**Chapter 1**

**INTRODUCTION**

The project deals with the design of a sales order data mart which is a subset of a data warehouse and producing dashboard interface as BI solution. It also deals with the application of data mining algorithms on the result data mart to find out hidden patterns and relations among data.

**1.1 Data Warehouse**

An organization manages information in two dominant forms: operational systems ofrecord and data warehouses. Operational systems are designed to support online transactionprocessing (OLTP) whereas data warehousing systems are designed to support online analyticalprocessing (OLAP).

While the operational systems primarily focus on current data management, the datawarehouses update and store historical data. They are generally subject-specific and usually carrydata from multiple operational systems to support organizational decision-making.

A data warehouse is a storehouse of an organization’s data. Information fromoperational systems is extracted and imported into the data warehouse on a regular basis. As aresult, complex inquiries, or queries, can be conducted through the data warehouse with minimalinterruptions to the operational systems. The imported data is read-only and only adds to the dataexisting in the data warehouse. With greater amounts of data, the value of the data warehouse tothe user increases, since analyses looking over a longer period of time become possible. When auser query is submitted to the warehouse, all relevant historical data addressing that query isreadily available and current to support in the decision-making.

**1.2 Data Mart**

The data mart is a subset of the data warehouse, usually oriented to a specific business line or team.

In other words, a data mart is a repository of data gathered from operational data and other sources that is designed to serve a particular community of knowledge workers. In scope, the data may derive from an enterprise-wide database or data warehouse or be more specialized.

The emphasis of a data mart is on meeting the specific demands of a particular group of knowledge users in terms of analysis, content, presentation,and ease-of-use. Users of a data mart can expect to have data presented in terms that are familiar.

A data mart can also be defined as a repository of a business organization's data implemented to answer very specific questions for a specific group of data consumers such as organizational divisions of marketing, sales, operations, finances, collections and others.

**1.3Dashboard Reporting**

In management information systems, a **dashboard** is an executive information system user interface that is designed to be easy to read. The end user understands the data better when presented using a dashboard. The dashboards are built using various reports.

**1.4 Data Mining**

Data mining(sometimes called data or knowledge discovery), *the extraction of hidden predictive information from large databases*, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses.

Data mining tools predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions.

The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

Also, data mining can also be used for analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

**Chapter 2**

**PROBLEM DEFINITION**

**2.1 Introduction**

In many ways, the potential for success of any data integration/business intelligence solution correlates directly to the clarity and focus of its business scope. If the business objectives are vague, there is a much higher risk of failure or, at least, of a less-than-direct path to likely limited success. Hence, in this chapter we define the business drivers, business objectives and the business goals of the project.

**2.2 Business Drivers**

The business drivers explain why the solution is needed and is being recommended at a particular time by identifying the specific business problems, issues, or increased business value that the project is likely to resolve or deliver. Business drivers include background information necessary to understand the problems and/or needs. This is explained in the following paragraphs.

An enterprise’s profitability depends on a clear picture of product sales and delivery. The Sales module functionality allows us to understand the product sales in terms of volume and value, by examining how much of the product we are selling, and how much we are selling it for.

The Sales functional area encompasses the activities of Opportunity and Pipeline, and Bookings and Revenue. Measuring cycle times across the entire span of sales allows us to evaluate costs in the bigger picture. Prospects enter the opportunity pipeline and fall out or become your customers. Customers enter the sales order process, ultimately contributing to the company bookings and to revenue reports. As shown in the process flow in figure below, the Sales module allows us to rollup prospect and customer activities to an enterprise evaluation. Opportunity and Pipeline information rolls into Bookings and Revenue, and content from both are rolled into Sales Performance.

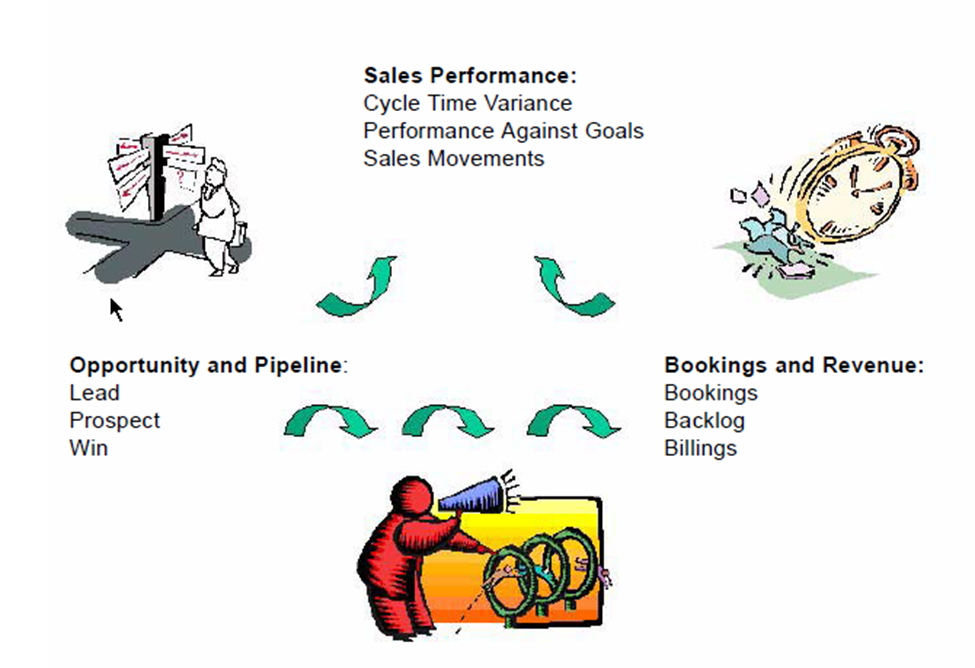


Fig 2.1Sales Module Process Flow

The sales order is the entry point in the sales order process, and is the record, of a customer placing an order with the organization for the supply of goods or services. The analysis of sales performance allows us to examine data at a high level (across many sales orders), or down to the detail of a single line item on one sales order.

The sales order lines document enables us to analyze demand placed by the customer on the organization. Details about quantity and shipment dates are all in the sales order lines; we may further break down the level of detail within our sales order lines to view average sales prices, order, confirmed and shipped quantities, creation dates, and sales orders confirmations.

**2.3 Business Objectives**

Objectives are concrete statements describing what the project is trying to achieve. Objectives should be explicitly defined so that they can be evaluated at the conclusion of a project to determine if they were achieved.

The main objective of this project is to build a Sales Order Data Mart. And it is achieved through Informatica Power Center (ETL Tool). Once the Data Mart is built reports are generated using a business intelligence toolset called MicroStrategy to facilitate reporting and analysis of following metrics.

* Number of Orders: This metric measures the total number of orders that were placed by the customers. This metric can be combined with the Total Ordered Amount metric to calculate the Average Ordered Amount.
* Number of Bookings: This metric tracks the total number of bookings placed by customers. It can be combined with the Number of Orders to calculate Orders to Bookings Close Rate.
* Orders to Bookings Close Rate: The Orders to Bookings Close Rate identifies the percentage of all closed orders by comparing orders closed (booked) to all orders placed during a chosen time period. This metric calculates the close rate of an organization’s orders. A closed order is an order that has been booked; booked orders show as a percentage of all orders placed. This helps analyze an organization’s order fulfillment performance.
* Product Metric: This metric gives the details of all the products. It gives total number of customers for that product, number of orders placed by the customers for a particular product and number of orders booked for that product.

These reports generated containing the above mentioned metrics are further used for creating dashboards using MicroStrategy Web.

Data mining applications are built using the data in the data mart. FP-Growth algorithm is implemented to mine the frequent itemsets in the data mart and Bayesian classification is implemented to classify the data in to classes and for prediction.

A common descriptive category for analysis includes the following dimensions- Customers, Products, Region and Time Period.

**2.4 Business Goals**

1. Build extraction, transformation, and load processes in Informatica to extract data from Oracle Applications (ERP) instance to load dimension tables such as Customers, Products,Region and Time Period
2. Build extraction, transformation, and load processes in Informatica to extract sales order data and populate Sales Order and Bookings Fact data with appropriate linkages to dimension data
3. Design and Build reports for Business users using MicroStrategy tools to report metrics and attributes above and facilitate analysis across multiple dimensions
4. Design and Build data mining applications for business users using Java to mine hidden data and facilitate analysis of hidden patterns in data.

**Chapter 3**

**LITERATURE SURVEY**

**3.1 Introduction**

The project constituted of building a data mart using Informatica PowerCenter, building reports and dashboards using MicroStrategy and implementing data mining applications on the data mart. We have implemented this project as interns in Informatica Corp.

**3.2 Informatica**

**3.2.1 About Informatica**

Informatica Corporation (NASDAQ: INFA) is the world’s number one independent provider of data integration software. More than 4,200 enterprises worldwide rely on Informatica to access, integrate and trust their information assets held in the traditional enterprise, off premise and in the Cloud.

**3.2.2 Informatica Products**

The Informatica Platform is the first comprehensive, unified, and open software platform specifically designed for data integration. The open, platform-neutral software accesses data of virtually all types and makes it accessible, meaningful, and usable to the people and processes that need it. With products that encourage collaboration across the enterprise, Informatica reduces costs, speeds time to results, and scales to handle data integration projects of any size or complexity.

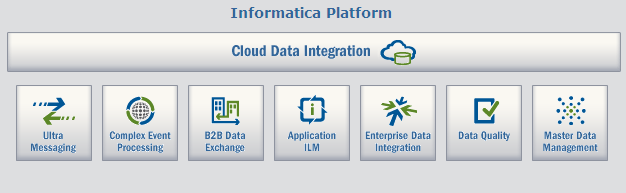


Fig 3.1 Informatica Products

**3.3 Data Warehousing Concepts**

**3.3.1 Introduction**

“A Data Warehouse is a subject oriented, integrated, non-volatile, and time variant collection of data in support of management’s decisions.”

The data in the data warehouse is:

* **Subject-Oriented Data:**The data in the data warehouse is organized so that all the data elements relating to the same real-world event or object are linked together.
* **Non-volatile:** Data in the data warehouse are never over-written or deleted-once committed; the data are static, read-only, and retained for future reporting.
* **Integrated:** The data warehouse contains data from most or all of an organization's operational systems and these data are made consistent.
* **Time Variant:**In order to discover trends in business, analysts need large amounts of data. A data warehouse's focus on change over time is what is meant by the term time variant.

**3.3.2 Components of a Data Warehouse**

There are four separate and distinct components to be considered as we explore the data warehouse environment-operational source systems, data staging area, data presentation area, and data access tools.

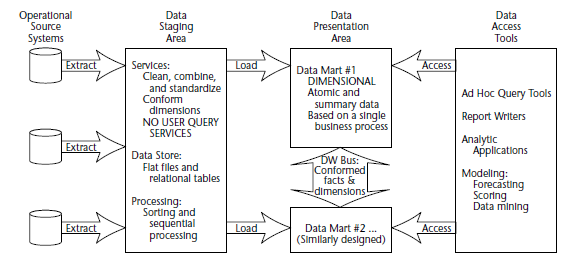
****

Fig 3.2 Components of a data warehouse

**Operational Source Systems**

These are the operational systems of record that capture the transactions of the business. We make the strong assumption that source systems are not queried in the broad and unexpected ways that data warehouses typically are queried.

**Data Staging Area**

The Data Staging Areais a temporary location where data from source systems is copied.  A staging area is mainly required in a Data Warehousing Architecture for timing reasons.

Due to varying business cycles, data processing cycles, hardware and network resource limitations and geographical factors, it is not feasible to extract all the data from all Operational databases at exactly the same time. Hence data staging area is used.

**Data Presentation**

The data presentation area is where data is organized, stored, and made available for direct querying by users, report writers, and other analytical applications. Since the backroom staging area is off-limits, the presentation area *is* the data warehouse as far as the business community is concerned.

If the presentation area is based on a relational database, then these dimensionally modelled tables are referred to as star schemas*.* If the presentation area is based on multidimensional database or online analytic processing (OLAP) technology, then the data is stored in cubes*.*

**Data Access Tools**

The final major component of the data warehouse environment is the data accesstool(s*)*. We use the term tool loosely to refer to the variety of capabilities that can be provided to business users to leverage the presentation area for analytic decision making.

A data access tool can be as simple as an ad hoc query tool or as complex as a sophisticated data mining or modelling application.

**3.3.3 Dimensional Data Modelling**

It is a logical design technique to structure the business dimensions and the metrics that are analysed along these dimensions.We use star schema for modelling the data.

**Star Schema**

In the star schema design, a single object (the fact table) sits in the middle and is radially connected to other surrounding objects (dimension tables) like a star. Each dimension is represented as a single table. The primary key in each dimension table is related to a foreign key in the fact table. All measures in the fact table are related to all the dimensions that fact table is related to. In other words, they all have the same level of granularity.

**3.3.4Bill Inmon vs. Ralph Kimball**

Bill Inmon's paradigm: Data warehouse is one part of the overall business intelligence system. An enterprise has one data warehouse, and data marts source their information from the data warehouse. In the data warehouse, information is stored in 3rd normal form.

Ralph Kimball's paradigm: Data warehouse is the conglomerate of all data marts within the enterprise. Information is always stored in the dimensional model.

There is no right or wrong between these two ideas, as they represent different data warehousing philosophies. In reality, the data warehouse in most enterprises is closer to Ralph Kimball's idea. This is because most data warehouses started out as a departmental effort, and hence they originated as a data mart. Only when more data marts are built later do they evolve into a data warehouse.

**3.4Power Center-ETL Tool**

PowerCenter provides an environment that allows us to load data into a centralized location, such as a data warehouse or operational data store (ODS). We can extract data from multiple sources, transform the data according to business logic we build in the client application, and load the transformed data into file and relational targets.

PowerCenter includes the following components:

**Informatica Domain.** The Informatica domain is the primary unit for management and administration within PowerCenter. The domain supports PowerCenter and Informatica application services. Some of the PowerCenter application services include the PowerCenter Repository Service, PowerCenter Integration Service.

**PowerCenter Repository.** The PowerCenter repository resides in a relational database. The repository stores information required to extract, transform, and load data. It also stores administrative information such as permissions and privileges for users and groups that have access to the repository.

We can view repository metadata in the Repository Manager. We can also create a Reporting Service in the Administration Console and run the PowerCenter Repository Reports to view repository metadata. **Administration Console.** Informatica Administration console is a web application that is used to administer the Informatica domain and PowerCenter security.

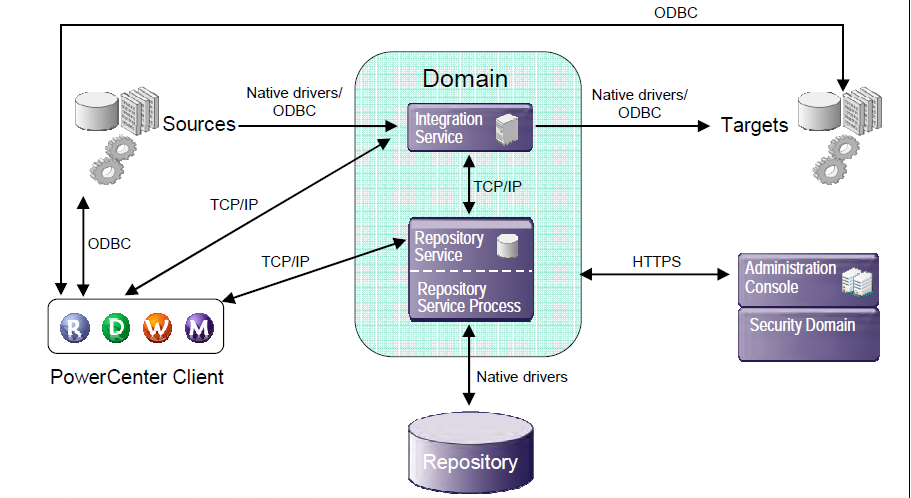


Fig 3.3 PowerCenter Architecture

**3.4.1 PowerCenter Client**

The PowerCenter Client application consists of the tools to manage the repository and to design mappings, mapplets, and sessions to load the data. The PowerCenter Client application has the following tools:

**Designer** Use the Designer to create mappings that contain transformation instructions for the Integration Service.

**Repository Manager** Use the Repository Manager to assign permissions to users and groups and manage folders.

**Workflow Manager** Use the Workflow Manager to create, schedule, and run workflows. A workflow is a set of instructions that describes how and when to run tasks related to extracting, transforming, and loading data.

**Workflow Monitor** Use the Workflow Monitor to monitor scheduled and running workflows for each Integration Service

**3.4.2Transformations Overview**

A transformation is a repository object that generates, modifies, or passes data. The Designer provides a set of transformations that perform specific functions

The transformations which we have used in this project are

* **Expression Transformation**

Transformation type: Passive, Connected

We use the Expression transformation to calculate values in a single row. We can also use the Expression transformation to test conditional statements before we pass the results to a target or other transformations.

* **Filter Transformation**

Transformation type: Active, Connected

We use the Filter transformation to filter out rows in a mapping. The Filter transformation allows rows that meet the specified filter condition to pass through. It drops rows that do not meet the condition. We can filter data based on one or more conditions.

* **Lookup Transformation**

Transformation type: Passive/Active, Connected/Unconnected

Use a Lookup transformation in a mapping to look up data in a flat file, relational table, view, or synonym.The Integration Service queries the lookup source based on the lookup ports in the transformation and a lookup condition. The Lookup transformation returns the result of the lookup to the target or another transformation. We can configure the Lookup transformation to return a single row or multiple rows.

* **Router Transformation**

Transformation type: Active, Connected

A Router transformation is similar to a Filter transformation because both transformations allow us to use a condition to test data. A Filter transformation tests data for one condition and drops the rows of data that do not meet the condition. However, a Router transformation tests data for one or more conditions and gives us the option to route rows of data that do not meet any of the conditions to a default output group.

* **Sequence Generator Transformation**

Transformation type: Passive, Connected

The Sequence Generator transformation generates numeric values. Use the Sequence Generator to create unique primary key values, replace missing primary keys, or cycle through a sequential range of numbers.

* **Update Strategy Transformation**

Transformation type: Active, Connected

Used to flag the incoming rows for insert, update, or delete based on business logic.

**3.5MICROSTRATEGY**

**3.5.1 INTRODUCTION**

MicroStrategy is a global leader in Business Intelligence**.** This software is used for unlimited data analysis, reports and dashboards, mobile BI, data mining, forecasting, operations management, and executive decision making.

**3.5.2 MicroStrategy Architecture**

The MicroStrategy platform components work together to provide an analysis and reporting environment to the user community, as shown in the following diagram.



Fig 3.4MicroStrategy Architecture

**MicroStrategy Intelligence Server**

Report Services is an available extension to MicroStrategy Intelligence Server that delivers the most flexible report layout, with drag-and-drop simplicity, and provides comprehensive formatting capabilities to MicroStrategy Desktop users.

**MicroStrategy project**

A projectis where we build and store all schema objects and information we need to create application objects such as reports in the MicroStrategy environment, which together provide a flexible reporting environment.

**MicroStrategy Desktop**

MicroStrategy Desktop is the business intelligence software component that provides integrated query and reporting, powerful analytics, and decision support workflow on the personal computing desktop. It provides a wide variety of features for online analysis of data.

**MicroStrategy Web and MicroStrategy Web Universal**

MicroStrategy Web is a powerful and user-friendly environment for interactive analysis. A full set of capabilities for data browsing, drilling, and reporting development enable stream-of-consciousness navigation. Boardroom quality reports can be generated using a wide range of graphing and formatting options.

**MicroStrategy Architect**

MicroStrategy Architect is a rapid development environment for business intelligence systems. The information-mapping module separates underlying information structures from applications, providing flexibility.

**3.6 Data Mining Concepts**

In today’s world, the need to understand large, complex, information-rich data sets is common tovirtually all fields of business, science, and engineering. Theability to extract useful knowledge hidden in these data and to act on that knowledgeis becoming increasingly important in today's competitive world. The entireprocess of applying a computer-based methodology, including new techniques, fordiscovering knowledge from data is called data mining.

In practice, the two primary goals of data mining tend to be prediction anddescription. Prediction involves using some variables or fields in the data set topredict unknown or future values of other variables of interest. Description, on theother hand, focuses on finding patterns describing the data that can be interpretedby humans. Therefore, it is possible to put data-mining activities into one of twocategories:

1)Descriptive data mining2)Predictive data mining

In descriptive data mining we are concentrating on mining of frequent itemsets and in predictive data mining we are implementing classification and prediction using Bayesian classification.

**3.6.1 Frequent itemset mining using FP-Growth algorithm**

Mining frequent patterns in transaction databases, time-series databases, and many other kinds of databases are extremely popular in data mining.Previously people used to adopt an *Apriori*-like approach, which is based on the *anti-monotone Apriori heuristic-if any length k pattern is not frequent in the database, its length* (*k* + 1) *super-pattern can never be frequent*

The *Apriori*heuristic achieves good performance gained by (possibly significantly) reducing the size of candidate sets. However, in situations with a large number of frequent patterns, long patterns, or quite low minimum support thresholds, an *Apriori*-like algorithm suffers from the following two nontrivial costs:

* It is costly to handle a huge number of candidate sets.
* It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

For the above reasons, we implement the FP-growth method. First, a novel, compact data structure, called *frequent-pattern tree*, or FP-tree in short, is constructed, which is extended prefix-tree structure storing crucial, quantitative information about frequent patterns.

Second, an FP-tree-based pattern-fragment growth mining method is developed, which starts from a frequent length-1 pattern (as an initial *suffix pattern*), examines only its *conditional-pattern base* (a “sub-database” which consists of the set of frequent items co-occurring with the suffix pattern), constructs its (*conditional*) FP-tree, and performs mining recursively with such a tree. The pattern growth is achieved via concatenation of the suffix pattern with the new ones generated from a conditional FP-tree.

Third, the search technique employed in mining is a partitioning-based, divide-and conquers method rather than Apriori-like level-wise generation of the combinations of frequent itemsets.

**3.6.2 Naïve-Bayesian Classification**

Bayesian classifiers are statistical classifiers. They can predictclass membership probabilities, such as the probability that a given tuple belongs toa particular class.

Naïve Bayesian classifiers assume that the effect of an attribute value on a given classis independent of the values of the other attributes. This assumption is called class conditionalindependence. It is made to simplify the computations involved and, in this sense,is considered “naïve.”

The idea behind is for the training set learner to “learn” from a set of labeled examples in the training set so that it can test set identify unlabeled examples in the test set with the highest possible accuracy.

**Chapter 4**

**Project Requirement Definition**

**4.1 Introduction**

The project requirement definition phase is based on the data in the problem definition phase and the knowledge gained in the literature survey phase. This phase involves defining thebusiness scope and functional requirement of the project.

**4.2 Define Business Scope**

The business scope forms the boundary that defines where the project begins and ends.We have various source tables containing historic information on sales order lines in the oracle apps database. We use this data to populate the Staging area and then the presentation area of the data mart respectively applying suitable transformations on the data.

**4.2.1 Identify Source Data Systems**

We have received the xml documentation of the oracle apps source system which houses all of our source data. As Oracle Apps would be standard database housing data for many enterprise wide applications, we have recognized where exactly the tables reside under which ownership.

The different source tables are located in the Oracle Apps database and their schema owners are APPS\AR, APPS\ONT, APPS\INV and APPS\QP.

**4.2.2 Determine Target Requirements**

We have identified what columns have to present in the target tables and also the transformations required extracting the target data from the source which are depicted in the mapping specifications.

**4.3 Define Functional Requirements**

We are generating many BI reports and dashboard using MicroStrategy desktop and Web as the project deliverables. The functional requirements of those reports are as follows

* Report 1:- The report should describe the number of orders, the total ordered amount and the percentage difference between them for any specified year, month and customer.
* Report 2:- The report should describe the number of bookings, the total booked amount and the percentage difference between them for any specified year, month and customer.
* Report 3:- The report should describe the number of orders, number of bookings and the orders to bookings closure rate for any specified year and month.
* Report 4:- The report should describe the product type, unit of measure, number of orders, and number of unique customers, top customer and rank by sales volume for any specified product.

The data mining applications result in the display of the frequent itemsets depending on the input minimum support count and the display of the accuracy percentages or the predictions of the Naïve-Bayesian classifier.

**Chapter 5**

**System requirements specification**

**5.1 Introduction**

The System requirements phase takes in to account the requirements of the project defined in the previous phase so as to define the system requirements. During this Phase of the project, the technical requirements are defined, the project infrastructure is developed and the development standards and strategies are defined.

**5.2 Design Development Architecture**

The development environmentwould encompass the following

* Informatica power center client 9.0.1 for the extraction, transformation and loading of data from the different tables of the Oracle apps to the data warehouse environment.
* Informatica power center server hosting the integration service to accept requests from the power center clients.
* Design and Build reports for Business users using MicroStrategy Desktop client tool to report metrics and attributes above and facilitate analysis across multiple dimensions
* MicroStrategy Intelligence Server must be running for users to get information from your data warehouse using MicroStrategy clients such as MicroStrategy Web or Desktop.
* Java SDK 1.6 for coding the implementation of the data mining application.
* Eclipse is an integrated development environment (IDE) and an extensible plug-in system which is used to develop applications in Java.
* Oracle SQL Developer is a fully supported graphical tool for database development which can be used to browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements.

**5.3 Procure Hardware and Software**

**5.3.1 HARDWARE USED**

CPU: Dual Core 2

RAM: 4GB RAM

Hard Disk: 300 GB Hard Disk

**5.3.2 SOFTWARE USED**

Data Mart Construction: Informatica 9.0.1

Business Intelligence: MicroStrategy 9

Database: Oracle 11g

Database Development: SQL Developer

Java IDE: Eclipse

Operating Systems: Windows 7, Linux

**Chapter 6**

**Gantt chart**

The project was accomplished in the months of January, February, March, April and May. The Gantt charts for the months are as follows:-

**January**



Fig 6.1 Gantt chart for the month of January

**February**

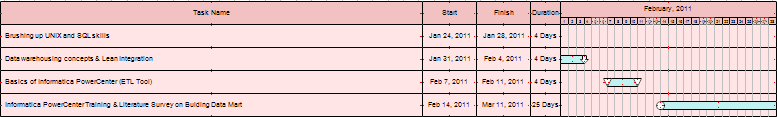
****

Fig 6.2 Gantt chart for the month of February

**March**

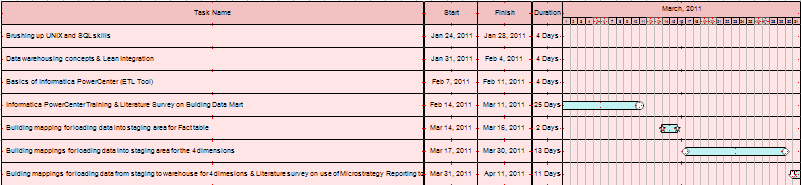


Fig 6.3 Gantt chart for the month of March

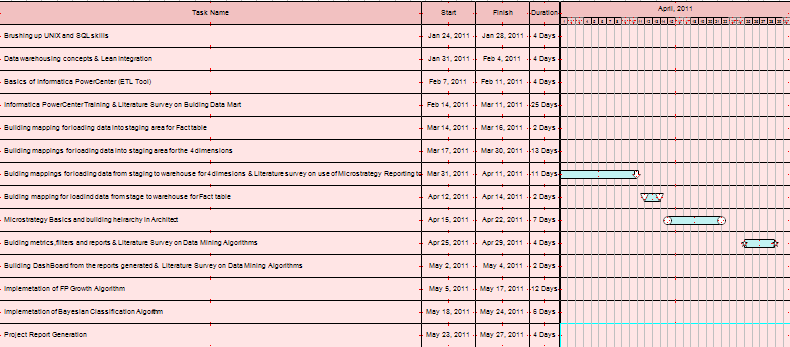
**April**

Fig 6.4 Gantt chart for the month of April

**May**

Fig 6.5 Gantt chart for the month of May

**Chapter 7**

**System Design**

**7.1 Introduction**

In this phase, all data models are developed which lays the foundation for the source systems to be analyzed and physical databases to be designedin the next phases.

**7.2 Develop Data Model**

The data model has the star schema comprising of the dimensions for product, customer, time and a fact table of sales order lines

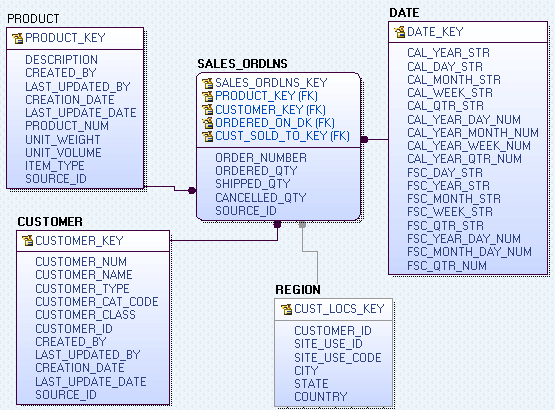


Fig 7.1 Star Schema for Data Mart

**7.3 Develop MicroStrategyattribute hierarchy**

Accordingly we establish the hierarchy among the attributes along the dimensions.

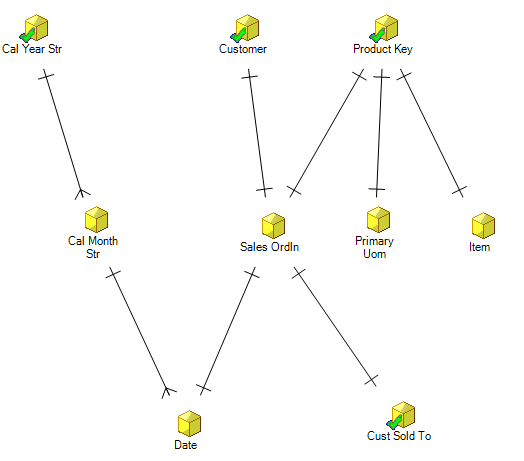
****

Fig 7.2 MicroStrategy Attribute hierarchy

**7.4 Develop design of mining algorithms**

The purpose of this subtask is to develop a design and prototype for the implementation of the two mining algorithms, namely, FP-Growth and Bayesian classification algorithm.

**7.4.1 FP-Growth Algorithm**

The first step in the implementation of the FP-Tree algorithm is to find the frequent items in the transactional database. This step can be implemented by taking the minimum support as an input from the end user and writing in to the database the frequent items and their respective support counts. Frequent items are those items whose support count is greater than the minimum support specified by the end user.

The second step in the FP-Growth algorithm is the process of building the FP-Tree. FP-Tree is normally built by reading each transaction one by one. This results in each node of the FP-Tree having a list of parent-child linkages. This list has to be searched linearly each time a child node has to be inserted to check if the node is already available.

In our implementation the initial FP-tree is built from a main memory representation of the (preprocessed) transaction database as a simple list of integer arrays. This listis sorted lexicographically (thus respecting the order of theitems in the transactions, which reflects their frequency).The sorted list can easily be turned into an FP-tree with astraightforward recursive procedure: at recursion depth k,the k-th item in each transaction is used to split the databaseinto sections, one for each item. For each section a node ofthe FP-tree is created and labeled with the item correspondingto the section. Each section is then processed recursively,split into subsections, a new layer of nodes (one per subsection)is created etc. This implementation results in the elimination of parent-child pointers which is used only in the construction of the tree.

The third step involves mining the FP-Tree for frequent itemsets which is done in a bottom up approach using the combinatorics process.

**7.4.2 Bayesian Classification**

Here, we develop prototypes for the implementation of the Bayesian classification. We are dividing the data in the transactional database in to training and test sets accordingly. 2 /3 of the data are used as the training set and the rest as the test set. The prototypes are:-

The first prototype involves predicting the buying patterns of the given customer in the test set by classifying the training data on the months of the year. It also calculates the accuracy of the prediction of the buying pattern.

The second prototype ranks the products bought by a particular customer and hence predicts which product he may be buying next.

The third prototype predicts whether a given customer in the test set buys a laptop or not depending on the data in the training set. It also involves the calculation of the accuracy of prediction of the Bayesian classification.

**Chapter 8**

**Detailed Design**

**8.1 Introduction**

The detailed design phase is directly based on the system design phase .The detailed design involves the designing of the source to target data flow and the design of the presentation layer for the reports to be developed.

**8.2 Develop Source to target Data flow diagram**

This task provides an idea on how the data should flow from the various sources to the targets specified. This task provides us with an idea of how to build the mappings for the various stages of building the data warehouse.

The mapping specifications for the project are given below:-

**High level process overviewfor m\_load\_Flatfile\_Warehouse\_Date**

Target

Expression transformation

Source Qualifier transformation

Source

Fig 8.1 Mapping of loading date from flat file to warehouse

**High level process overviewfor m\_load\_OrclApps\_Stage\_Customer**

Source

Expression transformation

Source

Target

Source Qualifier transformation

Fig 8.2Mapping of loading customer from Oracle Apps to Staging

**High level process overviewfor m\_load\_OrclApps\_Stage\_Fact**

Source

Target

Expression transformation

Source

Source

Source

Source Qualifier transformation

Fig 8.3 Mapping of loading fact table from Oracle Apps to Staging

**High level process overviewfor m\_load\_OrclApps\_Stage\_Product**

Expression transformation

Source Qualifier transformation

Source

Target

Fig 8.4 Mapping of loading productfrom Oracle Apps to Staging

**High level process overviewfor load\_Stage\_Warehouse\_Customer**

Sequence Generator transformation

Unconnected Lookup transformation

Lookup transformation

Target

Router transformation

Update Strategy transformation

Expression transformation

Source Qualifier transformation

Source

Target

Fig 8.5 Mapping of loading customer from Staging to Warehouse

**High level process overviewfor m\_load\_Stage\_Warehouse\_Product**

Sequence Generator transformation

Unconnected Lookup transformation

Target

Lookup transformation

Router transformation

Update Strategy transformation

Expression transformation

Source Qualifier transformation

Source

Target

Fig 8.6 Mapping of loading product from Staging to Warehouse

**High level process overviewfor m\_load\_OrclApps\_Stage\_Region**

Expression transformation

Aggregator transformation

Source Qualifier transformation

Sources

Target

Fig 8.7 Mapping of loading region fromOracle Apps to Staging

**High level process overviewfor m\_load\_Stage\_Warehouse\_Region**

Sequence Generator transformation

Unconnected Lookup transformation

Lookup transformation

Update Strategy transformation

Source Qualifier transformation

Router transformation

Source

Target

Target

Expression transformation

Fig 8.8 Mapping of loading region from Staging to Warehouse

**High level process overviewfor m\_load\_Stage\_Warehouse\_Fact**

Lookup transformation1

Unconnected Lookup transformation1

Lookup transformation2

Sequence Generator transformation

Unconnected Lookup transformation2

Target

Lookup transformation3

Update Strategy transformation

Router transformation

Source Qualifier transformation

Expression transformation1

Source

Target

Expression transformation2

Fig 8.9 Mapping of loading Fact from

Expression transformation3

Staging to Warehouse

**8.3 Design Presentation Layer**

The objective of this task is to design a presentation layer for the end-user community. The developers will use the design that results from this task and its associated subtasks in the Implementation phase to build the presentation layer.

We are building the reports using microstrategy 9.0. These reports can be built and accessed by the end user. So the presentation layer constitutes the reports built using MicroStrategy desktop and the dashboard built using the MicroStrategy Web.

**8.3.1 Design Presentation Layer Prototype**

The purpose of this subtask is to develop a prototype of the end-user presentation layer "application".

The prototypes of the reports are as follows

**Report: Number of Orders**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | No Of Orders | Total Ordered | % Avg Diff |
| Year | Month | Customer |  |  |  |

Table 8.1 Prototype of Number of Orders Report

**Report: Number of Bookings**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | No Of Bookings | Total Bookings | % Avg Diff |
| Year | Month | Customer |  |  |  |

Table 8.2 Prototype of Number of Bookings Report

**Report: Orders to Bookings Close Rate**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | No of Orders | No Of Bookings | % Closure rate |
| Year | Month |  |  |  |

Table 8.3 Prototype of Orders to Bookings Close Rate Report

**Report: Product Metric**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Type | Unit Of Measure | No Of Orders | No Of Unique Customers | Top Customer | Rank By Sales Volume |
| Product |  |  |  |  |  |  |

Table 8.4 Prototype of Product Metric Report

**8.3.2 Design algorithms for data mining applications**

The data mining applications consists of the following two algorithms:-

**FP-Growth Algorithm**

**Algorithm (*ComputeItemFrequency***)**.**

**Input**: Minimum support threshold.

**Output**: Frequent-patterns of the products

Procedure ***ComputeItemFrequency (*conn, minimumSupport)**

**{**

1. Use the following query

select product\_key , count (distinct(order\_number)) from wh\_sales\_ordlns group by product\_key having count(distinct(order\_number)) >="+*minimumSupport*+" order by count(sales\_ordln\_key) desc,product\_key asc

1. Process each row of the resultset above and insert into freqcnt table values of product key and the frequency count

**}**

**Algorithm** (***ReadTransaction***)**.**

**Input**: A transaction database

**Output**: Store frequent items in a 2D transaction array

Procedure**CreateFPTree.*ReadTransaction*(conn)**

**{**

1. Set no of rows of transaction array =total no of orders

select count(distinct(order\_number)) from freqcnt, wh\_sales\_ordlns where wh\_sales\_ordlns.product\_key = freqcnt.item

1. Set no of columns of transaction array=max no of products ordered

select max(count(product\_key)) from freqcnt,wh\_sales\_ordlns where wh\_sales\_ordlns.product\_key = freqcnt.item group by order\_number

1. Join source table and freqcnt table so as to process the records which are frequent and neglect others

select distinct wh\_sales\_ordlns.order\_number, freqcnt.item,freqcnt.cnt from freqcnt,wh\_sales\_ordlns where wh\_sales\_ordlns.product\_key = freqcnt.item order by order\_number asc,freqcnt.cnt desc, item asc

Process each row as:

when previous order\_no not equal to the present then increment the rowcountandinsert the item else increment the col count and insert the item

}

**Algorithm**(***ComputeTree***)**.**

**Input**: 2D transaction array of frequent items

**Output**: Frequent-patterns of the products

Procedure **CreateFPTree.*ComputeTree* ()**

**{**

1. Initialize arrInd array of indices representing the indexes of orders
2. Create root node

parent = FP1.CreateNode(-1,0,-1);

1. Call Recursive function to add remaining nodes to the FP-Tree.

addNode(arrInd , recCount , parent );

1. Construct the Conditional Pattern Base for each frequent item and call the combinatorics functions to produce frequent patterns

MineFPTree.FreqPattern( FP1.getMap() , items,FP1 , minimumSupport )

**}**

**Algorithm** (***CreateNode***)**.**

**Input**: product\_key,frequency,parent

**Output**: Node created with appropriate hash populated

Procedure **CreateNode(int name , int frequency , int parent )**

**{**

1. Initialize the array element using

FPNodes[treeCount] = **new**FPTreeNode(name,frequency,parent);

1. **If** the hash with the name of the product as key doesn’t exist

**Then**Create a linked list and add the index of array at which the product is inserted

**Else**Get the linked list and add the index of array at which the product is inserted

**}**

**Algorithm Recursive** (***addNode***)**.**

**Input**: arrInd - list of indices to be considered for extracting information from trans[][] matrix.

recCount - The Count of the recursive call which is used to extract information from trans[][] matrix.

parent - Parent of the node to be created **Output**: Node created with appropriate hash populated

**Output**: Fp Tree created

Procedure **addNode(int[] arrInd, intrecCount, int parent )**

**{**

1. **Loop** through array of indices passed and increment integer NoOfTrans if array element is non zero
2. **Base Condition**: **If**NoOfTrans to be processed is 0 return
3. **Loop** through array of indices passed and populate arrTransCol : array to store the product\_keys passed to the function by the current recursion then increment the recount
4. Sort the array based on product\_key so that we have same product\_keystogether,adjust the arrRowNum indices accordingly

using**HeapSort.*startSort*(arrTransCol, arrRowNum)**

1. now process elements of sorted arrTransCol

**if** current product\_key is same as the previous

**then**increase the count of occurence of the product in the present column being processedalso record the index at which the product occurs

**else** (i.e. when different create the FP node for the product)

create a node ***FP1*.CreateNode(prevName, count, parent)**

store the indices where the product appears in current column in**tempArray**make a recursive call***addNode*( tempArray , recCount , RecParent )**

**}**

**Algorithm** (***FreqPattern***)**.**

**Input**: map - Specifies the node-links of each node created in the FP-Tree.

items - The list of frequent items.

FP1 The FP - Tree created in the CreateFPTree class.

minimumSup - Specifies the minimum support to be satisfied by the items to be deemed as frequent items.

**Output**: Frequent Patterns of the products

Procedure **FreqPattern(HashMap<Integer , LinkedList<Integer>> map , int[] items , FPTree FP1 , intminimumSup )**

**{**

Start mining from less frequent product, **Loop** the items array in the reverse order

**{**

1. create hash map tempMap for conditional pattern base
2. get the linked list of the item being processed ll = map.get(items[i]);
3. **Loop** through the indices of fpnode array using the elements of the linked list

**{**

1. nodeIndex = ll.get( j );//get the index in fpnode array at which the product occurs
2. base = FP1.FPNodes[nodeIndex].name;
3. frequency = FP1.FPNodes[nodeIndex].frequency;
4. supportCount = supportCount + frequency;
5. nodeIndex = FP1.FPNodes[nodeIndex].parent; **//move up to parent**
6. //move till we reach root

**while**( FP1.FPNodes[nodeIndex].parent != -1 )

**{**

populate product's prefix path to root with frequency of the base through the use of hash map

**if**(tempMap.get( FP1.FPNodes[nodeIndex].name ) == **null)**{

tempMap.put( FP1.FPNodes[nodeIndex].name , frequency );}

**Else**{

tempFrequency =(Integer) tempMap.get( FP1.FPNodes[nodeIndex].name)

tempFrequency = tempFrequency + frequency;//**combine common paths**

tempMap.put( FP1.FPNodes[nodeIndex].name , tempFrequency );}

nodeIndex = FP1.FPNodes[nodeIndex].parent; **//move up to parent**

}**while ends**

**}loop ends**

1. Iterate over the keys of **tempMap** (prefix path)

**if** the frequency is greater than or equal to minimum support

**then** add to the condition base string

1. generate combinations of the prefix pattern

*Combinatorics*( Integer.*toString*( base ).trim() , arrCondition , tempMap , supportCount , conn )  **}**

**}**

**Algorithm** (***Combinatorics***)**.**

**Input**: base - The item for which the Conditional Pattern Base is created.

conditionBase - The Conditional Pattern base used for generating different combinations.

tempMap The - Temporary HashMap created to store the support counts of each item in the Conditional Pattern Base.

supportCount - The support count of the base.

**Output**: Node created with appropriate hash populated

Procedure **Combinatorics( String base , String[] conditionBase , HashMap<Integer, Integer>tempMap , intsupportCount , Connection conn )**

**{**

1. Use *generate*( conditionBase ) to generate all combinations of the product key in the pattern base
2. get the descriptions of the product\_key from the database and for each set of patterns obtain the min of occurrence frequency and report as the support count of the frequent item set

**}**

**Algorithm Recursive** (***generate***)**.**

**Input**: string containing the conditional pattern base

**Output**: generated combination populated in a hash set

Procedure **generate( String[] words ){**

1. Add words to combination results set*combinations*.add( result );
2. **Base Condition**:

**If** the words has only one word we break the recursion

1. **Iterate** through every word of the words , in each iteration remove the string at the current position a call this method with remaining

**}**

**Bayesian Classification**

**Algorithm** (computeClassProb)**.**

**Input**: Training database

**Output**: Prior Probabilites of the attributes with respect to the classes buys\_laptop or not

Procedure **computeClassProb**()

{

1. Use the query to import the data

selectproduct\_desc,customer\_type,industrial\_sector,region from training\_bayes\_table

1. Read each record
2. If prodDesccription contains substring "Laptop"

Thenindex = 0; //buys laptop

Elseindex = 1; //doesn't buy laptop

1. record the frequency count of customers buying laptop into appropriate classestotal[ index ] ++
2. record the frequency count of customer\_type(organization/person)

buying laptop into appropriate classes

1. record the frequency count of region(US/non\_US) buying laptop into appropriate classes
2. record the frequency count of industrialSector(financial/technology/services) buying laptop into appropriate classes
3. Compute totalOrders - total no of orders for buys\_laptop=yes and buys\_laptop=no

}

**Algorithm** (ClassifyTestSet)**.**

**Input**: Test database

**Output**: Based Posterior Probabilites of the attributes with respect to the classesbuys\_laptop or not data is classified and accuracy is estimated.

Procedure ClassifyTestSet()

{

1. Use the query to import the data

selectproduct\_desc,customer\_type,industrial\_sector,region from test\_bayes\_table

1. Read each record .Iterate for each class (index-0,1)
2. if the test data's cust\_type is organization then get the approriate frequency count for buys\_laptop={yes,no}

else if the test data's cust\_type is person then get the approriate frequency count for buys\_laptop={yes,no}

1. if the test data's region is US then get the approriate frequency count for buys\_laptop={yes,no}

else if the test data's region is NON\_US then get the approriate frequency count for buys\_laptop={yes,no}

1. if the test data's Industrial Sector is Financial then get the approriate frequency count for buys\_laptop={yes,no}

else if the test data's Industrial Sector is Technology then get the approriate frequency count for buys\_laptop={yes,no}

else if test data's Industrial Sector is Technology then get the approriate frequency count for buys\_laptop={yes,no}

else (in case of Services)get the approriate frequency count for buys\_laptop={yes,no}

1. compute bayes formula for classes buys\_laptop={yes,no}
2. compare which either classes have highest probability and predict buys\_laptop behavior

if(result[0] > result[1])res = 0; //predicted buys\_laptop=yes

elseres = 1; //predicted buys\_laptop=no

1. Accuracy computation

if(rs.getString(1).contains("Laptop"))test = 0

else test = 1; //actual buys\_laptop=no

if(res == test)correct++; elsewrong++;

3.Compute Accuracy as correct/(correct+wrong)\*100)

}

**Chapter 9**

**Implementation**

**9.1 Introduction**

This Phase involves implementing the processes defined in the design phases of the project. Implementation phase involves designing and developing the data integration processes.

**9.2 Design and Develop Data Integration Processes**

This task includes the necessary steps for developing a comprehensive design plan for the data integration process, which incorporates high-level standards such as data load schedule and dependency, initial and ongoing load plan and specific details and snapshots of individual mappings.

**9.2.1 Design High Level Load Process**

This subtask incorporates two steps as follows:

**Data Load Schedule and Dependency**

Data Warehouse contains the staging area and the presentation area. The tables in the staging area reflect the tables present in the presentation area. The periodic snapshots of the operational data source are taken and the data warehouse is updated. As the design of the data warehouse involves both facts and dimensions, snapshots of both of them are taken and updated in a periodic manner.

During the updating of the data warehouse, first data is loaded into the staging area. In the staging area, the fact table data is loaded first and then the dimensions data are loaded. This is done in this way to avoid any discrepancy in the data of the staging area. For example, if the dimensions data are loaded first and then the fact data is loaded and if the fact data involves a new tuple which corresponds to a new dimension data which is not loaded, it leads to referential integrity constraint error.

If it is done the other way round, even though a dimension contains some new data, it will not lead to errors.

When data is loaded form the staging area to the presentation area, dimensions data are loaded first and then the fact data is loaded. This is done in this manner as the process of loading the fact table involves looking up the various dimensions for validation of the referential integrity constraint of the composite key of the fact table.

**Create initial and ongoing load plan** In this step, the Data Integration Developer and Business Analyst use information created in the two earlier steps to develop a load plan document; this lists the estimated run times for the batches and sessions required to populate the data warehouse and/or data marts.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dimension/Fact** | **Workflow** | **Number of Rows processed** | **Estimated Duration (in hh:mm:ss)** |
| Customer |  |  | 00:00:15 |
|  | wkf\_STG\_CUSTOMERS | 818 | 00:00:08 |
|  | wkf\_DWH\_CUSTOMERS | 818 | 00:00:07 |
| Product |  |  | 00:00:18 |
|  | wkf\_STG\_PRODUCTS | 24533 | 00:00:08 |
|  | wkf\_DWH\_PRODUCTS | 24533 | 00:00:10 |
| Dates | wkf\_DWH\_DATES | 20005 | 00:00:03 |
| Region | wkf\_STG\_REGION  wkf\_DWH\_REGION | 3128  3128 | 00:00:04  00:00:03 |
| Sales |  |  | 00:01:15 |
|  | wkf\_STG\_FACT | 90668 | 00:00:38 |
|  | wkf\_DWH\_FACT | 90668 | 00:00:37 |
| **Total Estimated Duration** | |  | **00:01:58** |

Table 9.1 Load Window for the Data Mart

**9.2.2 Design Individual Mappings**

Before designing a mapping, it is important to have a clear picture of the end-to-end processes that the data will flow through. Then, design a high-level view of the mapping and document a picture of the process within the mapping, using a textual description to explain exactly what the mapping is supposed to accomplish and the methods or steps it follows to accomplish its goal.

The mapping detailed designs are a crucial input for building the data integration processes, and can also be useful for system and unit testing. The specific details used to build an object are useful for developing the expected results to be used in system testing.

The descriptions of the mappings are given below:-

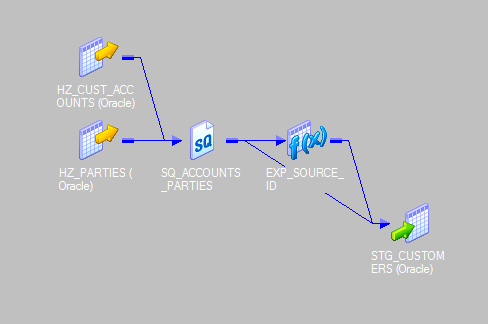
1. **m\_load\_OrclApps\_Stage\_Customer**

This mapping loads customer information from oracle apps to staging area. In this mapping the sources HZ\_CUST\_ACCOUNTS and HZ\_PARTIES are joined in Source Qualifier transformation by join condition

HZ\_CUST\_ACCOUNTS.PARTY\_ID = HZ\_PARTIES.PARTY\_ID

HZ\_CUST\_ACCOUNTS.PARTY\_ID = HZ\_PARTIES.PARTY\_ID.

The expression transformation is used to generate KEY\_ID and SOURCE\_ID. KEY\_ID is generated by converting CUST\_ACCOUNT\_ID from HZ\_CUST\_ACCOUNTS table to string and SOURCE\_ID is generated by assigning the value ‘ORACLE APPS’.

All the other required columns are directly mapped to the target table STG\_CUSTOMERS.

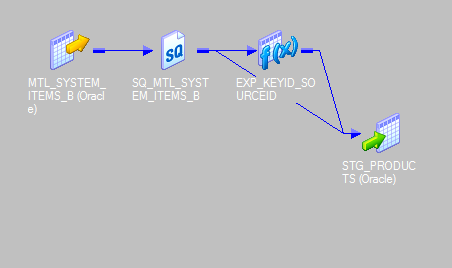
**Fig 9.1 Snapshot of m\_load\_OrclApps\_Stage\_Customer**

1. **m\_load\_OrclApps\_Stage\_Product**

This mapping loads product information from oracle apps to staging area.

The expression transformation is used to generate KEY\_ID and SOURCE\_ID. KEY\_ID is generated by concatenating INVENTORY\_ITEM\_ID and ORGANIZATION\_ID from the table MTL\_SYSTEM\_ITEMS\_B. SOURCE\_ID is generated by assigning the value ‘ORACLE APPS’.

All the other required columns are directly mapped to the target table STG\_PRODUCTS.



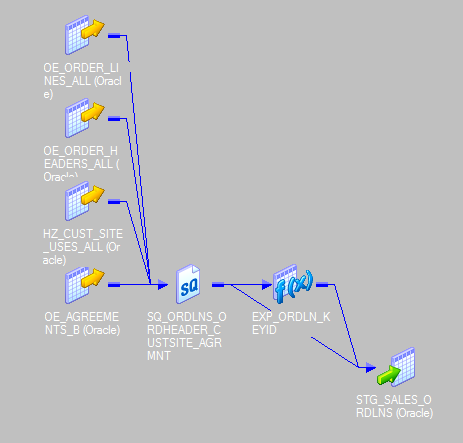
**Fig 9.2 Snapshot of m\_load\_OrclApps\_Stage\_Product**

1. **m\_load\_OrclApps\_Stage\_Fact**

This mapping loads sales fact information from oracle apps to staging area. In this mapping only required records are fetched from the sources OE\_ORDER\_LINES\_ALL,OE\_ORDER\_HEADERS\_ALL, HZ\_CUST\_SITE\_USES\_ALL, OE\_AGRREMENTS\_B using the condition OE\_ORDER\_HEADERS\_ALL.HEADER\_ID=OE\_ORDER\_LINES\_ALL.HEADER\_ID ANDOE\_AGREEMENTS\_B.AGREEMENT\_ID(+)=OE\_ORDER\_HEADERS\_ALL.AGREEMENT\_ID AND

HZ\_CUST\_SITE\_USES\_ALL.SITE\_USE\_ID(+)=OE\_ORDER\_LINES\_ALL.SHIP\_TO\_ORG\_ID in the Source Qualifier transformation.

Using expression transformation KEY\_ID and SOURCE\_IDs are generated. String format of LINE\_ID form the source table OE\_ORDER\_LINES\_ALL is assigned to the port KEY\_ID. SOURCE\_ID is generated by assigning the value ‘ORACLE APPS’ in a new port.

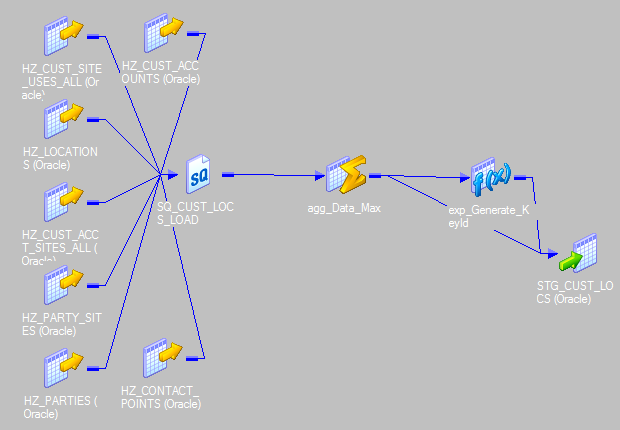


**Fig 9.3 Snapshot of m\_load\_OrclApps\_Stage\_Fact**

1. **m\_load\_OrclApps\_Stage\_Region**

This mapping loads customer location information from Oracle Apps to staging area. Only required information is fetched from the sources HZ\_CUST\_ACCOUNTS, HZ\_CUST\_SITE\_USES\_ALL, HZ\_LOCATIONS, HZ\_CUST\_ACCOUUNT\_SITE\_ALL, HZ\_PARTY\_SITES, HZ\_PARTIES, HZ\_CONTACT\_POINTS to the source qualifier transformation.

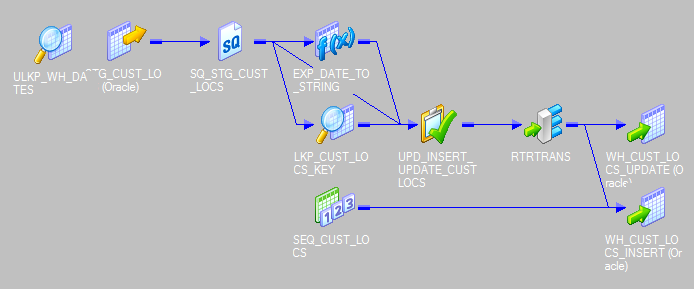
With the help of the expression transformation KEY\_ID and SOURCE\_ID are generated. Finally all the ports are mapped to the Target table STG\_CUST\_LOCS.

****

**Fig 9.4 Snapshot of m\_load\_OrclApps\_Stage\_Region**

1. **m\_load\_Stage\_Warehouse\_Region**

This mapping loads customer location information from staging area to warehouse. Initially the target table WH\_CUST\_LOCS is looked up for the CUST\_LOCS\_KEY through the lookup transformation. If the record is present in the target table then that record is tagged as UPDATE otherwise as INSERT. This tagging process is done by the transformation called update strategy. The records which are tagged as INSERT are routed to the WH\_CUST\_LOCS\_INSERT table and the records tagged as UPDATE to the WH\_CUST\_LOCS\_UPDATE using the router transformation. Expression transformation and unconnected lookup transformation helps in substituting the DATE\_KEY in the dates column of the target table. Sequence generator transformation helps in generating the CUST\_LOCS\_KEY.

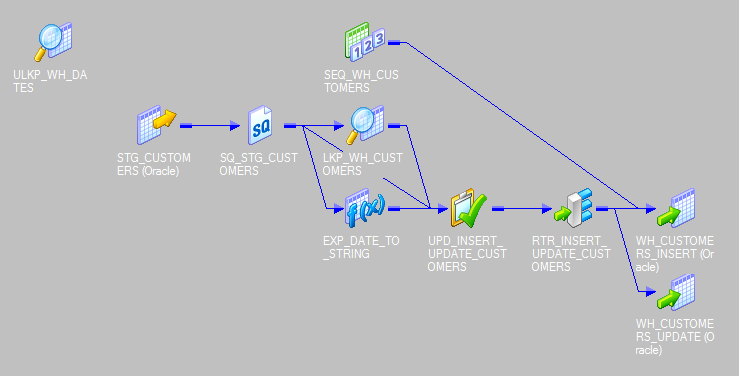
****

**Fig 9.5 Snapshot of m\_load\_Stage\_Warehouse\_Region**

1. **m\_load\_Stage\_Warehouse\_Customer**

This mapping loads customer information from staging to warehouse. In this mapping we will pass all the columns of the source table STG\_CUSTOMERS to the source qualifier transformation. Using the expression transformation and unconnected lookup transformation DATE\_KEY is looked up from the WH\_DATES table by condition CAL\_DAY\_STR=i\_CREATION\_DATE\_STRING and assigned to the CREATION\_DATE column.

Before loading data to the target table WH\_CUSTOMERS, CUSTOMER\_KEY is looked up from the lookup tableWH\_CUSTOMERS through connected lookup. If the CUSTOMER\_KEY is null for a record present in the target that record is flagged as INSERT otherwise the record is flagged as UPDATE using the update strategy transformation. With the help of the router transformation the records which are flagged as INSERT are directed to the target WH\_CUSTOMERS\_INSERT and the records flagged as UPDATE are directed to the target WH\_CUSTOMERS\_UPDATE. To generate the CUSTOMER\_KEY for WH\_CUSTOMERS\_INSERT sequence generator transformation is used.

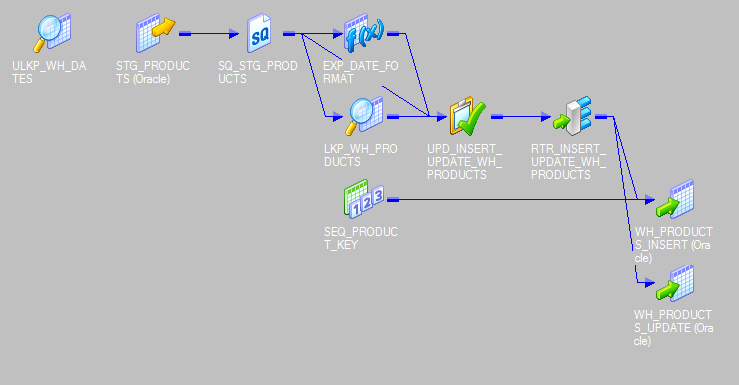


**Fig 9.6 Snapshot of m\_load\_Stage\_Warehouse\_Customer**

1. **m\_load\_Stage\_Warehouse\_Product**

This mapping loads product information from staging to warehouse. In this mapping we will pass all the columns of the source table STG\_PRODUCTS to the source qualifier transformation. Using the expression transformation and unconnected lookup transformation DATE\_KEY is looked up from the WH\_DATES table by condition CAL\_DAY\_STR=i\_CREATION\_DK and assigned to the CREATION\_DATE column.

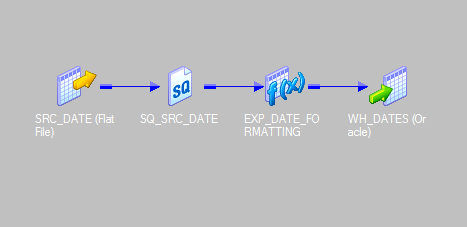
Before loading data to the target table WH\_PRODUCTS, PRODUCT\_KEY is looked up from the lookup tableWH\_ PRODUCTS through connected lookup. If the PRODUCT\_KEY is null for a record present in the target that record is flagged as INSERT otherwise the record is flagged as UPDATE using the update strategy transformation. With the help of the router transformation the records which are flagged as INSERT are directed to the target WH\_ PRODUCTS\_INSERT and the records flagged as UPDATE are directed to the target WH\_ PRODUCTS\_UPDATE. To generate the PRODUCT\_KEY for WH\_ PRODUCTS\_INSERT sequence generator transformation is used.



**Fig 9.7 Snapshot of m\_load\_Stage\_Warehouse\_Product**

1. **m\_load\_Flatfile\_Warehouse\_Date**

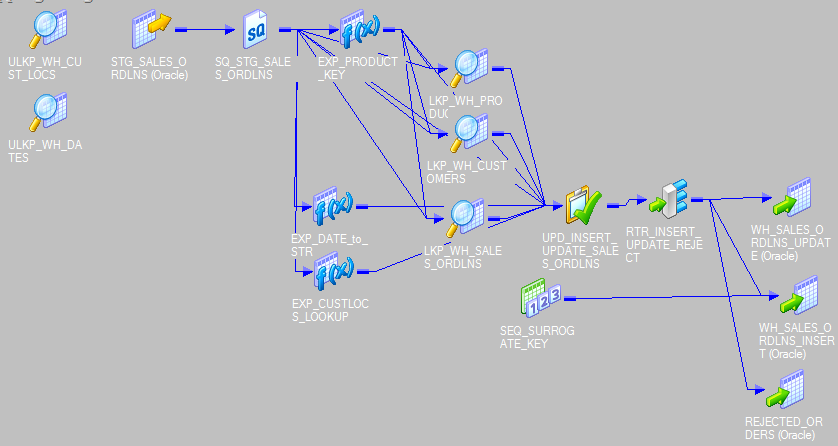
This mapping generates Dates information from a flat file and loads into warehouse. Here the source is a flat file named DUMMY\_DATES.txt. This file contains only 1s. Expression transformation is used to get the required columns for the target WH\_DATES. Required date formats are obtained by applying various date functions on the data of DUMMY\_DATES.txt in expression transformation.



**Fig 9.8 Snapshot of m\_load\_Flatfile\_Warehouse\_Date**

1. **m\_load\_Stage\_Warehouse\_Fact**

This mapping loads sales fact information from staging area to the warehouse. PRODUCT\_KEY, CUSTOMER\_KEY, CUST\_LOCS\_KEY are loaded to the target WH\_SALES\_ORDLNS from the warehouse tables WH\_PRODUCTS, WH\_CUSTOMERS, WH\_CUST\_LOCS respectively using lookup transformation. Date fields in the target are replaced by the corresponding DATE\_KEY by looking to the WH\_DATES table. Before loading data to the target we need to check for the record’s presence in the target table. This is done by the lookup transformation. Then update strategy transformation comes into existence by tagging the record’s presence as UPDATE and absence as INSERT. Once the tagging process completes for a row router transformation will directs these rows the appropriate target. SALES\_ORDLN\_KEY is generated by sequence generator transformation.

****

**Fig 9.9 Snapshot of m\_load\_Stage\_Warehouse\_Fact**

**9.3 Build Presentation Layer**

We are building many reports using the microstrategy 9 tool. The reports can be rerun whenever needed by the end user, that is, whenever data is added to the data mart. The columns in the report can be rolled up or drilled up as the end user wants.

**9.3.1 Develop Presentation Layer**

In this phase we have to decide what measurements must be displayed, what parameters must be included, what format should the report be in and also decide the data limiting criteria for the report.

Before generating these reports the first step is to bring required tables to the MicroStrategy platform. This can be achieved through MicroStrategy Architect. Once the tables came into existence the next step is to identify the attributes and facts in that tables.

Metrics and filters are created to fulfil the requirements. These Metrics are similar to formulas in spread sheet software. Filters are helpful in clarifying large quantities of data and only displaying subsets of that data, so reports show users what they really need to see. These metrics and filters are added to the report editor along with the attributes. Derived metrics can also be part of the reports which can have the scope within the defined report.

These reports are further used to build the dashboards. A dashboard is a display of related sets of data on one screen

The snapshots of the reports we have generated are as follows

**Report 1: Number of Orders**

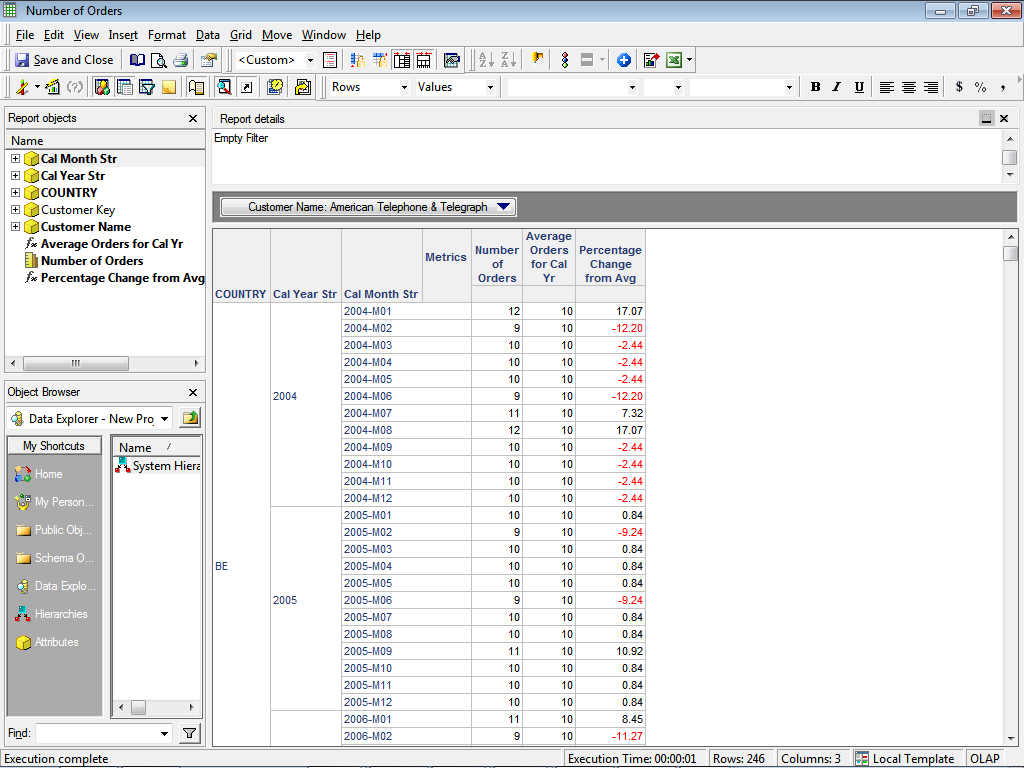
**

Fig 9.10 Snapshot of Number of Orders Report

**Report 2: Number of Bookings**

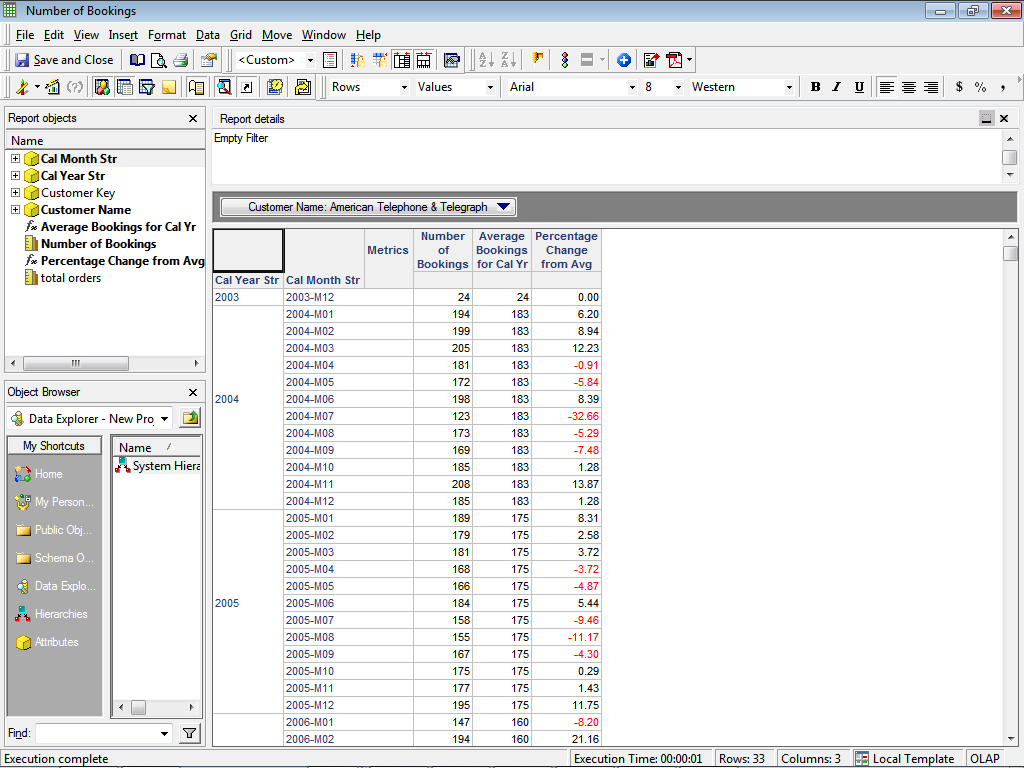


Fig 9.11 Snapshot of Number of Bookings Report

**Report 3: Orders to Bookings Close Rate**

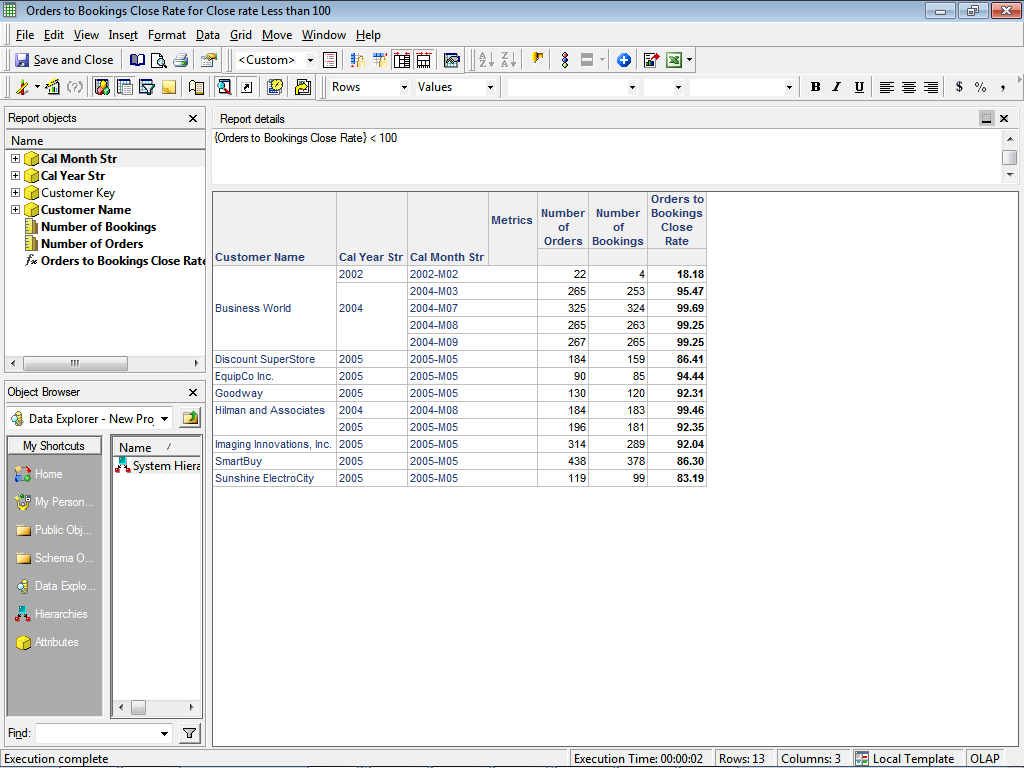


Fig 9.12Snapshot ofOrders to Bookings Close Rate Report

**Report 4: Product Metric**

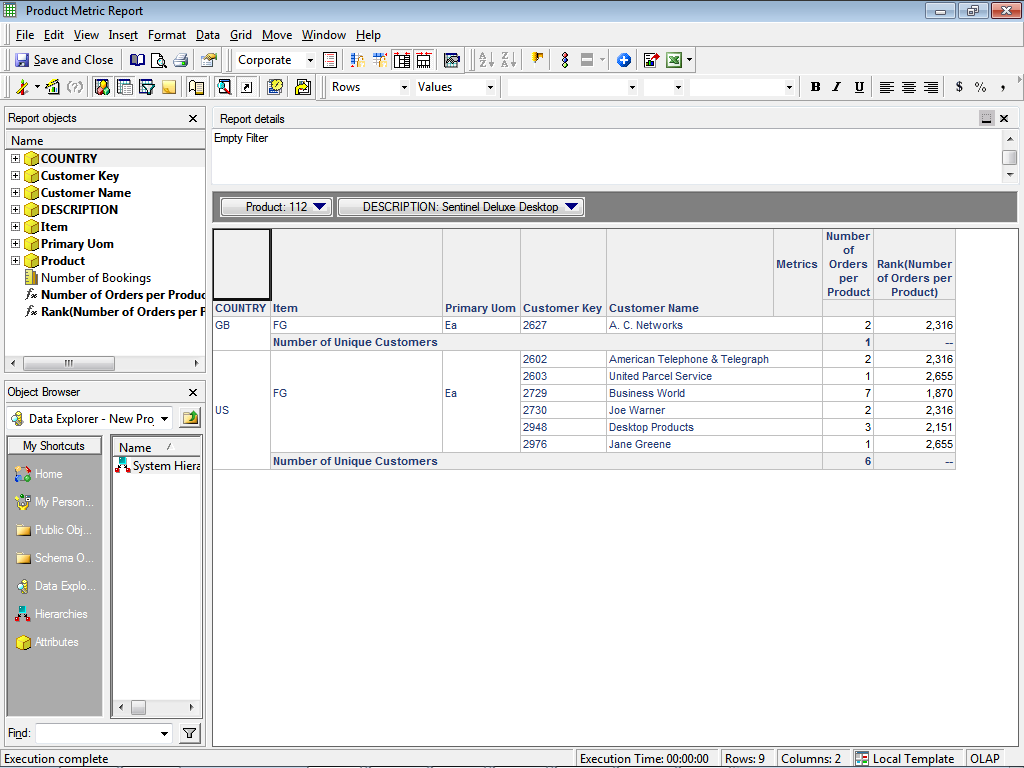


Fig 9.13Snapshot of Product Metric Report

**Dashboard Snapshots**

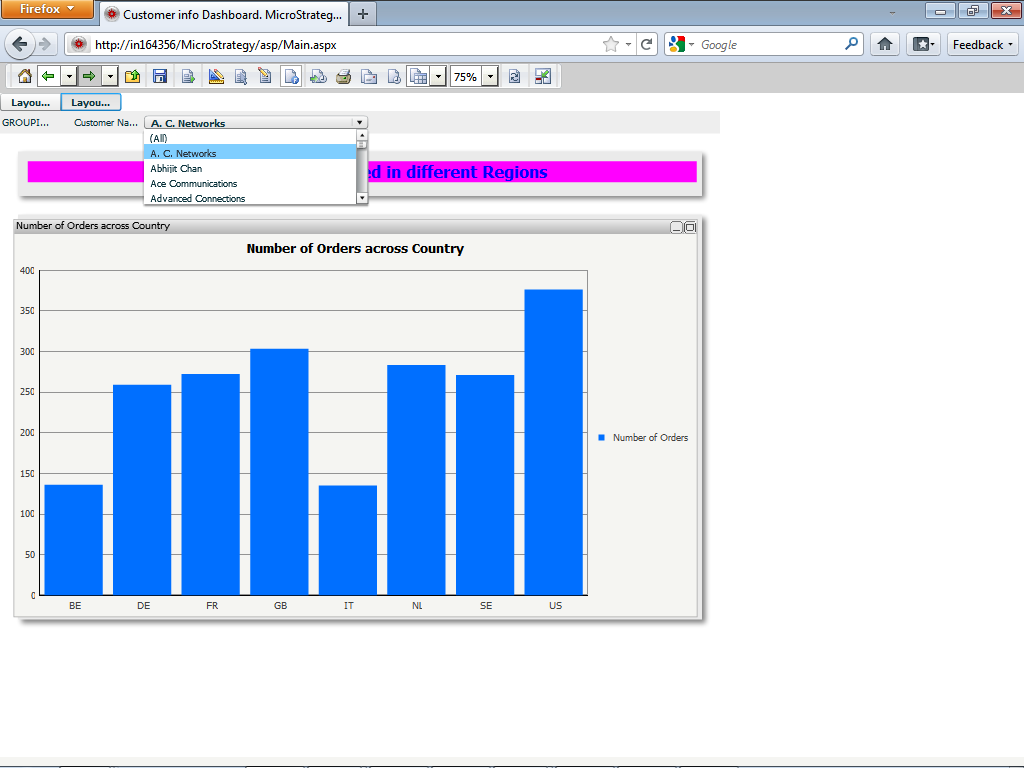


Fig 9.14 Snapshot of Number of Orders Placed in different regions for a customer Dash Board

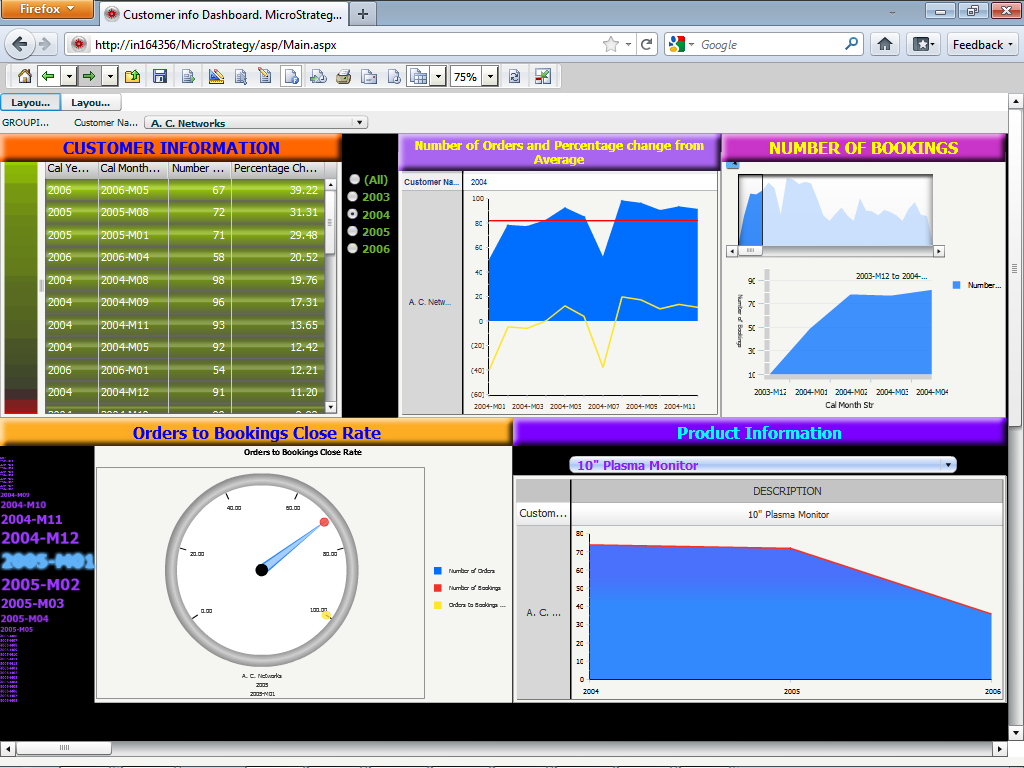


Fig 9.15 Snapshot of Customer Information Dash Board

**Chapter 10**

**Testing**

**10.1 Introduction**

The purpose of testing is to verify that the software has been developed according to the requirements and design specifications. The detailed object level testing plans are continually updated and modified as the development process continues since any change to development work is likely to create a new scenario to test.

**10.2 Perform Unit Test**

The Unit test case for the project is given below

**Unit Test Plan for project**

|  |  |
| --- | --- |
| **PURPOSE/**  **SCOPE** | **Testing Objectives**  Test all logic, data errors, and system errors for all the mappings  in the project.  **In Scope**  m\_load\_Flatfile\_Warehouse\_Date, m\_load\_OrclApps\_Stage\_Customers  m\_load\_OrclApps\_Stage\_Fact, m\_load\_OrclApps\_Stage\_Product  m\_load\_Stage\_Warehouse\_Customer, m\_load\_Stage\_Warehouse\_Product, m\_load\_Stage\_Warehouse\_Fact  m\_load\_OrclApps\_Stage\_Region, m\_load\_Stage\_Warehouse\_Region  m\_load\_Stage\_Warehouse\_Fact |

**TEST CASE 1**

**Mapping Name:** m\_load\_OrclApps\_Stage\_Fact

**Source**: OE\_ORDER\_LINES\_ALL, OE\_ORDER\_HEADERS\_ALL, HZ\_CUST\_SITE\_USES\_ALL, OE\_AGRREMENTS\_B

**Target**: STG\_SALES\_ORDRLNS

**Mapping Name:** m\_load\_OrclApps\_Stage\_Product

**Source:** MTL\_SYSTEM\_ITEMS\_B

**Target:** STG\_PRODUCTS

**Mapping Name:** m\_load\_OrclApps\_Stage\_Customer

**Source:**HZ\_CUST\_ACCOUNTS, HZ\_PARTIES

**Target:** STG\_CUSTOMERS

**Mapping Name:** m\_load\_OrclApps\_Stage\_Region

**Source:**HZ\_CUST\_ACCOUNTS, HZ\_CUST\_SITE\_USES\_ALL, HZ\_LOCATIONS, HZ\_CUST\_ACCOUUNT\_SITE\_ALL, HZ\_PARTY\_SITES, HZ\_PARTIES, HZ\_CONTACT\_POINTS

**Target:** STG\_REGION

**Cycle:** 1

**Description:**Target-Source Port Match

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Refer to the Target-Source matrix document and propagate all the ports from target to source to see if they are correctly mapped.   1. Right click on target table port, click on “Select Link Path” and choose “Back”. 2. Screen shot below shows an example of the same 3. You will see “**Red**” line propagated all the way to beginning of the port. |
|  |  |

Table 10.1 Test Case 1, Cycle 1

**Cycle:**2

**Description**: Check for Join Condition working (If there are any) .Enter a new row in to one of the source table such that it violates the join condition and verify that this row is not in the staging table when the mapping is run.

| **Condition Number** | **Description of Conditions to be Tested** |
| --- | --- |
| 1 | Session inserts records containing typical values to the target table |
| 2 | Verify in the target table that the new row is not added |
| 3 | $$MDATE\_VAR mapping variable is incremented |

Table 10.2 Test Case 1, Cycle 2

**Cycle:**3

**Description**: Initial Run (Inserts)

This will be initial run which helps to test the mapping flow. Records should pass through the mapping flow. This will helps to test the mapping flow, specifically insert flow and Error flow.

This will ensure that session is valid, working in terms of database connectivity, data fetch and data flow.

| **Condition Number** | **Description of Conditions to be Tested** |
| --- | --- |
| 1 | Session inserts records containing typical values to the target table |
| 2 | Session writes some/no records to the Target tables. It may also insert some records into Error Tables. |
| 3 | $$MDATE\_VAR mapping variable is incremented appropriately if the workflow is successful |

Table 10.3 Test Case 1, Cycle 3

**Cycle:**4

**Description**: Re-Run (Filtering)

Use the same input data as for the previous test cycle and note that records are not inserted as the variable $$MDATE\_VAR is incremented. This helps in testing whether the filtering done in the source qualifier work properly. This validates whether the incremental load is working properly.

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session updates an input record containing typical values to the target table |
| 2 | Session writes no records to the error tables |
| 3 | $$MDATE\_VAR mapping variable is incremented appropriately |

Table 10.4 Test Case 1, Cycle 4

**TEST CASE 2**

**Mapping Name:** m\_load\_Stage\_Warehouse\_Customer

**Source:** STG\_CUSTOMERS

**Target:** WH\_CUSTOMERS\_INSERT, WH\_CUSTOMERS\_UPDATE

**Mapping Name:**m\_load\_Stage\_Warehouse\_Product

**Source:** STG\_CUSTOMERS

**Target:** WH\_PRODUCTS\_INSERT, WH\_ PRODUCTS \_UPDATE

**Mapping Name:**m\_load\_Stage\_Warehouse\_Region

**Source:** STG\_REGION

**Target:** WH\_REGION\_INSERT, WH\_ REGION \_UPDATE

**Cycle:**1

**Description:**Target-Source Port Match

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Refer to the Target-Source matrix document and propagate all the ports from target to source to see if they are correctly mapped.   1. Right click on target table port, click on “Select Link Path” and choose “Back”. 2. Screen shot below shows an example of the same. 3. You will see “**Red**” line propagated all the way to beginning of the port. |
|  |  |

Table 10.5 Test Case 2, Cycle 1

**Cycle:**2

**Description:** Initial Run (Inserts)

This will be initial run which helps to test the mapping flow. Records should pass through the mapping flow. This will helps to test the mapping flow, specifically insert flow and Error flow.

This will ensure that session is valid, working in terms of database connectivity, data fetch and data flow.

| **Condition Number** | **Description of Conditions to be Tested** |
| --- | --- |
| 1 | Session inserts records containing typical values to the target table |
| 2 | Session writes some/no records to the Target tables. It may also insert some records into Error Tables. |

Table 10.6 Test Case 2, Cycle 2

**Cycle:**3

**Description:** Re-Run (Updates)

Use the same input data as for the previous test cycle and note that records are updated rather than inserted. This helps in testing the update flow in the mapping.

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session updates input records containing typical values to the target table |
| 2 | Session writes no records to the error tables |
| 3 | Session does not insert a single record |

Table 10.7 Test Case 2, Cycle 3

**Cycle:**4

**Description:** Verifying Lookup and Expression Transformation

Use Typical Input data to verify whether the conversion of creation\_date and last\_update date is happening properly and whether the unconnected lookup returns the correct date key from wh\_dates

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session inserts record to the target table |
| 2 | Verify with the wh\_dates whether the date written in the target table is same as the date in the source table for creation\_date column |
| 3 | Verify with the wh\_dates whether the date written in the target table is same as the date in the source table for last\_update\_date column |

Table 10.8 Test Case 2, Cycle 4

**TEST CASE 3**

Mapping Name: m\_load\_Flatfile\_Warehouse\_Date

**Source: Flatfile named Dummy\_Dates.txt**

**Target: Warehouse Table Wh\_Dates**

**Cycle:1**

Description: Target-Source Port Match

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Refer to the Target-Source matrix document and propagate all the ports from target to source to see if they are correctly mapped.   1. Right click on target table port, click on “Select Link Path” and choose “Back”. 2. Screen shot below shows an example of the same 3. You will see “**Red**” line propagated all the way to beginning of the port. |
|  |  |

Table 10.9 Test Case 3, Cycle 1

**Cycle: 2**

**Description**: Initial Run (Inserts).This will be initial run which helps to test the mapping flow. Records should pass through the mapping flow. This will helps to test the mapping flow, specifically insert flow and Error flow.This will ensure that session is valid, working in terms of database connectivity, data fetch and data flow.

| **Condition Number** | **Description of Conditions to be Tested** |
| --- | --- |
| 1 | Session inserts a record containing typical values to the target table |
| 2 | Session writes some/no records to the Target tables. It may also insert some records into Error Tables. |
| 3 | $$TEMPDATE mapping variable is incremented |

Table 10.10 Test Case 3, Cycle 2

**Cycle: 3**

**Description**: Check for Expression Transformation validity

Use a typical set of dummy data to check whether the data in all the columns of the target table are valid for a specified value of the $$TEMPDATE variable

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session inserts an input record containing typical values to the target table |
| 2 | Check for Validity of the data in all the columns of the table |
| 3 | $$TEMPDATE mapping variable is incremented |

Table 10.11 Test Case 3, Cycle 3

**TEST CASE 4**

**Mapping Name:** m\_load\_Stage\_Warehouse\_Fact

**Source:** STG\_SALES\_ORDLNS

**Target:** WH\_ SALES\_ORDLNS \_INSERT, WH\_ SALES\_ORDLNS \_UPDATE,REJECTED\_ROWS

**Cycle:**1

**Description:**Target-Source Port Match

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Refer to the Target-Source matrix document and propagate all the ports from target to source to see if they are correctly mapped.   1. Right click on target table port, click on “Select Link Path” and choose “Back”. 2. Screen shot below shows an example of the same 3. You will see “**Red**” line propagated all the way to beginning of the port. |
|  |  |

Table 10.12 Test Case 4, Cycle 1

**Cycle:**2

**Description:** Initial Run (Inserts)

This will be initial run which helps to test the mapping flow. Records should pass through the mapping flow. This will helps to test the mapping flow, specifically insert flow and Error flow.

This will ensure that session is valid, working in terms of database connectivity, data fetch and data flow.

| **Condition Number** | **Description of Conditions to be Tested** |
| --- | --- |
| 1 | Session inserts records containing typical values to the target table |
| 2 | Session writes some/no records to the Target tables. It may also insert some records into Error Tables. |

Table 10.13 Test Case 4, Cycle 2

**Cycle:**3

**Description:** Re-Run (Updates)

Use the same input data as for the previous test cycle and note that records are updated rather than inserted. This helps in testing the update flow in the mapping.

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session updates input records containing typical values to the target table |
| 2 | Session writes no records to the error tables |
| 3 | Session does not insert a single record |

Table 10.14 Test Case 4, Cycle 3

**Cycle:**4

**Description:** Verifying Lookup and Expression Transformation for Date

Use Typical Input data to verify whether the conversion of creation\_date and last\_update date is happening properly and whether the unconnected lookup returns the correct date key from wh\_dates

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session inserts record to the target table |
| 2 | Verify with the wh\_dates whether the date information for the date key written in the target table is same as the date information in the source table for creation\_date and last\_update column |

Table 10.15 Test Case 4, Cycle 4

**Cycle:**5

**Description:** Verifying Lookup and Expression Transformation for Product

Use Typical Input data to verify whether the conversion of inventory\_item\_id and warehouse\_id to customer\_key\_source is happening properly and whether the connected lookup returns the correct product key from wh\_products

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session inserts record to the target table |
| 2 | Verify that the product\_key\_source is formed properly from the inv\_item\_id and warehouse\_id columns of the source table |
| 3 | Verify with the wh\_products whether the product information for the product key written in the target table is same as the product information in the source table. |

Table 10.16 Test Case 4, Cycle 5

**Cycle:**6

**Description:** Verifying Lookup and Expression Transformation for Customer

Use Typical Input data to verify whether the conversion of creation\_date and last\_update date is happening properly and whether the connected lookup returns the correct customer key from wh\_customers

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session inserts record to the target table |
| 2 | Verify that the customer\_key\_source is formed properly from the customer\_id |
| 3 | Verify with the wh\_customers whether the customer information for the customer key written in the target table is same as the customer information in the source table. |

Table 10.17 Test Case 4, Cycle 6

**Cycle:**7

**Description:** Check for data in REJECTED\_ROWS table

Add a new row in STG\_SALES\_ORDRLNS where the customer\_id will generate a customer key which returns null upon lookup from the wh\_customers table. This row should be written to the REJCTED\_ROWS table

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session returns customer\_key as null after lookup from the wh\_customers |
| 2 | Session writes the record to the error table rejected\_rows |
| 3 | Session does not insert a single record |

Table 10.18 Test Case 4, Cycle 7

**Cycle:**8

**Description:** Check for data in REJECTED\_ROWS table. Add a new row in STG\_SALES\_ORDRLNS where the inventory\_item\_id and warehouse\_id will generate a product key which returns null upon lookup from the wh\_products table. This row should be written to the REJCTED\_ROWS table.

|  |  |
| --- | --- |
| **Condition Number** | **Description of Conditions to be Tested** |
| 1 | Session returns product\_key as null after lookup from the wh\_products |
| 2 | Session writes the record to the error table rejected\_rows |
| 3 | Session does not insert a single record |

Table 10.19 Test Case 4, Cycle 8

**TEST CASE 5**

For testing the implementation of the presentation layer which includes reporting and data mining applications we use the Oracle SQL Developer tool. We test whether the reports are showing valid data by using the data in the SQL View of MicroStrategy and executing it in the SQL Developer.

While developing the FP-Growth Algorithm, we first successfully implemented it on a small verifiable database and then scaled it to the transactional database present in the data mart. We tested the code for various scaling issues by implementing various methods such as providing various minimum support counts at the user interface.

Bayesian classification is tested by dividing the transactional database in to training and test data and accuracy of the prediction for the test data is calculated and displayed.