SYLLABUS

- What Is a Data Warehouse?
- Data Warehousing Today
  - Business intelligence
  - Customer Relationship Management
  - Data Mining
- Future Trends in Data Warehousing.
  - Unstructured Data
  - Search
  - service oriented architecture (SOA)
  - Real-time data warehousing
- Data Warehouse Architecture
  - Data flow architecture
  - System architecture

Table of Contents

1. What Is a Data Warehouse?

Let's begin by defining what a data warehouse is. A data warehouse is a system that retrieves and consolidates data periodically from the source systems into a dimensional or normalized data store. It usually keeps years of history and is queried for business intelligence or other analytical activities. It is typically updated in batches, not every time a transaction happens in the source system. Figure1 shows a diagram of a data warehouse system, including the applications.

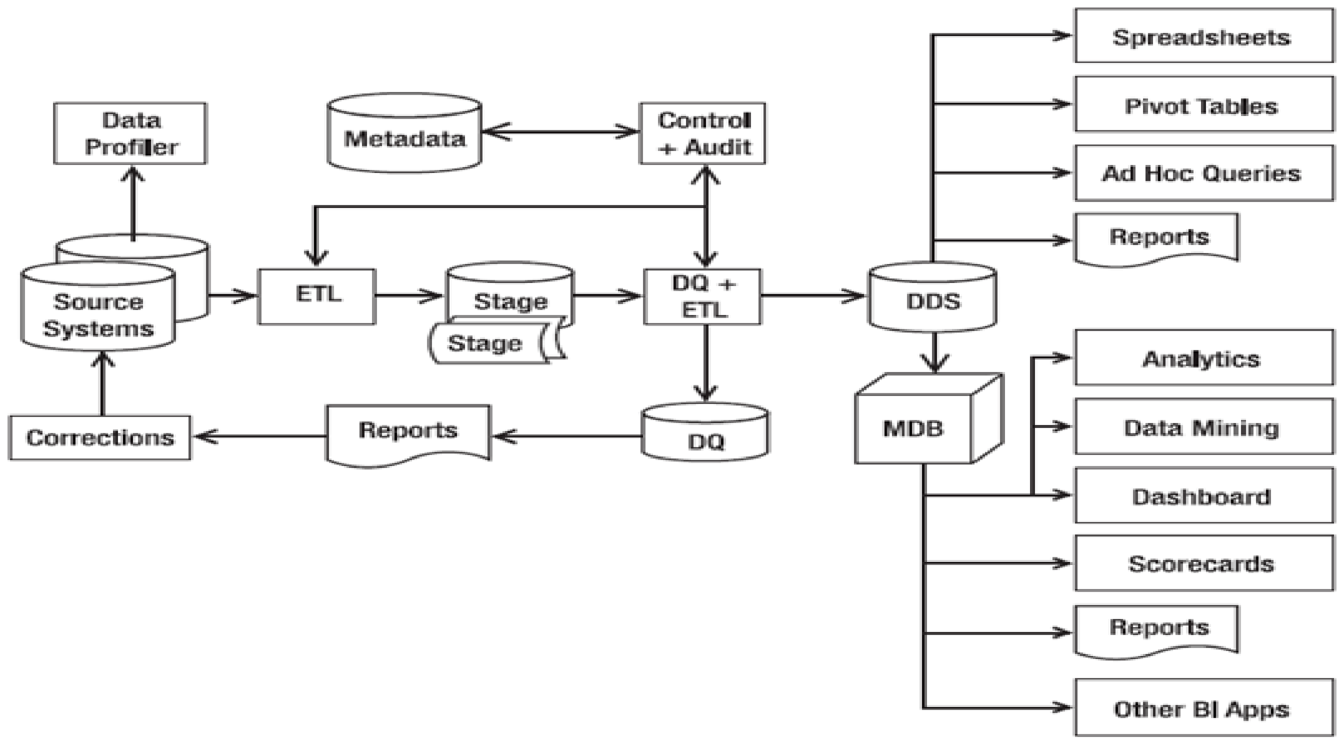

Figure 1 - A diagram of a data warehouse system

Let's go through the diagram in Figure 1, component by component, from left to right. The source systems are the OLTP systems that contain the data you want to load into the data warehouse**. Online Transaction Processing (OLTP)** is a system whose main purpose is to capture and store the business transactions. The source systems' data is examined using a data profiler to understand the characteristics of the data. A data profiler is a tool that has the capability to analyze data, such as finding out how many rows are in each table, how many rows contain NULL values, and so on.

The **extract, transform, and load (ETL)** system then brings data from various source systems into a staging area. ETL is a system that has the capability to connect to the source systems, read the data, transform the data, and load it into a target system (the target system doesn't have to be a data warehouse). The ETL system then integrates, transforms, and loads the data into a **dimensional data store (DDS)**. A DDS is a database that stores the data warehouse data in a different format than OLTP The reason for getting the data from the source system into the DDS and then querying the DDS instead of querying the source system directly is that in a DDS the data is arranged in a dimensional format that is more suitable for analysis. The second reason is because a DDS contains integrated data from several source systems.

When the ETL system loads the data into the DDS, the data quality rules do various data quality checks. Bad data is put into the **data quality (DQ)** database to be reported and then corrected in the source systems. Bad data can also be automatically corrected or tolerated if it is within a certain limit. The ETL system is managed and orchestrated by the control system, based on the sequence, rules, and logic stored in the metadata. The metadata is a database containing information about the data structure, the data meaning, the data usage, the data quality rules, and other information about the data.

The audit system logs the system operations and usage into the metadata database. The audit system is part of the ETL system that monitors the operational activities of the ETL processes and logs their operational statistics. It is used for understanding what happened during the ETL process.

Users use various front-end tools such as spreadsheets, pivot tables, reporting tools, and SQL query tools to retrieve and analyze the data in a DDS. Some applications operate on a multidimensional database format. For these applications, the data in the DDS is loaded into **multidimensional databases (MDBs)**, which **are also known as cubes**. A multidimensional database is a form of database where the data is stored in cells and the position of each cell is defined by a number of variables called dimensions. Each cell represents a business event, and the values of the dimensions show when and where this event happened.

Figure 2 shows a cube with three dimensions, or axes: Time, Store, and Customer. Assume that each dimension, or axis, has 100 segments, so there are 100 X 100 X 100 = 1 mil- lion cells in that cube. Each cell represents an event where a customer is buying something from a store at a particular time. Imagine that in each cell there are three numbers: Sales Value (the total value of the products that the customer purchased), Cost (the cost of goods sold + proportioned overheads), and Profit (the difference between the sales value and cost). This cube is an example of a multidimensional database.

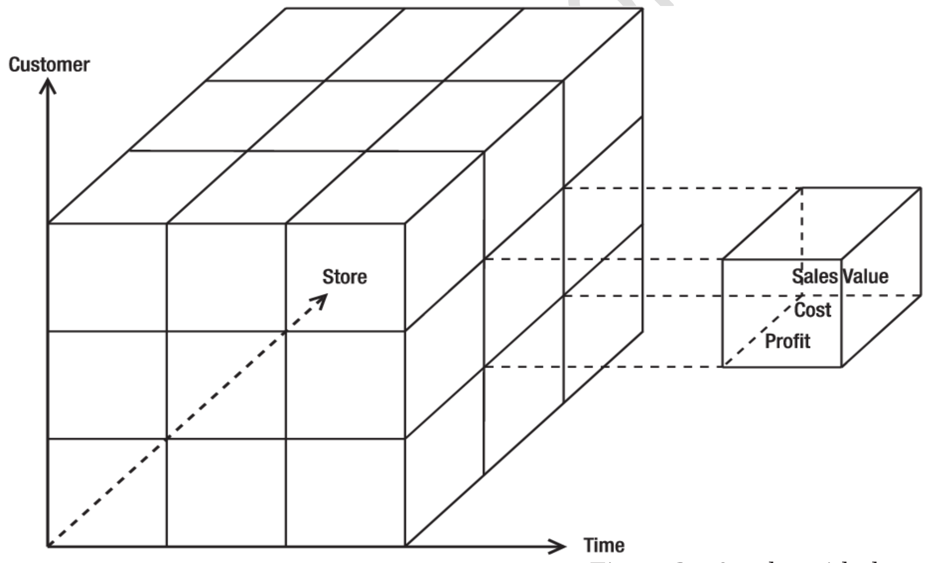

Figure 2 - A cube with three dimensions

Tools such as analytics applications, data mining, scorecards, dashboards, multidimensional reporting tools, and other BI tools can retrieve data interactively from multidimensional databases. They retrieve the data to produce various features and results on the front-end screens that enable the users to get a deeper understanding about their businesses. An example of an analytic application is to analyze the sales by time, customer, and product. The users can analyze the revenue and cost for a certain month, region, and product type.

Not all data warehouse systems have all the components pictured previously. Even if a data warehouse system does not have a data quality mechanism, a multidimensional data- base, any analytics applications, a front-end application, a control system or audit system, metadata, or a stage, you can still call it a data warehouse system. In its simplest form, it is similar to Figure 3.

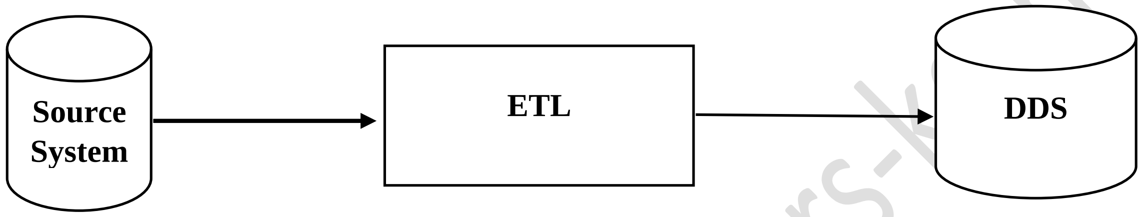

Figure 3 - Simplest form of a data warehouse system

In this case, the data warehouse system contains only an ETL system and a dimensional data store. The source system is not part of the data warehouse system. This is pretty much the minimum. If you take out just one more component, you cannot call it a data warehouse sys- tem anymore. In Figure 3, even though there is no front-end application such as reports or analytic applications, users can still query the data in the DDS by issuing direct SQL select statements using generic database query tools such as the one hosted in SQL Server Management Studio.

Now that you have an idea about what a data warehouse system is and its components, let's take a look at the data warehouse definition in more detail.

1.1 Retrieves Data

The data retrieval is performed by a set of routines widely known as an ETL system, which is an abbreviation for extract, transform, and load. The ETL system is a set of processes that retrieve data from the source systems, transform the data, and load it into a target system. The transformation can be used for changing the data to suit the format and criteria of the target system, for deriving new values to be loaded to the target system, or for validating the data from the source system. ETL systems are not only used to load data into the data warehouse. They are widely used for any kind of data movements.

1.2 Consolidates Data

A data warehouse consolidates many transactional systems. The key difference between a data warehouse and a front-office transactional system is that the data in the data ware- house is integrated. This consolidation or integration should take into account the data availability (some data is available in several systems but not in others), time ranges (data in different systems has different validity periods), different definitions (the term total weekly revenue in one system may have a different meaning from total weekly revenue in other systems), conversion (different systems may have a different unit of measure or currency), and matching (merging data based on common identifiers between different systems).

Let’s go through the previous concepts one by one:

1.2.1 Data availability: When consolidating data from different source systems, it is possible that a piece of data is available in one system but is not in the other system. For example, system A may have seven address fields (address1, address2, address3, city, county, ZIP, and country), but system Bodes not have the address3 field and the country field. In system A, an order may have two level order header and order line. However, in system B, an order has four levels—order header, order bundle, order line item, and financial components. So when consolidating data across different transaction systems, you need to be aware of unavailable columns and missing level sin the hierarchy. In the previous examples, you can leave address3 blank in the target and set the country to a default value. In the order hierarchy example, you can consolidate into two levels, order header and order line.

1.2.2 Time ranges: The same piece of data exists in different systems, but they have different time periods. So, you need to be careful when consolidating them. You always need to examine what time period is applicable to which data before you consolidate the data. Otherwise, you are at risk of having inaccurate data in the warehouse because you mixed different time periods. For example, say in system A the average supplier overhead cost is calculated weekly, but in system B it is calculated monthly. You can't just consolidate them. In this example, you need to go back upstream to get the individual components that make up the average supplier overhead cost in both systems and add them up first.

1.2.3 Definitions: Sometimes the same data may contain different things. In system A, a column called “Total Order Value” may contain taxes, discounts, credit card charges, and delivery charges, whereas in system B it does not contain delivery charges. In system a, the term weekly fraJc may refer to unique web site visitors, whereas in system B it means non unique web site visitors. In this matter, you always need to examine the meaning of each piece of data. Just because they have the same name doesn’t mean they are the same. This is important because you could have inaccurate data or meaningless data in the data warehouse if you consolidate data with different meanings.

1.2.4 Conversion: When consolidating data across different source systems, sometimes you need to do conversion because the data in the source system is in different units of meas- ure. If you add them up without converting them first, then you will have incorrect data in the warehouse. In some cases, the conversion rate is fixed (always the same value), but in other cases the conversion rate changes from time to time. If it changes from time to time, you need to know what time period to use when converting. For example, the conversion between the times in one country to another country is affected by daylight savings time, so you need to know the date to be able to do the conversion. In addition, the conversion rate between one currency and another currency fluctuates every day, so when convert- ing, you need to know when the transaction happened.

1.2.5 Matching: Matching is a process of determining whether a piece of data in one system is the same as the data in another system. Matching is important because if you match the wrong data, you will have inaccurate data in the data warehouse. For example, say you want to consolidate the data for customer 1 in system A with the data for customer 1 in system B. In this case, you need to determine first whether those two are the same customer. If you match the wrong customers, the transaction from one customer could be mixed up with the data from another customer. The matching criteria are different from company to company. Sometimes criteria are simple, such as using user IDs, customer IDs, or account IDs. But sometimes it is quite complex, such as name + e-mail address + address. The logic of determining a match can be simply based on the equation sign (=) to identify an exact match. It can also be based on fuzzy logic or matching rules.

1.2.6 Periodically

The data retrieval and the consolidation do not happen only once; they happen many times and usually at regular intervals, such as daily or a few times a day. If the data retrieval happens only once, then the data will become obsolete, and after some time it will not be useful.

You can determine the period of data retrieval and consolation based on the business requirements and the frequency of data updates in the source systems. The data retrieval interval needs to be the same as the source system's data update frequency. If the source sys- tem is updated once a day, you need to set the data retrieval once a day. There is no point extracting the data from that source system several times a day.

On the other hand, you need to make sure the data retrieval interval satisfies the business requirements.

2. Dimensional Data Store

A data warehouse is a system that retrieves data from source systems and puts it into a dimensional data store or a normalized data store. Yes, some data warehouses are in dimensional format, but some data warehouses are in normalized format. Let's go through both formats and the differences between them.

A DDS is one or several databases containing a collection of dimensional data marts.

A dimensional data mart is a group of related fact tables and their corresponding dimension tables containing the measurements of business events categorized by their dimensions.

A dimensional data store is denormalized, and the dimensions are conformed. Conformed dimensions mean either they are exactly the same dimension table or one is the subset of the other. Dimension A is said to be a subset of dimension B when all columns of dimension A exist in dimension B and all rows of dimension A exist in dimension B.

A dimensional data store can be implemented physically in the form of several different schemas. Examples of dimensional data store schemas are a star schema a snowflake schema, and a galaxy schema. In a star schema, a dimension does not have a sub- table (a sub dimension). In a snowflake schema, a dimension can have a sub dimension. The purpose of having a sub dimension is to minimize redundant data. A galaxy schema is also known as a foci come schema. In a galaxy schema, you have two or more related fact tables surrounded by common dimensions. The benefit of having a star schema is that it is simpler than snowflake and galaxy schemas, making it easier for the ETL processes to load the data into DDS. The benefit of a snowflake schema is that some analytics applications work better with a snowflake schema compared to a star schema or galaxy schema. The other benefit of a snowflake schema is less data redundancy, so less disk space is required. The benefit of galaxy schema is the ability to model the business events more accurately by using several fact tables.

2.1 Normalized Data Store

Other types of data warehouses put the data not in a dimensional data store but in a normalized data store. A normalized data store is one or more relational databases with little or no data redundancy. A relational database is a database that consists of entity tables with parent- child relationships between them. Normalization is a process of removing data redundancy by implementing normalization rules. There are five degrees of normal forms, from the first normal form to the fifth normal form. A normalized data store is usually in third normal form or higher, such as fourth or fifth normal form.

2.2 History

One of the key differences between a transactional system and a data warehouse system is the capability and capacity to store history. Most transactional systems store some history, but data warehouse systems store very long history. In my experience, transactional systems store only one to three years of data; beyond that, the data is purged. For example, let's have a look at a sales order—processing system. The purpose of this system is to process customer orders. Once an order is dispatched and paid, it is closed, and after two or three years, you want to purge the closed orders out of the active system and archive them to maintain system performance.

2.3 Query

Querying is the process of getting data from a data store, which satisfies certain criteria. Here is an example of a simple query: "How many customers do you have now?""

A data warehouse is built to be queried. That is the number-one purpose of its existence. Users are not allowed to update the data warehouse. Users can only query the data ware- house. Only the ETL system is allowed to update the data warehouse. This is one of the key differences between a data warehouse and a transaction system.

2.4 Business Intelligence

Business intelligence is a collection of activities to understand business situations by per forming various types of analysis on the company data as well as on external data from third parties to help make strategic, tactical, and operational business decisions and take necessary actions for improving business performance. This includes gathering, analyzing, understanding, and managing data about operation performance, customer and supplier activities, financial performance, market movements, competition, regulatory compliance, and quality controls.

2.5 Reporting

In a data warehousing context, a report is a program that retrieves data from the data ware- house and presents it to the users on the screen or on paper. Users also can subscribe to these reports so that they can be sent to the users automatically by e-mail at certain times (daily or weekly, for example) or in response to events.

2.6 Online Analytical Processing (OLAP)

OLAP is the activity of interactively analyzing business transaction data stored in the dimensional data warehouse to make tactical and strategic business decisions. Typical people who do OLAP work are business analysts, business managers, and executives. Typical functionality in OLAP includes aggregating (totaling), drilling down (getting the details), and slicing and dicing

(Cutting the cube and summing the values in the cells). OLAP functionality can be delivered using a relational database or using a multidimensional database. OLAP that uses a relational database is known as relational online analytical processing (ROLAP). OLAP that uses a multidimensional database is known as multidimensional online analytical processing (MOLAP).

2.7 Data Mining

Data mining is a process to explore data to find the patterns and relationships that describe the data and to predict the unknown or future values of the data. The key value in data mining is the ability to understand why some things happened in the past and to predict what will happen in the future. When data mining is used to explain the current or past situation, it is called descriptive analytics. When data mining is used to predict the future, it is called predictive analytics.

In business intelligence, popular applications of data mining are for fraud detection (credit card industry), forecasting and budgeting (finance), developing cellular/mobile pack- ages by analyzing call patterns (telecommunication industry), market basket analysis (retail industry), customer risk profiling (insurance industry), usage monitoring (energy and utili- ties), and machine service times (manufacturing industry).

2.8 Other Analytical Activities

Other than for business intelligence, data warehouses are also used for analytical activities in nonbusiness purposes, such as scientific research, government departments (statistics office, weather office, economic analysis, and predictions), military intelligence, emergency and disaster management, charity organizations, server performance monitoring, and network traffic analysis.

2.9 Updated in Batches

A data warehouse is usually a read-only system; that is, users are not able to update or delete data in the data warehouse. Data warehouse data is updated using a standard mechanism called ETL at certain times by bringing data from the operational source system. This is different from a transactional system or OLTP where users are able to update the system at any time.

3. Data Warehousing Today

Today most data warehouses are used for business intelligence to enhance CRM and for data mining. Some are also used for reporting, and some are used for data integration. These usages are all interrelated; for example, business intelligence and CR Muse data mining, business intelligence uses reporting, and BI and CRM also use data integration.

3.1 Business Intelligence

It seems that many vendors prefer to use the term business intelligence rather than data warehousing. In other words, they are more focused on what a data warehouse can do for a business. As I explained previously, many data warehouses today are used for BI. That is, the purpose of a data warehouse is to help business users understand their business better; to help them make better operational, tactical, and strategic business decisions; and to help them improve business performance.

3.2 Customer Relationship Management

I defined CRM earlier in this chapter. A customer is a person or organization that consumes your products or services. In nonbusiness organizations, such as universities and government agencies, a customer is the person who the organization serves.

A CRM system consists of applications that support CRM activities (please refer to the definition earlier where these activities were mentioned). In a CRM system, the following functionality is ideally done in a dimensional data warehouse:

Single customer view: The ability to unify or consolidate several definitions or meanings of a customer, such as subscribers, purchasers, bookers, and registered users, through the use of customer matching

Permission management: Storing and managing declarations or statements from cus- tomers so you can send campaigns to them or communicate with them including subscription-based, tactical campaigns, ISP feedback loops, and communication preferences

Campaign segmentation: Attributes or elements you can use to segregate the customers into groups, such as order data, demographic data, campaign delivery, campaign response, and customer loyalty score Customer services/support. Helping customers before they use the service or product (preconception support), when they are using the service or product, and after they used the service/product; handling customer complaints; and helping them in emergencies such as by contacting them.

Customer analysis: Various kinds of analysis including purchase patterns, price sensitivity analysis, shopping behavior, customer attrition analysis, customer profitability analysis, and fraud detection

Personalization: Tailoring your web site, products, services, campaigns, and offers for a particular customer or a group of customers, such as price and product alerts, personal- ized offers and recommendations, and site personalization

Customer loyalty scheme: Various ways to reward highly valued customers and build loyalty among customer bases, including calculating the customer scores/point-based system, customer classification, satisfaction survey analysis, and the scheme administration.

3.3 Data Mining

Data mining is a field that has been growing fast in the past few years. It is also known as knowledge discovery, because it includes trying to find meaningful and useful information from a large amount of data. It is an interactive or automated process to find patterns des- cribbing the data and to predict the future behavior of the data based on these patterns.

Data mining systems can work with many types of data formats: various types of data- bases (relational databases, hierarchical databases, dimensional databases, object-oriented databases, and multidimensional databases), files (spreadsheet files, XML files, and struc- tured text files), unstructured or semi structured data (documents, e-mails, and XML files), stream data (plant measurements, temperatures and pressures, network traffic, and telecom- munication traffic), multimedia files (audio, video, images, and speeches), web sites/pages, and web logs.

Of these various types of data, data mining applications work best with a data warehouse because the data is already cleaned, it is structured, it has metadata that describes the data (useful for navigating around the data), it is integrated, it is nonvolatile (that is, quite static), and most important it is usually arranged in dimensional format that is suitable for various data mining tasks such as classification, exploration, description, and prediction. In data min- ing projects, data from the various sources mentioned in the previous paragraph are arranged in a dimensional database. The data mining applications retrieve data from this database to apply various data mining algorithms and logic to the data. The application then presents the result to the end users.

4. Future Trends in Data Warehousing

Several future trends in data warehousing today are unstructured data, search, service- oriented architecture, and real-time data warehousing.

4.1 Unstructured Data

Data that is in databases is structured; it is organized in rows and columns. I have talked in great length in previous sections about data warehousing using structured data; that is, the source system is a database. It can be a relational database (tables, rows, and columns), and it may be an object-oriented database (classes and types) or a hierarchical database (a tree-like structure). However, they all have data structure.

Unstructured data, on the other hand, does not have a data structure such as rows and columns, a tree-like structure, or classes and types. Examples of unstructured data are documents, images (photos, diagrams, and pictures), audio (songs, speeches, and sounds), video (films, animations), streaming data, text, e-mails, and Internet web sites. Arguably, some people say this kind of data is semi structured data, with the argument that there is some structure, so it has attributes. For example, an e-mail has attributes such as from, to, date sent, date created, date received, subject, and body; a document has attributes such as title, subject, author, and number of pages, number of words, creation date, and last-modified date.

How do you store unstructured data in the data warehouse? And, after you store it, how do you get the information that you need out of this data? Well, the answer to the first question is for each unstructured data item you define the attributes and then organize these items according to the attributes. You can store the unstructured data items in a relational database as a binary object column, with the attributes as other columns. Or you can store the unstructured data items in the file systems and just store the pointer to the file in the database.

Each type of unstructured data has different physical and content attributes. These attributes can be stored in a relational or multidimensional database to enable the users to easily find a particular piece of unstructured data. The content of the unstructured data itself can be analyzed, extracted, categorized, and stored to assist information retrieval.

For example, let's say you have 1 million e-mails as your unstructured data. They have attributes, such as from, to, cc, bcc, subject, date created, date sent, attachments, number of words in the body, host address, originator address, recipient address, and so on. You then store these attributes in a relational table, and the e-mails are stored as files with the file name and location stored in the table.

In addition to the physical attribute of the e-mail, you can also uncover the content attributes of the e-mail body. First, you do some text cleansing on the e-mail body to remove noises such as numbers and adjectives; convert past-tense verbs into present; correct plu- rals into singulars; eliminate meaningless words such as aft, the, irt (well, they are not really meaningless, but what you are looking for are verbs and nouns); convert synonyms using a dictionary; and so on. Then you can count how many times each word occurs, giving a score depending on how far a word is located from another word and categorizing the items to a certain hierarchy that you have created based on the words, the statistical properties of these words, and the score for these words. Then you can put these categories and proper- ties in those attribute columns to assist information retrieval.
It is probably not fair if we say that unstructured data is new. Text analytics have been around for a long time. It's just that recently people have started realizing that most of their data is stored in unstructured

data, especially text such as documents and e-mails, and only a little bit of their data is structured data. So why do we spend a lot of effort storing and analyzing structured data (such as numbers) and only a little effort on unstructured data? Hence, structured data has become one of the current trends in data warehousing.

4.2 Search

This section answers the second question, how do you get the information out? The answer is by searching. To get the information out of structured data, provided that you know the structure, you can do a select query, whether using a static report or manual interactive ad hoc queries. If you use a BI application, the application can go through the metadata and display the structure of the data and then assist you in navigating through the data to retrieve the information you need.

To get the information out of unstructured data, especially text data such as documents, e-mails, and web pages, you do a search. Like on the Internet, the search engine has already crawled the data warehouse and indexed the unstructured data. The search engine has categorized the unstructured data based on their types and their properties and, in the case of web pages, their links.

You can now type what you want to find in a search box, and the search engine will go through its index, find the locations of the information, and display the results. It can also offer predefined searches, wrapped in a nice hierarchical structure for you to navigate and choose. It can also memorize user searches that could assist you in defining what to type when searching.

4.3 Service-Oriented Architecture (SOA)

SOA is a method of building an application using a number of smaller, independent components that talk to each other by offering and consuming their services. These components can be distributed; in fact, they can be located on different sides of the world.

Almost every large application can benefit from an SOA approach. You don't build one giant application anymore. Instead, you build many smaller pieces that talk to each other. It is the nature of the IT industry that applications will need to be replaced every several years (I'd say every four to eight years). It could be because of obsolete technology or because of the functionality. Bankruptcy, mergers, and takeovers are also the other drivers to this.

If you make one giant application, it would be costly to replace it. If you make it from a number of smaller, independent components, it is easier to replace it. SOA gives us more flexi- bility to replace the components. In other words, you can do it in stages piece by piece without affecting the functionality. This is because the components are independent; that is, they don't care how the other components work internally as long as externally they get the responses they require. This enables you to rebuild one component with newer technology without affecting the others.

How does this SOA relate to data warehousing? If you refer to Figure 1, a data ware- house system consists of many components: source systems, ETL systems, a data quality mechanism, a metadata system, audit and control systems, a BI portal, a reporting application, OLAP/ analytic applications, data mining applications, and the database system itself. You can build it as one giant application with all the components tightly coupled; that is, you cannot replace one component without affecting the other components. Or you can build it in service-oriented architecture—you build it as a number of smaller, independent components that talk to each other by offering and consuming their services.

Let’s take a look at an example: ETL. During a recent project I was involved with, the project team built the ETL system as a service. It had various data retrieval services (for example, get service, get Account, and get Transaction), which retrieved the data from the source system, wrapped it around XML messages, and sent it to a message queue. It had a scheduler service that acted as a mini-batch system; it invoked various data retrieval services at a certain frequency that we set, such as every 5 or 30 minutes. It had a queue management
Service that controlled the message queue. It had data transformation and data loading serv- ices that transformed and loaded the data into the data warehouse.

More and more data warehousing applications on all fronts are built using SOA: ETL, reporting, analytics, BI applications, data mining, metadata, data quality, and data cleansing. In the future, it would be easier to update one component without impacting the others and to connect different components that are made using different technologies.

4.4 Real-Time Data Warehouse

A data warehouse, a few years ago, was usually updated every day or every week. In the past two to three years, there has been more and more demand to increase the frequency of updates. The users want to see the data in the data warehouse updated every two minutes or even in real time. A real-time data warehouse is a data warehouse that is updated (by the ETL) the moment the transaction happens in the source system.

For example, you can put triggers on the sales transaction table in the source system so that whenever there is a transaction inserted into the database, the trigger fires and sends the new record to the data warehouse as a message. The data warehouse has an active listener that captures the message the moment it arrives, cleanses it, DQs it, transforms it, and inserts it into the fact table immediately. I’m ta11‹lng about a two-second time difference here, between the moment a customer purchased a product on the web site and the moment data is available in the fact table.

The other approach of implementing a real-time data warehouse is to modify the operational source application to write to the data warehouse staging area, immediately after it writes the data in its internal database. In the staging database, you place triggers that would be invoked every time there is a new record inserted, and these triggers update the data warehouse.

Near real-time approaches can be implemented by using a mini-batch with two- to five- minute frequency, which pulls the data from the stage area instead of using triggers. This mini- batch also does the normal ETL job—transforming the data and loading it into the data warehouse dimensional database. The mini-batch can also pull the data directly from the source system, eliminating the need of modifying the source system to update the staging area.

5. Data warehouse Architecture system architecture

The previous sections covered data flow architecture. They showed how the data is arranged in the data stores and how the data flows within the data warehouse system. Once you have chosen a certain data flow architecture, you then need to design the system architecture, which is the physical arrangement and connections between the servers, network, software, storage system, and clients. Designing a system architecture requires knowledge about hardware (especially servers), networking (especially with regard to security and performance and in the last few years also fiber networks), and storage (especially storage area networks [SANs], redundant array of inexpensive disks [RAID], and automated tape backup solutions).

In this example, the system architecture consists of three servers: one ETL server, one database server, and one reporting server. The source system is an electronic point-of-sale system that records retail sales in the stores running on Oracle 9.2 on Sun Solaris. The ETL server is Informatics Power Center 7 installed on Red Hat Enterprise Linux. The data warehouse data stores (stage, DDS, and MDB) are hosted in the SQL Server relational database engine and Analysis Services, running on Windows Server 2003. The data is physically stored in a DAS consisting of fifteen 146GB disks, making the raw capacity 2TB. The disks are configured in RAID 1 and RAID 5 for data resilience. (I will discuss RAID in Chapter 6 when I cover physical database design.) The reports are created and managed using SSRS installed on Windows Server 2003.

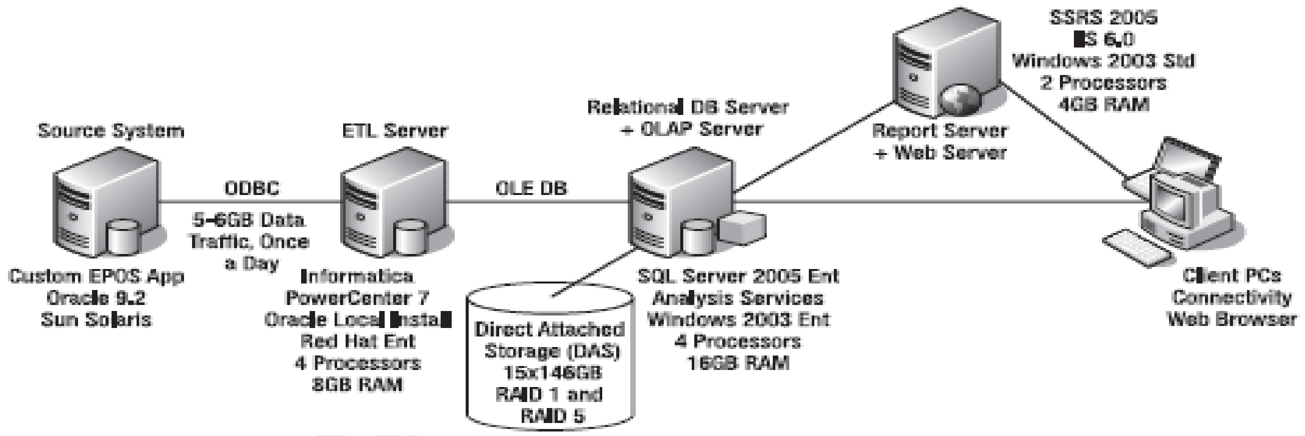

Figure 4 - Example of system architecture for data warehouse

I chose this example because it represents a typical architecture for a medium system. We have a dedicated ETL server, separated from the database server. It's a medium size of data; the raw capacity of 2TB is about 400GB to 500GB final usable database space, assuming we have both development and production environments. The platform is a bit of a mixture, as typically found in organizations: the source system and ETL are not Microsoft. The informatics was probably already there when the data warehouse project started, so they have to use what they already have. Therefore, you can create system architecture with different platforms.

A data warehouse architect does not usually design the system architecture of a data warehouse infrastructure, but in my opinion, it is helpful if they know these subjects—perhaps not at a very detailed level, but they need to know at a high level. For example, they don't need to understand how to build a four-node cluster using Windows 2003 servers, but they need to know what kind of high availability can be achieved using clustering technology.

To design a data warehouse system architecture, you first establish the technology stack you want to use for ETL, the database, and BI, such as Microsoft SQL Server (SSIS, SSAS, SSIS), informatics + Oracle 9i + Congo's, and so on. This is determined based on product capability and based on the company standard. After you determine the technology stack, you do a high-level design on the servers, network configuration, and storage configuration that supports the chosen technology, including high-availability design.

Then you determine the detailed technical specification of the servers, network, and storage. This is done based on capacity and system performance requirements. You then order the hardware and software and build the system in the data center together with the hardware and network vendor. You then install and configure the software. Designing and building the environment is fundamental and critical to the performance and stability of the data warehouse system that you are going to build on top of it.

The other factor that greatly affects the system architecture is the choice of software in building the data warehouse, such as specific versions of SQL Server, Oracle, or Teradata. The system architectures required to run this software are different. For example, Teradata runs on massively parallel processing hardware. It does not share the storage between the nodes. On the other hand, a SQL Server cluster uses central storage. It shares the storage between the nodes.

In terms of software, there are two different types of database software: symmetric multiprocessing (SMP) and massively parallel processing (MPP). An SMP database system is a database system that runs on one or more machines with several identical processors sharing the same disk storage. When an SMP database system runs on more than one machine, it is called a clustered configuration. The database is physically located in a single disk storage system. Examples of SMP database systems are SQL Server, Oracle, DB/2, Informix, and Sybase. An MPP database system is a database system that runs on more than one machine where each machine has its own disk storage. The database is physically located in several disk storage systems that are interconnected to each other. An MPP database system is also known as a parallel database system. Examples of MPP database systems are Teradata, Neoview, Netezza, and DATAllegro.

The machines in SMP and MPP database systems are called nodes. An MPP database system is faster and more scalable than an SMP database system. In an MPP database system, a table is physically located in several nodes, each with its own storage. When you retrieve data from this table, all nodes simultaneously read the data from their own storage, so the process of reading the data from disk is quicker. Similarly, when you load data into this table, all nodes simultaneously load a bit of the data into their disks. In SMP database systems, there is a bottleneck on the disk storage. SMP database systems, on the other hand, are simpler, are easier to maintain, and have lower costs.

5.1 Data Flow Architecture

In data warehousing, the data flow architecture is a configuration of data stores within a data warehouse system, along with the arrangement of how the data flows from the source systems through these data stores to the applications used by the end users. This includes how the data flows are controlled, logged, and monitored, as well as the mechanism to ensure the quality of the data in the data stores. I discussed the data flow architecture briefly in Chapter 1, but in this chapter I will discuss it in more detail, along with four data flow architectures: single DDS, NDS + DDS, ODS + DDS, and federated data warehouse.

The data flow architecture is different from data architecture. Data architecture is about how the data is arranged in each data store and how a data store is designed to reflect the business processes. The activity to produce data architecture is known as data modelling.
Data stores are important components of data flow architecture. I'll begin the discussion about the data flow architecture by explaining what a data store is. A data store is one or more databases or files containing data warehouse data, arranged in a particular format and involved in data warehouse processes. Based on the user accessibility, you can classify data warehouse data stores into three types:

- A user-facing data store is a data store that is available to end users and is queried by the end users and end-user applications.
- An internal data store is a data store that is used internally by data warehouse components for the purpose of integrating, cleansing, logging, and preparing data, and it is not open for query by the end users and end-user applications.
- A hybrid data store is used for both internal data warehouse mechanisms and for query by the end users and end-user applications.
- A master data store is a user-facing or hybrid data store containing a complete set of data in a data warehouse, including all versions and all historical data.
- Based on the data format, you can classify data warehouse data stores into four types: · a stage is an internal data store used for transforming and preparing the data obtained from the source systems, before the data is loaded to other data stores in a data warehouse.
- A normalized data store (NDS) is an internal master data store in the form of one or more normalized relational databases for the purpose of integrating data from various source systems captured in a stage, before the data is loaded to a user-facing data store.
- An operational data store (ODS) is a hybrid data store in the form of one or more normalized relational databases, containing the transaction data and the most recent version of master data, for the purpose of supporting operational applications.
- A dimensional data store (DDS) is a user-facing data store, in the form of one or more relational databases, where the data is arranged in dimensional format for the purpose of supporting analytical queries.
- A relational database is a database that consists of entity tables with parent-child relationships between them.
- A normalized database is a database with little or no data redundancy in third normal form or higher.
- A denormalized database is a database with some data redundancy that has not gone through a normalization process.
- A dimensional database is a denormalized database consisting of fact tables and common dimension tables containing the measurements of business events, categorized by their dimensions.

Some applications require the data to be in the form of a multidimensional database (MDB) rather than a relational database. An MDB is a form of database where the data is stored in cells and the position of each cell is defined by a number of variables called dimensions. Each cell represents a business event, and the value of the dimensions shows when and where this event happened. MDB is populated from DDS. Extract, transform, and load (ETL) is a system that has the capability to read the data from one data store, transform the data, and load it into another data store. The data store where the ETL reads the data from is called a source, and the data store that the ETL loads the data into is called a target.

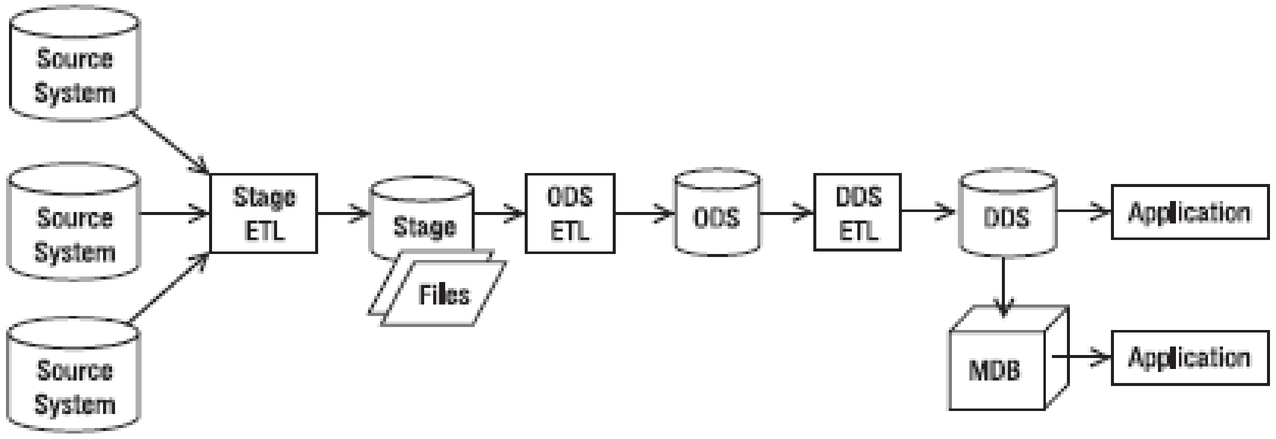

Figure 5 - A data flow architecture with a stage, ODS, DDS, and MDB

The arrows in Figure 5 show the flows of data. The data flows from the source systems to a stage, to ODS, to DDS, and then to the applications used by the users. In Figure 5 there are three ETL packages between the four data stores. A stage ETL retrieves data from the source systems and loads it into a stage. ODS ETL retrieves data from a stage and loads it into ODS. DDS ETL retrieves data from ODS and loads it into DDS.

An ETL package consists of several ETL processes. An ETL process is a program that is part of an ETL package that retrieves data from one or several sources and populates one target table. An ETL process consists of several steps. A step is a component of an ETL process that does a specific task. An example of a step is extracting particular data from a source data store or performing certain data transformations.

The ETL packages in the data warehouse are managed by a control system, which is a system that manages the time each ETL package runs, governs the sequence of execution of processes within an ETL package, and provides the capability to restart the ETL packages from the point of failure. The mechanism to log the result of each step of an ETL process is called ETL audit. Examples of the results logged by ETL audits are how many records are transformed or loaded in that step, the time the step started and finished, and the step identifier so you can trace it down when debugging or for auditing purposes.

The description of each ETL process is stored in metadata. This includes the source it extracts the data from, the target it loads the data into, the transformation applied, the parent process, and the schedule each ETL process is supposed to run. In data warehousing, metadata is a data store containing the description of the structure, data, and processes within the data warehouse. This includes the data definitions and mapping, the data structure of each data store, the data structure of the source systems, the descriptions of each ETL process, the description of data quality rules, and a log of all processes and activities in the data warehouse.

Data quality processes are the activities and mechanism to make sure the data in the data warehouse is correct and complete. This is usually done by checking the data on its way into the data warehouse. Data quality processes also cover the mechanism to report the bad data and to correct it. A data firewall is a program that checks whether the incoming data complies with the data quality rules. A data quality rule is the criteria that verify the data from the source systems are within the expected range and in the correct format. A data quality database is a database containing incoming data that fails data quality rules. Data quality reports read the data quality violations from the data quality (DQ) database and display them on paper or on the screen. Figure 6 shows a data flow architecture complete with the control system, the metadata, and the components of data quality processes.

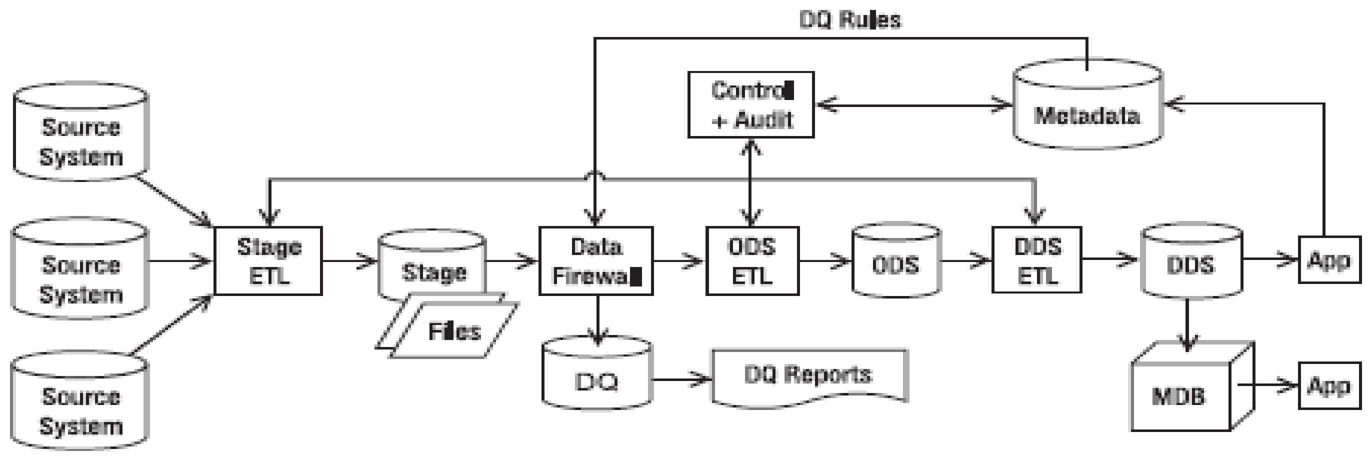

Figure 6 - A data flow architecture with control system, metadata, and data quality process

A data flow architecture is one of the first things you need to decide when building a data warehouse system because the data flow architecture determines what components need to be built and therefore affects the project plan and costs. The data flow architecture shows how the data flows through the data stores within a data warehouse.

The data flow architecture is designed based on the data requirements from the applications, including the data quality requirements. Data warehouse applications require data in different formats. These formats dictate the data stores you need to have. If the applications require dimensional format, then you need to build a DDS. If the applications require a normalized format for operational purposes, then you need to build an ODS. If the application requires multidimensional format, then you need to build an MDB. Once you determine the data stores you need to build, you can design the ETL to populate those data stores. Then you build a data quality mechanism to make sure the data in the data warehouse is correct and complete.

5.2 (System) Single DDS

In this section, you will learn about a simple data flow architecture that consists of only two data stores: stage and DDS. In this architecture, the core data warehouse store is in dimensional format.

In single DDS architecture, you have a one dimensional data store. The DDS consists of one or several dimensional data marts. A dimensional data mart is a group of related fact tables and their corresponding dimension tables containing the measurements of business events, categorized by their dimensions. An ETL package extracts the data from different source systems and puts them on the stage.

A stage is a place where you store the data you extracted from the store system temporarily, before processing it further. A stage is necessary when the transformation is complex (in other words, cannot be done on the fly in a single step in memory), when the data volume is large (in other words, not enough to be put in memory), or when data from several source systems arrives at different times (in other words, not extracted by a single

ETL). A stage is also necessary if you need to minimize the time to extract the data from the source system. In other words, the ETL processes dump the extracted data on disk and disconnect from the source system as soon as possible, and then at their own time they can process the data.

The physical form of a stage can be a database or files. The ETL that extracts data from the source system inserts the data into a database or writes them as files. A second ETL package picks up the data from the stage, integrates the data from different source system, applies some data quality rules, and puts the consolidated data into the DDS. Figure 7 describes general model of this architecture.

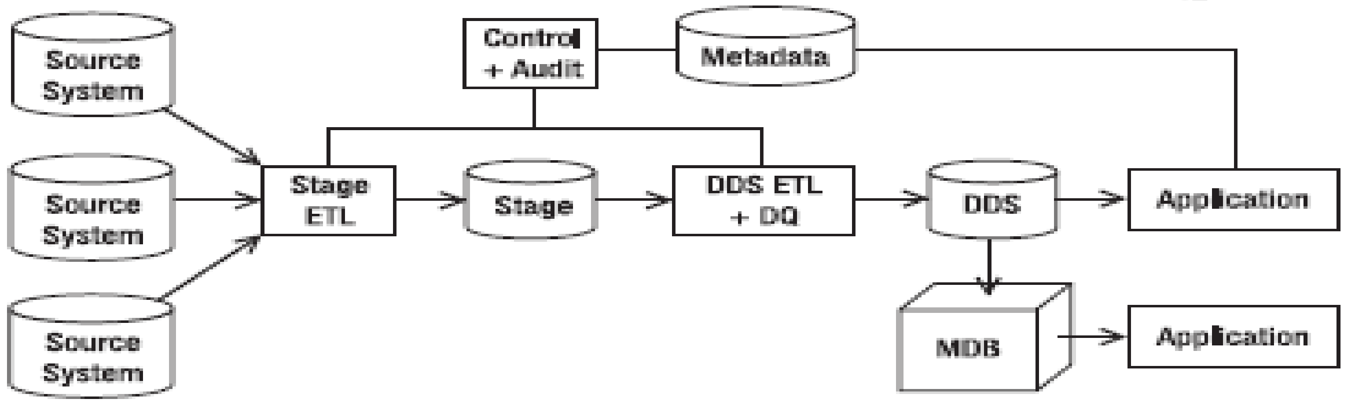

Figure 7 - Single DDS data warehouse architecture

In Figure 7, the "Control + Audit" box contains the ETL control system and audit, as I discussed earlier. They manage the ETL processes and log the ETL execution results. The metadata database contains the description of the structure, data, and processes within the data warehouse.

The data warehouse applications, such as business intelligence (BI) reports, read the data in the DDS and bring the data to the users. The data in the DDS can also be uploaded into multidimensional databases, such as SQL Server Analysis Services, and then accessed by the users via OLAP and data mining applications.

Some ETL architects prefer to combine the two ETL packages surrounding the stage in Figure 7 into one package, as pictured in Figure 8. In Figure 8, the stage ETL, the DDS ETL, and the data quality processes are combined into one ETL. The advantage of combining them into one package is to have more control over the timing of when the data is written to and retrieved from the stage. In particular, you can load the data directly into the DDS without putting it to disk first. The disadvantage is that the ETL package becomes more complicated.

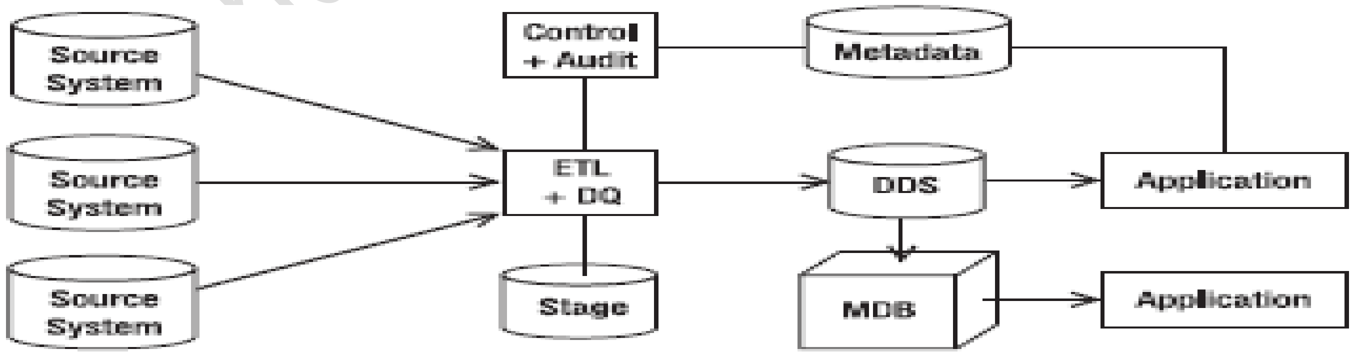

Figure 8 - Data architecture where the stage ETL and DDS ETL are combined

An advantage of a single DDS architecture is that it is simpler than the next three architectures. It is simpler because the data from the stage is loaded straight into the dimensional data store, without going to any kind of normalized store first. The main disadvantage is that it is more difficult, in this architecture, to create a second DDS. The DDS in the single DDS architecture is the master data store. It contains a complete set of data in a data warehouse, including all versions and all historical data. Sometimes you need to create a smaller DDS containing a subset of the data in the master DDS for the purpose of specific analysis where you want to be able to change the data or you want the data to be static. To create this smaller DDS, you would need to create a new ETL package that retrieves data from the master DDS and populates the smaller DDS. You need to build this ETL package from scratch. You cannot reuse the existing ETL package because the existing ETL package retrieves data from the stage and populates the master DDS. It's a totally different data flow altogether.

For example, the users may require a static smaller DDS containing only order data for the purpose of analyzing the impact of a price increase across different customer accounts.

They want the BI tool (Business Objects, Congo's, or Analysis Services, for example) to run on this smaller DDS so they can analyze the price increase. To create this smaller DDS, you need to write a new ETL package.

You would use a single DDS architecture when you need only a one dimensional store and you don't need a normalized data store. It is used for a simple, quick, and straight forward analytical BI solution where the data is used only to feed a dimensional data warehouse. A single DDS solution is particularly applicable when you have only one source system because you don't need additional NDS or ODS to integrate the data from different source systems.

Compared to the NDS + DDS or ODS + DDS architecture, the single DDS architecture is the simplest to build and has the quickest ETL run time because the data is loaded straight into DDS without going into the NDS or ODS data store first.

5.2.1 NDS + DDS

In NDS + DDS data flow architecture, there are three data stores: stage, NDS, and DDS. This architecture is similar to the single DDS architecture, but it has a normalized data store in front of the DDS. The NDS is in third normal relational form or higher. The purpose of having NDS is twofold. First, it integrates data from several source systems. Second, it is able to load data into several DDSs. Unlike the single DDS architecture, in the NDS + DDS architecture you can have several DDSs. Figure 9 shows the NDS + DDS data flow architecture.

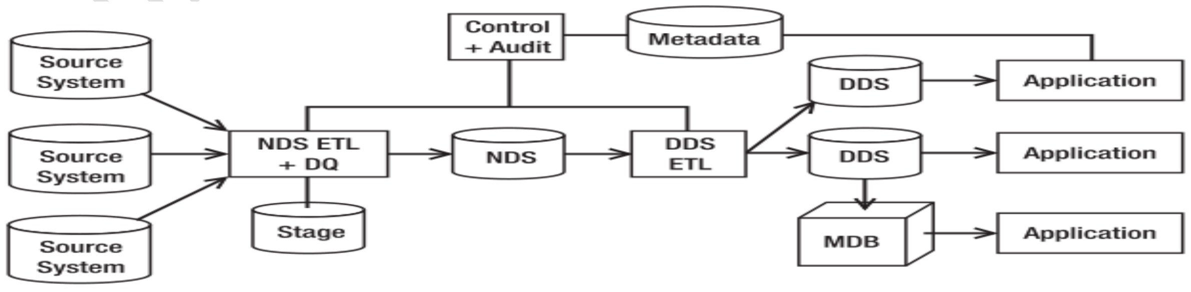

Figure 9 - NDS + DDS data flow architecture

In the NDS + DDS architecture, NDS is the master data store, meaning NDS contains the complete data sets, including all historical transaction data and all historical versions of master data. Historical transaction data means the business transactions that happened in the past. Data from every single year is stored in the NDS. The DDS, on the other hand, is not the master data store. It may not contain all transaction data for every single year. NDS contains all historical versions of master data. If there is a change in master data, the attributes are not overwritten by new values. The new values are inserted as a new record, and the old version (the old row) is kept in the same table.

Similar to an OLTP source system, there are two types of tables in the NDS: transaction tables and master tables. A transaction table is a table that contains a business transaction or business event. A master table is a table that contains the people or objects involved in the business event. A sales order table is an example of a transaction table. A product table is an example of a master table. The NDS transaction tables are the source of data for the DDS fact table. In other words, the fact tables in the DDS are populated from the transaction tables in the NDS. The NDS master tables are the source of data for DDS dimension tables. That is, the dimension.

Tables in the DDS are populated from the master tables in the NDS. NDS is an internal data store, meaning it is not accessible by the end user or the end-user applications. Data from NDS is loaded into DDSs in dimensional format, and the end users access these DDSs. The only application that is able to update NDS is the NDS ETL. No other application should be able to update NDS.

The principles about the stage, control, audit, and metadata discussed earlier are also applicable here. Some entities can fly through directly to NDS without being staged to disk first. In that case, the data integration/transformation is performed online in the memory of the ETL server. The ETL reads the data from the source system, transforms or integrates the data in memory, and writes the data to NDS directly without writing anything to the stage.

Data transformation is converting, calculating, or modifying the data to suit the target database. Data integration is combining the same record from several different source systems into one record or combining different attributes of master data or transaction data. In the NDS + DDS architecture, the DDS ETL that loads data into the DDS is simpler than the one in the single DDS architecture because the data in the NDS is already integrated and cleaned. In some cases, the DDS ETL needs to feed the data only incrementally to the DDS without any transformation. Most, if not all, fact table calculations have been done in the NDS.

The data in the NDS is uploaded to the DDSs. The flexibility of using a centralized NDS is that you can build a DDS that you need at any time with the scope of data as required. The ability to build a new DDS at any time is useful to satisfy requirements from projects involving data analysis. The ability to set the scope of data when building a new DDS means you can pick which tables, columns, and rows you want to transfer to the new DDS. For example, you can build a DDS containing only a customer profitability data mart (one fact table and all its dimensions) that contains only the last three months of data.

To populate a new DDS, you can use the existing DDS ETL. You just need to point the ETL to the new DDS. If you build the DDS ETL correctly, you can rebuild any DDS quickly, in other words, the only time we need is the time to run the ETL. You don't have to spend days or weeks to build a new ETL package to load the new DDS. To get this flexibility, the DDS ETL needs to be parameterized; that is, the date range, the fact tables, and the dimensions to be copied across are all set as parameters that can easily be changed to point to a new DDS. The database connection details are also written as parameters. This enables you to point the ETL to another database.

NDS - ODS architecture

In NDS + ODS architecture, you can have several DDSs. But there is one DDS you have to build and maintain: the one that contains all the fact tables and all the dimensions. This one is sort of obligatory; all other DDSs are optional—you build them as you need them. You need to have this one obligatory DDS because it contains a complete set of data warehouse data and is used by all BI applications that require dimensional data stores.

The NDS tables use surrogate keys and natural keys. A surrogate key is the identifier of the master data row within the data warehouse. In the DDS, the surrogate key is used as the primary key of dimension tables. The surrogate key is a sequential integer, starting from 0. So, it is 0, 1, 2, 3… and so on. Using the surrogate key, you can identify a unique record on a dimension table. Surrogate keys also exist in the fact tables to identify the dimension attributes for a particular business transaction. Surrogate keys are used to link a fact table and the dimension tables. For example, using surrogate keys, you can find out the details of the customer for a particular order. In the NDS + DDS architecture, the surrogate keys are maintained in the NDS, not in the DDS.

A natural key is the identifier of the master data row in the source system. When loading data from the stage to NDS, you need to translate the natural key from the source system to a data warehouse surrogate key. You can do this by looking up the surrogate key in the NDS for each natural key value from the source system. If the natural key exists in the NDS, it means the record already exists in NDS and needs to be updated. If the natural key doesn't exist in the NDS, it means the record does not exist in the NDS and needs to be created. Only internal administrative applications access the NDS directly. These are usually applications that verify the data loaded into NDS, such as data quality routines that check NDS data against certain firewall rules. End user applications, such as BI reports, access the DDS (dimensional model) and some applications, such as OLAP applications, access the multidimensional databases that are built from the DDSs. You need to understand what kind of data store is required by each application to be able to define the data flow architecture correctly.

The main advantage of this architecture is that you can easily rebuild the main DDS; in addition, you can easily build a new, smaller DDS. This is because the NDS is the master data store, containing a complete set of data, and because the DDS ETL is parameterized. This enables you to create a separate static data store for the purpose of specific analysis. The second advantage is that it is easier to maintain master data in a normalized store like the NDS and publish it from there because it contains little or no data redundancy and so you need to update only one place within the data store.

The main disadvantage is that it requires more effort compared to the single DDS architecture because the data from the stage needs to be put into the NDS first before it is uploaded into the DDS. The effort to build ETL becomes practically double because you need to build two ETL sets, while in single DDS it is only one. The effort for data modeling would be about 50 percent more because you need to design three data stores, whereas in single DDS you have two data stores.

The NDS + DDS architecture offers good flexibility for creating and maintaining data stores, especially when creating a DDS. The NDS is a good candidate for an enterprise data warehouse. It contains a complete

set of data, including all versions of master data, and it is normalized, with nearly no data redundancy, so data updates are more easily and quickly compared to the dimensional master data store. It also contains both source systems' natural keys and data warehouse surrogate keys, enabling you to map and trace data between the source systems and data warehouse.
You would use an NDS + DDS architecture when you need to make several DDSs for different purposes containing a different set of data and when you need to integrate data in a normalized form and use the integrated data outside of the dimensional data warehouse.

5.2.2 ODS + DDS
This architecture is similar to an NDS + DDS architecture, but it has an ODS in the place of the NDS. Like NDS, ODS is in third normal form or higher. Unlike the NDS, the ODS contains only the current version of master data; it has no historical master data. The structure of its entities is like an OLTP database. The ODS has no surrogate keys. The surrogate keys are maintained in the DDS ETL. The ODS integrates the data from various source systems. The data in the ODS is cleaned and integrated. The data flowing into the ODS has already passed the DQ screening.

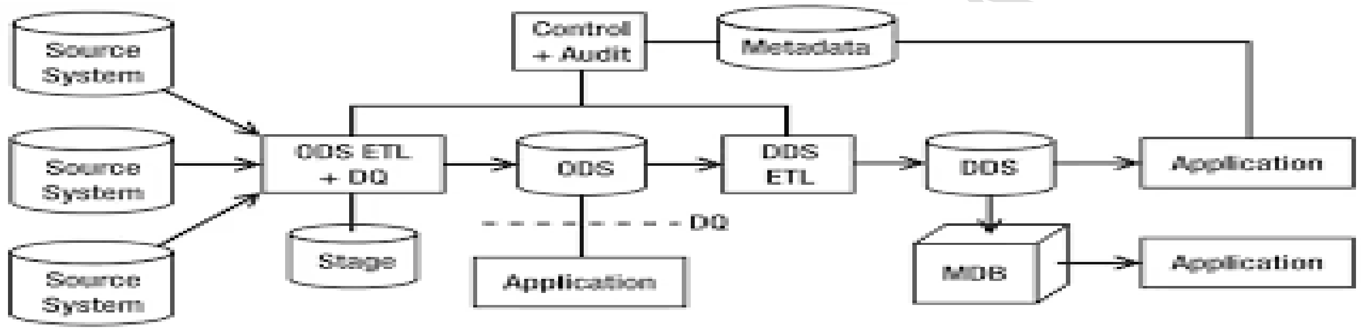

Figure 10 - ODS + DDS data flow architecture

Like NDS, ODS contains transaction tables and master tables. The transaction tables contain business events, and the master tables contain the people or objects involved in the business events. The fact tables in the DDS are populated from the transaction tables in the ODS. The dimension tables in the DDS are populated from the master tables in the ODS. Unlike NDS, ODS's master tables contain only the current version of master data.

ODS does not contain the historical versions of master data. Unlike NDS, which is an internal data store, ODS is a hybrid data store. This means ODS is accessible by the end users and end-user applications. In NDS + DDS applications, NDS is not accessible by the end users and end-user applications. Unlike NDS, ODS is updatable.

End-user applications can retrieve data from the ODS, but they can also update the ODS. To ensure the quality of the data in the ODS, data quality rules are also applied to these updates.

The end-user application must not update the data coming from the source systems; it can update only the data that itself generates to complement the source systems' data. If the ODS is used to support a CRM customer support application, data such as status and comments can be written on ODS directly, but all the customer data is still from the source systems.

In the ODS + DDS architecture, the DDS is the master data store. Unlike an NDS + DDS architecture, in an ODS + DDS architecture you have only one DDS. The DDS contains a complete set of fact tables and the dimension tables. The DDS contains both the current version and all historical versions of master data.

The principles about the stage, control, audit, and metadata discussed in regard to the single DDS architecture are also applicable here. Some entities can fly through directly to the ODS without being staged first. Integration and transformation happen in the memory of the ETL server. The DDS ETL is simpler than the one in the single DDS architecture because the data in the ODS is already integrated and cleaned. In many cases, it is literally feeding DDS incrementally without any transformation. Most, if not all, fact table calculations have been done in the ODS.

In the ODS + DDS architecture, applications can access the data warehouse in three places in three different formats: those that need the data in normalized form can access the ODS, those that need the data in relational dimensional format can access the DDS, and those that need the data in multidimensional format can access the MDBs.

This architecture has these advantages:

The third normal form is slimmer than the NDS because it contains only current values. This makes the performance of both ODS ETL and DDS ETL better than the ones in the NDS + DDS architecture.

Like the NDS + DDS architecture, in the ODS + DDS architecture you have a central place to integrate, maintain, and publish master data.

The normalized relational store is updatable by the end-user application, making it capable of supporting operational applications at the transaction level.

The disadvantage of this architecture is that to build a new, small DDS (say, 2007 Q4 sales data), you need to get it from the main DDS and cannot utilize the existing DDS ETL to do that.

You need either to write custom queries (in other words, create table from select), which is not preferred because of standardization and consistency reasons, or to build a new ETL, which is not preferred either because of the effort, especially if it is a one-off, throwaway thing. You would use an ODS + DDS architecture when you need only a one dimensional data store and you need a centralized, normalized data store to be used for operational purposes such as CRM. The ODS contains detailed, current-valued, integrated data that is useful for transactional queries.

QUESTION BANK

Q-1: 1 Mark Question

1. Give Full form of MDM.(July-2021,2020) Master Data Management.
2. Give the full form ROLAP. (July-2021,2019) Relational Online Analytical Processing.
3. Give full form of DDS. (July-2021,2020,2019) "Dimensional Data Store."
4. Give the full form of CRM. (July-2021,2019,2023) Customer Relationship Management.
5. Give Full form of SOA. (July-2021,2019) Service-Oriented Architecture
6. What is DWH? (July-2021) DWH stands for "Data Warehouse." A data warehouse is
7. What is stage in data store? (July-2021) a "stage" refers to a temporary storage area or data store where raw data is initially loaded before being processed and integrated into the data warehouse.
8. What is third party data? (July-2021)
9. Give the full form of CRM. (April–2020)
10. OLAP stands for (BKNMU - June-2021,2019) Online Analytical Processing
11. ERP Stand for___ (BKNMU - april-2020) Enterprise Resource Planning
12. NDS Stand for____ (BKNMU - april-2020) Normalized Data Store.
13. What is Multidimensional data store? (BKNMU - april-2020)
14. ODS stand for___. Operational Data Store
15. What is data warehousing? (BKNMU - april-2022)
16. SOA stands for ________. (BKNMU - april-2022)
17. Write full form of CRM. (BKNMU - april-2022)
18. EDW stands for_____ (2023) Enterprise Data Warehouse.
19. OLTP stands for_____ (2023) online transaction processing

Q-2: 3 Mark Question

1. What is Data Mining? (July-2021)(2023)
2. Give the advantages of Data warehousing. (July-2021,2020)
3. Explain Data warehouse types. (July-2021,2020)
4. Explain Characteristics of Data Warehouse. (July-2021,2020,2019)
5. What is Normalized and Demoralized Database? (April–2020,2019)
6. Explain Data Warehousing Today. (April–2020)
7. Explain future trends of data warehouse. (BKNMU - June-2021,2019)
8. What is Data Mart? (BKNMU - april-2020)
9. Different between OLTP v/s OLAP. (BKNMU - april-2020)
10. What is data warehouse? How it is works? (BKNMU - april-2022)
11. Differentiate OLTP v/s OLAP. (BKNMU - april-2022)
12. Discuss types of data warehouse. (BKNMU - april-2022)
13. Define data cube, virtual warehouse and unstructured data. (BKNMU - april-2022)
14. Write advantages and disadvantages of data warehouse. (2023)
15. Explain dataflow architecture of data warehouse.(2023)
16. Discuss CRM in detail. (2023)

Q-3: 5 Mark Question

1. Explain Data Flow Architecture. (July-2021,2019)
2. Explain IOT and Cloud in the term of data warehousing. (July-2021)
3. Explain Data Warehouse Architecture. (April–2020)
4. What is data warehouse? Explain its characteristics. (BKNMU- June-2021,2019,2023)
5. Explain Structure data, Semi- Structure data and unstructured data. (BKNMU- april-2020)
6. Explain architecture of data warehouse. (BKNMU- april-2022)
7. Explain future trends of data warehouse. (BKNMU- april-2022,2023)