CHAPTER 1

Introduction to Data Warehousing

In this chapter, I will discuss what a data warehouse is, how data warehouses are used today, and the future trends of data warehousing.

I will begin by defining what a data warehouse is. Then I'll walk you through a diagram of a typical data warehouse system, discussing its components and how the data flows through those components. I will also discuss the simplest possible form of a data warehouse. After you have an idea about what a data warehouse is, I will discuss the definition in more detail. I will go through each bit of the definition individually, exploring that bit in depth. I will also talk about other people's definitions.

Then, I will move on to how data warehouses are used today. I will discuss business intelligence, customer relationship management, and data mining as the popular applications of data warehousing. I will also talk about the role of master data management and customer data integration in data warehousing.

Finally, I will talk about the future trends of data warehousing, such as unstructured data, search, real-time data warehouses, and service-oriented architecture. By the end of this chapter, you will have a general understanding of data warehousing.

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.

In the next few pages, I will discuss each of the italicized terms in the previous paragraph one by one. But for now, I'll walk you through a diagram of a data warehouse system, discussing it component by component and how the data flows through those components. After this short walk-through, I will discuss each term in the previous definition, including the differences between dimensional and normalized data stores, why you store the data in the data store, and why data warehouses are updated in batches. Figure 1-1 shows a diagram of a data warehouse system, including the applications.

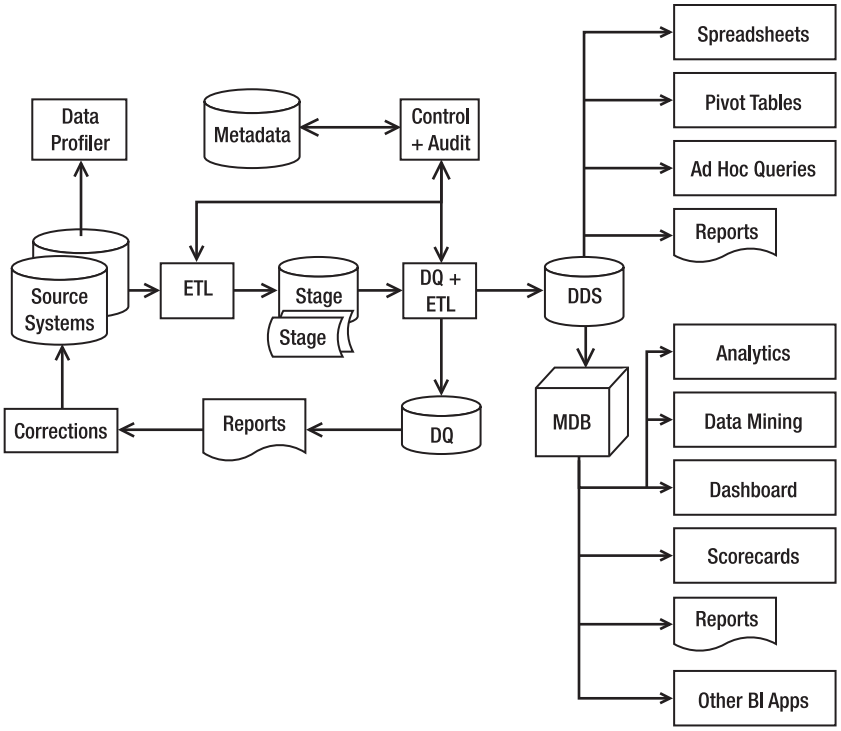

Figure 1-1. *A diagram of a data warehouse system*

Let's go through the diagram in Figure 1-1, 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 1-2 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 × 100 × 100 = 1 million 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.

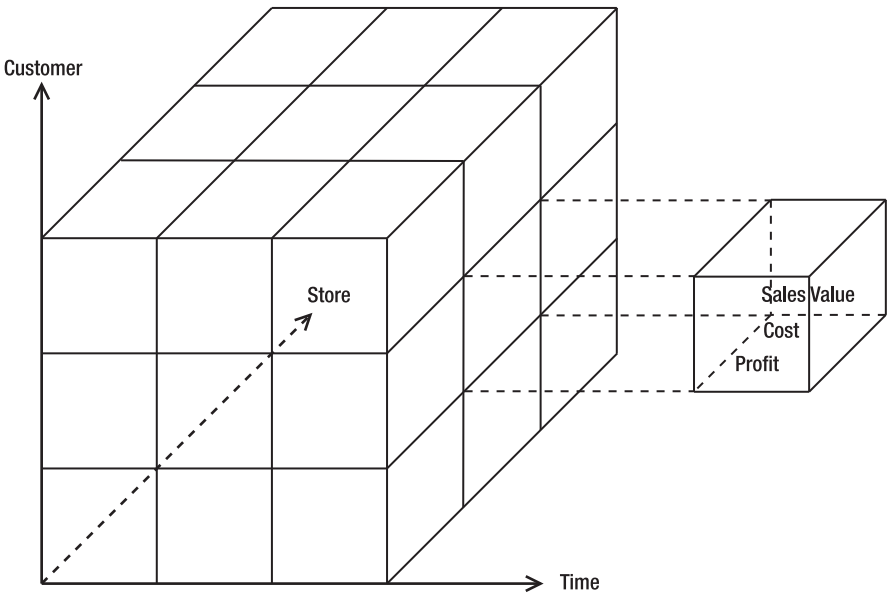

Figure 1-2. *A cube with three dimensions*

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 database, 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 1-3.

Figure 1-3. *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 system anymore. In Figure 1-3, 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. I will be discussing data warehouse architecture in Chapter 2.

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. Again, in the next few pages, I will discuss each italicized term in the following data warehouse definition one by one: 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.

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.

Most ETL systems also have mechanisms to clean the data from the source system before putting it into the warehouse. Data cleansing is the process of identifying and correcting dirty data. This is implemented using data quality rules that define what dirty data is. After the data is extracted from the source system but before the data is loaded into the warehouse, the data is examined using these rules. If the rule determines that the data is correct, then it is loaded into the warehouse. If the rule determines that the data is incorrect, then there are three options: it can be rejected, corrected, or allowed to be loaded into the warehouse. Which action is appropriate for a particular piece of data depends on the situation,

the risk level, the rule type (error or warning), and so on. I will go through data cleansing and data quality in more detail in Chapter 9.

There is another alternative approach to ETL, known as extract, load, and transform (ELT). In this approach, the data is loaded into the data warehouse first in its raw format. The transformations, lookups, deduplications, and so on, are performed inside the data warehouse. Unlike the ETL approach, the ELT approach does not need an ETL server. This approach is usually implemented to take advantage of powerful data warehouse database engines such as massively parallel processing (MPP) systems. I will be discussing more about the ELT approach in Chapter 7.

Consolidates Data

A company can have many transactional systems. For example, a bank may use 15 different applications for its services, one for loan processing, one for customer service, one for tellers/cashiers, one for ATMs, one for bonds, one for ISA, one for savings, one for private banking, one for the trading floor, one for life insurance, one for home insurance, one for mortgages, one for the call center, one for internal accounts, and one for fraud detection. Performing (for example) customer profitability analysis across these different applications would be very difficult.

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 warehouse 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:

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 B does not have the address3 field and the country field. In system A, an order may have two levels—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 levels in 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.

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.

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 traffic* may refer to unique web site visitors, whereas in system B it means nonunique 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.

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 measure. 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 time 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 converting, you need to know when the transaction happened.

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. (I will talk more about data matching in Chapter 9.)

When building the data warehouse, you have to deal with all these data integration issues.

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 system 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. For example, if the business needs the product profitability report once a week,

then the data from various source systems needs to be consolidated at least once a week. Another example is when a company states to its customer that it will take 24 hours to cancel the marketing subscriptions. Then the data in the CRM data warehouse needs to be updated a few times a day; otherwise, you risk sending marketing campaigns to customers who have already canceled their subscriptions.

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 (shown in Figure 1-4), a snowflake schema, and a galaxy schema. In a star schema, a dimension does not have a subtable (a subdimension). In a snowflake schema, a dimension can have a subdimension. The purpose of having a subdimension is to minimize redundant data. A galaxy schema is also known as a *fact constellation 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.

■ **Note** A data store can be physically implemented as more than one database, in other words, two databases, three databases, and so on. The contrary is also true: two or more data stores can be physically implemented as one database. When designing the physical layer of the data store, usually you tend to implement each data store as one database. But you need to consider physical database design factors such as the physical data model, database platform, storage requirement, relational integrity, and backup requirements when determining whether you will put several data stores in one database or split a data store into several databases. Putting one data store in one database is not always the best solution. (I will discuss physical database design in Chapter 6.)

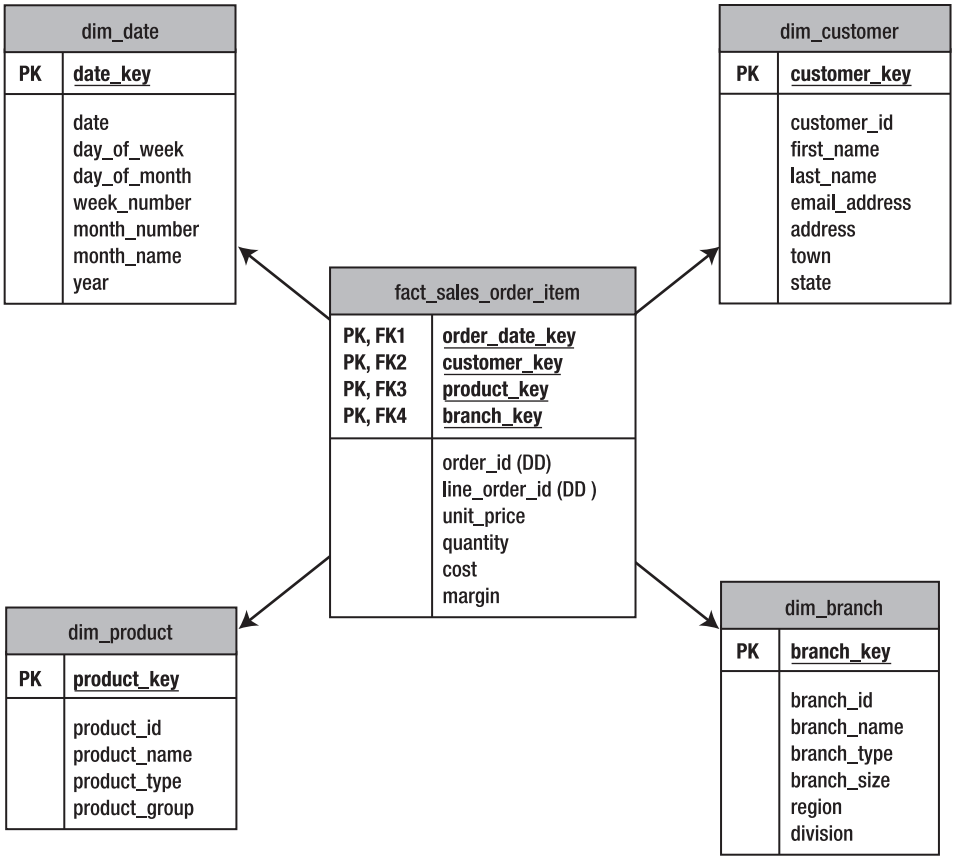

Figure 1-4. *Star schema dimensional data store*

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. I will discuss the normalization process and normalization rules in Chapter 5.

Figure 1-5 shows an example of a normalized data store. It is the normalized version of the same data as displayed in Figure 1-4.

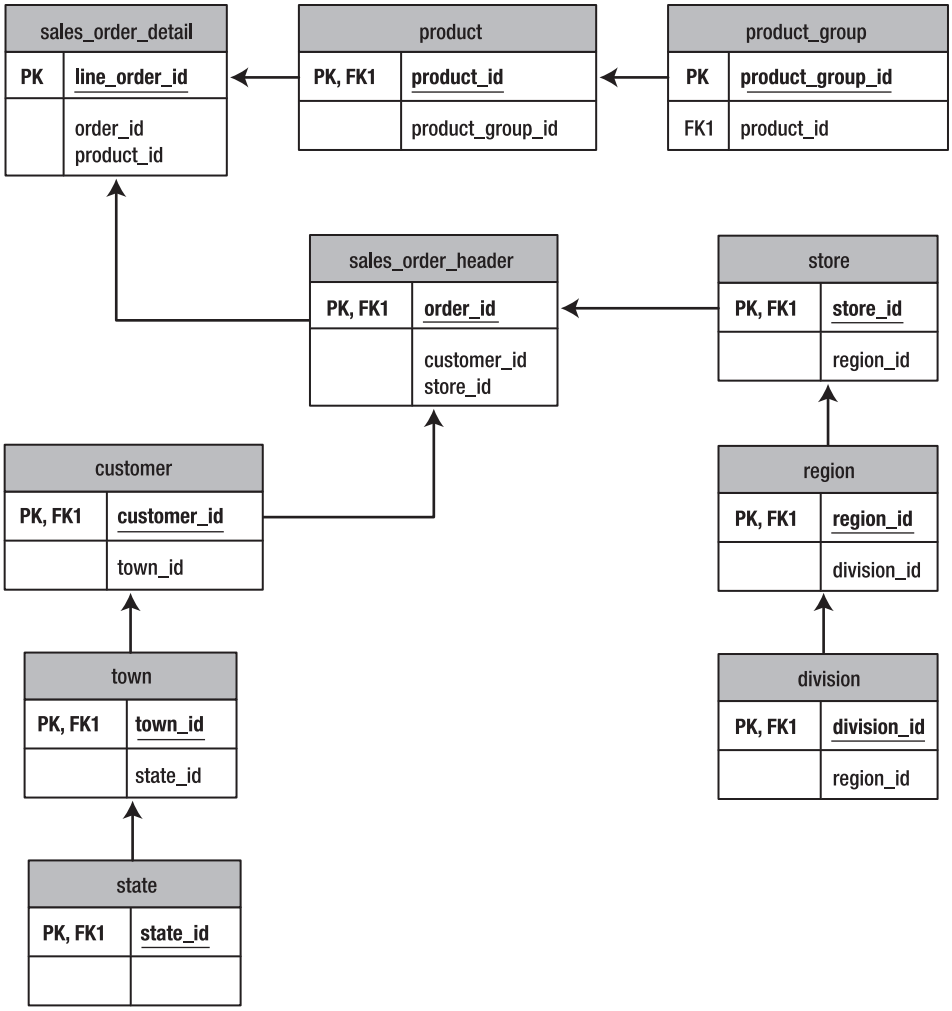

Figure 1-5. *Normalized data store*

A dimensional data store is a better format to store data in the warehouse for the purpose of querying and analyzing data than a normalized data store. This is because it is simpler (one level deep in all directions in star schema) and gives better query performance. A normalized data store is a better format to integrate data from various source systems, especially in third normal form and higher. This is because there is only one place to update without data redundancy like in a dimensional data store.

The normalized data store is usually used for an enterprise data warehouse; from there the data is then loaded into dimensional data stores for query and analysis purposes. Figure 1-6 shows a data warehouse system with a normalized data store used for an enterprise data warehouse (labeled as "EDW" in the figure).

Some applications run on a DDS, that is, a relational database that consists of tables with rows and columns. Some applications run on a multidimensional database that consists of cubes with cells and dimensions. I will go through cubes and multidimensional database concepts later in this chapter and in Chapter 12.

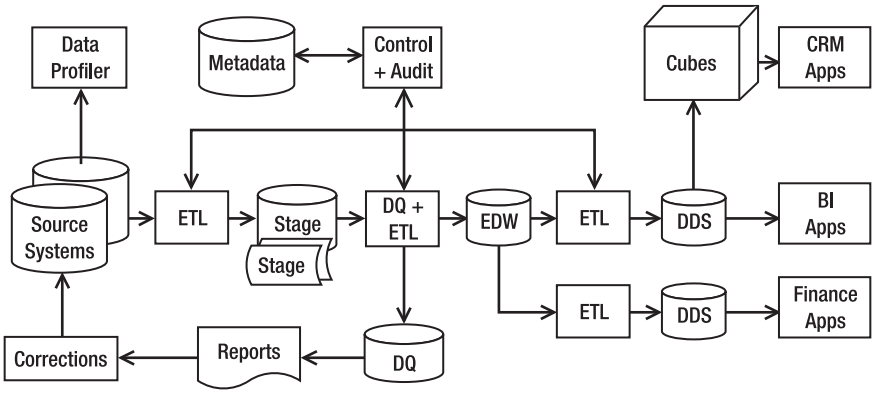

Figure 1-6. *A data warehouse system that uses an enterprise data warehouse*

I will discuss more about dimensional data stores, dimensional schemas, conformed dimensions, normalized data stores, the normalization process, and third normal form in Chapter 5 when I talk about data modeling.

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.

You may want to keep the records for, say, two years, in case the customer queries their orders, but you don't want to keep ten years worth of data on the active system, because that slows the system down. Some regulations (which differ from country to country) require you to keep data for up to five or seven years, such as for tax purposes or to adhere to stock exchange regulations. But this does not mean you must keep the data on the active system. You can archive it to offline media. That's what a typical transaction system does: it keeps only two to three years of data in the active system and archives the rest either to an offline media or to a secondary read-only system/database.

A data warehouse, on the other hand, stores years and years of history in the active system. I have seen ten years of historical data in a data warehouse. The amount of historical data to store in the data warehouse depends on the business requirements. Data warehouse tables can become very large. Imagine a supermarket chain that has 100 stores. Each store welcomes 1,000 customers a day, each purchasing 10 items. This means $100 \times 1000 \times 10 = 1$ million sales

order item records every day. In a year, you will have 365 million records. If you store 10 years of data, you will have 3.65 billion records. A high volume like this also happens in the telecommunications industry and in online retail, especially when you store the web page visits in the data warehouse. Therefore, it is important for a data warehouse system to be able to update a huge table bit by bit, query it bit by bit, and back it up bit by bit. Database features such as table partitioning and parallel query would be useful for a data warehouse system. Table partitioning is a method to split a table by rows into several parts and store each part in a different file to increase data loading and query performance. Parallel query is a process where a single query is split into smaller parts and each part is given to an independent query-processing module. The query result from each module is then combined and sent back to the front-end application. I will go through parallel database features such as table partitioning in Chapter 6, when I discuss physical database design.

Most transaction systems store the history of the transactions but not the history of the master data such as products, customers, branches, and vehicles. When you change the product description, for example, in my experience most of the transaction systems update the old description with the new one; they do not store the old description. There are some exceptions, however; for example, some specialized applications such as medical and customer service applications store historical master data such as old customer attributes.

In a data warehouse, on the other hand, storing the history of the master data is one of the key features. This is known as a slowly changing dimension (SCD). A slowly changing dimension is a technique used in dimensional modeling for preserving historical information about dimensional data. In SCD type 2, you keep the historical information in rows; while in SCD type 3, you keep the historical information in columns. In SCD type 1, you don't keep the historical information. Please refer to Chapter 5 for more information about SCD.

Also related to history, a data warehouse stores a periodic snapshot of operational source systems. A *snapshot* is a copy of one or more master tables taken at a certain time. A periodic snapshot is a snapshot that is taken at a regular interval; for example, the banking industry takes snapshots of customer account tables every day. The data warehouse applications then compare the daily snapshots to analyze customer churns, account balances, and unusual conditions. If the size of a source system is, say, 100MB, then in a year you would have accumulated 37GB. Storing source system daily snapshots could have a serious impact on data warehouse storage, so you need to be careful.

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?"[1] Here is an example of a complex query: "Show me the names and revenue of all product lines that had a 10 percent loss or greater in Q3 FY 2006, categorized by outlet."

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 warehouse. 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.

---

1. Note: "How many customers do you have now?" is a simple question if you have only one application, but if you have 15 applications, it could be quite daunting.

If you refer once again to Figure 1-1, you can ask yourself this question: "Why do I need to get the data from the source system into the DDS and then query the DDS? Why don't I query the source system directly?"

For the purpose of simple querying and reporting, you usually query the source system directly. But for conducting heavy analysis such as customer profitability, predictive analysis, "what if?" scenarios, slice and dice analytical exercises, and so on, it is difficult to do it on the source system.

Here's why: the source system is usually a transactional system, used by many users. One important feature of a transactional system is the ability to allow many users to update and select from the system at the same time. To do so, it must be able to perform a lot of database transactions (update, insert, delete, and select) in a relatively short period of time. In other words, it should be able to perform database transactions very quickly. If you stored the same piece of data—say, unit price—in many different places in the system, it would take a long time to update the data and to maintain data consistency. If you stored it in only one place, it would be quicker to update the data, and you wouldn't have to worry about maintaining data consistency between different places. Also, it would be easier to maintain the concurrency and locking mechanism to enable many people to work together in the same database. Hence, one of the fundamental principles of a transaction system is to remove data redundancy.

Performing a complex query on a normalized database (such as transactional systems) is slower than performing a complex query on a denormalized database (such as a data warehouse), because in a normalized database, you need to join many tables. A normalized database is not suitable to be used to load data into a multidimensional database for the purpose of slicing-and-dicing analysis. Unlike a relational database that contains tables with two dimensions (rows and columns), a multidimensional database consists of cubes containing cells with more than two dimensions. Then each cell is mapped to a member in each dimension. To load a multidimensional database from a normalized database, you need to do a multijoin query to transform the data to dimensional format. It can be done, but it is slower. I will go through normalization in more detail in Chapter 5 and data loading in Chapter 8.

The second reason why you don't query the source systems directly is because a company can have many source systems or front-office transactional systems. So, by querying a source system, you get only partial data. A data warehouse, on the other hand, consolidates the data from many source systems, so by querying the data warehouse, you get integrated data.

Business Intelligence

Business intelligence is a collection of activities to understand business situations by performing 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.

Examples of business intelligence are the following:

- Business performance management, including producing key performance indicators such as daily sales, resource utilization, and main operational costs for each region, product line, and time period, as well as their aggregates, to enable people to take tactical actions to get operational performance on the desired tracks.
- Customer profitability analysis, that is, to understand which customers are profitable and worth keeping and which are losing money and therefore need to be acted upon. The key to this exercise is allocating the costs as accurately as possible to the smallest unit of business transaction, which is similar to activity-based costing.
- Statistical analysis such as purchase likelihood or basket analysis. Basket analysis is a process of analyzing sales data to determine which products are likely to be purchased or ordered together. This likelihood is expressed in terms of statistical measures such as support and confidence level. It is mainly applicable for the retail and manufacturing industries but also to a certain degree for the financial services industry.
- Predictive analysis such as forecasting the sales, revenue, and cost figures for the purpose of planning for next year's budgets and taking into account other factors such as organic growth, economic situations, and the company's future direction.

According to the depth of analysis and level of complexity, in my opinion you can group business intelligence activities into three categories:

- Reporting, such as key performance indicators, global sales figures by business unit and service codes, worldwide customer accounts, consolidated delivery status, and resource utilization rates across different branches in many countries
- OLAP, such as aggregation, drill down, slice and dice, and drill across
- Data mining, such as data characterization, data discrimination, association analysis, classification, clustering, prediction, trend analysis, deviation analysis, and similarity analysis

Now let's discuss each of these three categories in detail.

Reporting

In a data warehousing context, a *report* is a program that retrieves data from the data warehouse 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.

The reports are built according to the functional specifications. They display the DDS data required by the business user to analyze and understand business situations. The most common form of report is a tabular form containing simple columns. There is another form of report known as *cross tab* or *matrix*. These reports are like Excel pivot tables, where one data attribute becomes the rows, another data attribute becomes the columns, and each cell on the report contains the value corresponding to the row and column attributes.

Data warehouse reports are used to present the business data to users, but they are also used for data warehouse administration purposes. They are used to monitor data quality, to monitor the usage of data warehouse applications, and to monitor ETL activities.

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).

An example of OLAP is analyzing the effectiveness of a marketing campaign initiative on certain products by measuring sales growth over a certain period. Another example is to analyze the impact of a price increase to the product sales in different regions and product groups at the same period of time.

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 packages by analyzing call patterns (telecommunication industry), market basket analysis (retail industry), customer risk profiling (insurance industry), usage monitoring (energy and utilities), and machine service times (manufacturing industry).

I will discuss the implementation of data warehousing for business intelligence in Chapter 13.

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.

Data warehouses are also used for customer relationship management (CRM). CRM is a set of activities performed by an organization (business and nonbusiness) to manage and conduct analysis about their customers, to keep in contact and communicate with their customers, to attract and win new customers, to market product and services to their customers, to conduct transactions with their customers (both business and nonbusiness transactions), to service and support their customers, and to create new ideas and new products or services for their customers. I will discuss the implementation of data warehouses for CRM later in this chapter and in Chapter 14.

Data warehouses are also used in *web analytics*. Web analytics is the activity of understanding the behavior and characteristics of web site traffic. This includes finding out the number of visits, visitors, and unique visitors on each page for each day/week/month; referrer sites; typical routes that visitors take within the site; technical characteristics of the visitors' browsers; domain and geographical analysis; what kind of robots are visiting; the exit rate of each page; and the conversion rate on the checkout process. Web analytics are especially important for online businesses.

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.

The reason for not allowing users to update or delete data in the data warehouse is to maintain data consistency so you can guarantee that the data in the data warehouse will be consistent with the operational source systems, such as if the data warehouse is taking data from two source systems, A and B. System A contains 11 million customers, system B contains 8 million customers, and there are 2 million customers who exist in both systems. The data warehouse will contain 17 million customers. If the users update the data in the data warehouse (say, delete 1 million customers), then it will not be consistent with the source systems. Also, when the next update comes in from the ETL, the changes that the users made in the warehouse will be gone and overwritten.

The reason why data warehouses are updated in batches rather than in real time is to create data stability in the data warehouse. You need to keep in mind that the operational source systems are changing all the time. Some of them change every minute, and some of them change every second. If you allow the source system to update the data warehouse in real time or you allow the users to update the data warehouse all the time, then it would be difficult to do some analysis because the data changes every time. For example, say you are doing a drilling-down exercise on a multidimensional cube containing crime data. At 10:07 you notice that the total of crime in a particular region for Q1 2007 is 100. So at 10:09, you drill down by city (say that region consists of three cities: A, B, and C), and the system displays that the crime for city A was 40, B was 30, and C was 31. That is because at 10:08 a user or an ETL added one crime that happened in city C to the data warehouse. The drilling-down/summing-up exercise will give inconsistent results because the data keeps changing.

The second reason for updating the data warehouse in batches rather than in real time is the performance of the source system. Updating the data warehouse in real time means that the moment there is an update in the source systems, you update the data warehouse immediately, that is, within a few seconds. To do this, you need to either

- install database triggers on every table in the source system or
- modify the source system application to write into the data warehouse immediately after it writes to the source system database.

If the source system is a large application and you need to extract from many tables (say 100 or 1,000 tables), then either approach will significantly impact the performance of the

source system application. One pragmatic approach is to do real-time updates only from a few key tables, say five tables, whilst other tables are updated in a normal daily batch. It is possible to update the data warehouse in real time or in near real time, but only for a few selected tables.

In the past few years, real-time data warehousing has become the trend and even the norm. Data warehouse ETL batches that in the old days ran once a day now run every hour, some of them every five minutes (this is called a *mini-batch*). Some of them are using the push approach; that is, rather than pulling the data into the warehouse, the source system pushes the data into the warehouse. In a push approach, the data warehouse is updated immediately when the data in the source system changes. Changes in the source system are detected using database triggers. In a pull approach, the data warehouse is updated at certain intervals. Changes in the source system are detected for extraction using a timestamp or identity column. (I will go through data extraction in Chapter 7.)

Some approaches use messaging and message queuing technology to transport the data asynchronously from various source systems into the data warehouse. Messaging is a data transport mechanism where the data is wrapped in an envelope containing control bits and sent over the network into a message queue. A message queue (MQ) is a system where messages are queued to be processed systematically in order. An application sends messages containing data into the MQ, and another application reads and removes the messages from the MQ. There are some considerations you need to be careful of when using asynchronous ETL, because different pieces of data are arriving at different times without knowing each other's status of arrival. The benefit of using MQ for ETL is the ability for the source system to send out the data without the data warehouse being online to receive it. The other benefit is that the source system needs to send out the data only once to an MQ so the data consistency is guaranteed; several recipients can then read the same message from the MQ. You will learn more about real-time data warehousing in Chapter 8 when I discuss ETL.

Other Definitions

I will close this section with data warehouse definitions from Bill Inmon and from Ralph Kimball, the fathers of data warehousing:

- According to Bill Inmon, a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions.[2]
- According to Ralph Kimball, a data warehouse is a system that extracts, cleans, conforms, and delivers source data into a dimensional data store and then supports and implements querying and analysis for the purpose of decision making.[3]

Both of them agree that a data warehouse integrates data from various operational source systems. In Inmon's approach, the data warehouse is physically implemented as a normalized data store. In Kimball's approach, the data warehouse is physically implemented in a dimensional data store.

In my opinion, if you store the data in a normalized data store, you still need to load the data into a dimensional data store for query and analysis. A dimensional data store is a better

2. See *Building the Data Warehouse, Fourth Edition* (John Wiley, 2005) for more information.
3. See *The Data Warehouse ETL Toolkit* (John Wiley, 2004) for more information.

format to store data in the warehouse for the purpose of querying and analyzing the data, compared to a normalized data store. A normalized data store is a better format to integrate data from various source systems.

The previous definitions are amazingly still valid and used worldwide, even after 16 years. I just want to add a little note. It is true that in the early days data warehouses were used mainly for making strategic management decisions, but in recent years, especially with real-time data warehousing, data warehouses have been used for operational purposes too. These days, data warehouses are also used outside decision making, including for understanding certain situations, for reporting purposes, for data integration, and for CRM operations.

Another interesting definition is from Alan Simon: the coordinated, architected, and periodic copying of data from various sources into an environment optimized for analytical and informational processing.[4]

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 CRM use data mining, business intelligence uses reporting, and BI and CRM also use data integration. In the following sections, I will describe the main usages, including business intelligence, CRM, and data mining. In Chapters 13 to 15, I will go through them again in more detail.

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.

Many companies have built business intelligence systems to help these processes, such as understanding business processes, making better decisions (through better use of information and through data-based decision making), and improving business performance (that is, managing business more scientifically and with more information). These systems help the business users get the information from the huge amount of business data. These systems also help business users understand the pattern of the business data and predict future behavior using data mining. Data mining enables the business to find certain patterns in the data and forecast the future values of the data.

Almost every single aspect of business operations now is touched by business intelligence: call center, supply chain, customer analytics, finance, and workforce. Almost every function is covered too: analysis, reporting, alert, querying, dashboard, and data integration. A lot of business leaders these days make decisions based on data. And a business intelligence tool running and operating on top of a data warehouse could be an invaluable support tool for

4. See `http://www.datahabitat.com/datawarehouse.html` for more information.

that purpose. This is achieved using reports and OLAP. Data warehouse reports are used to present the integrated business data in the data warehouse to the business users. OLAP enables the business to interactively analyze business transaction data stored in the dimensional data warehouse. I will discuss the data warehouse usage for business intelligence in Chapter 13.

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 customers 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 (preconsumption 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, personalized 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

Other functionality such as customer support and order-processing support are better served by an operational data store (ODS) or OLTP applications. An ODS is a relational, normalized data store containing the transaction data and current values of master data from the OLTP system. An ODS does not store the history of master data such as the customer, store,

and product. When the value of the master data in the OLTP system changes, the ODS is updated accordingly. An ODS integrates data from several OLTP systems. Unlike a data warehouse, an ODS is updatable.

Because an ODS contains integrated data from several OLTP systems, it is an ideal place to be used for customer support. Customer service agents can view the integrated data of a customer in the ODS. They can also update the data if necessary to complement the data from the OLTP systems. For example, invoice data from a finance system, order data from an ERP system, and subscription data from a campaign management system can be consolidated in the ODS.

I will discuss the implementation of data warehousing for customer relationship management in Chapter 14.

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 describing 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 databases (relational databases, hierarchical databases, dimensional databases, object-oriented databases, and multidimensional databases), files (spreadsheet files, XML files, and structured text files), unstructured or semistructured data (documents, e-mails, and XML files), stream data (plant measurements, temperatures and pressures, network traffic, and telecommunication 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 mining 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.

You can use data mining for various business and nonbusiness applications including the following:

- Finding out which products are likely to be purchased together, either by analyzing the shopping data and taking into account the purchase probability or by analyzing order data. Shopping (browsing) data is specific to the online industry, whilst order data is generic to all industries.
- In the railway or telecommunications area, predicting which tracks or networks of cables and switches are likely to have problems this year, so you can allocate resources (technician, monitoring, and alert systems, and so on) on those areas of the network.
- Finding out the pattern between crime and location and between crime rate and various factors, in an effort to reduce crime.

- Customer scoring in CRM in terms of loyalty and purchase power, based on their orders, geographic, and demographic attributes.
- Credit scoring in the credit card industry to tag customers according to attitudes about risk exposure, according to borrowing behaviors, and according to their abilities to pay their debts.
- Investigating the relationship between types of customers and the services/products they would likely subscribe to/purchase in an effort to create future services/products and to devise a marketing strategy and effort for existing services/products.
- Creating a call pattern in the telecommunication industry, in terms of time slices and geographical area (daily, weekly, monthly, and seasonal patterns) in order to manage the network resources (bandwidth, scheduled maintenance, and customer support) accordingly.

To implement data mining in SQL Server Analysis Services (SSAS), you build a mining model using the data from relational sources or from OLAP cubes containing certain mining algorithms such as decision trees and clustering. You then process the model and test how it performs. You can then use the model to create predictions. A prediction is a forecast about the future value of a certain variable. You can also create reports that query the mining models. I will discuss data mining in Chapter 13 when I cover the implementation of data warehousing for business intelligence.

Master Data Management (MDM)

To understand what master data management is, we need to understand what master data is first. In OLTP systems, there are two categories of data: transaction data and master data. Transaction data consists of business entities in OLTP systems that record business transactions consisting of identity, value, and attribute columns. Master data consists of the business entities in the OLTP systems that describe business transactions consisting of identity and attribute columns. Transaction data is linked to master data so that master data describes the business transaction.

Let's take the classic example of sales order–processing first and then look at another example in public transport.

An online music shop with three brands has about 80,000 songs. Each brand has its own web store: Energize is aimed at young people, Ranch is aimed at men, and Essence is aimed at women. Every day, thousands of customers purchase and download thousands of different songs. Every time a customer purchases a song, a transaction happens. All the entities in this event are master data.

To understand which entities are the transaction data and which entities are the master data, you need to model the business process. The business event is the transaction data. In the online music shop example, the business event is that a customer purchases a song. Master data consists of the entities that describe the business event. Master data consists of the answers of who, what, and where questions about a business transaction. In the previous example, the master data is customer, product, and brand.

Here's the second example: 1,000 bus drivers from 10 different transport companies are driving 500 buses around 50 different routes in a town. Each route is served 20 times a day, which is called a *trip*. The business process here is driving one trip. That is the transaction.

You have 50 × 20 = 1000 transactions a day. The master data consists of the business entities in this transaction: the driver, the bus, and the route. How about the companies? No, the company is not directly involved in the trip, so the company is not master data in this process. The company is involved in a trip through the buses and the drivers; each driver and each bus belong to a company. The company, however, may be a master data in another business process.

In the previous examples, you learned that to identify the transaction data and the master data in a business process, you need to identify what the business event is in the process first. Then, you identify the business entities that describe the business event.

Examples of master data are the supplier, branch, office, employee, citizen, taxpayer, assets, inventory, store, salespeople, property, equipment, time, product, tools, roads, customer, server, switch, account, service code, destination, contract, plants (as in manufacturing or oil refineries), machines, vehicles, and so on.

Now you are ready to learn about MDM, which is the ongoing process of retrieving, cleaning, storing, updating, and distributing master data. An MDM system retrieves the master data from OLTP systems. The MDM system consolidates the master data and processes the data through predefined data quality rules. The master data is then uploaded to a master data store. Any changes on master data in the OLTP systems are sent to the MDM system, and the master data store is updated to reflect those changes. The MDM system then publishes the master data to other systems.

There are two kinds of master data that you may not want to include when implementing an MDM system:

- You may want to exclude date and time. A date explains a business event, so by definition it is master data. A date has attributes such as month name, but the attributes are static. The month name of 01/11/2007 is November and will always be November. It is static. It does not need to be maintained, updated, and published. The attributes of a customer such as address, on the other hand, keep changing and need to be maintained. But the attributes of a date are static.

- You may want to exclude master data with a small number of members. For example, if your business is e-commerce and you have only one online store, then it may not be worth it to maintain store data using MDM. The considerations whether to exclude or include a small business entity as master data or not are the number of members and frequency of change. If the number of members is less than ten and the frequency of change is less than once a year, you want to consider excluding it from your MDM system.

Now let's have a look at one of the most widely used types of master data: products. You may have five different systems in the organization and all of them have a product table, and you need to make sure that all of them are in agreement and in sync. If in the purchase order system you have a wireless router with part number WAR3311N but in the sales order system you have a different part number, then you risk ordering the incorrect product from your supplier and replenishing a different product. There is also a risk of inaccuracy of the sales report and inventory control. It's the same thing with the speed, protocol, color, specification, and other product attributes; they also expose you to certain risk if you don't get them synchronized and corrected. So, say you have five different systems and 200,000 part numbers. How do you make sure the data is accurate across all systems all the time? That's where MDM comes into play.

An MDM system retrieves data from various OLTP systems and gets the product data. If there are duplicate products, the MDM system integrates the two records. The MDM system integrates the two records by comparing the common attributes to identify whether the two records are a match. If they are a match, survivorship rules dictate which record wins and which record loses. The winning record is kept, and the losing record is discarded and archived. For example, you may have two different suppliers supplying the same product but they have different supplier part numbers. MDM can match product records based on different product attributes depending on product category and product group. For example, for digital cameras, possible matching criteria are brand, model, resolution, optical zoom, memory card type, max and min focal length, max and min shutter speed, max and min ISO, and sensor type. For books, the matching is based on totally different attributes. MDM can merge two duplicate records into one automatically, depending on the matching rule and survivorship rules that you set up. It keeps the old data so that if you uncover that the merge was not correct (that is, they are really two different products), then MDM can unmerge that one record back into two records.

Once the MDM has the correct single version of data, it publishes this data to other systems. These systems use this service to update the product data that they store. If there is any update on any of these applications to product data, the master store will be updated, and changes will be replicated to all other systems.

The master data is located within OLTP systems. There are changes to this master data in the OLTP systems from time to time. These master data changes flow from OLTP systems to the master data store in the MDM system. There are two possible ways this data flow happens. The OLTP system sends the changes to the MDM system and the MDM system stores the changes in the master data store, or the MDM system retrieves the master data in OLTP systems periodically to identify whether there are any changes. The first approach where the OLTP system sends the master data changes to the MDM system is called a *push approach*. The second approach where the MDM system retrieves the master data from the OLTP systems periodically is called a *pull approach*. Some MDM systems use the push approach, and some MDM systems use the pull approach.

MDM systems have metadata storage. Metadata storage is a database that stores the rules, the structure, and the meaning of the data. The purpose of having metadata storage in an MDM system is to help the users understand the meaning and structure of the master data stored in the MDM system. Two types of rules are stored in the metadata storage: survivorship rules and matching rules. Survivorship rules determine which of the duplicate master data records from the OLTP system will be kept as the master data in the MDM system. Matching rules determine what attributes are used to identify duplicate records from OLTP systems. The data structure stored in metadata storage explains the attributes of the master data and the data types of these attributes.

MDM systems have a reporting facility that displays the data structure, the survivorship rules, the matching rules, and the duplicate records from OLTP systems along with which rule was applied and which record was kept as the master data. The reporting facility also shows which rules were executed and when they were executed.

The master data management system for managing product data such as this is known as *product information management* (PIM). PIM is an MDM system that retrieves product data from OLTP systems, cleans the product data, and stores the product data in a master data store. PIM maintains all product attributes and the product hierarchy in the master data

store and keeps it up-to-date by getting the product data changes from OLTP systems. PIM publishes the product data to other systems.

It is important to remember that the master data in the MDM system's master data store needs to be continuously updated. There is no point in synchronizing all the systems today when next week they will be out of sync again.

Now that you understand what MDM is, how does it relate to data warehousing? It relates for two reasons:

- If all the master data is clean, then it would make the tasks of creating and maintaining a data warehouse much easier. The data is already integrated and cleaned. All the data warehouse has to do is connect to this MDM store and synchronize all its dimensions with it. Of course, you would still have to maintain the surrogate key and SCD,[5] and so on, but the task is easier.

- You could use a data warehouse to store the master data. If you already have a data warehouse in place that pulls data from many different systems, cleans them, and integrates them in one place, then why don't you set up a service to publish this data and get those systems to subscribe or consume this service?

There is one more point that I need to discuss: *hierarchy*. Hierarchy is a structure where the master data is grouped into several categories, and the categories are classified into related levels. The top-level category has one or more second-level categories. Each of these second-level categories has one or more third-level categories. The same applies for further lower levels. The lowest-level categories have the individual master data as their members. Hierarchy is used to determine which attributes are applicable to which data. Hierarchy is also used when moving an item from one category to another.

For example, a product such as Canon EOS 400D belongs to a product category, such as SLR Digital Camera. A product category belongs to a product group, such as Digital Camera. A product group belongs to a product family, such as Camera. A product family belongs to a product type, such as Electronics. If you want to change the Camera product family to Video and Photo Camera and move the Video Camera product group under it, then you can do it in one place: the MDM hub. The MDM will then distribute this new hierarchy to other systems.

It is important that an MDM system is capable of maintaining hierarchy; otherwise, the product data in different applications would not be in sync, and you are exposed to those risks. And it is not an easy task; application A can have only three levels of product hierarchy, whereas application B has four levels.

Customer Data Integration

Customer data integration (CDI) is the MDM for customer data. CDI is the process of retrieving, cleaning, storing, maintaining, and distributing customer data. A CDI system retrieves customer data from OLTP systems, cleans it, stores it in a customer master data store, maintains the customer data, keeps it up-to-date, and distributes the customer data to other systems.

---

5. I will discuss surrogate keys and SCD in Chapter 5.

A CDI system enables you to have a cleaner, single, reliable version of customer data that other applications in the enterprise can use. This in turn can deliver business benefits such as increased customer satisfaction and better business analysis, and it reduces the complexity of the processes that use customer data. Of all the various kinds of master data management, CDI is the most widely used because every organization has customers. CDI provides clean integrated data for customer relationship management.

The importance of having a single accurate source of customer data is illustrated in the following examples. If in the customer service system you have Carol Goodman's date of birth as May 12, 1953, but in the policy management system it is May 12, 1973, then you are risking underwriting her life policy incorrectly. It's the same thing with the address and phone number, which also expose you to certain risks if you don't get them synchronized and corrected, just like the case with the product data discussed previously. CDI has different matching criteria compared to products, of course; for example, it can match customer records based on name, date of birth, Social Security number, telephone number, address, credit card number, and many other attributes. This time the criteria are not different from product group to product group, but rather universal. A human being is a human being, not like products that have different attributes for each product type.

Future Trends in Data Warehousing

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

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 semistructured 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, 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 plurals into singulars; eliminate meaningless words such as *ah*, *the*, *in* (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 properties 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.

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.

If the unstructured data contains links to other data—for example, hypertext documents such as XML or HTML—the search engine can utilize what it knows best in the Internet world, which is to use the links to score the relevance of the information when displaying the search results to the users. For searching images, the search engine could use the words around or

near the image to verify the title or tag of the image. This is also applicable for other multimedia types, such as video and music.

Search in the Internet world has been around for quite a while (10 to 15) years, but in data warehousing it is relatively new (2 to 3 years). If the information in the data warehouse is huge, often it is easier to search than to browse. For example, a top international organization with 100 branches located in 50 countries could have 100 data marts totaling 100 terabytes for sales, purchases, finance, inventory, human resources, and a huge amount of documents. This is especially true for new users who have not learned the structure of the data warehouse, what reports are available, and how to navigate around the data using the OLAP interface. With search, you have only one box, and you can type whatever you want. What user interface could be simpler than that?

That is why currently search has become a trend in business intelligence and data warehousing—first, because people collect their unstructured data in a huge amount, and second, because search is easier to use, even for structured data.

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 flexibility 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 1-1, a data warehouse 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, getService, getAccount, and getTransaction), 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 services 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.

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 talking 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.

Summary

This chapter introduced data warehousing. I showed many examples, with the hope that they would make the concepts easier to understand and would enrich your experience. I discussed a bit of today's situation so you know how it is now, and a bit of the future trends so you know what is coming. In the next chapter, I will discuss the architecture.