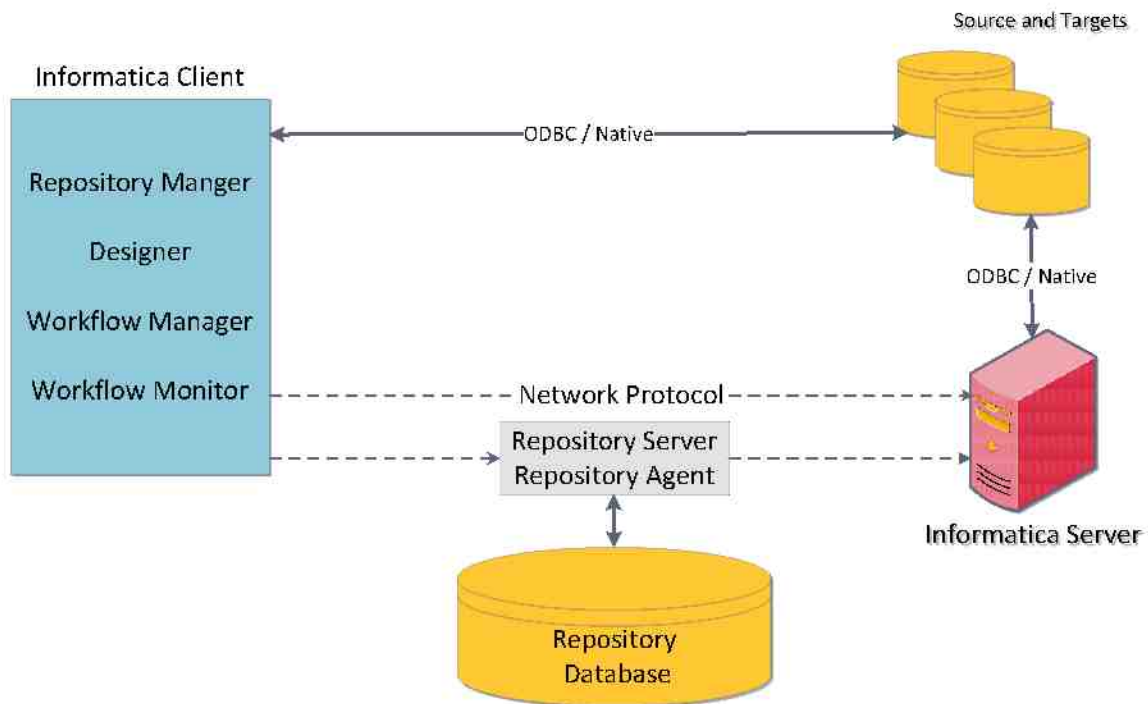


Figure 4.1: Overview of Power Center client tools

4.1.2 Designer

The Designer has sub tools that assist us to create mapplets and mappings so that we can specify how to transform and move data from source to target.

Tools in Designer

There are different types of sub tools available in the designer window to move and transform data from source to target. These tools are listed below:

Source Analyzer allows us to create or import the source definitions from multiple sources like flat file, COBOL files, XML document, application sources, word documents, pdf, and relational databases.

In **Target Designer**, we create or import target table definitions based on the requirements of the user.

Mapping Designer is used to move data from source to target through the transformations. These transformations are applied between the source and the target definitions.

Transformation Developer is used to design individual reusable transformations.

Mapplet Designer is used to design Mapplets, which contains set of transformations and can be reused in the Mappings.

Transformation

Transformations are used for cleaning, and scrubbing data as per the data requirements. There are various types of transformation to manipulate the source data according to the needs of the end user.

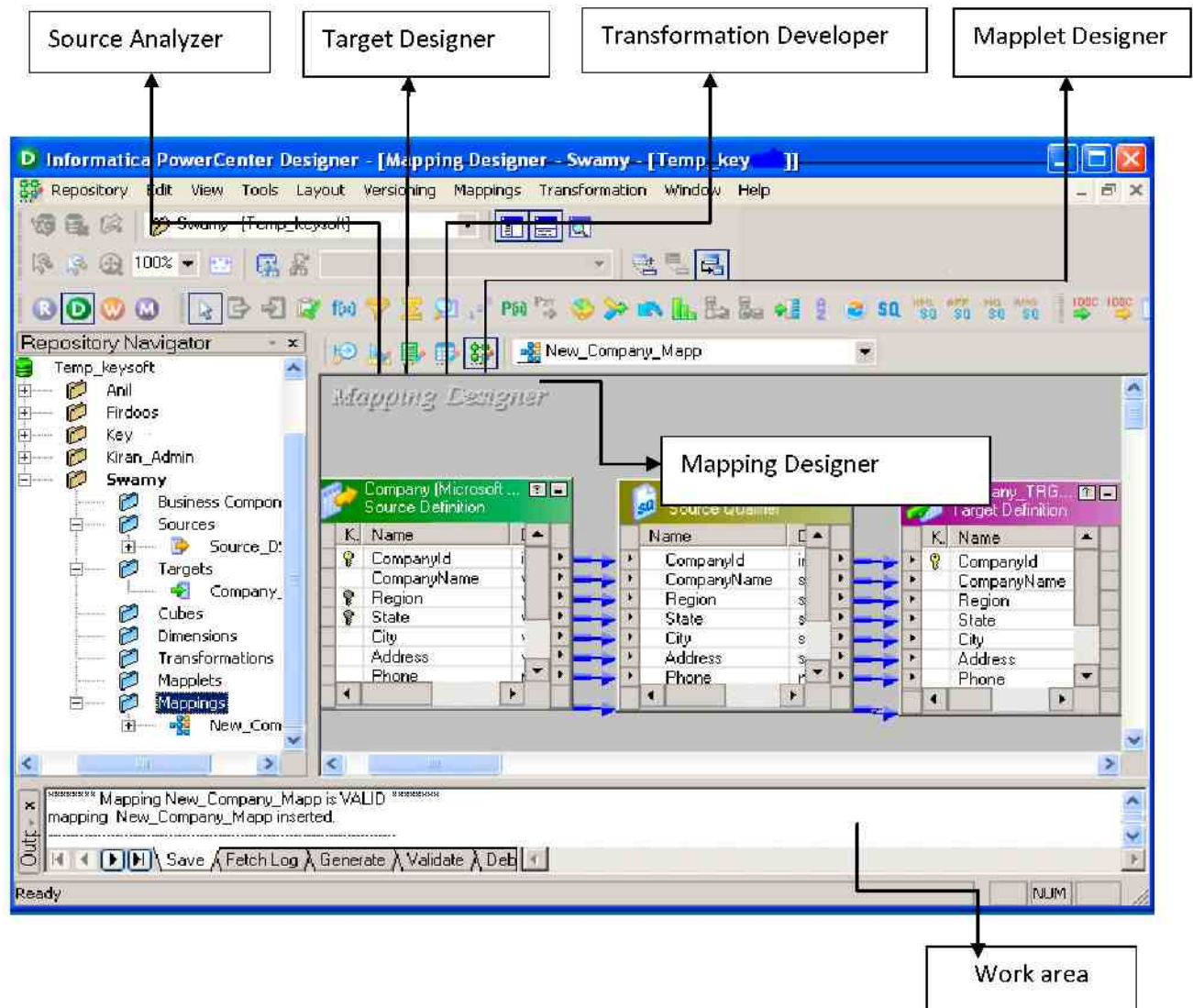


Figure 4.2: Designer

4.1.3 Workflow Manager

A workflow is a set of instruction that instructs the Integration Service to run tasks such as sessions, email notifications, and shell commands. After we create tasks in the Task Developer and Workflow Designer, we can connect the tasks with links to create a workflow.

4.1.4 Workflow Monitor

The Workflow Monitor window monitors all the scheduled and running processes in the Workflow Manager. This monitors the status of the processes like whether the task created in workflow manager has succeeded or failed and other information like the total time taken to complete the task.

Once the workflow has succeeded, Integration service successfully delivers all the desired records from source to target. The diagram below shows the stepwise process to integrate data:

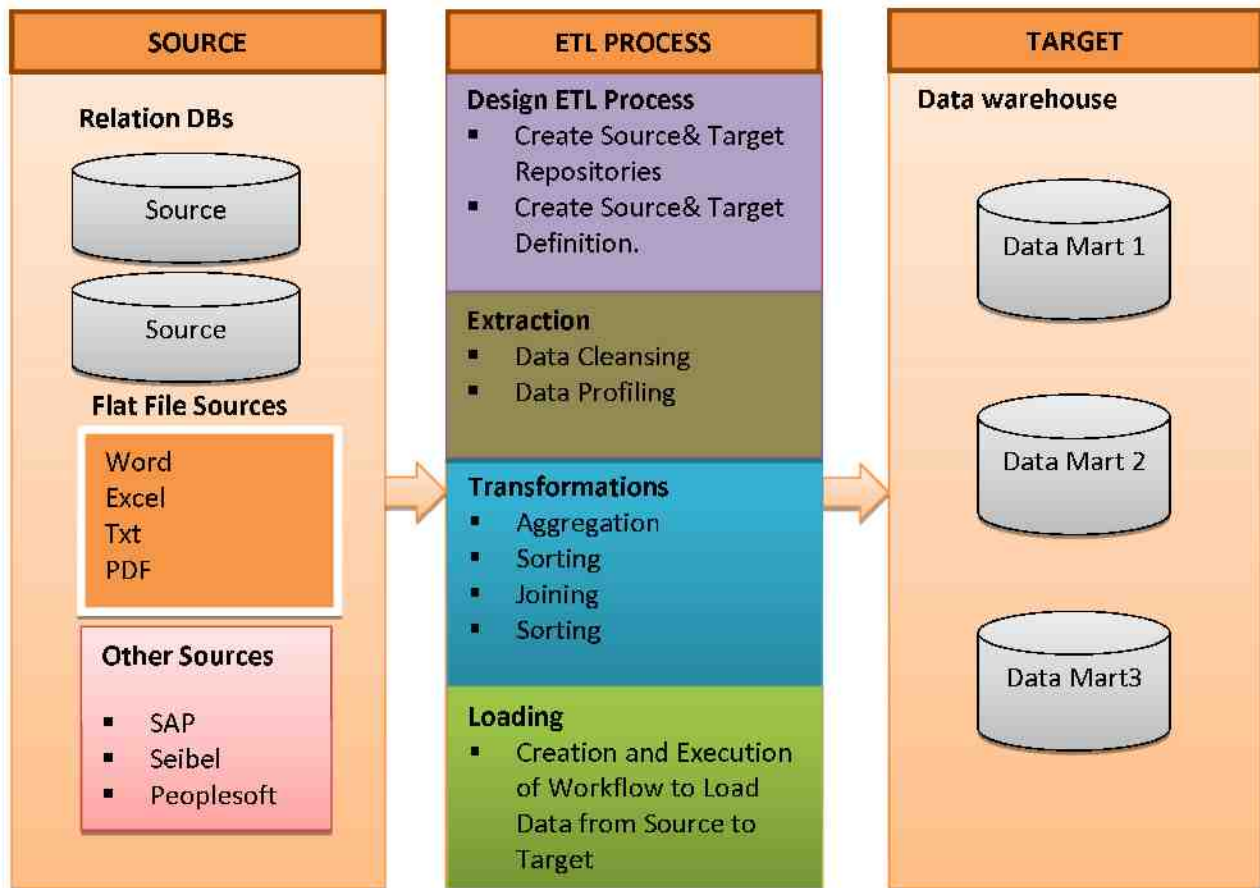


Figure 4.3: Data Flow diagram from source to target

4.2 Extraction

In Informatica when we import an XML definition, the Designer creates a schema in the repository for the definition. The repository schema provides the structure from which we edit and validate the XML definition. We can create metadata from following file types:

- XML files
- DTD files

- XML schema files
- Relational tables
- Flat files (txt, csv, xls files)
- Word document
- Pdf

4.3 Importing XML definition

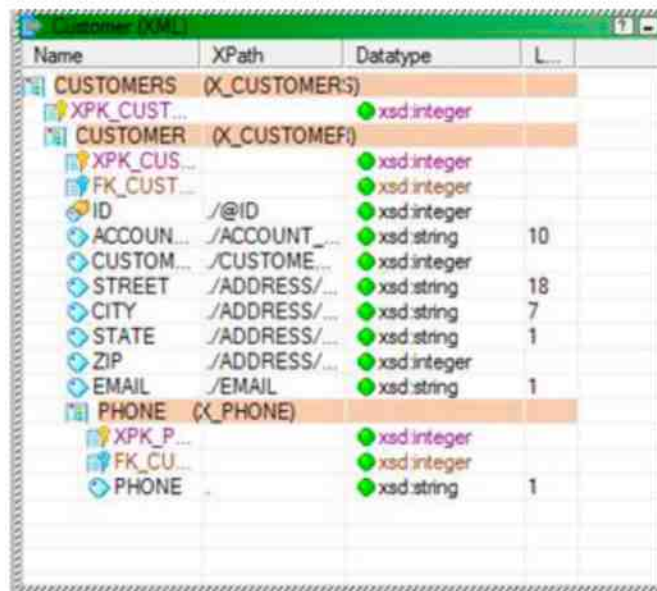
XML is a self-explanatory language. Each data element has the tags at the beginning which shows the start of the element. And a tag at the end is end of data element. A designer tool in ETL is dependent of these tags. These tags help to define the elements, their occurrences and hierarchy. Designer also verifies the data of the elements and picks a data type based on the representation. We are allowed to change the data type in the imported XML definition.

A Sample customer XML file is shown in below figure. It contains multiple occurring elements Customer, Phone and email etc. 'Customer' has been defined as root element. Based on XML data, the structure is defined in designer.

Figure 4.4: Sample Customer XML file with multiple-occurring elements

35

We should have sufficient data in the XLM file to display the hierarchy, multiple occurring element, root etc. Designer uses this sample information to create a structure. A default XML source definition can be shown as in below figure. It has separate views for the root and multiple occurring elements.



Name	XPath	Datatype	L...
CUSTOMERS (X_CUSTOMERS)			
XPX_CUST...		xsd:integer	
CUSTOMER (X_CUSTOMEF)			
XPX_CUS...		xsd:integer	
FK_CUST...		xsd:integer	
ID	/@ID	xsd:integer	
ACCOUN...	/ACCOUNT...	xsd:string	10
CUSTOM...	/CUSTOME...	xsd:integer	
STREET	/ADDRESS/...	xsd:string	18
CITY	/ADDRESS/...	xsd:string	7
STATE	/ADDRESS/...	xsd:string	1
ZIP	/ADDRESS/...	xsd:integer	
EMAIL	/EMAIL	xsd:string	1
PHONE (X_PHONE)			
XPX_P...		xsd:integer	
FK_CU...		xsd:integer	
PHONE		xsd:string	1

Figure 4.5: Customer XML source definition

A DTD (Document Type Definition) file is considered better than a XML file for creating definition. It clearly defines attributes, elements, entities of an XML file. It displays hierarchy, relation between the components and also provides the constraints on the XML files structure. Similar to the XML file, when we import a DTD file we can change the data type of the elements. What we cannot change is the cardinality. If we have an XML file which comes with DTD then import reference is given to the DTD. The definition is created based on DTD structure. Below definition shows the sample DTD file:-

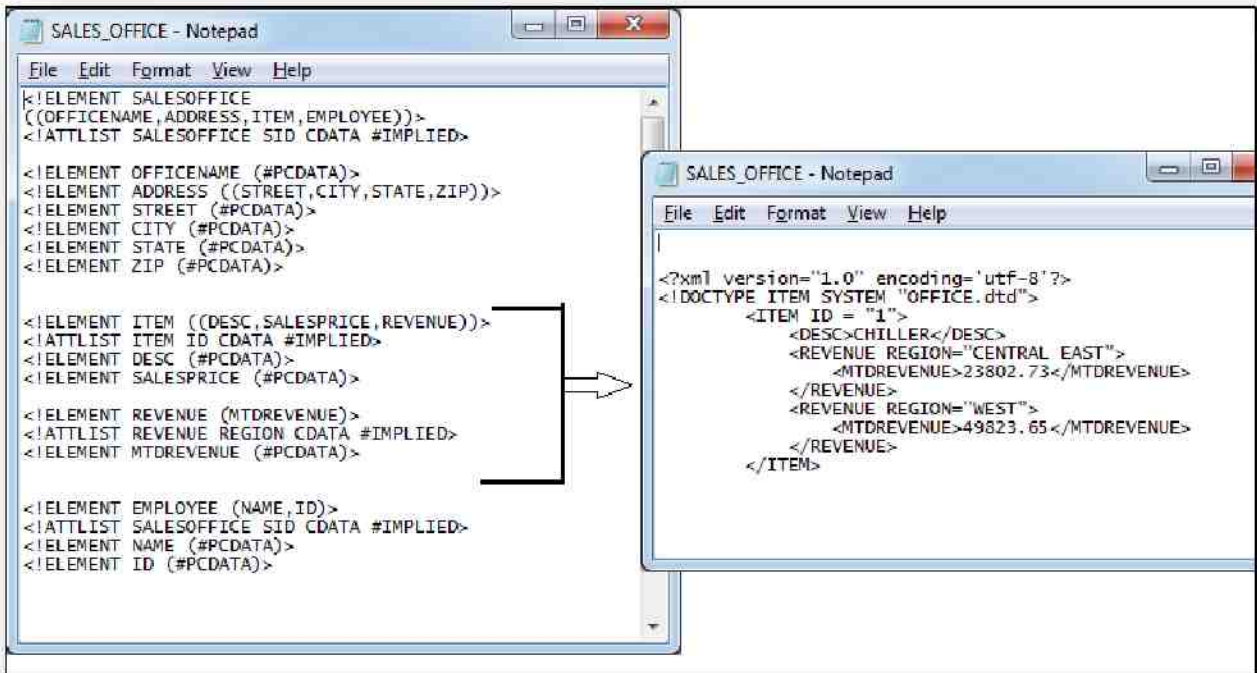


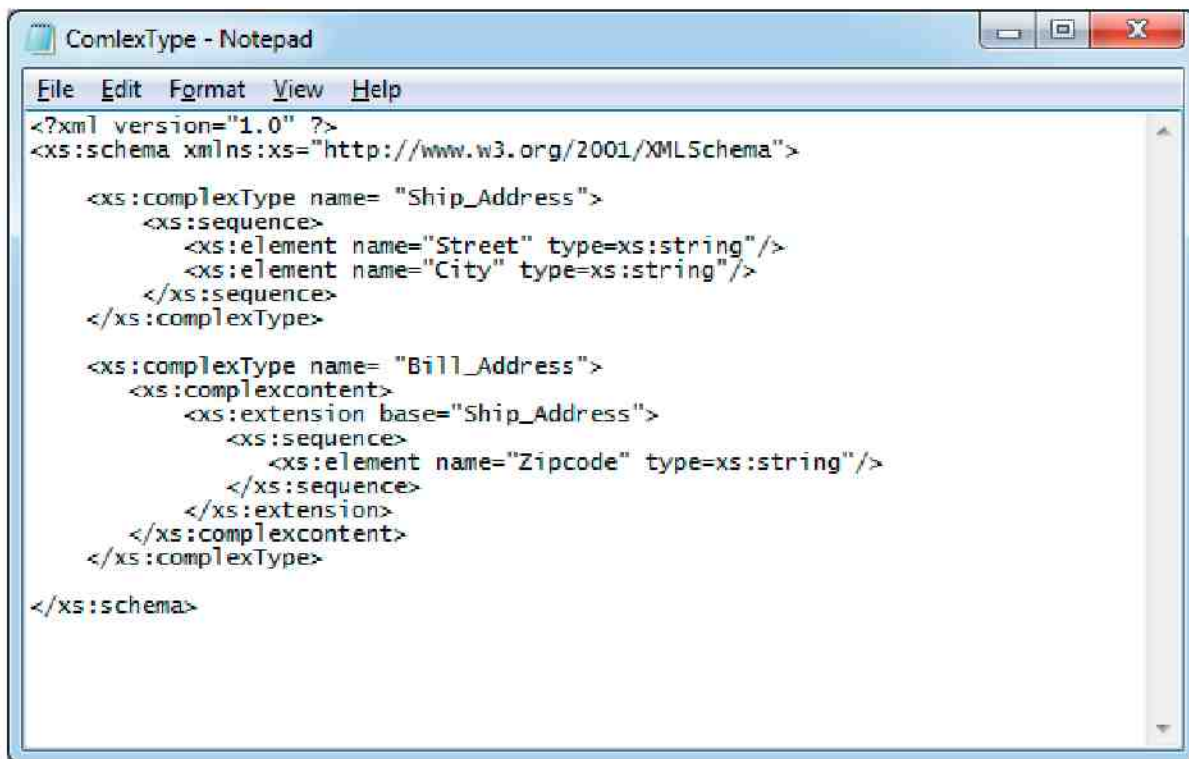
Figure 4.6: Sample DTD file

The below figure shows a source definition created in designer from above DTD.

Name	XPath	Datatype	Length/Pr...
SALESOFFICE	(X_SALESOFFICE)		
XPk_SALES...		xsd:integer	
SID	/@SID	xsd:string	infinite
OFFICENAME	/OFFICENAME	xsd:string	infinite
STREET	/ADDRESS/...	xsd:string	infinite
CITY	/ADDRESS/...	xsd:string	infinite
STATE	/ADDRESS/...	xsd:string	infinite
ZIP	/ADDRESS/...	xsd:string	infinite
ID	/ITEM/@ID	xsd:string	infinite
DESC	/ITEM/DESC	xsd:string	infinite
SALESPRICE	/ITEM/SALE...	xsd:string	infinite
REGION	/ITEM/REV...	xsd:string	infinite
MTDREVEN...	/ITEM/REV...	xsd:string	infinite
NAME	/EMPLOYEE...	xsd:string	infinite
ID	/EMPLOYEE...	xsd:string	infinite

Figure 4.7: Sample DTD Source Definition

We can also use schema file to define the structure of XML file elements. Designer has the capability to decide data type, precision, cardinality and hierarchy of the elements. We would not be able to change the definition if it is created by a schema file. We can define new data type from the existing ones. We can create new data type which contains elements of an existing data type and customize the new data type like adding restriction on the elements or add new elements. In this case designer creates view for new derived data type and does not repeat the inherited elements, thus it will reduce the size of metadata XML file.



```
ComplexType - Notepad
File Edit Format View Help
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Ship_Address">
    <xs:sequence>
      <xs:element name="Street" type=xs:string"/>
      <xs:element name="City" type=xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Bill_Address">
    <xs:complexContent>
      <xs:extension base="Ship_Address">
        <xs:sequence>
          <xs:element name="Zipcode" type=xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Figure 4.8: Sample XML Schema

In above XML Ship Address is an address and Bill Address is a derived from ship address. Bill address will have all the elements of ship address adding zip code to it.

The designer is also capable of creating an XML definition by multiple definitions. The definitions should be related to each other and designer creates link between them. It also generates primary key and foreign key relationship. An example of this scenario is creating a file which contains order header and its lines. Order line will have foreign keys that point to the order header. Flat files can also be used to create the definition in designer. We can also import multiple files at a time and designer will create one definition for each. We can also select to create relation between these files.

Similarly we have following ways to create the XML target definition:-

- 1) XML file or schema or DTD is used to import the definition. XML definitions can be imported from URL or a local system.
- 2) Target XML definition can be created by copying the source definition.
- 3) XML target definition can be created from the relational flat files.

So to conclude this we have multiple ways to create XML source or target definitions for different source systems.

4.4 Implementation

In this section, we take an example where there will be several backend database systems. First, we will extract data from different databases, and then we will do transformation, manipulation, cleaning of data, and finally load the data into a unified XML repository.

In this example, we have taken data coming from flat file, relational database, and XML database. **Salesorderheader** table data are derived from a relational database, i.e., Oracle 10g, **Salesterritory** table is in XML format, and **Customer** table is in flat file format in the source data. We will take the most meaningful data from all these sources and merge them into an XML Data warehouse. Source metadata for different source systems will look like below tables:

Table 4.1: Metadata for Customer Table

Database Type	Column	Particular	ATTRIBUTE_TYPE
Flat File	CustomerID	Number	NUMERIC
Flat File	TerritoryID	Number	NUMERIC
Flat File	AccountNumber	String	TEXT
Flat File	CustomerType	String	TEXT
Flat File	Rowguid	String	TEXT
Flat File	ModifiedDate	String	TEXT

Table 4.2: Metadata for Salesterritory Table

Database Type	Column	Particular	ATTRIBUTE_TYPE
XML	TerritoryID	Integer	NUMERIC
XML	Name	String	TEXT
XML	CountryRegionCode	String	TEXT
XML	Group	String	TEXT
XML	SalesYTD	Decimal	NUMERIC
XML	SalesLastYear	Decimal	NUMERIC
XML	CostYTD	Integer	NUMERIC
XML	CostLastYear	Integer	NUMERIC
XML	Rowguid	String	TEXT
XML	ModifiedDate	String	TEXT

Table 4.3: Metadata for SalesOrderHeader Table

Database Type	Column	Particular	ATTRIBUTE_TYPE
Oracle Database	SalesOrderID	Number	NUMERIC
Oracle Database	RevisionNumber	Number	NUMERIC
Oracle Database	OrderDate	varchar2	TEXT
Oracle Database	DueDate	varchar2	TEXT
Oracle Database	ShipDate	varchar2	TEXT
Oracle Database	Status	Number	NUMERIC
Oracle Database	OnlineOrderFlag	Number	NUMERIC
Oracle Database	SalesOrderNumber	varchar2	TEXT
Oracle Database	PurchaseOrderNumber	varchar2	TEXT
Oracle Database	AccountNumber	varchar2	TEXT
Oracle Database	CustomerID	Number	NUMERIC
Oracle Database	ContactID	Number	NUMERIC
Oracle Database	SalesPersonID	varchar2	TEXT
Oracle Database	TerritoryID	Number	NUMERIC
Oracle Database	BillToAddressID	Number	NUMERIC
Oracle Database	ShipToAddressID	Number	NUMERIC
Oracle Database	ShipMethodID	Number	NUMERIC
Oracle Database	CreditCardID	varchar2	TEXT
Oracle Database	CreditCardApprovalCode	varchar2	TEXT
Oracle Database	CurrencyRateID	varchar2	TEXT

Oracle Database	SubTotal	Number	NUMERIC
Oracle Database	TaxAmt	Number	NUMERIC
Oracle Database	Freight	Number	NUMERIC
Oracle Database	TotalDue	Number	NUMERIC
Oracle Database	Comment	varchar2	TEXT
Oracle Database	Rowguid	varchar2	TEXT

The user should understand the source schema and the relationship that exists between tables within the database, to capture metadata for the heterogeneous sources. Metadata can be captured manually or with the help of wizards in the ETL tool.

Below diagram shows a sample XSD for the SalesTerritory Table

```
<?XML version = '1.0' encoding = 'UTF-8'?>
<xs:schema xmlns:xs = 'http://www.w3.org/2001/XMLSchema'>
<!-- SalesTerritory definition -->
<xs:element name = 'SalesTerritory'>
<xs:complexType>
<xs:attribute name = 'TerritoryID' type = 'xs:integer'/>
<xs:attribute name = 'Name' type = 'xs:string'/>
<xs:attribute name = 'CountryRegionCode' type = 'xs:string'/>
<xs:attribute name = 'Group' type = 'xs:string'/>
<xs:attribute name = 'SalesYTD' type = 'xs:decimal'/>
<xs:attribute name = 'SalesLastYear' type = 'xs:decimal'/>
<xs:attribute name = 'CostYTD' type = 'xs:string'/>
<xs:attribute name = 'CostLastYear' type = 'xs:integer'/>
<xs:attribute name = 'rowguid' type = 'xs:string'/>
<xs:attribute name = 'ModifiedDate' type = 'xs:string'/>
</xs:complexType>
</xs:element>

</xs:schema>
```

Figure 4.9: XML Schema Definition (XSD) of the SalesTerritory Table

Integrating source metadata into a global metadata

We have heterogeneous database systems used as a source database schema; the metadata collected from different sources is merged in a global metadata schema. This integration helps us to resolve the possible conflicts in source tables. The rules to resolve such a kind of conflicts are stored in global metadata.

Example of different rules to resolve conflicts in source metadata are: different sources can follow different naming conventions, so the table names may differ. For example, Employee table in two sources can be named as: 'EMP_Dim' or 'EMPLOYEE_D'. Following guidelines can be used to integrate different database schema:

Schema Integration

The integration of the source database schemas is possible in several ways. The final integrated global schema is a view of all the source systems, which consists of heterogeneous data sources of an organization. Each source system or application is considered as an object for the integration process. The purpose of global metadata schema is to ensure that we avoid duplicate data, and all the conflicts among source systems are resolved. Some of the processes to handle these issues are listed below:

- *Eliminate inconsistencies between source schemas*

This process handles the conflicts related to definition, e.g., inconsistent key column names or data type. All the conflicts between sources are stored in global metadata.

- *Merging entities*

The entities can be integrated into a single entity by using UNION operator, only if they contain a similar kind of data in terms of business domain. For example, customer data from different locations or applications can be merged into a single entity. For this integration process, we need to identify the same keys with similar entities for business logic. The entities can be merged using UNION by subtype relationship, aggregation, generalization or cardinality.

All source metadata schemas are integrated into global metadata schema. The integrated database global schema is useful for analysis. The global metadata table reflects the merged view of data. The integrated attributes define merged source attributes. The mapping rules are the methods to deal with conflicts of source systems.

Global schema generated from source schema

The global metadata schema is populated from source metadata of different source systems or applications. To build global metadata schema, we should have the knowledge of business logic and source metadata. The global metadata to show the relation between different sources are captured in below table (table 4.4)

Table 4.4: Global metadata table

GLOBAL_TABLE	GLOBAL_FIELD_NAME	JOIN_FIELD_ID	KEY_TYPE
CUSTOMER	CUSTOMER_ID	CUSTOMER_ID	PRIMARY
SALESORDER	CUSTOMER_ID	CUSTOMER_ID	FOREIGN
SALESORDER	ORDER_ID	ORDER_ID	PRIMARY
SALESORDER	TerritoryID	TerritoryID	FOREIGN
SALES_TERRITORY	TerritoryID	TerritoryID	PRIMARY

The integrated source database schema from the heterogeneous sources is stored in a global metadata schema. Integrated global schema will remove the duplicate data issues in the global schema. The metadata of source system contains data about database, tables and their attributes.

Below is the ER diagram of the data we have integrated into our database:

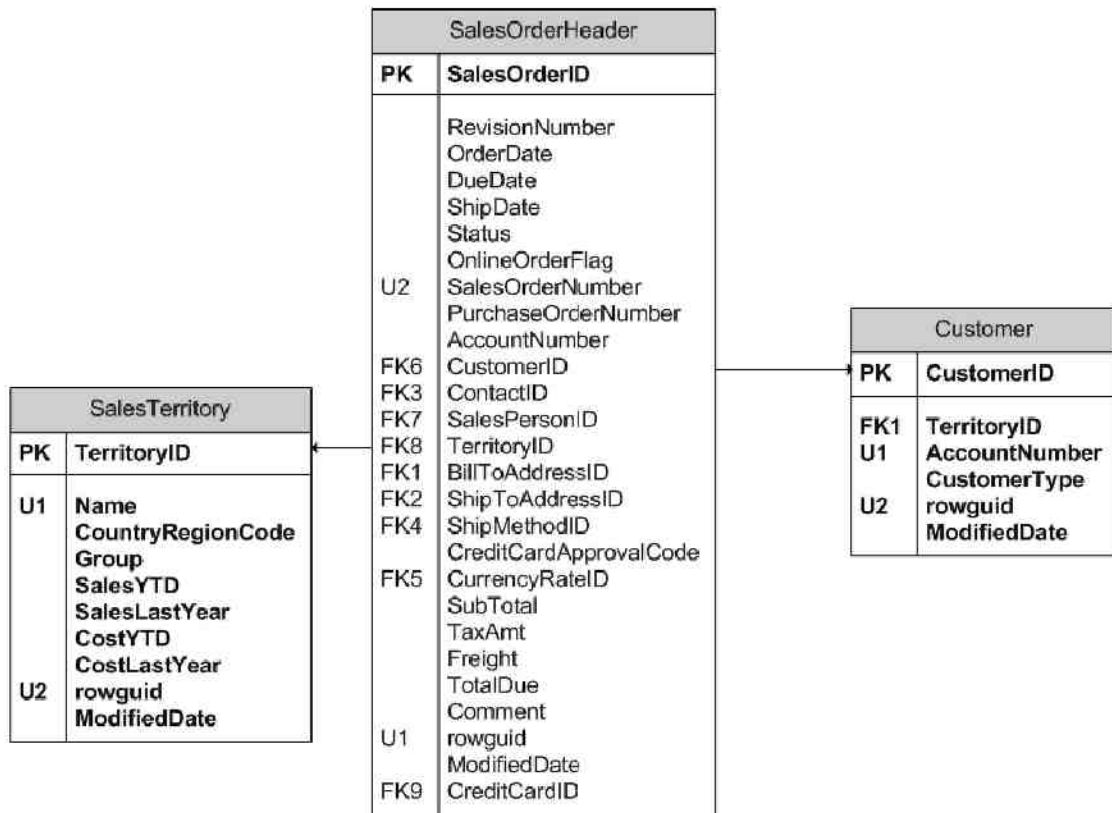


Figure 4.10: ER diagram of Sales department

Similarly, we will integrate the data for the HR department. Below is the ER diagram for the HR department. In HR department, source data are from Oracle database, flat file, and XML database. We will integrate this heterogeneous data into a unified XML Data warehouse. We will follow the same step to integrate this data.

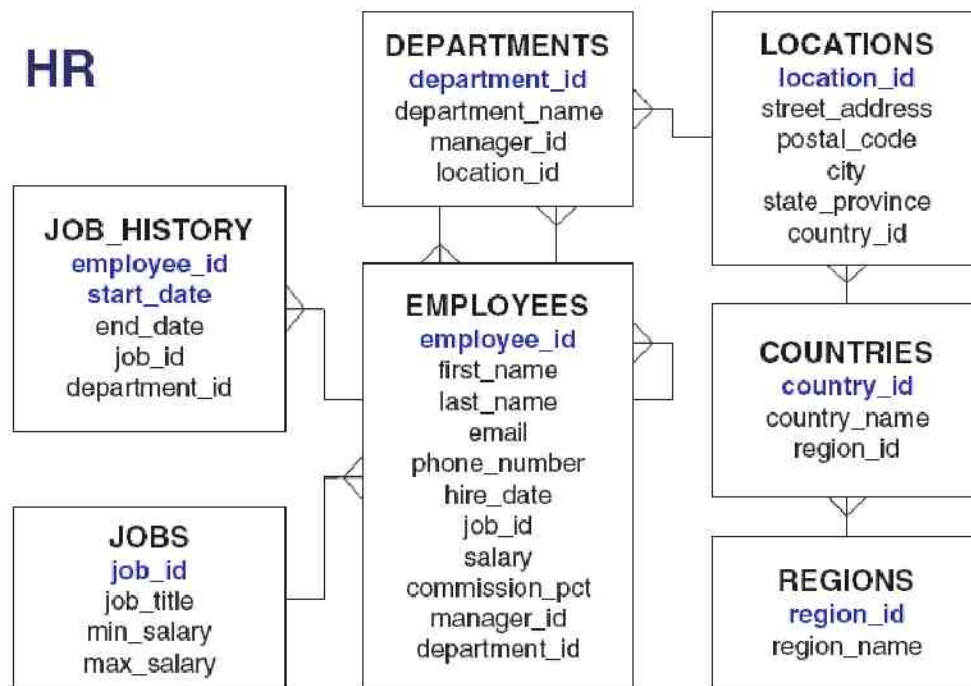


Figure 4.11: ER diagram of HR department

Once the schemas are generated, this schema data is utilized to integrate heterogeneous data into XML Data warehouse. As data is loaded from multiple heterogeneous databases, different steps are followed to build the XML Data warehouse. So following all the above procedure our XML Data warehouse is built and is ready for analysis.

4.5 Performance comparison

The objective of performance analysis is to examine the performance of XML Data warehouse against traditional Data warehouse using sample data.

4.5.1 Loading data in Data warehouse

Here, we have taken sample data set for source data and then recorded the time taken to load the data from heterogeneous sources into XML Data warehouse to test its performance. Also, to compare the performance of XML Data warehouse with traditional Data warehouse, we have used the same sample data and recorded the time taken to load data from heterogeneous sources into traditional Data warehouse. For the traditional Data warehouse, we have chosen the target database as Oracle 10g. Below table (table 4.5) shows the time taken to load the

data into XML Data warehouse and traditional Data warehouse for the same set of sample data. The time taken to load the data into the Data warehouse is subjected to change depending on many factors like hardware and software configuration, mapping, transformation used, and data load etc.

The hardware and software configurations are following:

- Intel Centrino Duo 1.66 GHz CPU
- 4 GB RAM
- Windows 7
- 400 GB Hard disk
- Oracle 10g

Table 4.5: Table for time taken to load data in Data warehouse

Number of records	Time taken to load data into XML Data warehouse(sec)	Time taken to load data into relational Data warehouse(sec)
500	3	20
1000	4	37
3000	4	39
5000	5	63
10000	6	87
20000	8	112
30000	12	184

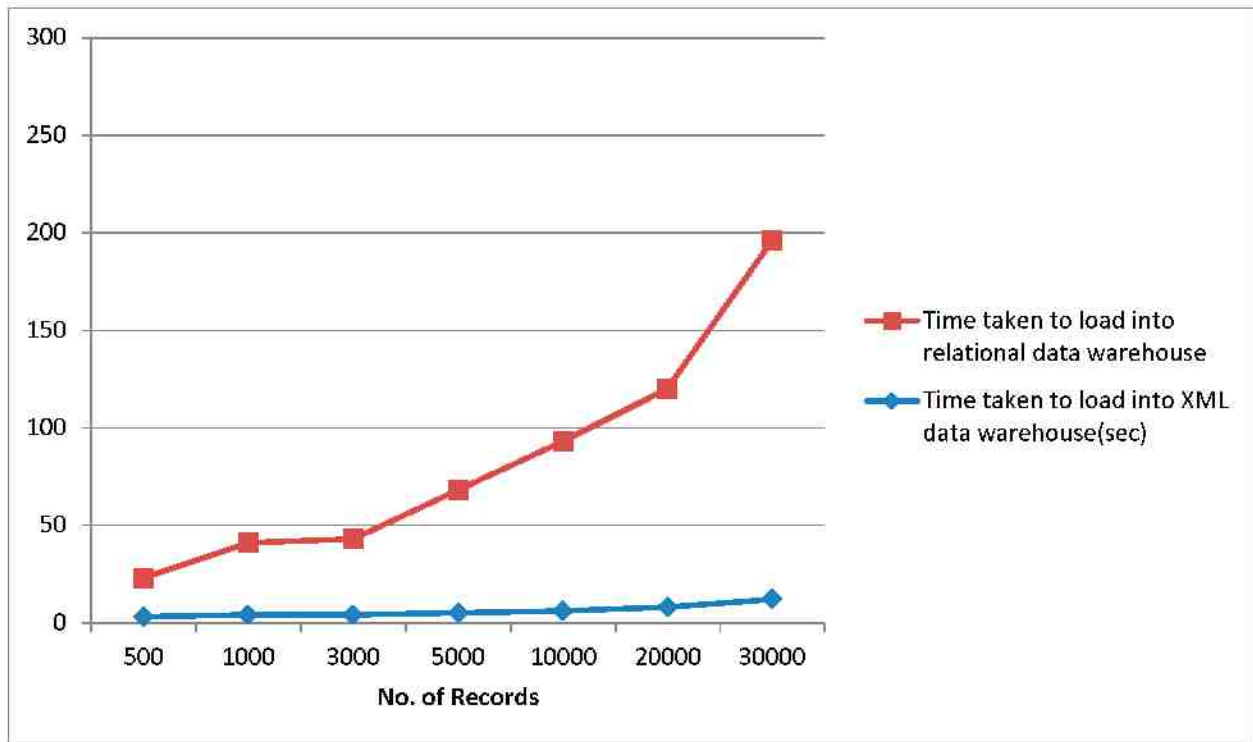


Figure 4.12: Comparative graph for Load time

From the table and graph, we can easily say that XML Data warehouse takes less time to load heterogeneous data source into XML Data warehouse than the traditional Data warehouse. So XML Data warehouse has a reduced load time in comparison to the traditional Data warehouse.

4.5.2 Front-end Information Delivery

After building a Data warehouse different reporting tools (e.g., Cognos 8BI, OBIEE, Business Objects, and SSRS etc.,) are used to make reports to deliver information to the end user. The extensive utilization of XML will be helpful in eliminating reporting and query tools from end user's terminals. We can use XSLT (XSL Transformations) to transform XML documents and produce the desired report for the end user. The desired business report could be published with HTML pages in the company's intranet website. Below is the report which I have made with XML document using XSLT. I have not used any Data warehouse reporting tool to create this report.

Below is the sample Report created from XML document using XSLT:-

Territory	Territory Name	Country/Region	Year To Date Sales	Last Year Sales
1	Northwest	US	5767341.975	3298694.494
2	Northeast	US	3857163.633	3607148.937
3	Central	US	4677108.269	3205014.077
4	Souththwe	US	8351296.741	5366575.710
5	Southeast	US	2851419.044	3925071.432
6	Canada	CA	6917270.884	5693988.860
7	France	FR	3899045.694	2396539.760
8	Germany	DE	2481039.179	1307949.792
9	Australia	AU	1977474.810	2278548.978
10	United Kingdom	GB	3514865.905	1635823.397

4.5.3 Disk space utilization

XML Data warehouse takes less space to store data as compared to the traditional Data warehouse. Semi-structured database has irregular structure therefore when we try to store it in relational format it may result in creation of many columns with null values, which in turn will increase the size of wasted space. Also, this may result in creation of unwanted tables that will again occupy database space. XML Data warehouse is a better alternative to store data that includes large number of attributes, many of which are null. Thus XML Data warehouse will use less disk space as compared traditional Data warehouse.

4.5.4 Data Retrieval in Reports

Once the data is loaded in warehouse, we can build reports on it. We will test the performance of Data warehouse by creating similar reports in XML Data warehouse and relational Data warehouse. The time taken by reports to display the content, can work as a parameter for performance analysis.

Time taken by reports depends on various factors like

- Type of report (Detail or Summary)
- Number of dimensions or facts used in reports
- Data volume of fact table
- Data volume of dimension
- Type of summarization used like sum or average

Query fetch time also depends on the indexes on the tables. A report may take more time if more fact and dimension tables are used in it. Usually multiple dimensions are used in a report

and joining them with facts consumes time in most of the cases. Attributes used for joins are preferred to have indexes.

In our analysis we will vary the count for dimension and fact table used and analyze the time taken by reports. This time will be compared between XML Data warehouse and relational Data warehouse. I have built these reports using OBIEE 10g.

Table 4.6: Table for time taken to retrieve data in Reports

Report type	No. of Fact Table Used	No. of Dimension Tables Used	No. of Records	Time Taken (in sec)	
				XML Data warehouse	Relational Data warehouse
Summary	1	1	4	6	13
Summary	1	1	29	11	23
Summary	1	2	32	24	37
Summary	2	2	35	33	49
Summary	2	3	121	36	56
Detail	1	1	237	13	20

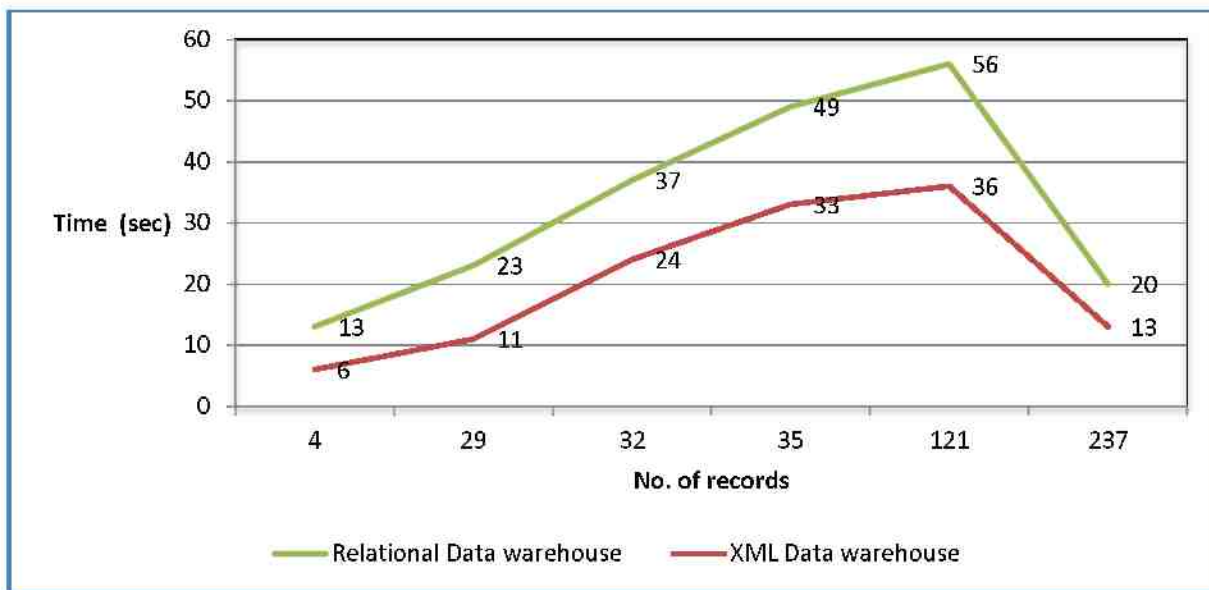


Figure 4.13: Graph for time taken to retrieve data

From the above table we can say that the XML Data warehouse is taking less time to retrieve data in report then traditional Data warehouse. This behavior is because of following reasons:-

Traditional Data warehouse is taking more time because many of the reporting tools use XML at the backend to display the content at the front end. In traditional Data warehouse, reporting tools fetch data from relational database and convert them into XML format. Reporting tool create XML at backend which has SQL query embedded in it. Query runs on the relational database and the result is converted into XML format. So when we have XML database at the backend, conversion of data from relation format to XML format is not required and thus it takes less time to display report data.

The other cause of the delay in traditional Data warehouse is the hierarchies used in the reports. Most of the reports are based on some kind of hierarchy. In relational database these hierarchies are stored using multiple tables. Creating a single table to maintain hierarchy will cause duplication of data and consume more space. For example, while capturing customer details like address, address type or customer type, the details could be stored in multiple tables. In relation database, we might use four tables to store this information. One table will store customer detail like customer type or account number. The second table will have customer id and address id. Next table will contain address details like street, city and state. Another table will contain details about address type. Tables in our traditional Data warehouse should look like the tables shown below:

Customer	
PK	CustomerID
FK1	TerritoryID
U1	AccountNumber
	CustomerType
U2	rowguid
	ModifiedDate

CustomerAddress	
PK,FK3	CustomerID
PK,FK1	AddressID
FK2	AddressTypeID
U1	rowguid
	ModifiedDate

Address	
PK	AddressID
U2	AddressLine1
U2	AddressLine2
U2	City
FK1,U2	StateProvinceID
U2	PostalCode
U1	rowguid
	ModifiedDate

AddressType	
PK	AddressTypeID
U1	Name
U2	rowguid
	ModifiedDate

Figure 4.14: Sample Traditional Data warehouse tables

Here customeraddress, address type, and address table will contain child data. Customer table can be considered to be parent. Child data alone does not make much sense here without

parent. The child will be more descriptive when it is seen with parent and other child. In XML Data warehouse these types of hierarchies can be easily maintained in the single XML document.

Below figure shows a single XML document where the above described inherent hierarchy relationship is displayed. In XML Data warehouse a single xml document is able to manage the inherent hierarchy as compared to traditional Data warehouse where four tables were used. In case of traditional Data warehouse while fetching data in reports, all four tables need to be joined to get complete details. These joins will cause delay to display report data. Having a single XML document will definitely save the time to retrieve data.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<Customer>
  <CustomerID>1</CustomerID>
  <CustomerAddress>
    <AddressID>832</AddressID>
    <Address>
      <AddressLine1>2251 Elliot Avenue</AddressLine1>
      <AddressLine2></AddressLine2>
      <City>Seattle</City>
      <StateProvinceID>79</StateProvinceID>
      <PostalCode>98104</PostalCode>
      <AdressTypeID>3</AdressTypeID>
      <AddressType>
        <Name>Main Office</Name>
      </AddressType>
    </Address>
  </CustomerAddress>
  <TerritoryID>1</TerritoryID>
  <AccountNumber>AW00000001</AccountNumber>
  <CustomerType>S</CustomerType>

  <CustomerID>2</CustomerID>
  <CustomerAddress>
    <AddressID>297</AddressID>
    <Address>
      <AddressLine1>7943 Walnut Ave</AddressLine1>
      <AddressLine2></AddressLine2>
      <City>Renton</City>
      <StateProvinceID>79</StateProvinceID>
      <PostalCode>98055</PostalCode>
      <AdressTypeID>5</AdressTypeID>
      <AddressType>
        <Name>Shipping</Name>
      </AddressType>
    </Address>
  </CustomerAddress>
  <TerritoryID>1</TerritoryID>
  <AccountNumber>AW00000002</AccountNumber>
  <CustomerType>S</CustomerType>
</Customer>
```

Figure 4.15: Sample Customer XML document

4.5.5 Speed of operations

In-order to test the speed of operations for traditional Data warehouse and XML Data warehouse we executed queries in batches against both the Data warehouse and compared relative results in terms of execution time. The sample we took for our test contained 103 data entries.

Below are the results from performing an Insert operation on a XML data source and a traditional data source. You may notice that on Insert operation the execution time for the traditional data source is 77% of the entire batch whereas for XML data source it is merely 23% relative to the batch execution time. This proves that Insertion is faster in XML based Data warehouse.

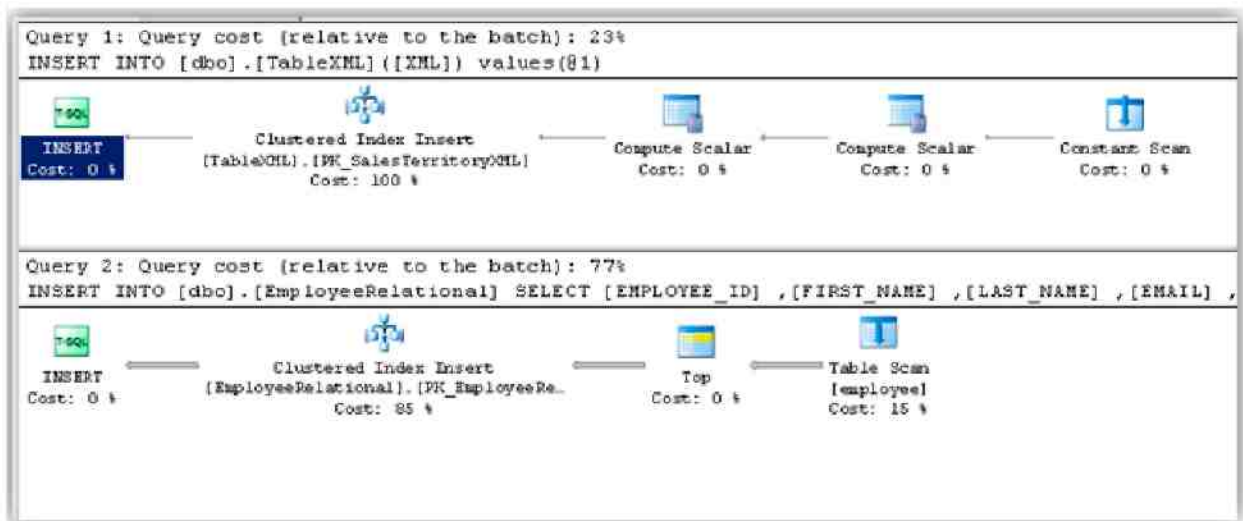


Figure 4.16: Execution time comparison for Insert Statement

Similarly we can compare the speed when performing operations like Delete, Select as shows below. From the figure below it is quite evident that XML database outperforms traditional databases in all these queries.

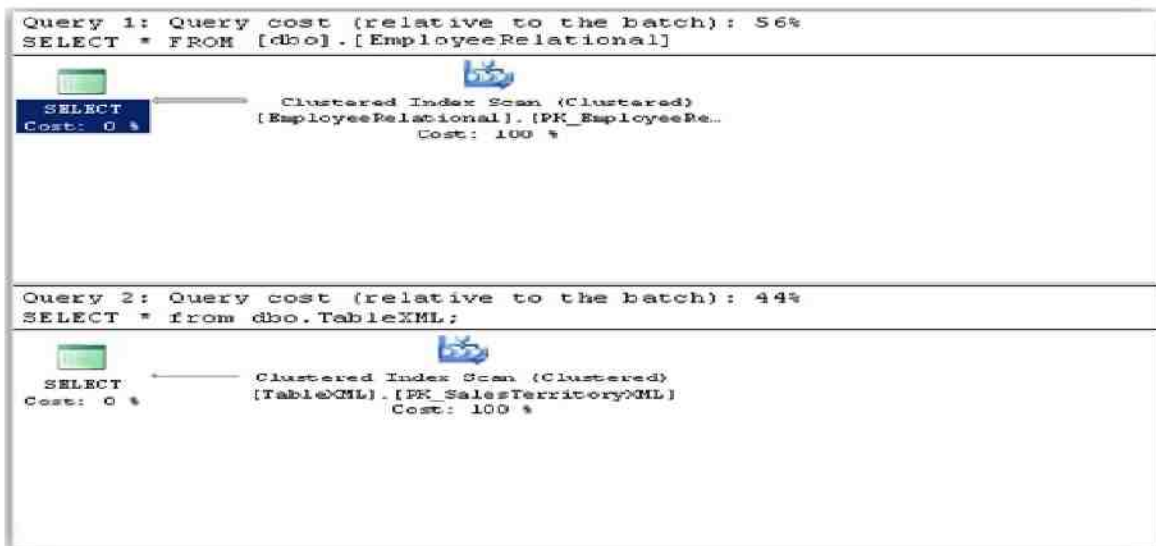


Figure 4.17: Execution time comparison for Select Statement

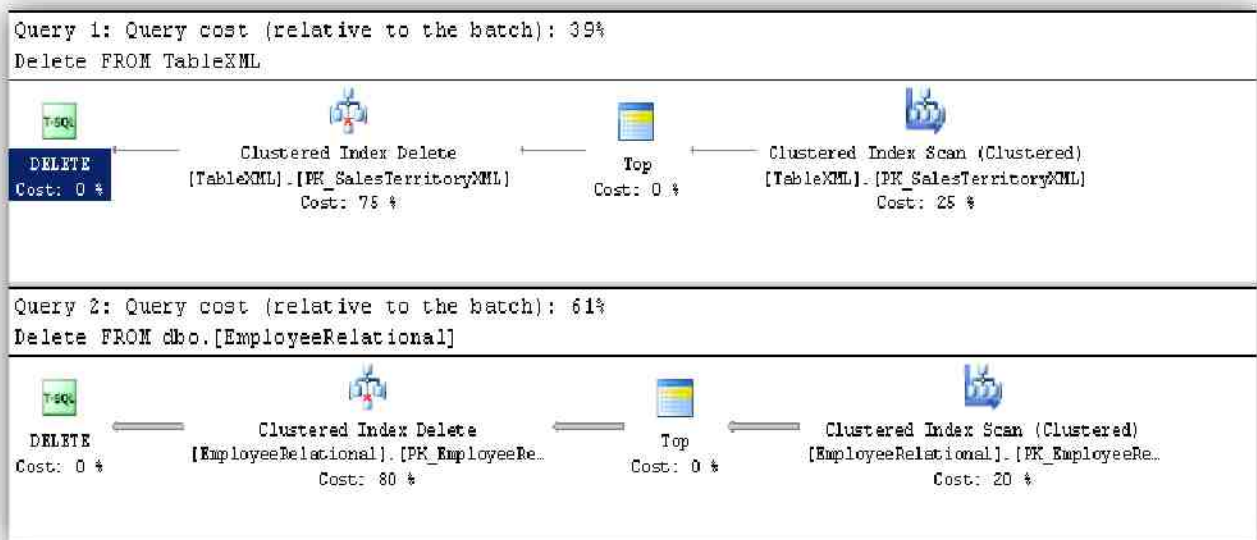


Figure 4.18: Execution time comparison for Delete Statement

The figure below shows a comparative graph of Relative Execution time between XML Data warehouse and traditional Data warehouse for all the three queries.

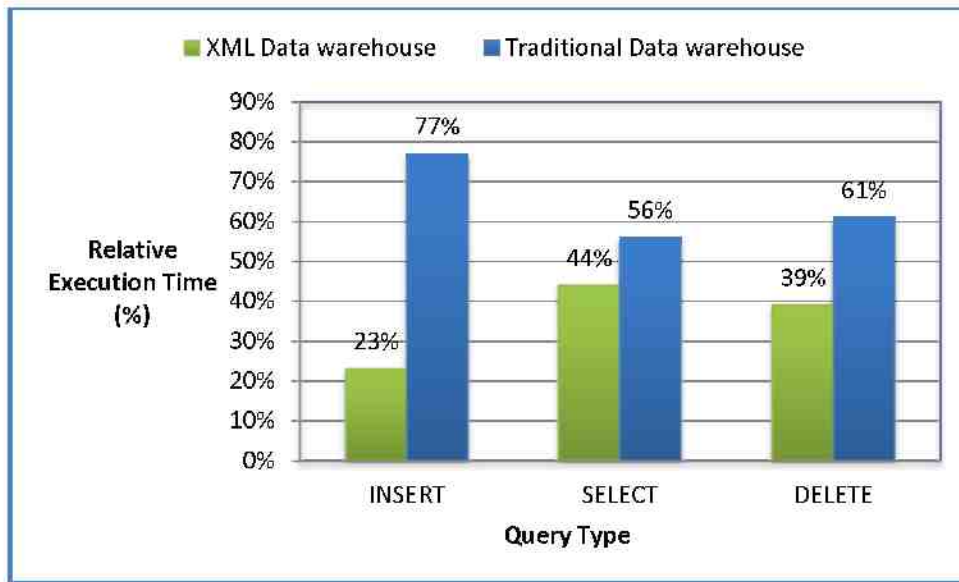


Figure 4.19: Relative Execution time graph

Based on these results we can draw safe conclusion that XML Data warehouse is much faster than traditional databases.

Thus in this section we have we have compared the performance of XML Data warehouse and traditional Data warehouse in terms of data load time, disk space utilization, data retrieval time and speed of operations. The results prove that XML Data warehouse has better performance than traditional Data warehouse.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

The aim of this thesis was to show how XML technologies can be effective in improving the Data warehousing process. We have demonstrated how XML technology can successfully increase the performance of Data warehouse. The basic idea behind this thesis is to associate XML with Data warehousing.

In this thesis first we have successfully built a conceptual architecture for Active XML Data warehouse. This AXML Data warehouse architecture has integrated complex data sources into XML Data warehouse using web services.

We have established a framework for building an Active XML Data warehouse, in term of quality of data and procedures to follow. This framework has been build following few important steps.

At last, we have implemented an XML Data warehouse with sample data from heterogeneous data sources and measured its performance in terms of data load time, disk space utilization, data retrieval time and speed of operation. We have compared the performance and found out that XML Data warehouse has a relatively better performance than traditional Data warehouse. In the end, we have proved that XML is the better format to represent and integrate different data sources inside the Data warehouse architecture

To conclude, this thesis has been an exciting experience and discovery of areas which were unknown to me in the past. This thesis has also given me an opportunity to work on various state of art Data warehousing tools like Informatica, MSBI, and OBIEE.

5.2 Future Work

In this thesis, we described a system that implements an architecture to integrate heterogeneous data into a Data warehouse using XML based technologies. The architecture uses web services to update XML documents on demand. This Active XML based Data warehouse implementation is still an ongoing activity. We are facing many challenges in integrating some of the heterogeneous data sources; for e.g., handling multimedia sources and

dynamic web sources are problematic. Our main focus in the future will be to incorporate data from such sources in our integration system. This can be done by embedding some external component into the integration system architecture such that the external component can mine relevant data from these heterogeneous sources and can also detect any modification in the Dynamic web source.

As indicated earlier, no browser can read Active XML documents as of now. Furthermore, there is no support in processing Active XML documents by end user applications. Hence, we plan to develop an Active XML Data warehouse as a web-based application and in that way allow it to be managed by a web browser.

We also plan to continue research on intelligent ETL that can be accomplished by analyzing and mining the transaction repository using more types of event. Intelligent ETL should also be capable of capturing user requirement because for any successful data integration user requirement are extremely useful.

References

1. *Comparison of the XML model and the relational model*. (n.d.). Retrieved Feb 15, 2012, from IBM:
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.apdv.embed.doc%2Fdoc%2Fc0023811.htm>
2. Khosrow-Pour, M. (2003). *Information Technology and Organizations: Trends, Issues, Challenges and Solutions*. Hershey, Pennsylvania, USA: IGI Global, 522-526
3. Darmont, J., Boussaid, O., Ralaivao, J.-C. & Aouiche, K., (2005). *An architecture framework for complex Data warehouses*. In 7th International Conference on Enterprise Information Systems (ICEIS'05), Miami, USA. INSTICC, 370–373.
4. Inmon, W. H. (1996). *Building the Data Warehouse* (2nd ed.). Hoboken, NJ, USA: Wiley, 55-72
5. Ralph Kimball, (April 26 2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)*. Wiley Computer Publishing, 81-120
6. Tseng F.S.C., Chou A.Y.H, 2006. *The concept of document warehousing for multi-dimensional modeling of textual-based business intelligence*. In journal of Decision Support Systems (DSS), vol.42(2), Elsevier, 727-744.
7. XSLT Introduction. Accessed on March 2011, from W3schools.com :
http://www.w3schools.com/xsl/xsl_intro.asp
8. Software AG Corporation, Tamino Native XML Database Management System. Accessed on Feb 2011, Available at: www.softwareag.com/tamino/technical/description.html
9. Sullivan D., 2001. *Document warehousing and Text Mining*, Wiley John & Sons.
10. Franck Ravat, Olivier Teste, Ronan Tournier and Gilles Zurlfluh, 2007. *A Conceptual Model for Multidimensional Analysis of Documents*. Published in: ER'07 Proceedings of the 26th international conference on Conceptual modeling. Springer-Verlag Berlin, Heidelberg.
11. Thomas Thalhammer, Michael Schrefl, Mukesh Mohania, 2001. *Active Data warehouses: Complementing OLAP with analysis*. Published in Journal Data & Knowledge Engineering - Data warehousing Volume 39 Issue 3, December 2001
12. ActiveXML home page. Available at <http://www.activexml.net>. Accessed on Jan 2012
13. L. Xyleme, 2001. *A Dynamic Warehouse for XML Data of the web*, IEEE Data Eng. Bull., vol. 24, no. 2,40-47.
14. M.C. Daconta, L.J. Obrst, and K.T. Smith, 2003. *The Semantic web: A Guide to the Future of XML, web services, and Knowledge Management*. John Wiley & Sons.
15. *Complete reference to Informatica*. Accessed on April 2011, Available at: <http://informaticatutorials-naveen.blogspot.com/#axzz1tn4jNwTs>,

16. E. Miller and F. Manola, RDF Primer, World Wide web Consortium (W3C) recommendation, <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>. Accessed on April 2012.
17. S. Abiteboul, T. Milo, O. Benjelloun, PODS'05. June 2005. *Regular and Unambiguous Rewritings for Active XML*, Published In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM New York, NY, USA
18. Rashed SALEM, Jerome Darmont and Omar Boussaid, Oct 2009. *Toward Active XML Data warehousing*. International ACM Conference on Management of Emergent Digital EcoSystems, New York.
19. Ladjel Bellatreche, August 2009. *Data warehousing Design and Advanced Engineering Applications: Methods for Complex Construction*. Poitiers University, France , 189-204
20. BenMessaoud R., Loudcher-Rabaseda S., & Boussaid O., (2006). *A data mining-based OLAP aggregation of complex data: Application on XML Document*. International Journal of Data warehousing and Mining, 2(4), 1–26
21. Mahboubi, H., & Darmont, J. (2007). *Indices in XML databases*. In *Encyclopedia of database technologies and applications, second edition*. Hershey, PA: IGI Global
22. Mahboubi, H., Aouiche, K., & Darmont, J. (2008). *A join index for XML Data warehouses*. In Proceedings of the 2008 International Conference on Information Resources Management (Conf- IRM'08), Niagara Falls, Canada.
23. Nassis, V., Rajugan, R., Dillon, T. S., & Rahayu, J. W. (2004). *Conceptual design of XML document warehouses*. In *Proceedings of the 6th International Conference on Data warehousing and Knowledge Discovery (DaWaK'04)*, Zaragoza, Spain (LNCS 3181, pp. 1-14). Berlin, Germany: Springer.
24. Nassis, V., Rajugan, R., Dillon, T. S., & Rahayu, J. W. (2005). *A requirement engineering approach for designing XML-view driven, XML document warehouses*. In Proceedings of the 29th International Conference on Computer Software and Applications (COMPSAC'05), Edinburgh, UK (pp. 388-395). Washington, DC: IEEE Computer Society.
25. Nassis, V., Rajugan, R., Dillon, T. S., & Rahayu, J. W. (2005). *Conceptual and systematic design approach for XML document warehouses*. International Journal of Data warehousing and Mining, 1(3), 63–86.
26. Pokorný, J. (2002). *XML Data warehouse: Modelling and querying*. In Proceedings of the 5th International Baltic Conference (BalticDB&IS'06), Tallin, Estonia (pp. 267-280). Institute of Cybernetics at Tallin Technical University.
27. Wang, H., Li, J., He, Z., & Gao, H. (2005) *OLAP for XML data*. In Proceedings of the 1st International Conference on Computer and Information Technology (CIT'05), Shanghai, China. Washington, DC: IEEE Computer Society, 233-237
28. Rusu, L. I., Rahayu, J. W., & Taniar, D. (2005). *A methodology for building XML Data warehouse*. International Journal of Data warehousing and Mining, 1(2), 67–92.

29. Vrdoljak, B., Banek, M., & Rizzi, S. (2003). *Designing web warehouses from XML schemas*. In *Proceedings of the 5th International Conference on Data warehousing and Knowledge Discovery (DaWaK'03)*, Prague, Czech Republic Berlin Germany, 89-98.
30. Boufares, F., & Hamdoun, S. (2005). *Integration techniques to build a Data warehouse using heterogeneous data sources*. *Journal of Computer Science*, 48-55.
31. XML Guide by Informatica Power Center (Version 8.6), 198-234
32. S. Abiteboul, T. Allard, P. Chatalic, G. Gardarin, A. Ghitescu, F. Goasdoué, I. Manolescu, B. Nguyen, M. Ouazara, A. Somani, N. Travers, G. Vasile, and S. Zoupanos, , August 2008. *webContent: Efficient P2P warehousing of web Data*. Published in: *Journal Proceedings of the VLDB Endowment*, Volume 1 Issue 2.
33. S. Abiteboul, I. Manolescu, and E. Taropa. *A Framework for Distributed XML Data Management*. *Advance in database technology - EDBT 2006, Lecture Notes in Computer Science*, 2006, Volume 3896/2006
34. Joseph Fong, Herbert Shiu and Davy Cheung, September 2008. *A Relational-XML Data warehouse for data aggregation with SQL and XQuery*. Published in *Journal: Software—Practice & Experience* Volume 38 Issue 11, John Wiley & Sons, Inc. New York, NY, USA
35. A. Arion, V. Benzaken, I. Manolescu, and R. Vijay, 2005. *ULoad, Choosing the right storage for your XML application*. *International Conference on Very Large Databases (VLDB)*, 1330–1333.
36. Rashed Salem, Omar Boussaid and Jerome Darmont, 2006. *Conceptual Workflow for Complex Data Integration using AXML*. Universite de Lyon (ERIC Lyon 2) 5 av. P. Mendes-France, 69676 Bron Cedex, France
37. Darmont, J. & Boussaïd, O. (Eds.), 2006. *Managing and Processing Complex Data for Decision Support*. Idea Group Publishing.
38. Codd, E., Codd, S. & Salley, C., (1994). *Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate*. White paper, E.F. Codd Associates.
39. Laura Irina Rusu, Wenny Rahayu, David Taniar, (2004). *On Building XML Data warehouses* LaTrobe University, Department of Computer Science and Computer Engineering
40. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Francois Yergeau, John Cowan, (2006). *Extensible Markup Language (XML) W3C, 2nd Edition*, Accessed at September 2011, Available at: <http://www.w3.org/TR/2006/REC-xml11-20060816/>
41. Boussaid O., BenMessaoud R., Choquet R. & Anthoard S., (2006). *X-warehousing: An XML Based Approach for warehousing Complex Data*. In: *10th East-European Conference on Advances in Databases and Information Systems (ADBIS'06)*, Thessaloniki, Greece. Vol. 4152 of *Lecture Notes in Computer Science*. Springer, 39–54.

42. Frank S.C. Tseng and Chia-Wei Chen, June 2005. *Integrating heterogeneous Data warehouses using XML technologies*. Published in Journal of Information Science Volume 31 Issue 3.
43. Ahmed Bahaa Farid, Prof.Dr. Ahmed Sharaf Aldin Ahmed., Prof.Dr. Yehia Mostafa Helmy, October 2008. *Designing new XML Based Multidimensional Messaging Interface for the new X-Warehouse Architecture*. In Proceedings of the International Multi-conference on Computer Science and Information Technology, 525 – 534
44. Serge Abiteboul, Omar Benjelloun¹, Ioana Manolescu, Tova Milo, Roger Weber, 2002. *Active XML: A Data-Centric Perspective on web services*. In Proceedings of the VLDB (Very Large Data Base) Endowment.
45. Laila Alami Kasri, 2010. *Model of Storage XML Database Based on the Relational-Object Model*. In International Journal of Engineering Science and Technology: Vol. 2(11), 6646-6656
46. Belen Vela, Carlos Blanco, Eduardo Fernandez-Medina, Esperanza, 2012. *A practical application of our MDD approach for modeling secure XML Data warehouses, Decision Support Systems* 52, 899–925
47. World Wide Web Consortium, XML Query(XQuery), W3C Working Draft (2003). Accessed on Jan 2012, Available at: www.w3.org/XML/Query
48. World Wide Web Consortium, XML Schema, W3C Recommendation (2001). Available at: www.w3c.org/TR/xmlschema-0
49. Stylus Studio: Powerful XML Integrated Development Environment (XML IDE). Available at: http://www.stylusstudio.com/xml_download.html