CHAPTER 2

Data Warehouse Architecture

A data warehouse system has two main architectures: the data flow architecture and the system architecture. The *data flow architecture* is about how the data stores are arranged within a data warehouse and how the data flows from the source systems to the users through these data stores. The *system architecture* is about the physical configuration of the servers, network, software, storage, and clients. In this chapter, I will discuss the data flow architecture first and then the system architecture.

Specifically, I will start this chapter by going through what the data flow architecture is, what its purpose is, and what its components are. Then I will discuss four data flow architectures: single DDS, NDS + DDS, ODS + DDS, and federated data warehouse. The first three use a dimensional model as their back-end data stores, but they are different in the middle-tier data store. The federated data warehouse architecture consists of several data warehouses integrated by a data retrieval layer.

I will then discuss the system architecture, which needs to be designed to suit the data flow architecture. So, the data flow architecture is determined first before you design the system architecture. The system architecture affects the actual performance of a data warehouse system delivered to the end users.

Toward the end of this chapter, I will introduce a case study that we will be using throughout the book. In subsequent chapters, I will show how to build a data warehouse for the company in the case study, Amadeus Entertainment Group. You can use this case study to learn all the data warehousing aspects covered in this book: architecture, methodology, requirements, data modeling, physical database design, ETL, data quality, metadata, reporting, multidimensional databases, BI, CRM, testing, and administration.

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 modeling*. I will

not discuss data architecture and data modeling in this chapter. I will discuss those topics in Chapter 5.

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.

I discussed the terms *relational*, *normalized*, *denormalized*, and *dimensional* in Chapter 1, but I'll repeat the definitions here briefly:

- 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 2-1 shows a data flow architecture with four data stores: stage, ODS, DDS, and MDB.

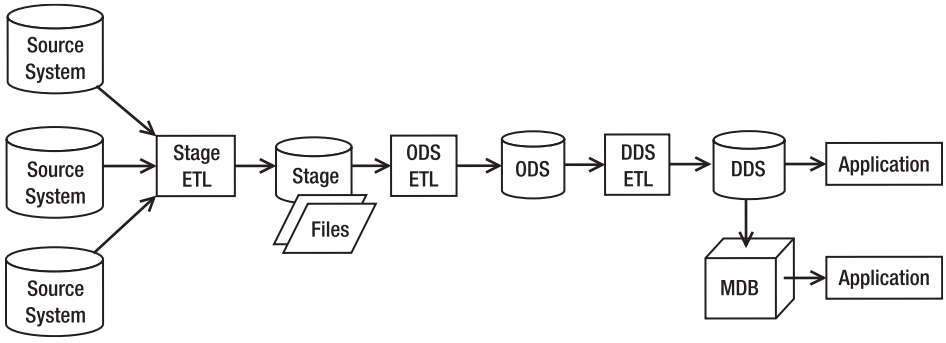

Figure 2-1. *A data flow architecture with a stage, ODS, DDS, and MDB*

The arrows in Figure 2-1 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 2-1 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. I will discuss more about ETL in Chapters 7 and 8.

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. I will discuss more about metadata in Chapter 10.

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. I will discuss more about data quality in Chapter 9.

Figure 2-2 shows a data flow architecture complete with the control system, the metadata, and the components of data quality processes.

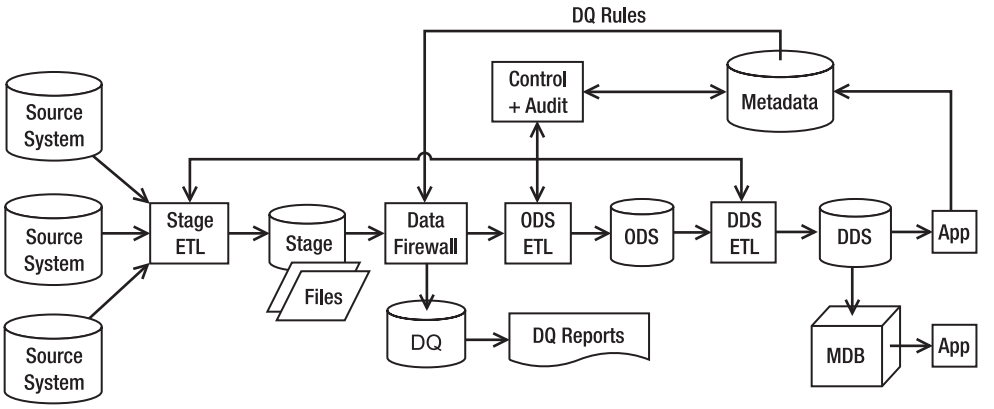

Figure 2-2. *A data flow architecture with control system, metadata, and data quality process*

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.

In the following four sections, I will discuss four data flow architectures with their advantages and disadvantages:

- The single DDS architecture has stage and DDS data stores.
- The NDS + DDS architecture has stage, NDS, and DDS data stores.
- The ODS + DDS architecture has stage, ODS, and DDS data stores.
- The federated data warehouse (FDW) architecture consists of several data warehouses integrated by a data retrieval layer.

These four data flow architectures are just examples. When building a data warehouse, you need to design the data flow architecture to suit the data requirements and data quality requirements of the project. These four architectures are by no means exhaustive; you should design your own data flow architecture. For example, is it possible to have a data flow architecture without a stage? Yes, it is possible, if you can do the transformation on the fly in the ETL server's memory, especially if you have a low data volume. Can you have a data flow architecture without a DDS? Yes, you can have stage + NDS data stores only if you want to have a normalized data warehouse rather than a dimensional one. Is it possible to have two DDSs? Yes, of course you can have multiple DDSs—you can have as many as you need.

Now I'll go through these four data flow architectures one by one.

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 2-3 describes a general model of this architecture.

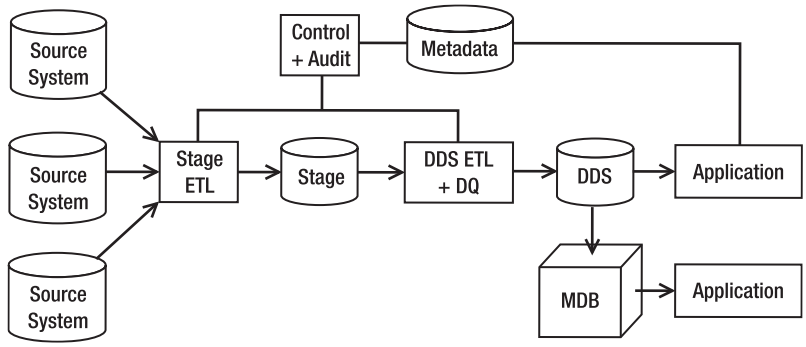

Figure 2-3. *Single DDS data warehouse architecture*

In Figure 2-3, 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 2-3 into one package, as pictured in Figure 2-4. In Figure 2-4, 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.

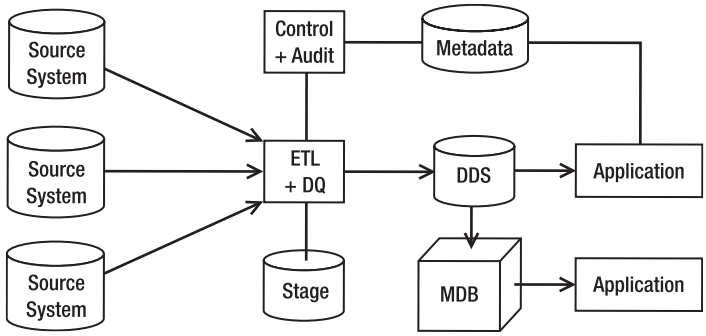

Figure 2-4. *Data architecture where the stage ETL and DDS ETL are combined*

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, Cognos, 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 straightforward 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.

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 2-5 shows the NDS + DDS data flow architecture.

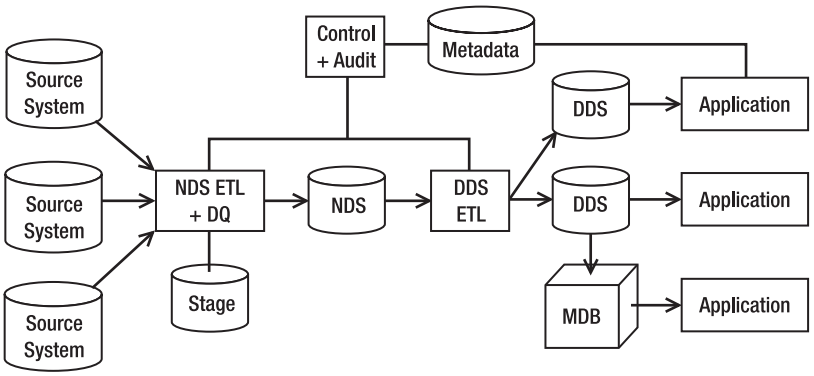

Figure 2-5. *NDS + DDS data flow architecture*

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. I will discuss more about NDS in Chapter 5 when I talk about dimensional modeling.

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.

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.

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 2-6 shows ODS + DDS data flow architecture.

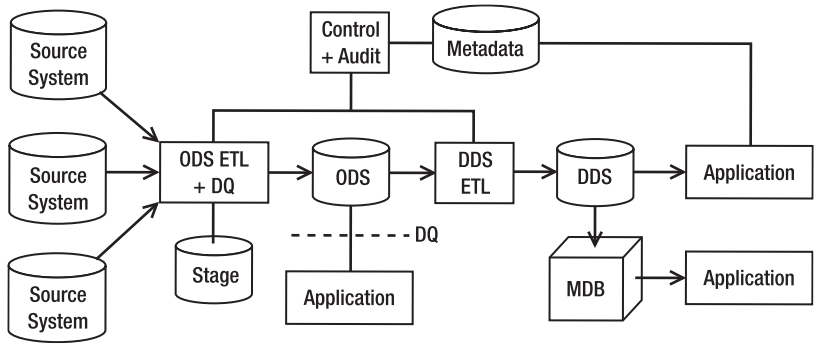

Figure 2-6. *ODS + DDS data flow architecture*

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