Microsoft EDW Architecture, Guidance and Deployment Best Practices

# Chapter 2: Data Architecture

**By Microsoft Corporation**

**Acknowledgements:**

**Contributing writers from Solid Quality Mentors**: Larry Barnes, Bemir Mehmedbasic

**Technical reviewers from Microsoft**: Eric Kraemer, Ross LoForte, Ted Tasker, Benjamin Wright-Jones

**Contributing editors from Solid Quality Mentors**: Kathy Blomstrom

**Published:**

**Applies to:** SQL Server 2008 R2

[Chapter 2: Data Architecture 1](#_Toc275837580)

[Introduction 3](#_Toc275837581)

[**Chapter Focus** 4](#_Toc275837582)

[**Roles and Responsibilities** 5](#_Toc275837583)

[**Requirements** 6](#_Toc275837584)

[**Challenges** 8](#_Toc275837585)

[**Data Warehouse Maturity Model Overview** 9](#_Toc275837586)

[**Data Quality and the Role of Governance and Stewardship** 10](#_Toc275837587)

[Data Warehouse Concepts 11](#_Toc275837588)

[**Data Warehouse Components** 12](#_Toc275837589)

[**Business Intelligence = Data Warehousing?** 13](#_Toc275837590)

[**Data Warehouse Architectures** 13](#_Toc275837591)

[**Data Models: Normalized to Denormalized** 16](#_Toc275837592)

[Database Architecture 20](#_Toc275837593)

[**One Data Area: Full Loads** 21](#_Toc275837594)

[**One Data Area: Incremental Loads** 21](#_Toc275837595)

[**Adding a Production Data Area** 23](#_Toc275837596)

[**The Consumption Data Area** 24](#_Toc275837597)

[**The *Data in* Data Area** 25](#_Toc275837598)

[**Exception and Logging Data Areas** 26](#_Toc275837599)

[**Archiving** 27](#_Toc275837600)

[**Metadata** 28](#_Toc275837601)

[**Operational Data Stores** 29](#_Toc275837602)

[**Consumer Interfaces** 30](#_Toc275837603)

[Master Data and Master Data Management 32](#_Toc275837604)

[**What Is Master Data?** 34](#_Toc275837605)

[**What Is Master Data Management?** 35](#_Toc275837606)

[**Where Do Data Warehousing and MDM Overlap?** 35](#_Toc275837607)

[Platform Architecture 42](#_Toc275837608)

[**Data Warehouse Server** 43](#_Toc275837609)

[**Server Virtualization** 44](#_Toc275837610)

[**SQL Server Fast Track Data Warehouse** 44](#_Toc275837611)

[**Data Warehouse Appliances** 44](#_Toc275837612)

[**Which Server Option Should You Choose?** 45](#_Toc275837613)

[Database Architecture 45](#_Toc275837614)

[**Databases, Schemas, and Filegroups** 46](#_Toc275837615)

[**Considerations for Physical Database Design** 50](#_Toc275837616)

[Data Modeling 51](#_Toc275837617)

[**Data Modeling Overview** 52](#_Toc275837618)

[**Conceptual Model** 53](#_Toc275837619)

[**Logical Model** 55](#_Toc275837620)

[**Physical Model** 56](#_Toc275837621)

[**Data Modeling – Column Types** 58](#_Toc275837622)

[**Keys** 63](#_Toc275837623)

[**Dimensions** 64](#_Toc275837624)

[**Fact Tables** 68](#_Toc275837625)

[**Reference Data** 69](#_Toc275837626)

[**Hierarchies** 71](#_Toc275837627)

[**Bridge Tables** 78](#_Toc275837628)

[**Nulls and Missing Values** 78](#_Toc275837629)

[**Referential Integrity** 79](#_Toc275837630)

[**Clustered vs. Heap** 79](#_Toc275837631)

[**SQL Server Data Type Considerations** 81](#_Toc275837632)

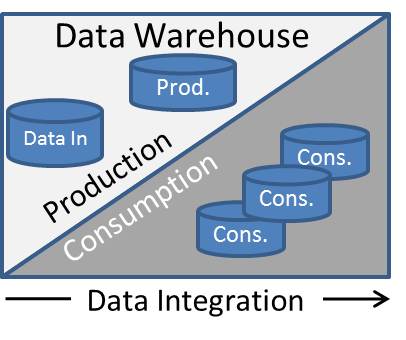
[**Very Large Data Sets** 83](#_Toc275837633)

[Conclusion and Resources 84](#_Toc275837634)

[**Resources** 85](#_Toc275837635)

## Introduction

Data architecture is an umbrella term for the standards, metadata, architectures, and data models used to ensure that an organization’s data warehouse meets the strategic decision-making needs of its business users. At its core, a data warehouse consists of a collection of databases organized into production and consumption areas, as shown in Figure 2-1.



**Figure 2-1**: Data warehouse organized into production and consumption areas

Each database contains tables and supporting objects. These tables are populated with large amounts of data. Data integration processes are responsible for moving and transforming the data as it flows from sources to the consumption area.

This simple concept is complicated by the following factors:

* Scope – Multiple, cross-organizational subject areas
* Scale – Very large volumes of time-variant data
* Quality – Cleansing, integrating, and conforming diverse data from multiple sources

There is a lot of literature available on data warehouse data architecture, but the two most visible data warehouse authors in the industry focus on different aspects of data architecture:

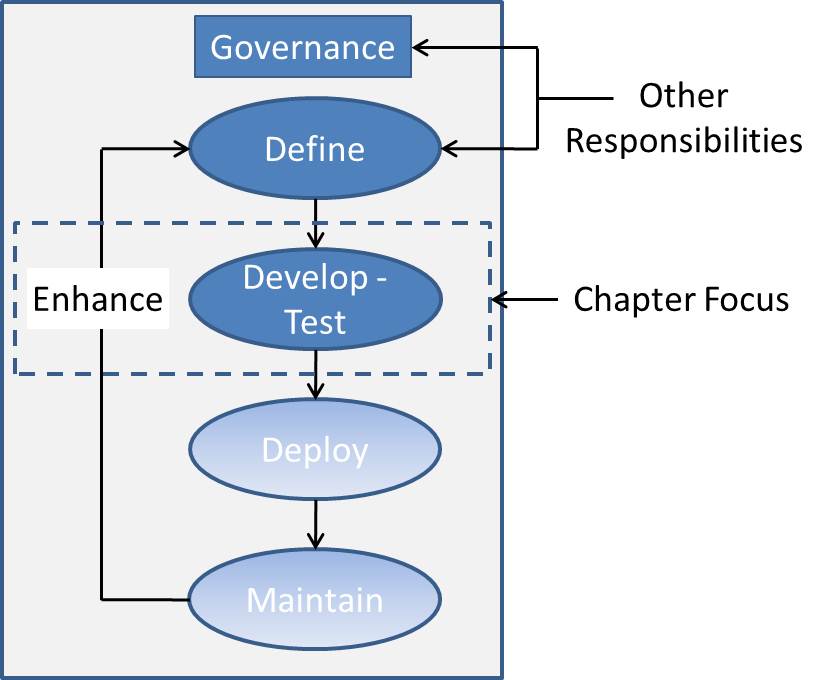
* Bill Inmon’s Corporate Information Factory (CIF) focuses on database architecture, a top-down approach.
* Ralph Kimball’s Enterprise Data Bus focuses on data modeling, a bottom-up approach.

We’ll look briefly at these different approaches in the next section. However, the objective of this chapter is to distill the available information into a set of concepts and patterns and then present tangible best practices for data warehouses implemented on the Microsoft SQL Server database platform.

The audience for this chapter is members of the data warehouse team responsible for the data architecture, database architecture, data models, and overall quality of the data warehouse.

### **Chapter Focus**

The data architecture team has responsibilities for oversight and specific deliverables throughout the data warehouse development life cycle, shown in Figure 2-2.

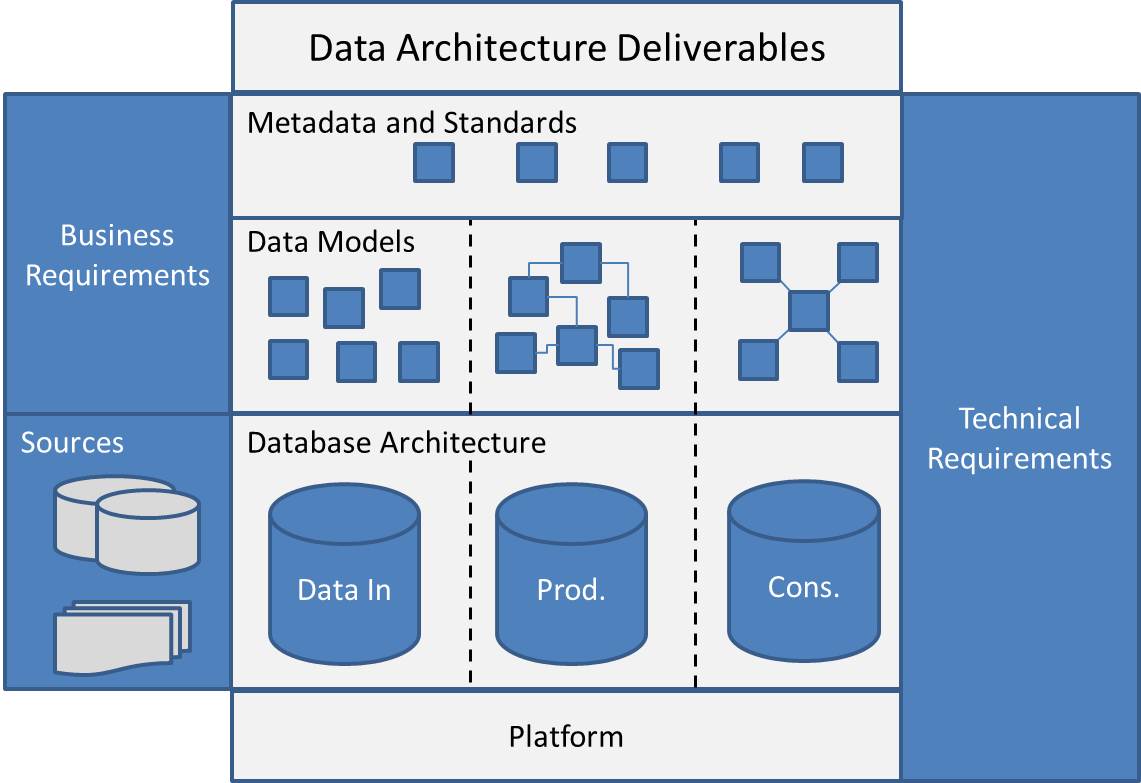


**Figure 2-2**: Focus of this chapter

This chapter focuses on the deliverables that the data architecture team produces as part of the development phase or in support of development—namely, architectures and data models.

Note that physical best practices and guidance within in this chapter are for the symmetric multi-processing (SMP) versions of SQL Server 2008 R2, i.e. guidance for [SQL Server 2008 R2 Parallel Data Warehouse](http://www.microsoft.com/sqlserver/2008/en/us/parallel-data-warehouse.aspx) (PDW) is out of scope for the initial release of this chapter and document.

Figure 2-3 shows the data architecture deliverables and the inputs into these deliverables.



**Figure 2-3**: Data architecture deliverables

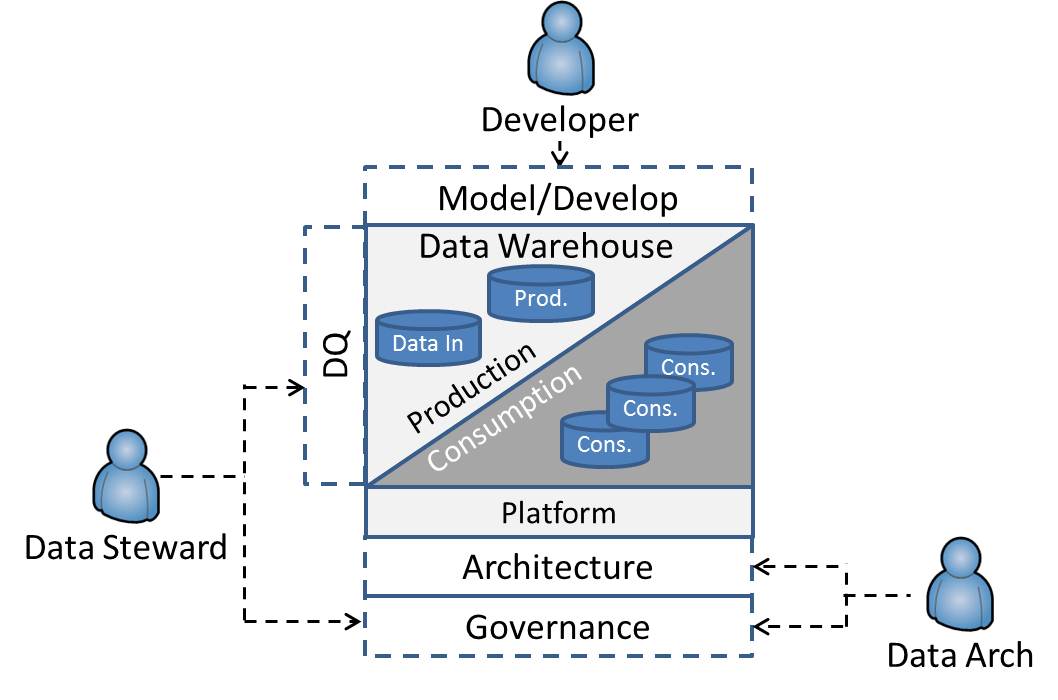
Data architecture team deliverables include:

* **Technical metadata and standards** – The team provides oversight and contributes to all database development standards and technical metadata used within the data warehouse.
* **Data models** – The team provides oversight and understanding of all data models within the data warehouse and acts as subject matter expert for data models within data governance and other cross-organizational efforts.
* **Database architecture** – The team has primary responsibility for the data warehouse database architecture.
* **Platform architecture** – The team contributes to the product selection and underlying hardware and software platform that hosts the data warehouse.

One primary driver behind these data architecture deliverables is the maturity level of an organization’s data warehouse. This is briefly covered in the next section.

### **Roles and Responsibilities**

The data architect, data developer, and data steward each play key roles within data architecture and are responsible for working with business analysts and the extended team to translate business requirements into technical requirements. Figure 2-4 shows these roles along with their responsibilities.



**Figure 2-4**: Roles and responsibilities on the data architecture team

Roles and responsibilities:

* The data architect is a member of the data governance team and is responsible for the data warehouse architecture, metadata, overall quality of the data warehouse solution, and in some cases, the initial data model.
* The database developer is responsible for the development of data models and other database objects within the data warehouse and contributes to data warehouse metadata.
* The data steward, also a member of the data governance team, contributes to business and technical metadata and is responsible for the quality of data within the data warehouse.

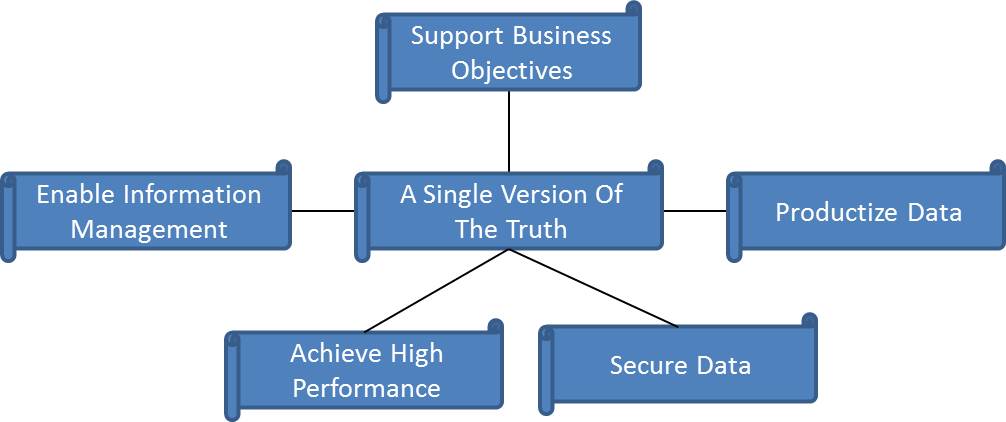
This chapter’s primary audience is data architects and database developers. Data stewardship is a key role within a data warehouse project and will be covered when it intersects with the core data architecture team. However, data stewards and their day-to-day activities are not a focus of Chapter 2.

### **Requirements**

Data architecture for data warehouses should be first driven by business need and should also conform to organizational and technology needs.

As Figure 2-5 illustrates, as enterprise data warehouse systems are being developed, data architecture should be implemented to:

* Support the business objectives
* Enable information management (data flows, validations, consumption)
* Productize data (i.e., turn data into an asset) for competitive advantage
* Produce a single version of the truth across time
* Achieve high performance



**Figure 2-5**: Requirements that the data architecture needs to support

**Support Business Objectives**

Ideally, business objectives are provided to teams developing data architecture. When business objectives are not clearly defined or don’t exist, data architecture team members need to be proactive in acquiring the missing information from business stakeholders and subject matter experts.

**Information Management – Data Consumption and Integration**

Business objectives are further broken down into data requirements that define the databases and data models used by business consumers. The data architecture team works closely with the data integration team to ensure that data requirements can be successfully populated by data integration processes. These data mappings, business rules, and transformations are often a joint effort between business analysts and data integration developers.

**Productize Data**

As data warehousing systems and business intelligence (BI) within the organization matures over time, organizations begin to realize the potential for productizing the data—meaning transforming data from a raw asset into a measure of business results that ultimately provides insight into the business.

**Single Version of the Truth**

There is no alternative to one version of the truth in successful data architecture. The data warehouse team shouldn’t underestimate the difficulty involved in achieving one version of the truth. The team must overcome technical, cultural, political, and technological obstacles to achieve what is probably the most challenging requirement when building a data warehouse.

**Achieve High Performance**

Achieving high performance for data retrieval and manipulation within the data warehouse is a requirement of data architecture. Later in this chapter, we discuss the different logical and physical data modeling techniques and best practices that directly relate to ensuring the most efficient performance for the very large databases within the data warehouse consumption area.

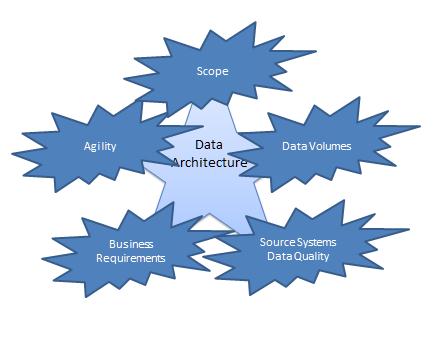
However, it’s often the performance of the data integration processes that drive the data models and architectures within the production area of very large data warehouses (VLDWs). These dual requirements for high-performance loads and high-performance queries result in different databases and data models for the production and consumption areas.

**Secure Data**

The data architecture must ensure that data is secured from individuals who do not have authorization to access it. Often, users are allowed to view only a subset of the data, and the data architecture is responsible for ensuring that the underlying constructs are in place to meet this requirement.

### **Challenges**

Providing high-performance access to one version of the truth that meets business objectives and requirements over time is not a simple task. It requires both a solid initial implementation and the ability to enhance and extend the data warehouse over a period of many years. Figure 2-6 presents some of the challenges for the data architecture team.



**Figure 2-6**: Challenges for the data warehouse team

These challenges include:

* The lack of complete business requirements or conflicting business requirements from business stakeholders
* Scope, or the cross-organizational communication toward a common goal and the need for common definitions and models
* Understanding source systems, their limitations (including data quality and antiquated systems), and the impact on the downstream data warehouse data stores
* Agility, or the ability to meet the needs of the business in a timely manner
* Data volumes and the need for high performance for both queries and data loads

**Business Challenges**

In a perfect world, business objectives are clearly defined and well understood by stakeholders and users. In reality, however, business needs are often not clearly defined or are not broken down into a clear set of requirements. Frequently, this is due to the difference between business descriptions of objectives and the technical interpretations of these objectives.

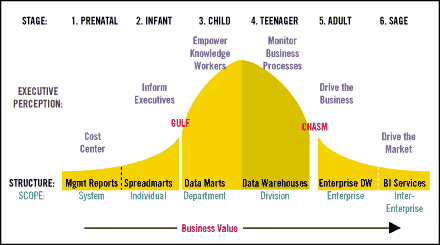
**Technical Challenges**

In addition, there will always be organizational change during the lifetime of the data warehouse, which requires that the data architecture be agile and able to respond to changes to the organization and environment.

The requirements and challenges depend upon the scope and scale of the data warehouse, which can often be mapped to its level of maturity. The next section provides a brief overview of a data warehouse maturity model.

### **Data Warehouse Maturity Model Overview**

Different organizations are at different levels of maturity with respect to their data warehouse initiatives. Several challenges, including scope and data volumes, are a function of a data warehouse’s maturity. Figure 2-7, from a 2004 *Information Management* article, [Gauge Your Data Warehouse Maturity](http://www.information-management.com/issues/20041101/1012391-1.html) , by Wayne Eckerson, provides an overview of a maturity model for data warehouses.



**Figure 2-7**: Data warehouse maturity model

The following section contains excerpts from this article.

*Data marts are defined as a shared, analytic structure that generally supports a single application area, business process, or department. These "independent" data marts do a great job of supporting local needs; however, their data can't be aggregated to support cross-departmental analysis.*

*After building their third data mart, most departments recognize the need to standardize definitions, rules, and dimensions to avoid an integration nightmare down the road. Standardizing data marts can be done in a centralized or decentralized fashion… The most common strategy is to create a central**data warehouse with logical dependent data marts. This type of data warehouse is commonly referred to as a hub-and-spoke data warehouse.*

*Although a data warehouse delivers many new benefits, it doesn't solve the problem of analytic silos. Most organizations today have multiple data warehouses acquired through internal development, mergers, or acquisitions. Divisional data warehouses contain overlapping and inconsistent data, creating barriers to the free flow of information within and between business groups and the processes they manage.*

*In the adult stage, organizations make a firm commitment to achieve a single version of the truth across the organization. Executives view data as a corporate asset that is as valuable as people, equipment, and cash. They anoint one data warehouse as the system of record or build a new enterprise data warehouse (EDW) from scratch. This EDW serves as an integration machine that continuously consolidates all other analytic structures into itself.*

*In the adult stage, the EDW serves as a strategic enterprise resource for integrating data and supporting mission-critical applications that drive the business. To manage this resource, executives establish a strong stewardship program. Executives assign business people to own critical data elements and appoint committees at all levels to guide the development and expansion of the EDW resource.*

*In summary, the need for a data warehouse arises from a desire by organizations to provide a single version of the truth. The complexity of this effort is magnified when done at the enterprise level (i.e., for an EDW). As stated above, stewardship programs are central to a successful data warehouse.*

The last section in this introduction briefly discusses stewardship, data governance, and how both are essential to delivering a high quality data warehouse.

### **Data Quality and the Role of Governance and Stewardship**

Acquiring and maintaining business trust is a foundational objective of a data warehouse that requires strong communication between the data warehouse team and the business users as well as the ability to provide accessible high quality data. Data governance and data stewardship are ongoing processes in support of maximizing business trust in a data warehouse.

The [Data Governance Institute](http://www.datagovernance.com/) defines data governance as:

*A system of decision rights and accountabilities for information-related processes, executed according to agreed-upon models which describe who can take what actions with what information, and when, under what circumstances, using what methods.*

This definition can be distilled down to the following statement:

*Data governance is a set of processes that ensures that important data assets are formally managed throughout the enterprise.*

The data governance conceptual process flows that intersect with data architecture include:

1. Identifying key data assets within the organization.
2. Ensuring that data assets have common definitions within the organization. Once common definitions are defined, business consumers can share this data, as opposed to re-creating it within each solution.
3. Ensuring that quality data is loaded into the data warehouse.

The set of processes to ensure maximum data integrity is called *data stewardship*. Data stewardship focuses on the management of data assets to improve reusability, accessibility, and quality. The team members implementing and enforcing these objectives are data stewards. These individuals should have a thorough understanding of business processes, data flows, and data sources.

Additionally, data stewards are liaisons between data warehouse architects and developers and the business community. Typically, one data steward is responsible for one subject area in a data warehouse.

Maintaining data warehouse data quality is an ongoing process. Explicitly creating direct ownership and accountability for quality data for data warehouse sources eliminates a “shadow” role that exists in many data warehouses. This shadow role often falls to business analysts and database developers once the business starts questioning results within the data warehouse. Failure to allocate resource to data stewardship activities can result in data warehouse team member burnout and negative attrition.

Data stewards are responsible for establishing and maintaining:

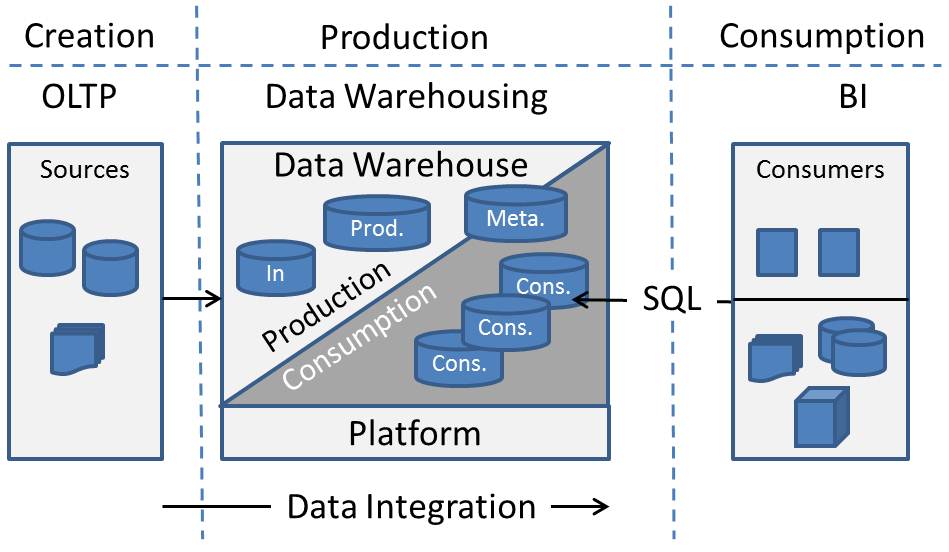
* Business naming standards
* Entity definitions
* Attribute definitions
* Business rules
* Base and calculated measures
* Data quality analysis
* Linkages to and understanding of data sources
* Data security specifications
* Data retention criteria

This chapter’s focus is less on the process and more on deliverables related to data architecture. The following links provide more information about data governance and data stewardship:

* [Data Governance Institute](http://www.datagovernance.com/)
* [Data Governance & Stewardship Community of Practice](http://www.datastewardship.com/)
* [Kimball University: Data Stewardship 101: First Step to Quality and Consistency](http://www.intelligententerprise.com/showArticle.jhtml?articleID=188101650)

## Data Warehouse Concepts

This chapter’s focus is on data warehousing, not business intelligence (BI). Figure 2-8 illustrates the different data responsibilities and what is considered a data warehouse component or process.



**Figure 2-8**: Data warehousing vs. business intelligence

Data responsibilities are segmented as follows:

* OLTP source systems and external data feeds are responsible for data *creation*
* Data warehousing focuses on data *production*
* BI focuses on data *consumption*

### **Data Warehouse Components**

The data warehouse is comprised of the following components:

* The **consumption area** serves up information to downstream consumers through SQL queries.
* The **production area** is where source data is transformed, normalized, and consolidated. Note that a **Data in** area is common within data warehouse implementations and is typically housed within the production area.
* The **metadata** area is populated with “data about data”—that is, business, technical, and process metadata providing detail behind the actual data, data models, and data integration processes.
* The **platform** is the system or systems where the data warehouse resides.

Data integration processes are responsible for the movement of data from sources to destination databases. See Chapter 3 – Data Integration for details about this topic.

The data warehouse is responsible for optimizing the query performance within the data consumption area. BI components are beyond the scope of this chapter but include:

* Data presentation, including decision-support systems, reports, and analytics
* Data delivery channels
* Downstream data stores, including data marts and semantic data models (e.g., OLAP)

### **Business Intelligence = Data Warehousing?**

There are many examples within the SQL Server community where the terms *business intelligence* and *data warehousing* are used interchangeably. In reality, these are two separate disciplines. This is especially true for enterprise data warehouses due to an EDW’s scope, complexity, and large volumes of data.

As seen in the data warehouse maturity model, as a data warehouse matures, the data warehouse team spends more and more resources on data integration. This shifts the focus from data consumption to data production.

Simply put, data warehousing and BI differ because:

* The primary focus for a data warehouse is the production of data.
* The primary focus for BI is the consumption, presentation, and delivery of the data produced by the data warehouse.

One example of where BI is confused with data warehousing within the SQL Server community is the AdventureWorks samples available at the [Microsoft SQL Server Community Projects & Samples](http://sqlserversamples.codeplex.com/) site. These samples include the following databases:

* A sample OLTP database (AdventureWorks2008R2)
* A sample data warehouse database(AdventureWorksDW2008R2)
* A sample SQL Server Analysis Services (SSAS) database, or cube

The [AdventureWorks sample data warehouse](http://msdn.microsoft.com/en-us/library/ms124623(v=SQL.100).aspx) link provides more information about the AdventureWorks data warehouse and supporting scenarios in which data warehouse, data mining, and Online Analytical Processing (OLAP) are relevant. However, note that these are BI scenarios because the focus is on consumption, not production. There are no samples for the data warehouse production scenario; instead, the data warehouse is populated directly from Comma Separated Values (CSV) files.

In summary, BI focuses more on data consumption and should not be equated to data warehousing, which concentrates more and more on data production as it matures, especially when you are working with very large data volumes.

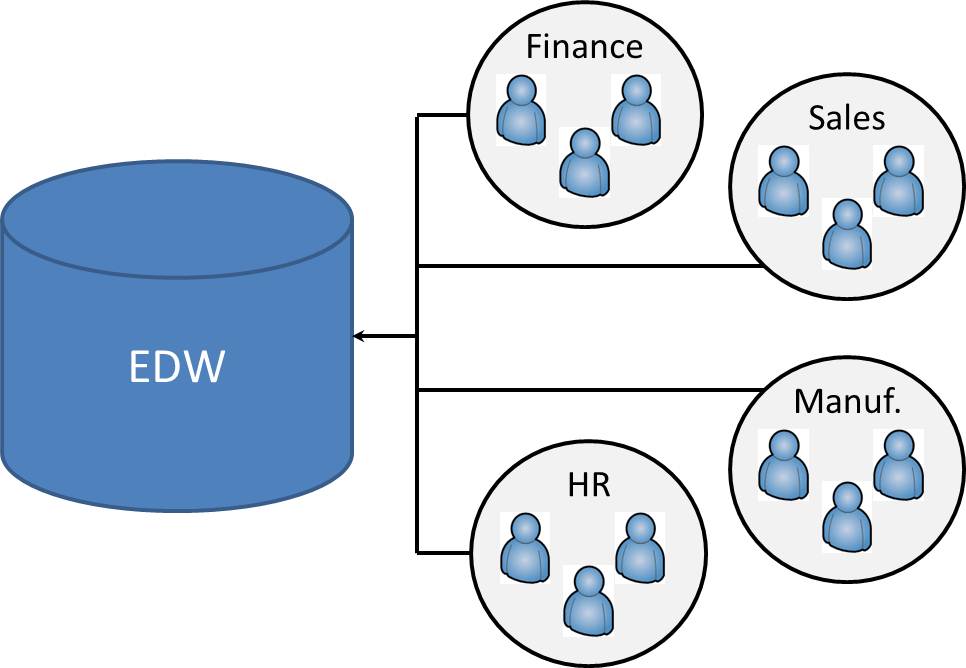
### **Data Warehouse Architectures**

As we noted earlier, much of the data warehouse literature is grounded in one of two different implementation approaches:

* The top-down approach is often used when describing [Bill Inmon’s Corporate Information Factory](http://www.inmoncif.com) reference architecture.
* The bottom-up approach is often used when describing [Ralph Kimball’s dimensional modeling and Enterprise Data Bus](http://www.ralphkimball.com) strategy.

The top-down approach historically has led to a centralized EDW, and the bottom-up approach has led to federated data marts. This section reviews these two approaches and presents a third approach used for many data warehouses.

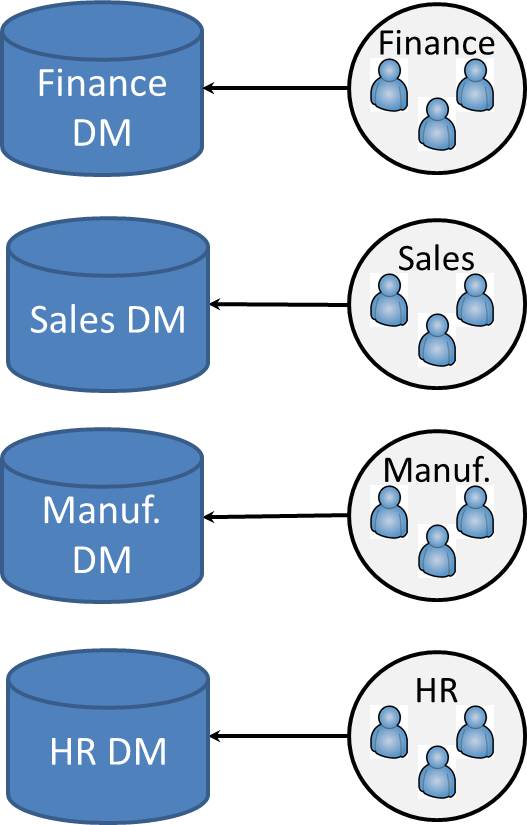
The centralized EDW, shown in Figure 2-9, was the first data warehouse database architecture. This architecture creates a central repository for all of an organization’s integrated data.



**Figure 2-9**: Centralized EDW

As stated earlier, the sheer scope of implementing an EDW often results in “analysis paralysis”—that is, inordinate amounts of time spent gathering requirements and designing subject area data models. These extended development cycles increase the risk of user requirement changes, user requirement misinterpretation, and ultimately, a failed implementation.

In 1996, Ralph Kimball published *The Data Warehouse Toolkit*. This book introduced dimensional data modeling to a large population and contained examples of dimensional data models for a variety of vertical scenarios. Kimball was also active on the lecture circuit and started to support federated data marts over a centralized EDW. Figure 2-10 illustrates the federated data mart approach.



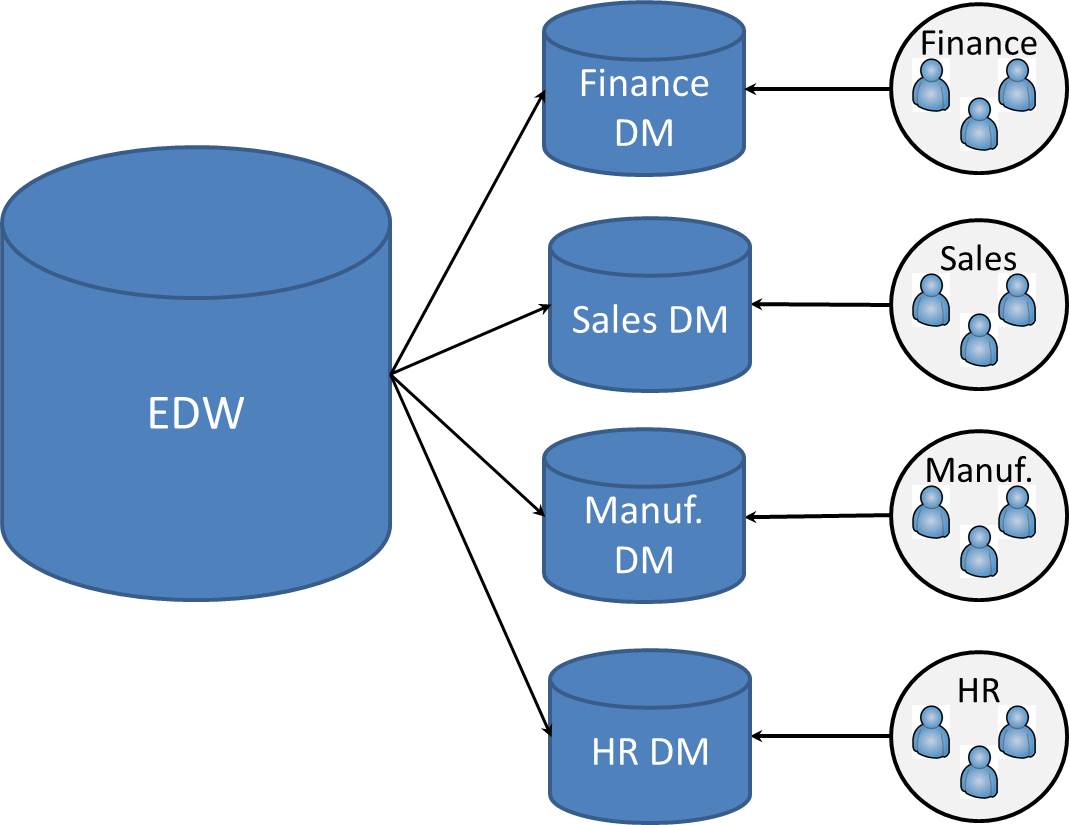
**Figure 2-10**: Federated data marts

In this architecture, a separate data mart is implemented for each business process and subject area. This strategy allows for shorter development cycles, lowers the risk of changing and misunderstanding user requirements, and delivers partial solutions faster.

However, a drawback of federated data marts is that it often results in “multiple versions of the truth,” even when the ideal approach for federated data marts is to have a common data model.

The data warehouse space started seeing the implementation of both centralized data warehouses, which often presented aggregated data to business users in support of faster SQL queries, and subject-oriented data marts created and populated from the central data warehouse. This resulted in a third approach, the hub-and-spoke architecture, shown in Figure 2-11. This model incorporates the benefits of a centralized data warehouse database and federated data marts:

* The central data warehouse database provides business consumers with “a single version of the truth.”
* Separate data marts provide business consumers with better performance.



**Figure 2-11**: Hub-and-spoke architecture

The hub-and-spoke approach has downstream data marts that are fed from a common data warehouse database. Data marts return consistent results because they are all populated from the data warehouse database, which contains a single version of the truth. Performance is improved for business unit analysis because the marts contain less information than the data warehouse database and also have a smaller user community.

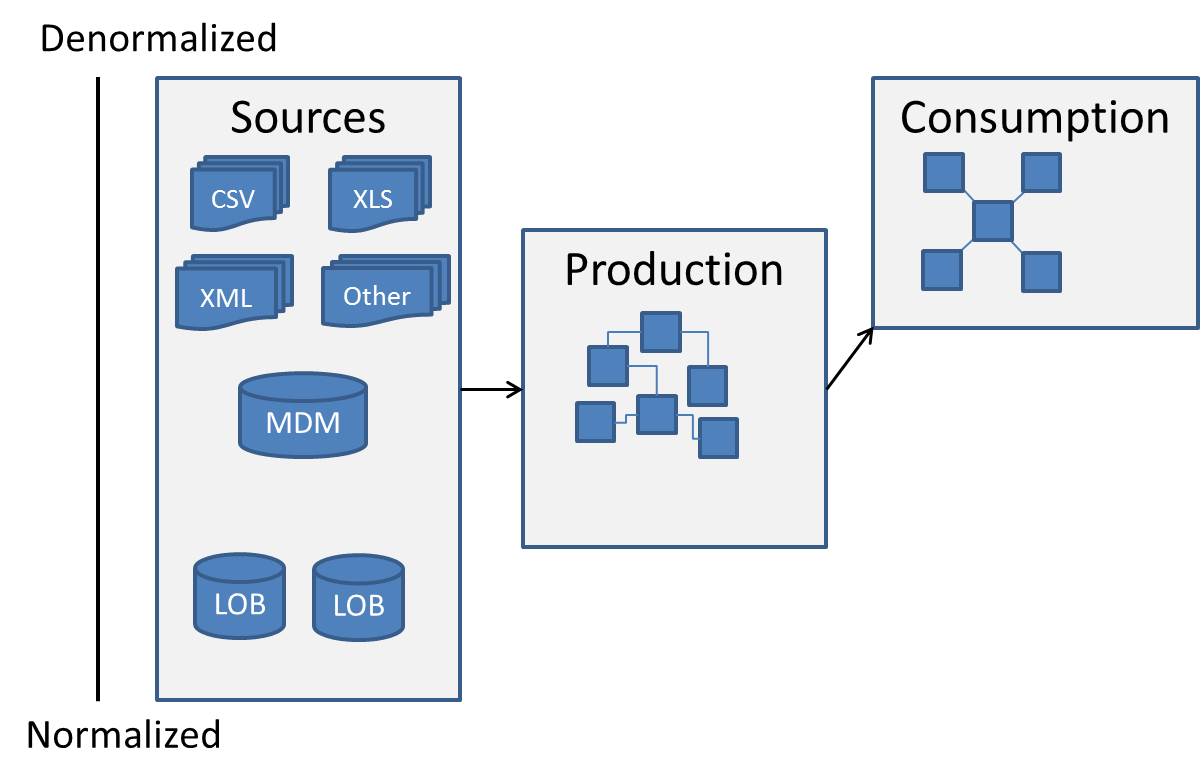
Note that scope issues still exist with the hub-and-spoke configuration—the data warehouse database still contains multiple subject areas and many consolidated sources.

Microsoft’s Parallel Data Warehouse (PDW) is a distributed architecture that supports the hub-and -spoke approach. PDW provides a publish model that supports the parallel loading of data mart spokes from the data warehouse hub. This publish model reduces data integration processes and the resources required to maintain these processes. You can read more about PDW in the MSDN article [Hub-And-Spoke: Building an EDW with SQL Server and Strategies of Implementation](http://msdn.microsoft.com/en-us/library/dd459147(SQL.100).aspx).

One of the primary deliverables for the data warehouse team is the data model or models. The next section provides an overview of this topic.

### **Data Models: Normalized to Denormalized**

One characteristic of many data warehouses is that the data is transformed from a normalized to denormalized form as it makes its way from sources to the production and consumption areas, as Figure 2-12 shows.

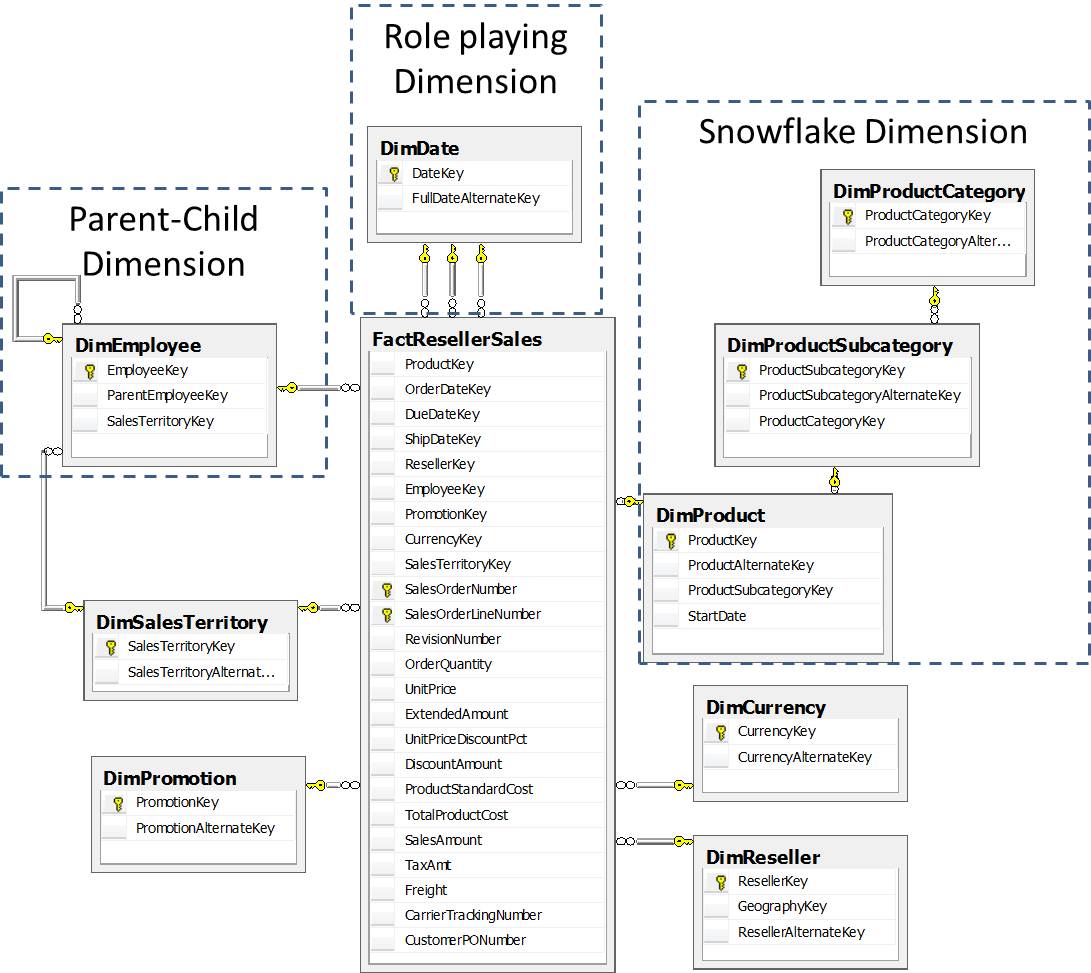


**Figure 2-12**: Normalized to denormalized data model

Most source data originates from transactional line of business (LOB) systems. These LOB systems’ data models are highly normalized in support of Online Transaction Processing (OLTP). Other sources, mostly file-based, are typically in a denormalized format and have various sources (e.g., external vendors and partners, exports from internal systems). In some cases, these files are created and maintained by business and technical users within applications such as Microsoft Excel and Access.

The production area is where source data is consolidated and rationalized. The subject areas within the production area are typically modeled in a normalized format, but less so than the source LOB systems.

This data is then transformed, denormalized, and aggregated when it flows to the consumption area. Dimensional data models are an example of a denormalized structure optimized for data consumption; Figure 2-13 shows one example, the AdventureWorksDW2008R2 FactResellerSales snowflake data model.



**Figure 2-13**: Snowflake dimension example: FactResellerSales

Note that this sample database is modeled to demonstrate the capabilities of SSAS. A normalized data model would not have the Parent-Child and the Snowflake dimensions highlighted above.

**Normalized vs. Denormalized**

The key differences between normalized and denormalized data structures are as follows:

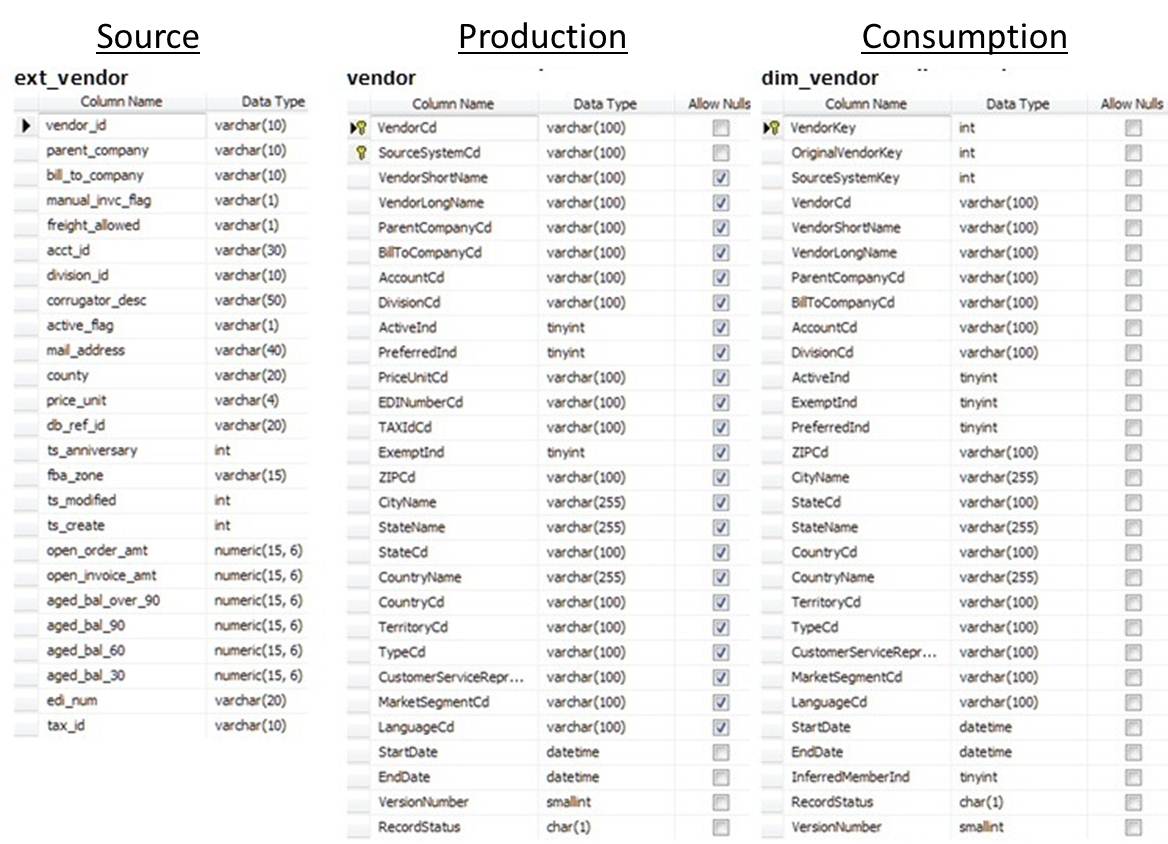
* OLTP systems are where data is created. Normalized data structures support a high-volume transactional workload, which consists of inserts, updates, deletes, and selects of individual or a small number of records.
* In a denormalized approach, data structures are optimized for data consumption (.i.e., workloads that process high volumes of records).

Dimensional modeling is one common example of a denormalized approach where information is grouped into dimensions and facts, as we see in Figure 2-13. Facts contain surrogate key pointers to dimensions along with mostly numeric measures. Dimensions contain flattened or snow-flaked hierarchies and relationships for entities and relevant attributes.

Deciding on the amount of denormalization within the consumption area involves balancing consumer requirements with source system models and load complexity, including the following considerations:

* **Whether to use a star schema or snowflake schema implementation.** A star schema is a fully denormalized implementation. See the [Kimball Group’s Web site](http://www.ralphkimball.com) for more information about star schemas, snowflake schemas, and dimensional modeling.
* **How much to denormalize data for the most efficient SQL access.** This, however, can increase load complexity. When choosing the appropriate level of denormalization and complexity of data integration processes related to it, consider the data warehouse requirements for analysis and data latency as well as overall delivery milestones and timelines for the data warehouse initiative.
* **Whether to use database views to present a denormalized view of normalized data.**
  + This implementation pattern allows for introducing denormalized data structures without having to materialize tables. With this approach, data integration for existing tables doesn’t need to change.
  + This approach is sometimes seen as a feed for a downstream semantic layer such as SSAS.
  + Views will have a negative impact on performance unless they are materialized.

Figure 2-14 shows different table structures for a hypothetical entity containing information about vendors for an organization. This illustrates how a data model changes as it flows from the source to the production area and then to the consumption area.



**Figure 2-14**: Example source, production, and consumption area table structures

Production area data store characteristics are as follows:

* Tables are denormalized some, but the overall data model is still normalized.
* Common naming conventions are applied to tables, attributes, and other database objects.
* Business keys are identified and enforced, as are relationships between entities.
* Business rules for data types for various types of string and numeric attributes are introduced and enforced.
* All of these are building blocks that help provide for consistency and manageability across a data warehouse.

A consumption data store has these characteristics:

* Tables are denormalized. These structures are designed for most effective data retrieval.
* Natural keys in dimensions are referenced via surrogate keys.

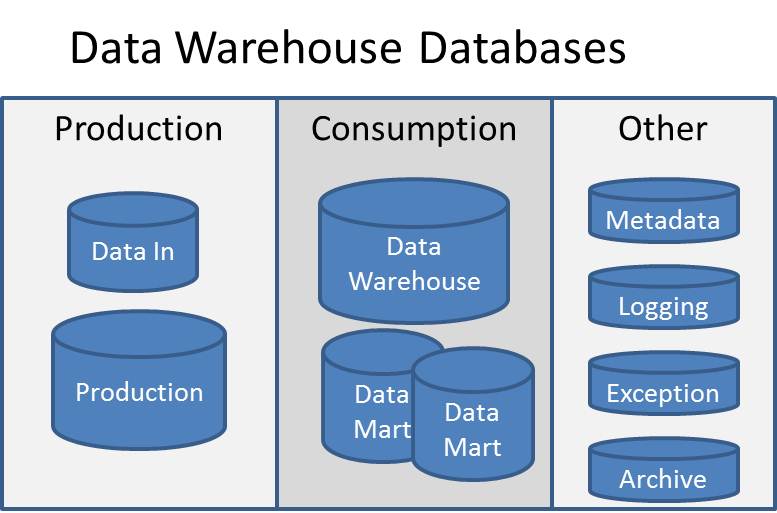
Natural and surrogate keys are discussed in more detail in the data modeling section later in this chapter.

In summary, choosing an appropriate data warehouse architecture and data model approach are central to the success of a data warehouse. The decisions about which data warehouse architecture and data models to use, however, are independent of one another—a federated data mart can have a more normalized data model, while a centralized EDW can have both normalized and denormalized data models, for example.

Once you’ve selected the architecture for your data warehouse, the next deliverable is the database architecture, which we cover in the following section.

## Database Architecture

This section expands on the data warehouse components and provides an overview of the different data areas within the data warehouse, as shown in Figure 2-15. Note that each data area can include one or more physical databases.



**Figure 2-15**: Data warehouse data areas

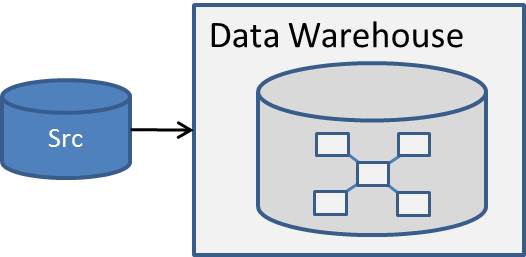
Data warehouse data areas include:

* **Production area**
  + **Production** databases are where data is cleansed, lineage is introduced, business rules are applied, and versioned data is introduced.
  + **Data in** databases contain a mirror copy of a subset of source system data.
* The **consumption area** is accessed by business consumers and can include a data warehouse, multiple data marts, or a combination of the two.
* The **exception** data area is where records that fail data quality checks and business rules are held.
* **Metadata** describes the data itself, both in business terms and technical terms. It includes definitions, rules, and origins of all data in the data warehouse. In addition, process metadata helps define configurations and implement security within the data warehouse.
* **Logging** databases are important for recording the day-to-day activity within the data warehouse and typically include logging activity from data integration processes and, optionally, consumption data area access.
* **Archived** databases hold aged data removed from data areas to improve performance.

These data areas are the focus of the remainder of this section, which is organized by data warehouse maturity level—starting with basic configurations seen in departmental data marts and moving to more advanced configurations seen in EDWs.

### **One Data Area: Full Loads**

The first implementation for many data warehouses and data marts is a full load, shown in Figure 2-16.



**Figure 2-16**: Full load

This implementation truncates and reloads the data warehouse or data mart directly from one source on a scheduled basis (i.e., daily, weekly, monthly). This is the simplest implementation and requires the least amount of data integration code. The data model is typically a denormalized dimensional model.

However, organizations soon realize that full loads have the following issues:

* **No history or no point-in-time history.** Many OLTP systems keep only current data, which precludes historical reporting. Even if the OLTP systems store history, these historical records are often modified over time. This makes it impossible to capture the state of data at a particular point in time.
* **Extended processing times.** As the data volumes grow, it takes longer and longer to drop indexes, truncate , reload, and reindex all data. This becomes an issue when processing extends into business-usage hours.
* **Data quality issues.** Data stewards have little to no visibility into record change histories, which makes it difficult to track data issues back to the source data and process.

These problems with full loads lead most organizations to an incremental load approach.

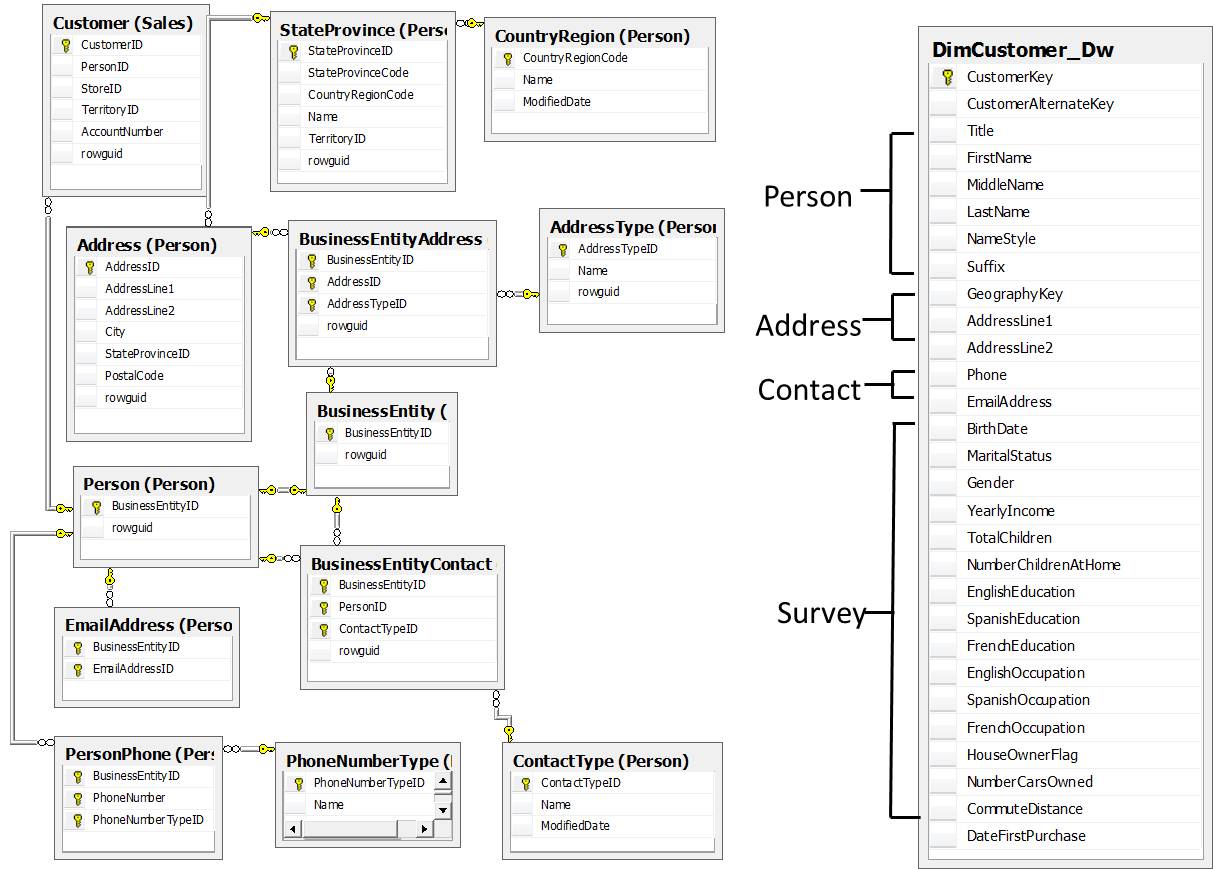
### **One Data Area: Incremental Loads**

The first step for incremental loads is to create a data model that supports history. The dimensional data model is extended to support record versions. When implemented within dimensions, this is referred to as a Slowly Changing Dimension II (SCD II).

With incremental loads, organizations can report on history, but this approach has the following issues:

* The fact that there is one data area for consumption and production results in the database being offline to business users while data integration processing is active.
* Merging multiple business entities into one dimension makes it difficult to enforce a single version of the truth, and tracking down and resolving data quality issues is a time-consuming process.

The AdventureWorksDW2008R2 DimCustomer dimension provides a simple example of how one dimension is populated from many source tables. Figure 2-17 shows the dimensional table along with its sources from the AdventureWorks2008R2 OLTP database. Note that 13 tables were identified as sources for the DimCustomer dimension.



**Figure 2-17**: DimCustomer dimension and its sources

Note that the AdventureWorks2008R2 OLTP data model presents an overly simplistic source data model. For example, the Person table contains two XML data columns—contact information and demographics—which contain columns used to populate DimCustomer columns, including education, number of cars owned, total children, and number of children at home. This information is typically obtained from a loan application, a credit card application, and/or a survey, which are often separate applications with separate data models.

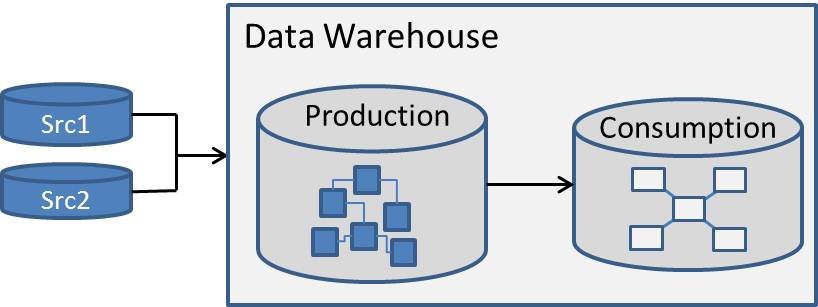
Here are additional issues encountered when the dimensional model is used to store and track history:

* Common entities (e.g., Address, Person) can’t be used to populate other data warehouse tables.
* Consolidating customer information from multiple source systems directly into the denormalized data warehouse table would make it more difficult for data stewards to track changes back to the respective source systems they’re responsible for.

Hopefully this simple example helps illustrate the level of complexity that exists within data, and why it’s beneficial to have a separate production data area.

### **Adding a Production Data Area**

Figure 2-18 shows separate data areas for production and consumption. The production area is where information loaded from sources is normalized and where business rules are applied, business keys are identified and enforced, lineage is introduced, and data is prepared for loading into downstream data layers.



**Figure 2-18**: Separate production data area

Production areas are typically seen in organizations that have moved from data marts to data warehouses within the data warehouse maturity model. At this point, the primary focus of the data warehouse shifts from providing results to consumers to integrating data from multiple sources.

The production area is the data integration working area where a “single version of the truth” is created and serves as the basis for data consistency and quality:

* The data model is typically normalized, although less so than in the source systems.
* This layer enforces relationships between entities using natural keys.
* The production area is referenced by data stewards and analysts when verifying and reconciling outputs with inputs.
* Production databases feed all consumption databases.

**What About Very Large Tables?**

Source system transaction and fact tables are the largest tables in a data warehouse and often have multi-billion record counts. Given these volumes, it’s tempting for the data warehouse team to decide to load this table directly into the consumption area and not keep a master copy of this table in the production area.

Although this strategy reduces the size of the production database, it also means that the production area no longer supports a single version of the truth. Given this issue, it’s generally recommended that the data warehouse team not use this approach.

See Chapter 3 for more information about loading patterns for very large tables using versioned inserts.

### **The Consumption Data Area**

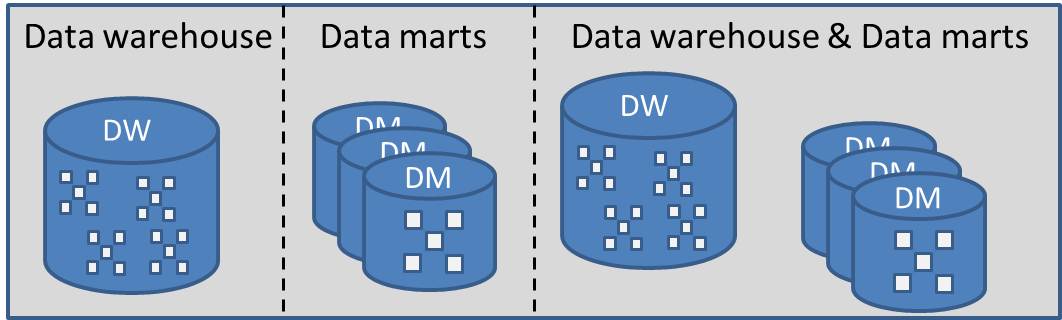
After data is loaded into the production area, the next step is to transform the data into a format more conducive to consumption—such as a dimensional data model. In addition, data will often be summarized and aggregated when there are large volumes of data.

Figure 2-19 shows the options for the consumption area: one data warehouse, multiple data marts, or a combination of both. This decision is based on many factors, including organizational structure, geographic location, security, data volumes, and platform architecture.

For the purpose of this discussion, the difference between a data mart and a data warehouse within the consumption area is scope:

* A data mart contains one subject area.
* Data warehouses have larger volumes of data than a data mart and contain multiple subject areas.

Note that there are strong views in the industry surrounding data marts and data warehouses. You can read Bill Inmon’s perspective in the article [Data Mart Does Not Equal Data Warehouse](http://csis.bits-pilani.ac.in/faculty/goel/Data%20Warehousing/Articles/Data%20Marts/dataWarehouse_com%20%20Article_DM%20VS%20DW.htm).

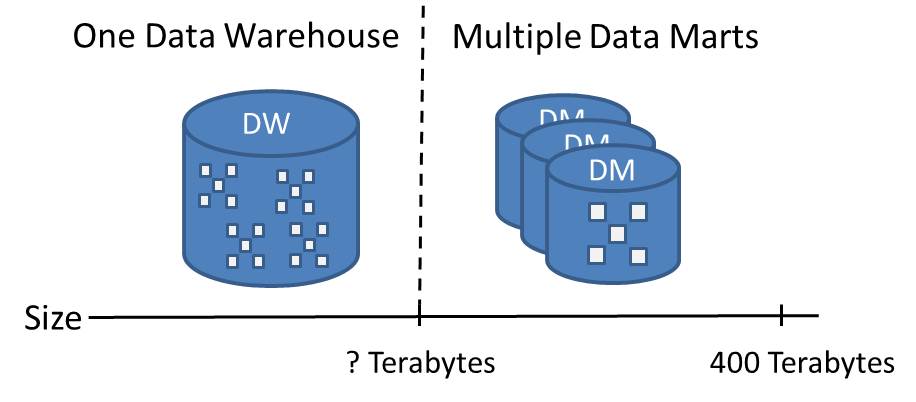


**Figure 2-19**: Consumption area options

As we noted, consumption area options are as follows:

* One data warehouse with multiple subject areas provides the greatest availability to data; users can drill down and drill across subject areas within the data warehouse.
* Multiple data marts typically provide better performance because the data volumes are less than those in a data warehouse.
* Having both a data warehouse and multiple data marts allows the business consumer to choose between completeness and performance.

Often, the option chosen by the data warehouse team depends on data volumes, as shown in Figure 2-20.

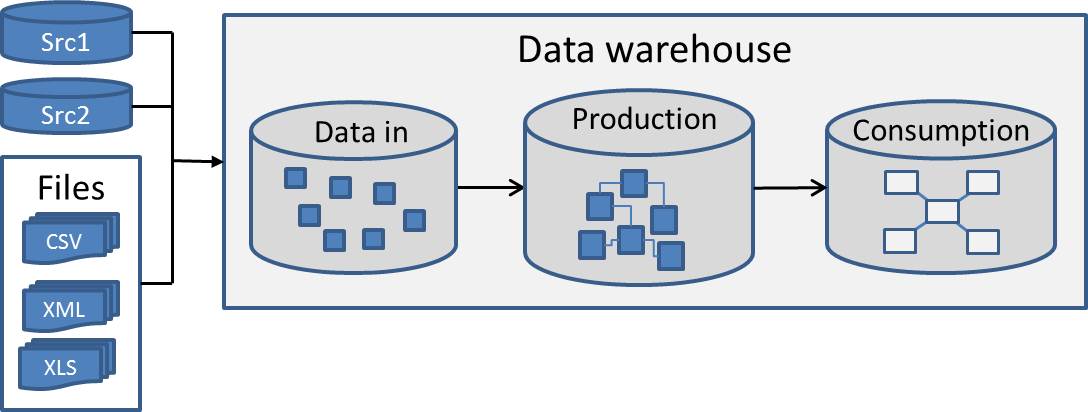


**Figure 2-20**: Data volume impact on the consumption area

Note that the question of when the consumption area can no longer support one data warehouse depends not only on data volumes but also on the underlying platform architecture.

### **The *Data in* Data Area**

The next decision is whether to have a *Data in* area or not. Figure 2-21 shows how the *Data in* area is populated with source data as the first step in the data integration process. The production area is then loaded from the *Data in* area.



**Figure 2-21**: *Data in* area

The decision about whether to have a *Data in* data store is based on business rules, available storage, data integration patterns, and whether your system can accommodate increased processing times.

*Data in* databases are probably not needed if data sources are fully available and there is an archive strategy in place for sources. However, here are some reasons to have a *Data in* area:

* Source data is preserved for auditing and reload.
* Preserving flat file sources within a database allows database backup and restore to be used as opposed to a separate file system backup.
* The *Data in* area facilitates very large data warehouse implementations:
  + Landing source data for very large data warehouses in extract databases lets you manage the amount of data being loaded by segmenting data in batches.
  + Processing batches can be introduced in parallel fashion, reducing data latency in a data warehouse.
  + Aggregations and calculations can be applied at the batch level, speeding up loads that require an intermediate area for applying aggregations and other calculations to source databases.
* The *Data in* area supports entities residing in multiple source systems:
  + Entities residing in multiple source systems typically have different designs. For example, the Products entity within a large financial services conglomerate can exist in many different systems.
  + Introducing extract databases can help when the complete definition of a dimension is not available until data from all relevant entities is made available or processed.
* Such an area helps handle data inconsistency and late-arriving facts and dimensions:
  + Issues related to data inconsistency and dirty data are preserved in extract layers so that this information can be analyzed and corrected at the source level.
  + Late-arriving facts and dimensions can be hosted in the extract area for consolidation and loading in downstream data warehouse layers once it’s complete.

Preserving the source data in an extract layer also helps in identifying common patterns in entities, which allows for more efficient design of relevant dimensions as they are shared across multiple data sources.

Chapter 3 covers source system extraction and data Integration patterns in more detail. However, note that some SQL Server technologies can be used to populate the *Data in* area. These technologies include database mirroring, log shipping, and database replication. And SQL Server 2008 introduced a fourth option: Change Data Capture.

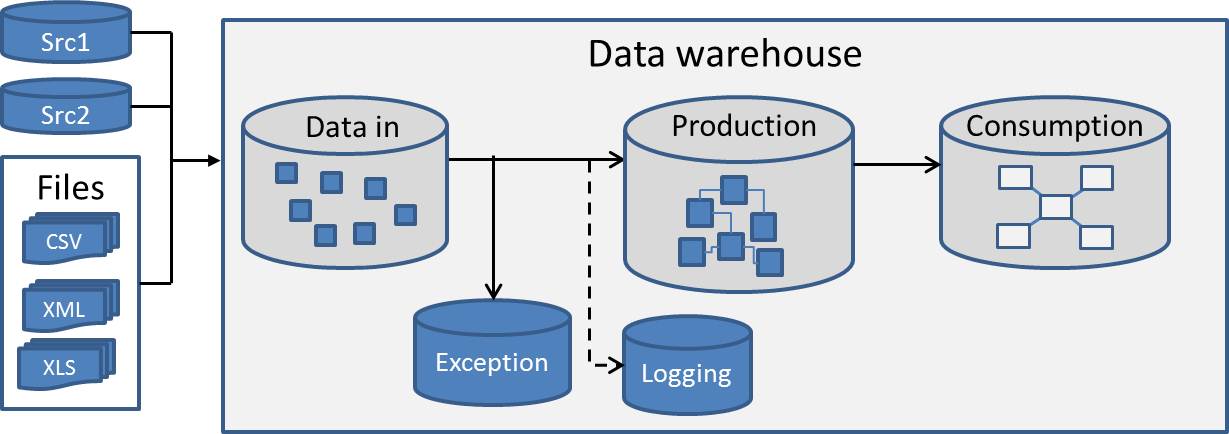
The following links provide details about SQL Server high availability options:

* [Database Mirroring](http://msdn.microsoft.com/en-us/library/bb934127.aspx)
* [Log Shipping Overview](http://msdn.microsoft.com/en-us/library/ms187103.aspx)
* [SQL Server Replication](http://msdn.microsoft.com/en-us/library/ms151198.aspx)
* [Basics of Change Data Capture](http://msdn.microsoft.com/en-us/library/cc645937.aspx)

For more information about whether to use a *Data in* area, see the “To Stage or not to Stage” section of Chapter 2 in [*The Data Warehouse ETL Toolkit*](http://www.amazon.com/Data-Warehouse-ETL-Toolkit-Techniques/dp/0764567578), by Ralph Kimball and Joe Caserta.

### **Exception and Logging Data Areas**

The exception and logging data areas are both populated by data integration processes, as Figure 2-22 shows.



**Figure 2-22**: Exception and logging data areas

Data integration processes use business rules, data quality checks, and data lookups to identify and move data records into the exception area. Whether the entire data record is infirmed or just the natural key depends on the implementation. Data stewards use the exception data area to troubleshoot and correct data exceptions.

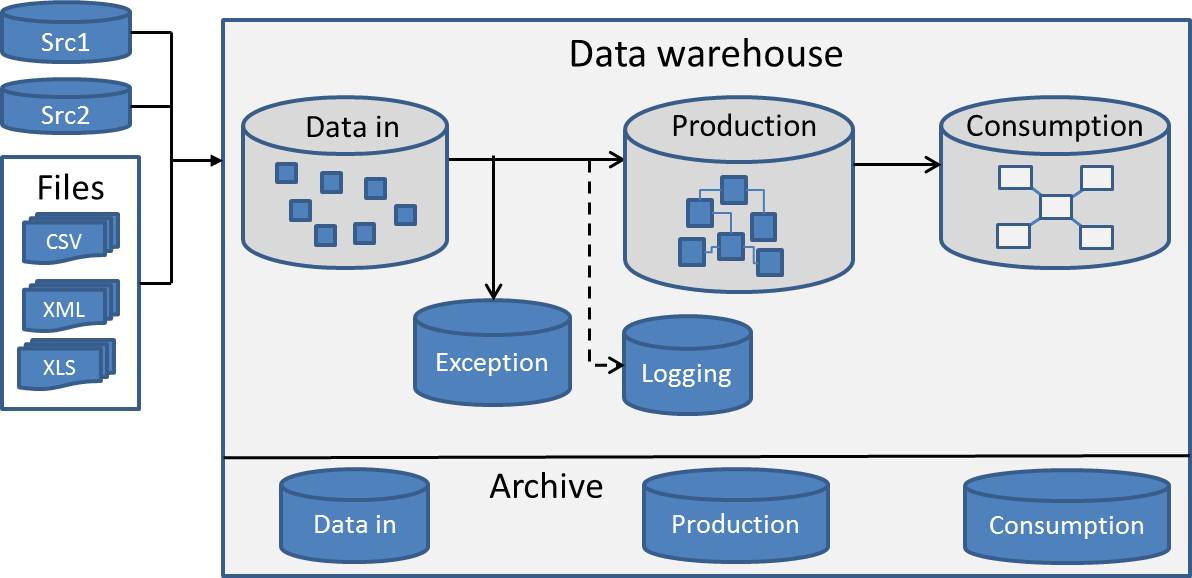
Data integration processes also populate the logging data area with information used to monitor and track the status of data loads. In addition, data integration processing errors are logged and used by data integration developers to troubleshoot the processes. See Chapter 3 on data integration for more information about these topics.

The data warehouse may also log consumer query activity. This is often useful in determining usage patterns, optimizing queries, and developing user charge-back models.

### **Archiving**

One way to optimize performance for any database is to reduce the size of the data being queried. Deleting data, however, conflicts with the data warehouse’s objective of providing one version of the truth over time. When data is deleted or purged, it is lost—along with the ability to obtain certain historical perspectives.

The alternative to deleting data is to archive it. Archiving data is the process of moving data from a primary data store to a secondary data store. Figure 2-23 contains an example of an archive data area within the data warehouse.



**Figure 2-23**: Data warehouse with an archive data area

Note that the secondary data store in the above example is a database, but it could also be a file or set of files. Archiving improves performance while supporting the preservation of data.

Archiving data is typically based on when the data was loaded, and the data archive schedule within a data warehouse is a function of:

* The data area that is being archived
* Data volumes over time
* How frequently business consumers query the data

The *Data in* data area will be frequently archived or truncated. Production data may require more frequent archiving for data warehouses with massive amounts of data. And examples of how vertical industry requirements affect consumption data include:

* Health care industry: Seven to 10 years of data is required for claims data for companies doing business in the US.
* Financial industry: Seven years of data is required for financial audit data for companies doing business in the US.

In general, every very large data warehouse should have an archiving strategy. This strategy also needs to support the repopulating of archived tables should the need arise for this data. In these cases, the destination can be the same or a different table. The architecture team should base its archiving requirements on legal and industry standards as well as user access patterns.

Note that the physical partitioning of tables by date simplifies the data archival processes.

### **Metadata**

As we mentioned earlier, metadata is data that describes the data itself. It includes definitions, rules, and origins of all data in the data warehouse. Metadata is important because it gives context to the data in the data warehouse and is typically classified into business and technical metadata. Ralph Kimball adds another category: process metadata, which is a variant of technical metadata.

* **Business metadata** – Provides business definitions for database objects, including databases, tables, and columns. It also includes additional information about columns, including but not limited to business rules and field types.
* **Technical metadata** – Documents technical aspects of data; the classic example of technical metadata is a data dictionary. Other examples include source-to-destination mappings used in data integration as well as results from data profiling and lineage.
* **Process metadata** – A type of technical metadata used to configure key components and processes within the data warehouse, including data integration and security.

The following are examples of process metadata.

**Configurations**

In data warehouses, there are databases used for storing information on various configurations, including:

* Data integration connection properties
* Environment variables, XML configuration files
* Size of ETL batches
* Data warehouse default values

**Security**

* Security-related scenarios are addressed in databases residing in the data warehouse security area.
* Access to data in the data warehouse is defined by business rules. There are often requirements to secure data both vertically and horizontally. Client-level access restrictions are a common example of these requirements, where data needs to be secured by specific client IDs. Additionally, there are often provisions for securing data on a row-level basis.

It is important to catalog business and technical metadata information in database structures so that this information is not scattered across various documents, diagrams, and meeting notes. Once introduced into databases, metadata information can be queried and analyzed. Reports can be developed to include answers to commonly asked questions about data transformation rules or data types and column defaults.

Using metadata also provides for more effective integration of data design tools and development tools; an entire software market is dedicated to metadata tools and metadata repositories.

This concludes our overview of data areas. Next, let’s look at brief overviews of operational data stores and data warehouse consumers.

### **Operational Data Stores**

Operational data stores (ODSs) are databases that support operational reporting outside of source systems. An ODS is a key data area within the Corporate Information Factory and is typically categorized into different classes, with the key variable being the delay between the live source data and the ODS data, as follows:

* Class I – One to two-second delay; this short delta often requires asynchronous “push” data integration processes instead of the traditional “pull” method.
* Class II – Intraday; typically a two- to four-hour delay.
* Class III – One day; this can be part of the daily data integration processes.
* Class IV – Loaded directly from the data warehouse.

Note that Class I and Class II ODSs require special data integration processes (i.e., processes that run more frequently than the nightly data integration batch schedule).

Also note that an ODS may be used as a *Data in* area depending upon the implementation.

You can read more about ODSs at the following links:

* [The Operational Data Store](http://www.information-management.com/issues/19980701/469-1.html)
* [Corporate Information Factory](http://www.inmoncif.com/home/)

### **Consumer Interfaces**

Data integrated and stored in data warehouses is consumed by a variety of interfaces. Typical data consumers include external data feeds, queries, reports, OLAP structures, BI semantic layer tools, and suites of interrelated applications and services. In addition, data marts are consumers when the data warehouse architecture is a centralized EDW.

This section covers the typical data consumer interfaces to consider in your data warehouse architecture. Note that there are many products within each of these categories. The products listed below are provided as examples within the Microsoft product suite for each category.

**Queries**

One of the most straightforward methods for consuming data from a data warehouse is using queries. These queries typically scan large numbers of records, are often compiled in stored procedures, and provide for a consistent source of data for analysts and decision makers. Queries are also often combined with other information delivery vehicles, such as reports, and provide for uniform representation of information. Managing changes to queries and managing security are some important aspects of using this delivery method.

**Reports**

SQL Server Reporting Services (SSRS) is one of the most commonly used tools to access data from data warehouses. This tool provides for enterprise-wide access to information in predefined forms as well as for ad hoc access to data.

SSRS, coupled with the powerful report-authoring environment provided by Microsoft Report Builder, frequently provides the primary data consumption methods within an organization.

For more information about SSRS and Report Builder, see the following links:

* [SQL Server 2008 Reporting Services Web site](http://www.microsoft.com/sqlserver/2008/en/us/reporting.aspx)
* SQL Server 2008 R2 Books Online topic [SQL Server Reporting Services](http://technet.microsoft.com/en-us/library/ms159106.aspx)
* TechNet article [Getting Started with Report Builder 3.0](http://technet.microsoft.com/en-us/library/dd220460.aspx)

**OLAP**

OLAP takes data consumption from data warehouses to a higher level. One of the most advanced OLAP technologies available is [SSAS](http://www.microsoft.com/sqlserver/2008/en/us/analysis-services.aspx), which gives organizations advanced enterprise-wide analytical capabilities, complementing the power of data contained in data warehouses.

How well data warehouse systems are architected directly affects how efficiently you can implement SSAS. In other words, straightforward and effective data warehouse design greatly reduces the complexities around OLAP models.

**BI Semantic Layer Tools**

SharePoint 2010 Insights is a powerful BI semantic layer tool that enables users to create BI dashboards that include powerful analytic reports, Key Performance Indicators (KPIs), and scorecards. Using PerformancePoint Services (PPS) in SharePoint 2010 provides for even greater insight into how business is performing and the status of key indicators for business processes. Adding PPS to a list of data consumers from a data warehouse enables organizations to make real-time decisions and improve the overall ROI of data warehouse efforts.

For more information about PPS, see the following links:

* [PerformancePoint Services](http://msdn.microsoft.com/en-us/library/bb848116.aspx)
* [What's new for PerformancePoint Services (SharePoint Server 2010)](http://technet.microsoft.com/en-us/library/ee661741.aspx)

**Embedded BI Applications**

This class of applications is developed to solve a targeted business problem, such as fraud detection. These applications often use data mining algorithms or provide a more guided user experience than their BI tool equivalents.

**Suites of Applications and Services**

Enabling ease of access to information contained in data warehouse systems is a main objective of any enterprise BI strategy. In this respect, having information users experience pervasive BI in their organization at a low cost through the Microsoft Office Suite tools they use every day efficiently accomplishes this goal. Applications and services within Office provide for easy and direct access to data in data warehouses, whether users connect directly from Microsoft Excel or use the advanced analytics of Microsoft PowerPivot.

For details about Microsoft Office Suite or PowerPivot, see the following links:

* [Microsoft Office Web site](http://office.microsoft.com/en-us/)
* [Microsoft PowerPivot Web site](http://www.powerpivot.com/)

**External Data Feeds**

External data feeds can include LOB systems, such as customer relationship management (CRM) systems. It is crucial for CRM systems to consume data that is put together from various data sources, so a data warehouse properly architected in this respect represents an ideal source of data for CRM. Data warehouses provide for data feeds that encompass a central, comprehensive, and consistent perspective of customers, making CRM systems more efficient and effective. Additionally, data warehouses represent a consistently cleansed and reliable source of customer data that is accessible enterprise-wide.

## Master Data and Master Data Management

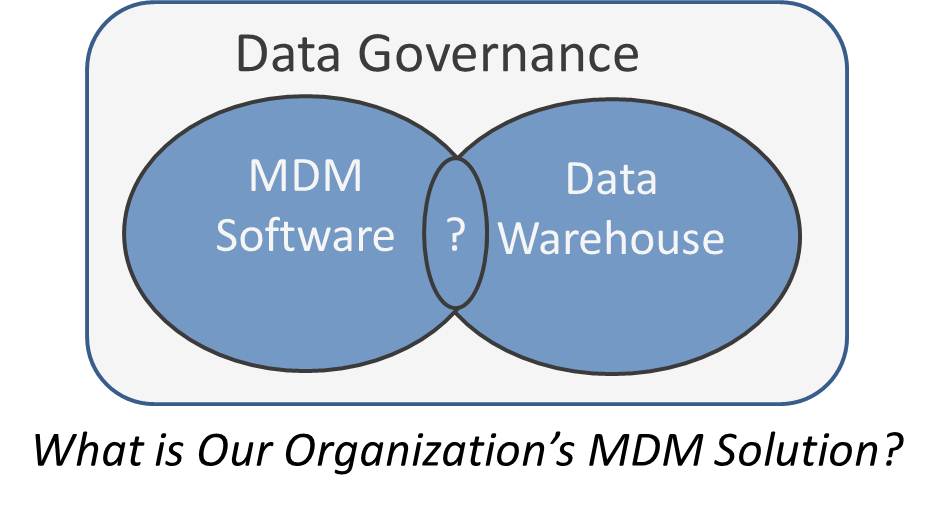
Master data and Master Data Management (MDM) are hot topics in the industry today. Best-of-breed MDM software products have hit the market, and major software vendors have added MDM capabilities to their product suites. SQL Server 2008 R2, for example, introduced Master Data Services (MDS).

However, it’s still early in the adoption curve. There is still confusion in this area, and few organizations are managing all of their master data within an MDM solution.

This section provides more clarity around the topics of master data and MDM and gives guidance around the options that organizations have for managing their master data.

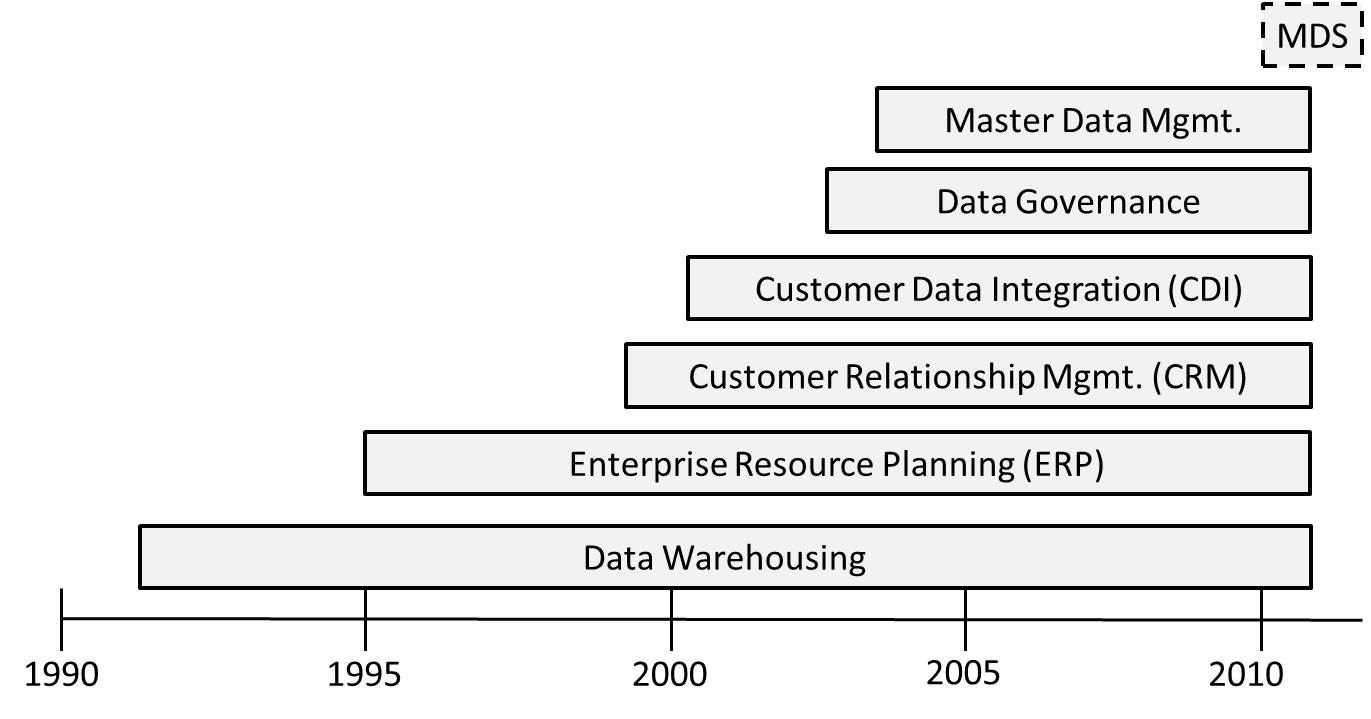
Figure 2-24 illustrates some questions organizations are asking today, such as:

* What is our organization’s MDM solution?
* Is there an overlap between data warehouses and MDM?
* Do I need an MDM software product? And if so, how does my MDM software communicate with my data warehouse?



**Figure 2-24**: Master Data Management questions

Before continuing, it’s useful to look back in time to provide a historical perspective on how the MDM industry arrived at where it is today. Figure 2-25 shows a timeline with significant events leading up to the present.



**Figure 2-25**: Master Data Management timeline

According to the timeline:

* Data warehousing has been part of the database industry’s lexicon since the early 1990s. A major selling point for an EDW was that it provided a “single version of the truth.”
* The 1990s was also a decade that saw wide adoption of LOB systems, starting with Enterprise Resource Planning (ERP). One business driver for these systems was Y2K concerns.
* The advent of the Internet and multiple customer channels drove demand for CRM systems. And wide adoption of CRM created challenges for data warehouses because there were now “multiple versions of the truth” implemented in complex enterprise-level systems such as SAP and Siebel (now Oracle).
* Customer Data Integration (CDI) started appearing in analyst writings around 2000. Although this software category never gained critical mass, it did raise awareness of the need to provide “a single version of the truth” for key business entities.
* The concept of a data governance discipline started to emerge after organizations recognized that data quality was an ongoing enterprise discipline, not a single-point-in-time solution.
* The term Master Data Management began to appear in 2002-2004 and was followed by the emergence of best-of-breed MDM products.
* In 2010, SQL Server 2008 R2 shipped with Master Data Services 1.0.

An Accenture article titled [Master Data Management (MDM) – Hot Topic Getting Hotter](http://www.accenture.com/Global/Technology/Information_Mgmt/Information_Mgmt_Services/R_and_I/MasterDataManagementHotter.htm) outlines MDM common business definitions and management processes for business and reference data. It also stresses the importance of data governance, stating:

*The technology side consists of master data management systems that provide “a single version of the truth.”*

Here’s where the confusion lies. Today’s consensus is that a data governance discipline is necessary, but how about “a single version of the truth”? Where does the authoritative version of the data reside? Is it within a master data database, the data warehouse, or both? What tools are used to create it: the data warehouse and data integration processes, an MDM system, or a combination of the two?

The first step is to define master data and Master Data Management.

### **What Is Master Data?**

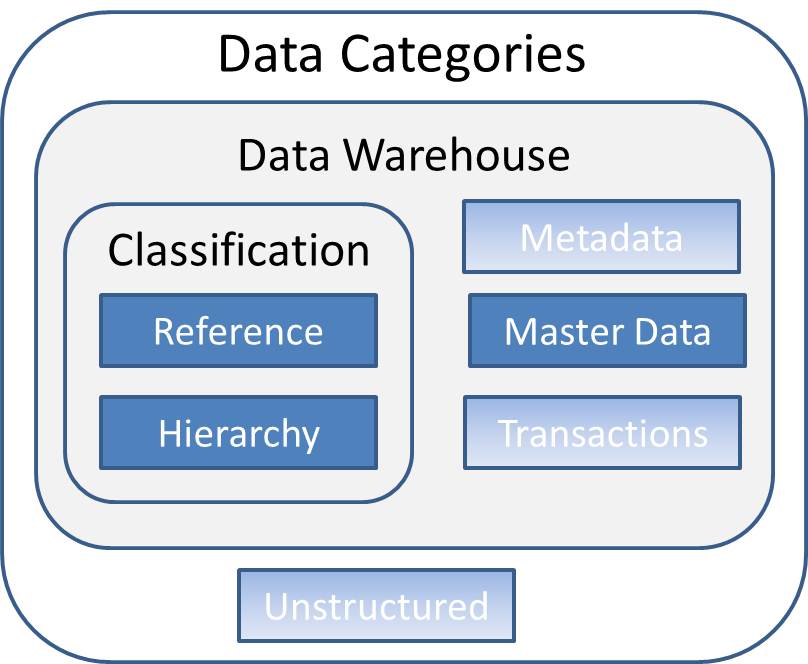
Many papers and articles include definitions of master data. The following definition is from the MSDN article [The What, Why, and How of Master Data Management](http://msdn.microsoft.com/en-us/library/bb190163.aspx), by Roger Wolter and Kirk Haselden:

*Master data is the critical nouns of a business. These nouns generally fall into four groupings: people, things, places, and concepts.*

(Note that this definition is abbreviated for readability.)

This article also categorizes an organization’s data into data types, as Figure 2-26 shows, and describes which ones are stored within a data warehouse.

**Note:** Instead of the term *data type*, which could be confused with the same term commonly used in physical database design, in this chapter, we use *data category*.



**Figure 2-26**: Data categories

The different data categories are:

* **Unstructured** – Data found in email, magazine articles, corporate intranet portals, and so on
* **Metadata** – Data about data, including report definitions, database column descriptions, and configuration files
* **Master** – The critical nouns of a business, which generally fall into four groupings: people, things, places, and concepts
* **Transactional** – Data related to sales, deliveries, invoices, trouble tickets, claims, and other monetary and non-monetary interactions
* **Hierarchical** – One- or multi-level relationships between other data, such as organizational charts and product lines
* **Reference** – A classification of a noun or transaction; often has two columns—code and description—and examples include marital status and gender

Notice in Figure 2-26 that:

* Master data exists within a data warehouse and can be fed from an MDM system.
* Reference and hierarchy data are used to classify master data and transaction data.
* Unstructured data is outside the scope of the data warehouse.
* Metadata is valuable but out of scope for this discussion.

### **What Is Master Data Management?**

The next step is to define Master Data Management. David Loshin in the book *Master Data Management* defines MDM as:

*A collection of best data management practices that orchestrate key stakeholders, participants, and business clients in incorporating the business applications, information management methods, and data management tools to implement the policies, procedures, services, and infrastructure to implement the capture, integration, and subsequent shared used of accurate, timely, consistent and complete master data.*

There are many other definitions, but the key points are:

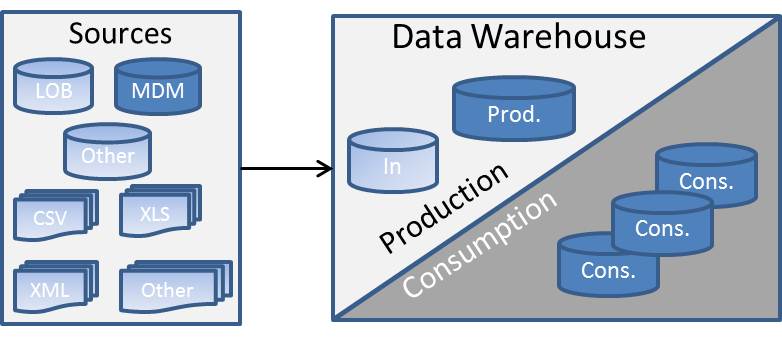
* MDM is not only a technical solution; it’s an ongoing set of processes and an organizational discipline.
* Transactional data is out of scope for MDM.
* Classification data—that is, reference and hierarchy data—is in scope for MDM.

### **Where Do Data Warehousing and MDM Overlap?**

Now that we’ve defined master data and MDM, the next question is: *Where’s the overlap between MDM and data warehousing?*

The simple answer to this question is master, reference, and hierarchy data.

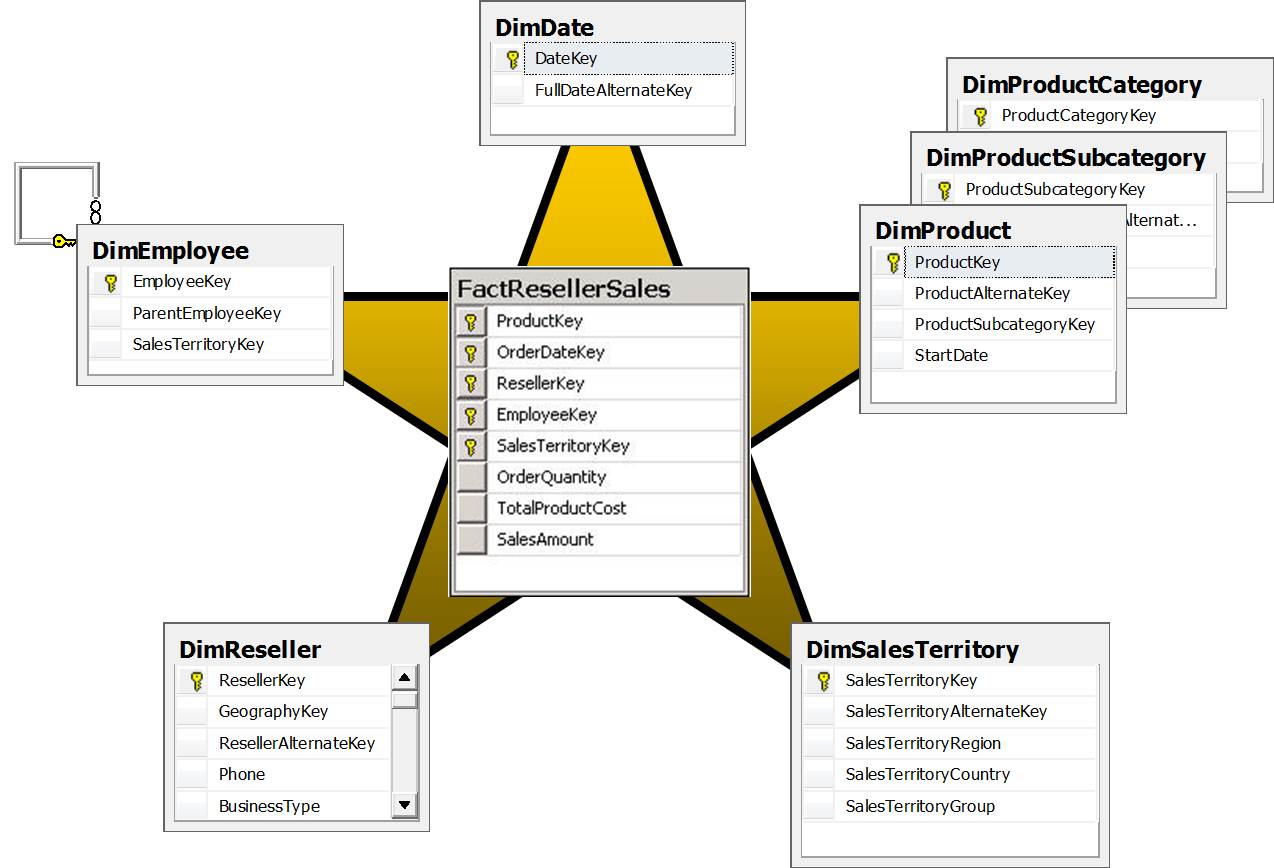
Master data has existed within the data warehouse long before the term Master Data Management was coined. This section shows where master data exists within a data warehouse; Figure 2-27 shows the data stores in scope for this discussion.



**Figure 2-27**: Data stores containing master data

Let’s start with the data warehouse destination and work our way back to the original source.

Most data marts, either within the consumption area or downstream from the data warehouse, use a denormalized data model for optimal query access. Figure 2-28 shows an example within the SQL Server sample data warehouse, AdventureWorksDW2008. (Note that this chapter assumes that the reader is familiar with dimensional data modeling concepts.)

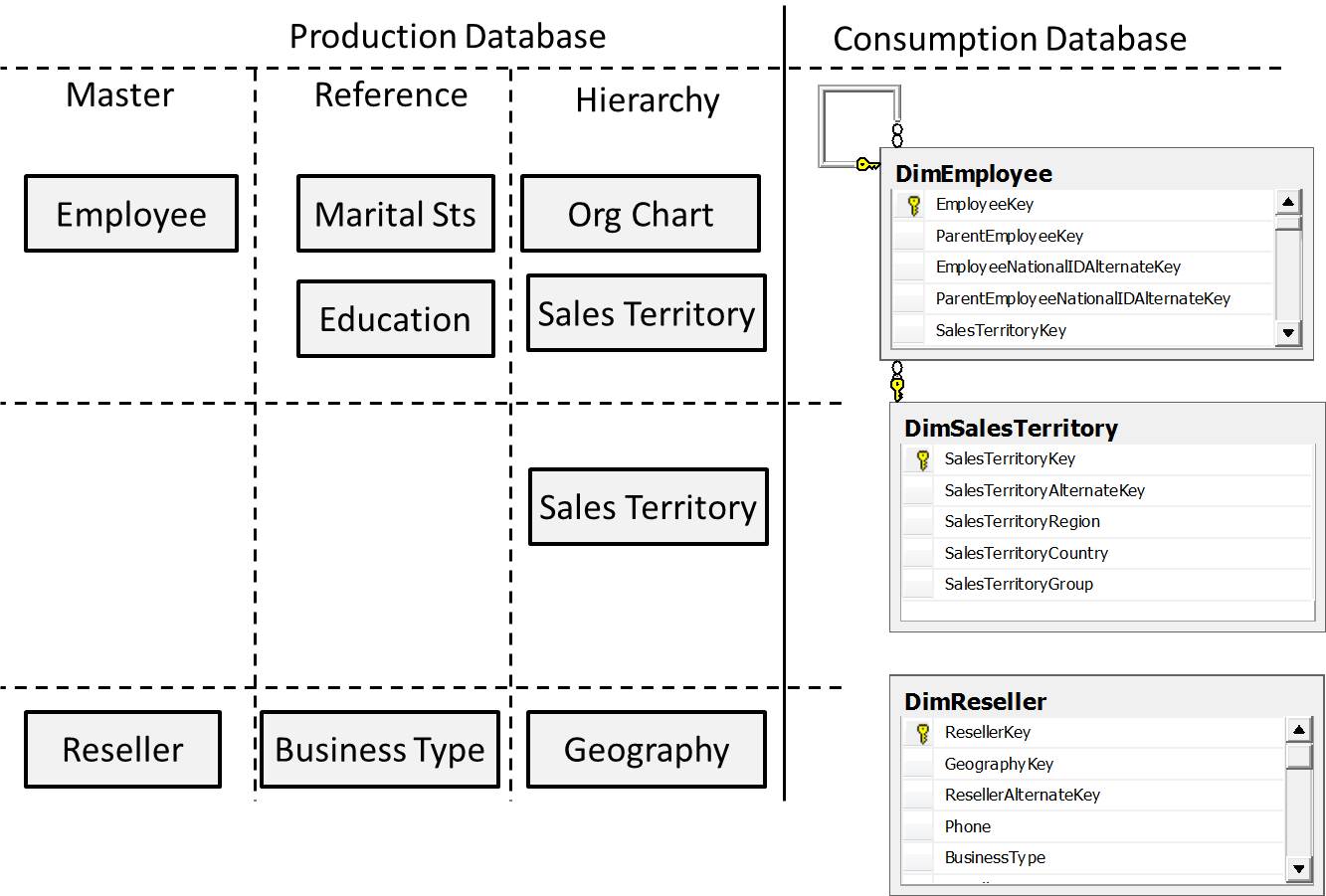


**Figure 2-28**: Reseller Sales snowflake schema

The Reseller Sales fact table contains all sales transactions for the Adventure Works bicycle shop. This fact table contains sales order quantity, product cost, and other numeric values recorded at the sales transaction. The fact table also consists of foreign keys to dimensions, which are used to filter the results from the joined fact and dimension tables. Note that transactional data is out of the scope of MDM and fact tables are not considered master data.

Each dimension in the data warehouse data store is sourced from master data, reference data, and hierarchy data residing in the staging data store. ETL processes process source data en route to the staging area. This processing includes cleansing, consolidating, and correlating master, reference, and hierarchy data as well as linking it to transaction data.

Figure 2-29 shows a partial mapping of three dimensions to their master, reference, and hierarchy data sources within the staging data store. Note that these mappings are conceptual and not actual mappings since the Adventure Works database samples do not include a staging data store.



**Figure 2-29**: Mapping three dimensions to their data categories

Note that the objective of this example is not completeness but rather to show that:

* Each dimension is in a denormalized form.
* Its source is the staging data store.
* The production area contains master, reference, and hierarchy data categories, which are the same categories managed by a data management system.
* The production area also contains transactional data.

Thus, the answer to the question, “Is there an overlap between data warehousing and MDM?” is:

*Yes, the master, reference, and hierarchy data categories exist in both.*

Given this overlap, the next question arises:

*Can an organization forego a data warehouse when it has an MDM software product?*

The answer is no, because MDM software does not manage transactional data. However, transactional data holds all the numeric and monetary values used to measure an organization’s performance—a key objective for a data warehouse.

The next question then becomes:

*Do I need an MDM software product? Does an MDM solution require an MDM software product?*

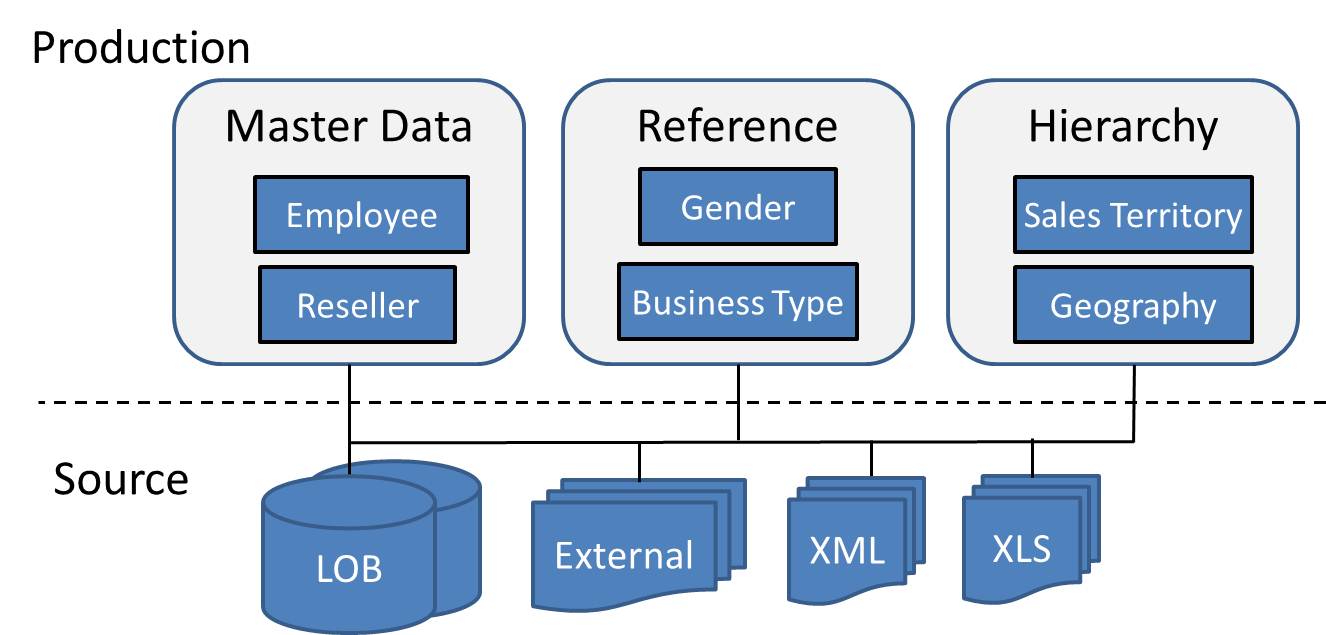
The answer here is maybe. Below are some questions to determine these answers for your organization:

* How many data sources contain master data?
* Does the master data have strong natural keys?
* How mature is the current data warehouse solution? Does it already handle master data?
* What level of manual effort is required for reference list maintenance?
* What if one reference table exists and is separately maintained within multiple source systems?
* What level of hierarchy management exists? What level is required?
* What if the internal hierarchy is a super-set of a standard? For example, what if a health care organization’s diagnostic codes are different than the ICD-9 standards?

Let’s look at how your answers to some of these questions might influence your decision to acquire and utilize an MDM software product.

**How Many Data Sources Contain Master Data?**

It’s useful to understand where master data is sourced from. Figure 2-30 shows an example of master data within a production area and its sources.



**Figure 2-30:** Master data by category and sources

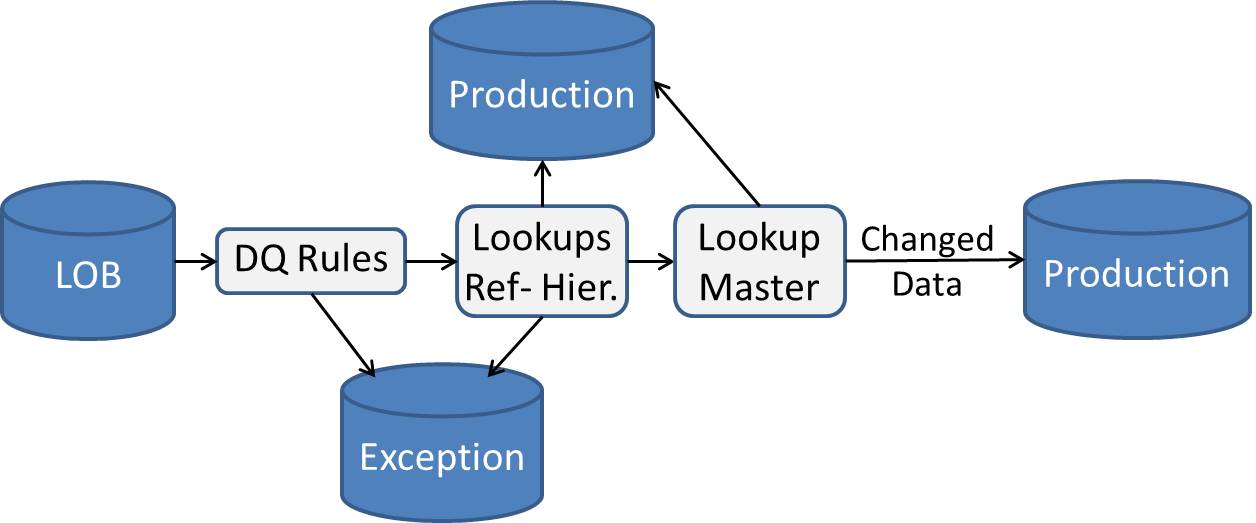
Some notes about this source data:

* Master data resides within the LOB systems.
* Master data can be loaded from multiple LOB systems; this requires the data to be normalized and consolidated.
* Reference and hierarchy data can reside in LOB systems, internal feeds, and external feeds.
* Traditionally, internal feeds were created and maintained with Excel.
* External data traditionally was structured in CSV format, but many industry standards are moving toward an XML format.

The need for a separate MDM software product increases as the number of sources increase.

**How Mature Is the Current Data Warehouse Solution? Does It Already Handle Master Data?**

The next step in determining whether an MDM software product is required is to review the data integration processes already in place. Figure 2-31 shows a high-level view of the data integration process used to load master data from one source to the staging area. Note that this chapter presents these data integration processes at a conceptual level; refer to Chapter 3 for details about data integration.



**Figure 2-31**: Master data integration flow

The data integration steps for master data are as follows:

1. Master data is extracted from the source.
2. Data quality business rules are applied, and data flows to an exception area if it fails a data quality check.
3. Master data is then normalized (i.e., attributes are used as lookups into reference lists and hierarchies). Data may flow to an exception area if the attribute does not match any entries in a reference list or hierarchy.
4. The process checks whether the master data already exists.
5. Master data flows to the production area if it doesn’t exist or if it has changed.

The above conceptual process for loading master data is very similar to processes used to populate dimensions and is familiar to data integration developers.

The simple answer to whether an MDM software product is required for loading master, reference, and hierarchy data is *no* if existing data integration processes already exist and are working without issues.

If these master data integration processes are not in place, the answer is then *maybe,* depending on the complexity of the master data integration processes, combined with other factors such as:

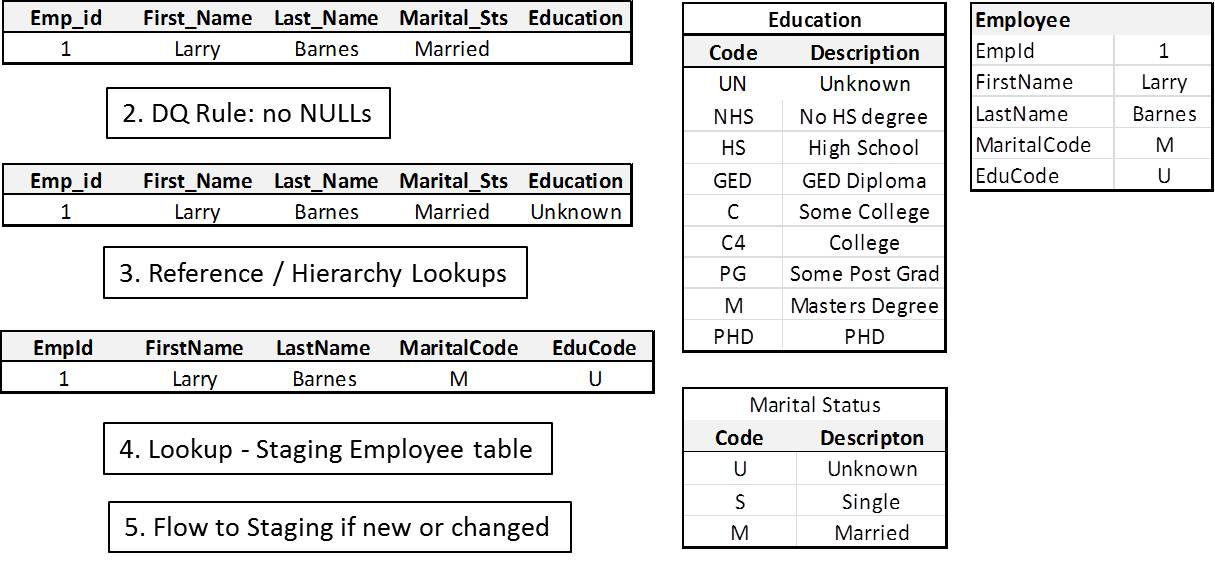
* Whether the existing data warehouse team (developers and operations) are comfortable learning a new technology
* Whether the new technology is affordable (both in acquisition and ongoing costs).

The takeaway from this section is that introducing an MDM software product has associated costs. The decision on whether an MDM software product is required should take these costs into consideration along with the complexity of the master data processes.

The next section introduces two master data scenarios and the data integration processing required for each.

**Employee Data – One Source**

In this first scenario, the data exists in one and only one source system, as Figure 2-32 shows. In this scenario, the Employee table is populated with Education and Marital Status reference data sourced from the same system.

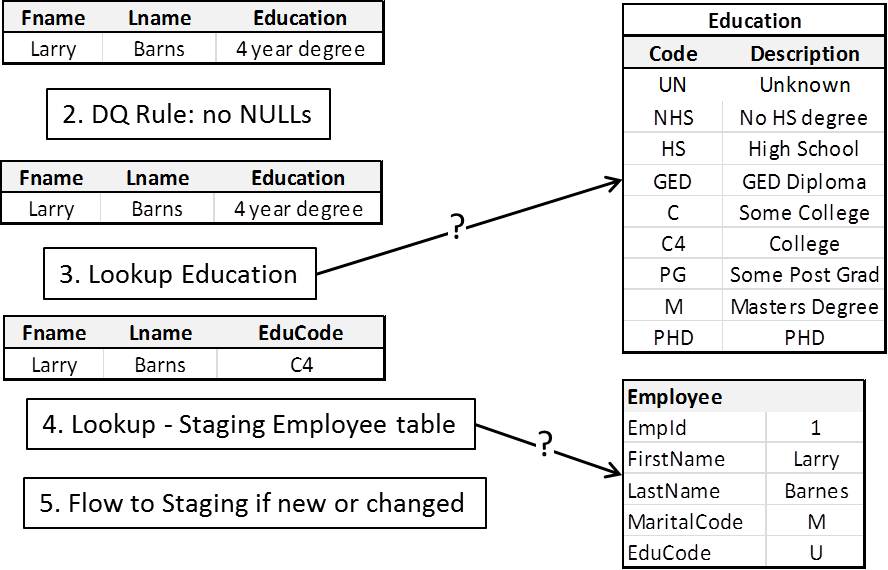


**Figure 2-32**: Master data example – one source

A separate MDM product is not required in the simplest of scenarios.

**Employee Data – Multiple Sources**

However, master data often exists in more than one place—typically in multiple source systems, but it can also be an external or internal feed. The next example assumes that Education data comes from an external feed, as shown in Figure 2-33.



**Figure 2-33**: Master data example – multiple sources

Note that even in this simple multi-source scenario the following issues arise:

* **Reference lookup failures** – The reference table in staging is sourced from the LOB system. The second source containing employee education information uses different descriptions. These cases require intermediate mapping tables, which take codes or descriptions from the source and map them into the staging reference table.
* **Production area table lookup failures** – The lookup against the production area Employee table fails in the above scenario. It demonstrates two common reasons for these lookups failures:
  + **Lack of a strong natural key** – Employee Id is a strong natural key and is ideal for comparing employee data across internal LOB systems if, and only if, this key exists in each system. If it doesn’t, then the source-to-stage lookup becomes more complicated and error-prone.
  + **Inexact matching** – Name and/or address matching are perfect examples of where the matching process is not completely deterministic. In these cases, either a fuzzy matching algorithm or product or product capabilities specifically developed for name and address matching is required.

SQL Server 2008 R2 Master Data Services (MDS) may be beneficial in this scenario if the implementation team thinks that leveraging MDS capabilities are preferable to adding capabilities to existing features (such as Fuzzy Lookups). Here are a couple of things to think about when considering MDS:

* **Reference table management** – What processes/interfaces are used for creating and maintaining reference tables and mappings that aren’t sourced directly from a LOB system?
* **Name/address matching** – Can the team leverage SQL Server Integration Services (SSIS) Fuzzy Lookups and provide tools for data stewards to make a human decision when the fuzzy matching doesn’t produce strong results?

**Complex Master Data: Multiple Sources and Hierarchy Management**

The multi-source Employee data example above provides a simple illustration of master data within organizations. Many organizations have more complicated master data needs as well as other advanced MDM needs such as creating and maintaining multiple versions of very complex hierarchies (e.g., organizational charts, chart of accounts, disease classification, etc.).

In these cases, organizations can benefit from advanced data integration capabilities such as merge/match. In addition, version control allows data stewards to work on new reference data and hierarchies prior to publishing them in the data warehouse. Workflow approval supports a multi-tier review process, and role based security lets data stewards work only on subsets of master data. Each of these capabilities supports complex master data environments and are features within SQL Server 2008 R2 MDS.

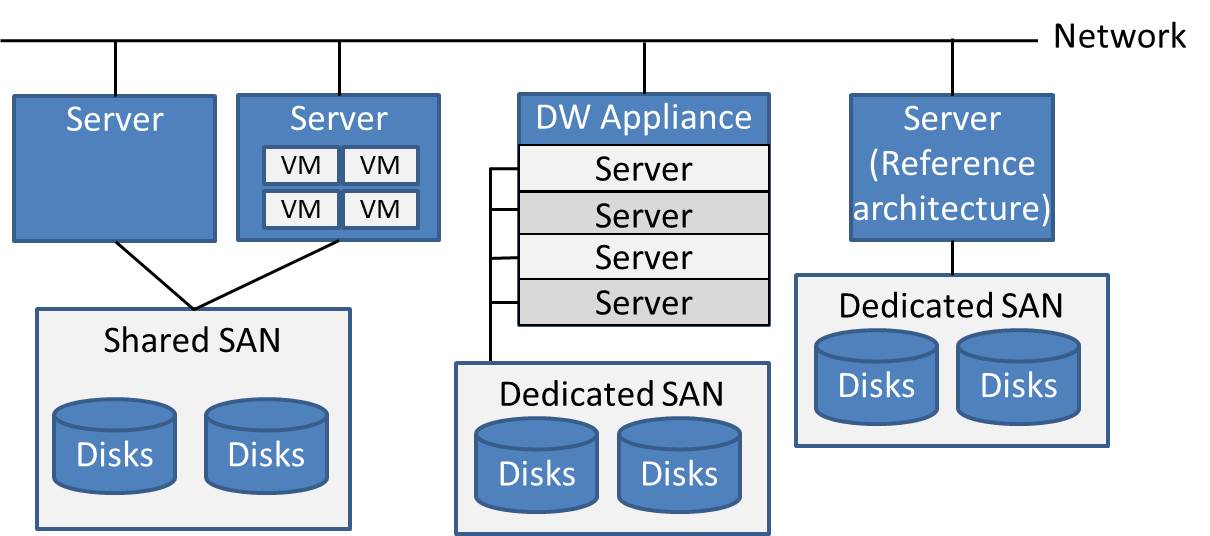
In summary, master data and MDM are hot topics today but are old problems within data warehousing. Many existing data warehouse implementations already process master data effectively. However, organizations with complex master data needs that don’t have enterprise MDM capabilities in place can benefit from adopting an enterprise MDM solution such as SQL Server 2008 R2 MDS.

Now that we’ve covered the key concepts involved in data warehouse architecture, we are ready to address the responsibilities that are owned or co-owned by the data architecture team.

## Platform Architecture

The data architect is typically the responsibility of the team that selects the platform architecture for a data warehouse. The final selection of a hardware and software platform depends on a combination of many factors, including performance and adherence to corporate standards. This section provides a brief overview of this topic; however, a detailed discussion of platform architecture is out of scope for this document.

Data warehouse data volumes and usage requirements require large dedicated hardware servers and storage hardware. Historically, organizations deployed SQL Server data warehouses by getting a big server with lots of CPUs and lots of memory and allocating lots of space in the Storage Area Network (SAN) for the data warehouse databases. However, alternative options have emerged over the past few years, as shown in Figure 2-34.



**Figure 2-34**: Platform architecture – server options

This illustration shows the following server options:

* **Server** – The historical Symmetrical Multiprocessor (SMP) configuration is a multi-processor architecture with lots of memory running SQL Server Enterprise Edition.
* **Server virtualization** – This configuration maximizes server resources by supporting the running of multiple virtual instances of SQL Server. This is a potential configuration for pre-production data warehouse environments, including test, staging, and QA.
* **Data warehouse appliance** – SQL Server 2008 R2 PDW is a massively parallel processor (MPP) architecture supporting both centralized EDW and hub-and-spoke architectures on dedicated SAN storage.
* **Server/reference architecture** – Microsoft’s Fast Track Data Warehouse architecture is a reference hardware configuration tailored to run a SQL Server data warehouse workload.

We’ll look briefly at each of these options, but the selection, acquisition, and deployment of data warehouse platform architectures is out of scope for this document.

### **Data Warehouse Server**

The traditional SQL Server data warehouse server for medium-size data volumes is a 64-bit machine running SQL Server Enterprise Edition and configured with a lot of CPU and memory. Given the memory- and compute-intensive workloads for both querying and populating the data warehouse, organizations should purchase hardware with the maximum CPUs and memory configuration they can afford.

These data warehouse servers are typically connected to the corporate SAN. This shared SAN provides centralized administration and is more flexible than storage directly attached to a server. This simplifies administration operations such as backup and disaster recovery. However, the SAN is the primary bottleneck for most data warehouse workloads experiencing performance issues, so pay careful attention to SAN configurations.

One issue with this large-server approach is that it becomes very expensive because multiple servers are required for different phases of the data warehouse lifecycle, including development, test, QA, and production. Traditionally, organizations typically purchased smaller servers for all servers not in production, but virtualization has provided another option.

### **Server Virtualization**

Server virtualization has emerged as a dominant theme in computing over the past 5 years. Virtualization reduces the number of servers in data centers, which results in lower IT costs from acquisition through ongoing management. It also provides a flexible mechanism for spinning up and tearing down server environments.

This flexibility is useful for non-production environments, but dedicated servers are still the norm for production data warehouse environments. SQL Server 2008 R2 Datacenter Edition is a new product offering from Microsoft for organizations looking for a virtualized SQL Server solution.

### **SQL Server Fast Track Data Warehouse**

The challenges involved in correctly acquiring and configuring a data warehouse hardware and software platform led Microsoft to introduce the SQL Server Fast Track Data Warehouse. This is a “scale-up” reference architecture targeted for data warehouses containing up to 48 TB of data. Options include different reference hardware configurations from HP, Dell, Bull, EMC, and IBM.

The benefits to this solution are that preconfigured, industry-standard hardware provides lower cost of ownership through better price/performance, rapid deployment, and correct configurations.

For details about SQL Server Fast Track Data Warehouse, see these links:

* [SQL Server Fast Track Data Warehouse Web site](http://www.microsoft.com/sqlserver/2008/en/us/fasttrack.aspx)
* MSDN white paper [An Introduction to Fast Track Data Warehouse Architectures](http://msdn.microsoft.com/en-us/library/dd459146(SQL.100).aspx)

### **Data Warehouse Appliances**

SQL Server 2008 R2 PDW is a highly scalable data warehouse appliance that uses MPP software architecture based on the DataAllegro acquisition. Its target workload is data warehouses containing up to 400 TB of data.

In a traditional, symmetric multi-processing (SMP) architecture, query processing occurs entirely within one physical instance of a database. CPU, memory, and storage impose physical limits on speed and scale.

A PDW MPP appliance partitions large tables across multiple physical nodes, each node having dedicated CPU, memory, and storage and each running its own instance of SQL Server in a parallel, shared-nothing design. All components are balanced against each other to reduce performance bottlenecks, and all server and storage components are mirrored for enterprise-class redundancy.

PDW is a distributed architecture that can act both as the centralized EDW and the hub in a hub-and-spoke architecture, as covered earlier.

You can read about PDW at these links:

* [SQL Server 2008 R2 Parallel Data Warehouse Web site](http://www.microsoft.com/sqlserver/2008/en/us/parallel-data-warehouse.aspx)
* TechNet white paper [Hub-And-Spoke: Building an EDW with SQL Server and Strategies of Implementation](http://technet.microsoft.com/en-us/library/dd459147(SQL.100).aspx)

### **Which Server Option Should You Choose?**

Large 64-bit servers running SQL Server Enterprise Edition can support multi-terabyte data warehouses and are a solid choice for organizations today. Here is some guidance for considering one of the alternative options:

* **Non-production environments** – SQL Server 2008 R2 Datacenter Edition provides the flexibility of quickly spinning up and tearing down environments, which lowers the total cost of ownership (TCO) for non-production data warehouse environments.
* **Performance issues** – Performance issues, both directly experienced and projected, can be addressed by scaling up with SQL Server Fast Track Data Warehouse. Pre-configured solutions help minimize common performance problems, such as SAN bottlenecks due to incorrect configurations.
* **Enterprise data warehouse** – PDW provides a platform for the largest organizations’ EDW needs. Its hub-and-spoke architecture supports data marts optimized for target business consumer communities and simplifies data integration between the data warehouse and downstream data marts.

These options allow organizations to standardize their data warehouse on the SQL Server platform—whether the data warehouse is measured in gigabytes or terabytes—first by selecting a platform and then by choosing more scalable platforms as their data warehouse matures.

Now that we’ve briefly covered platform architecture, the next section addresses the data warehouse database architectures implemented on that platform.

## Database Architecture

This section focuses on how different data warehouse data stores and data categories are organized into logical and physical databases.

Choosing the most appropriate database architecture for a data warehouse system is an important aspect of the overall data warehouse strategy: Poorly implemented database architecture will negatively affect data warehouse performance and overall user experience.

It’s not uncommon to find SQL Server data warehouse implementations in which no effort was made to create more than a single default filegroup, which results in all data, indexes, and logs residing on the same number of disks. Despite having a strong data model, such a data warehouse initiative will be labeled a failure because of:

* Slow queries
* Backup files becoming too large, resulting in excessive backup and restore times
* Data files becoming too large to manage, causing excessive execution times for rebuilding indexes and updating statistics

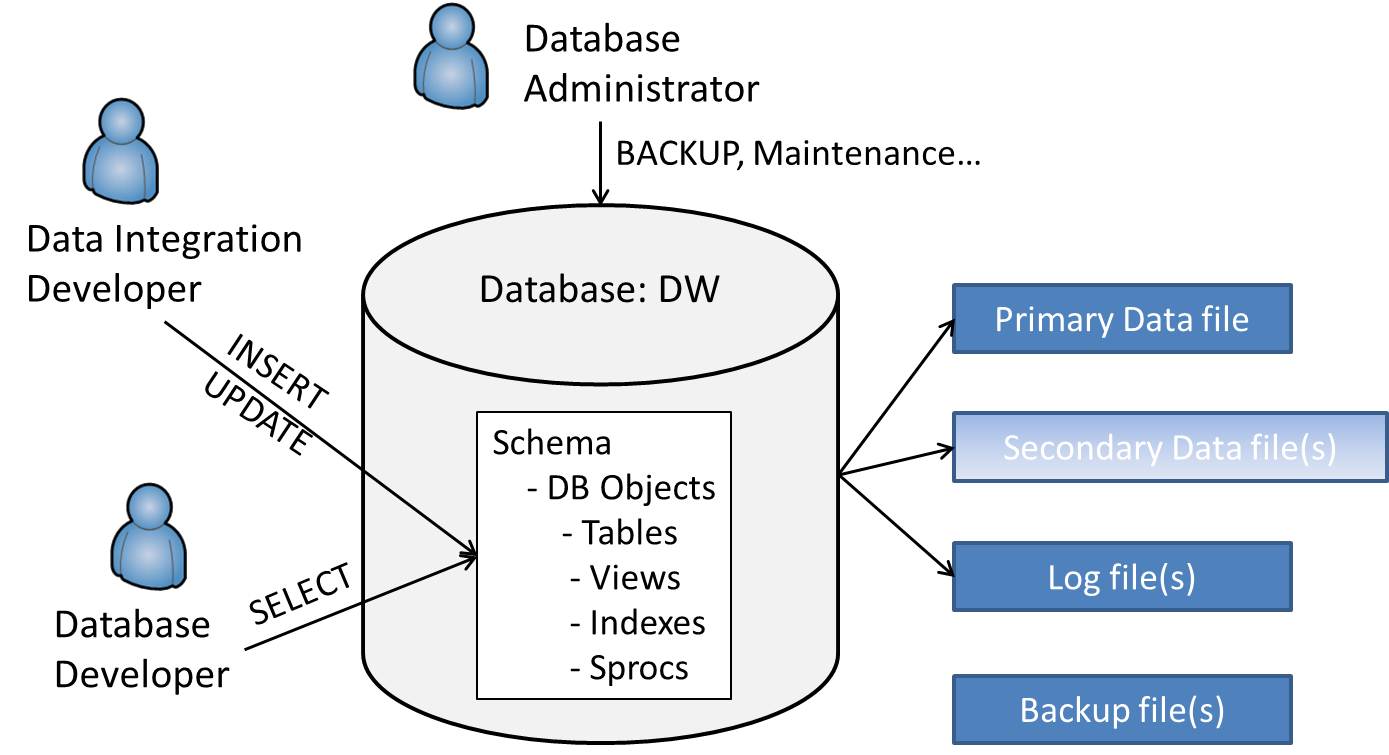
This underscores the need for an effective physical database design consisting of files and filegroups. In addition, a good database schema design can make database objects easier to access and simplify the security implementation.

### **Databases, Schemas, and Filegroups**

Data modelers are responsible for developing the data warehouse data models, which are accessed by the three roles mentioned below. A data warehouse can be viewed differently depending on your role and your objectives:

* **Database developers** view the data warehouse through a SQL lens. Their objective is to select the correct information from the data warehouse as efficiently as possible.
* **Data integration developers** view the data warehouse as a destination. Their objective is to populate the data warehouse as efficiently as possible.
* **Database administrators** view the data warehouse as a precious resource. Their objective is to ensure that the data warehouse is operational, secure, and high-performing—both when serving up data to business consumers and during database maintenance operations.

Figure 2-35 shows these different roles and how they relate to the data warehouse.



**Figure 2-35**: Data warehouse database logical and physical view

The data warehouse database can be viewed both as a logical entity accessed through SQL and as a physical entity consisting of multiple files stored within the disk subsystem.

Every SQL Server database is stored on disk and has two or more files:

* **Primary data file** – A database has only one primary data file, which contains startup information and pointers to other files in the database.
* **Log file(s)** – A database can have one or more log files.
* **Secondary data file(s)** – These optional files are used to store database objects. Although optional, the use of secondary data files is highly recommended for a SQL Server data warehouse.
* **Backup file(s)** – These files are used to back up information from the data warehouse.

All database files are organized into one or more *filegroups*. Every file has one filegroup parent, and every database object is associated with a filegroup. The primary filegroup stores all system tables.

A key database architecture task is determining which database objects reside in which filegroups. This task must take the different developer and DBA activities into consideration, including select, insert, and update activity; backup and restore processes; and database maintenance operations such as re-indexing, the updating of statistics, defragmentation, and so on.

**Databases**

Databases are the parent for both logical schemas and physical filegroups and files. Before we go deeper into physical filegroup configurations, it’s useful to review how database objects are logically accessed and organized within SQL Server.

The SQL Server Transact SQL (T-SQL) language supports three-level naming when accessing database objects within the same server:

*[Database].[Schema].Object*

Both database and schema are optional, as shown in the following query, which assumes our current database is AdventureWorksDW2008:

*SELECT \* FROM DimCustomer*

The following query, which specifies all three levels, returns identical results as the above query:

*SELECT \* FROM AdventureworksDW2008.dbo.DimCustomer*

Every database table has a schema, which we look at in a moment. When a schema isn’t included, the default schema is assumed. The default schema is dbo or the default schema explicitly assigned to the user account by the DBA.

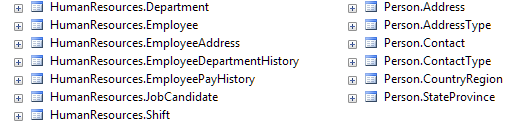
Note that SQL Server linked servers extends this query model to four-level naming, as shown below, to access an object on another server:

[Linked-Server-Name].[Database].[Schema].[Object]

However, linked servers are not regularly used in production due to performance and security reasons. Every linked server request has to first connect to a remote server and then retrieve the results before returning them to the requester. More often, SSIS data flows are used to populate tables local to the server, which are then directly accessible within a SQL Server instance.

**Schemas**

Schemas provide for logical grouping of tables. Figure 2-36 shows an example in which tables are grouped by the logical domain they belong to.



**Figure 2-36**: Subset of schemas from the AdventureWorks2008 sample database

The decision to create schemas is made by the data warehouse team in the design phase. The following are reasons for using schemas:

* **Logical grouping** – The ability to logically organize tables into groupings or subject areas makes it easier to navigate complex databases such as data warehouses. The example above illustrates how tables can be organized into subject areas.
* **Logical partitioning** – There are many other way to logically group tables; one example might be to group by entity. For example, the Delta-American Airlines merger creates the need to consolidate data from the LOB systems used by each entity prior to the merger. Creating a Delta and AmericanAirlines schema with identical table structures, such as DimCustomer, would be an intermediate structure prior to loading the consolidated DimCustomer table.
* **Security** – The ability to grant and restrict access to database users and roles through different schema simplifies the security approach for a database. For the example in Figure 2-29, you could:
  + Grant SELECT access to the HR department for the HumanResources schema
  + Grant INSERT, UPDATE, and DELETE access to the HR Admin role for the HumanResources and Person schema
  + Grant SELECT access to all data warehouse users for the Person schema

The downside to creating schemas is that the schema name must be included in all database object references. This may not seem like a significant issue for developers and DBAs, but it could be an issue for downstream business consumers. Also, any change in the schema name will require a change to every SQL statement accessing objects within the schema, except in the case where the user is referencing objects within their default schema. Changes in the schema name will not impact the underlying physical structure because schemas are a logical construct.

When using schemas in SQL Server, it is important to recognize that schemas are not equal to owners. Thus, it is not necessary to change owners of objects when owner accounts are being removed. A schema does have an owner, but the owner is not tied to the name.

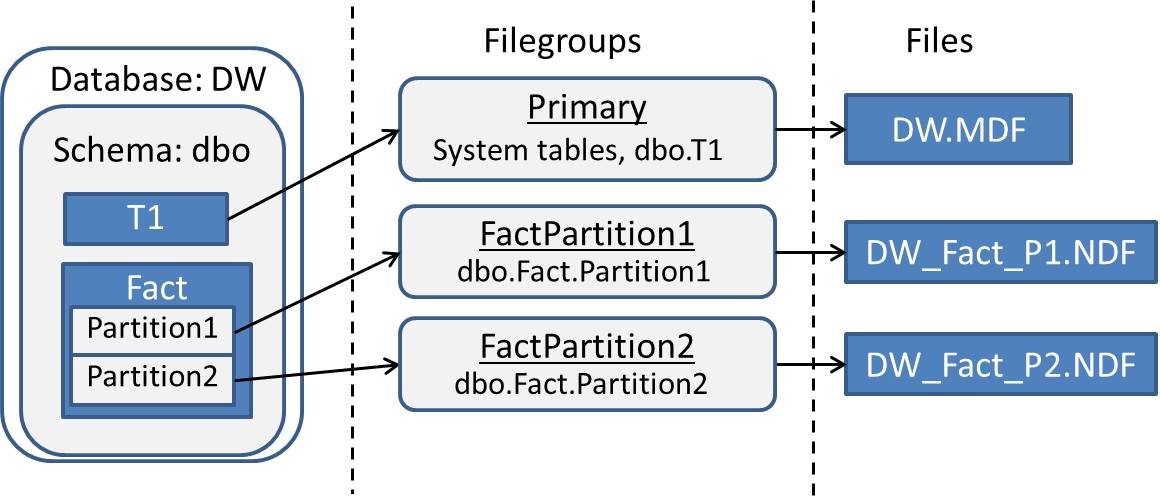
Note that dbo is the only schema supported in the initial PDW release.

**Filegroups**

A core database architecture concept is the filegroup, which contains database files and represents the physical aspect of the architecture.

Database tables, indexes, and logs are created on filegroups, while filegroups contain one or more physical files. Proper management of files and filegroups is instrumental to an efficient data warehouse design and maintenance. Here are some scenarios where wise filegroup architecture can provide significant value:

* **Partitioned fact tables**– Large fact tables (with more than 100 million rows) in one database file have benefit from partitioning. Modifying such tables to have their data divided among multiple physical files, with each file stored on a separate physical disk array, would enable the Database Engine to perform multiple parallel reads (one per file), improving read performance. Figure 2-37 shows a simple scenario of a fact table with two partitions.



**Figure 2-37**: Partitioned fact table

* **Backup/restore and archiving** – For VLDBs, it may become impossible to manage backups and restores for certain files in timely fashion unless multiple filegroups are introduced and separately backed up.
* **Other database maintenance activities** – Partitions also have advantages for indexing, consistency checking, updating statistics, and other maintenance tasks.

For more information about SQL Server filegroups, see the MSDN article [Files and Filegroups Architecture](http://msdn.microsoft.com/en-us/library/ms179316.aspx).

**Database Files**

Regarding database files, it is important to consider the physical architecture of your hardware infrastructure to take advantage of separate disk controllers. In addition:

* Placing non-clustered indexes on files separate from files containing data is a common practice to improve data warehouse performance.
* A strategy for where to store tempdb in SQL Server is an essential part of an overall database architecture because if not properly architected, write operations to this frequently used database often conflict with read/write operations for data and log files.

The next section of this chapter will address the questions: How many databases should be in my data warehouse, and what factors affect this decision?

**Database Files and Storage**

To have a high-performing data warehouse, you need to understand and properly manage storage for database objects. The MSDN article [Storage Top 10 Best Practices](http://msdn.microsoft.com/en-us/library/cc966534) is a good place to start.

Most data warehouses use SAN storage, with disks organized in applicable types of RAID configurations. Depending on disaster recovery policies, some data warehouse systems will reside on database clusters, but which services should be set up for failover depends on an organization’s particular business requirements.

### **Considerations for Physical Database Design**

This section outlines several considerations for physical database design as it relates to logical design and architecture decisions. For more about best practices for partitioning with SQL Server, see the partitioning section of Chapter 4.

**Capacity Planning and Designing for Change**

Initial capacity planning is required for every data warehouse and includes estimates for initial data size and data growth. In addition, physical designs need to be implemented so that new disks and servers can be added without affecting systems already in production.

**Templates in Physical Design**

The process of deploying the physical database architecture can be simplified by establishing templates. These templates are deployed to a new database with predefined architecture for filegroups and files. However, these patterns need to be adjusted for special cases—for example, when complexity and size of data in the fact tables requires multiple files for more efficient partitioning.

Some of the recommended practices for using templates in database architecture include maintaining scripts for template databases, including all relevant CREATE statements. These scripts are modified based on naming standards used in a particular data warehouse implementation and are executed to create new database objects.

**Partitioning**

Partitioning is a valuable technique for very large tables within the data warehouse. Partitioning creates smaller clusters of data, which enables maintenance operations to be applied on a partition-by-partition basis. Partitioning also enables minimum latency because source data is continuously loaded into a passive partition, which gets switched with active partitions on set schedules.

In physical design, typically for purposes of producing proof-of-concept models, the partitioning strategy is implemented after business rules, entity relationships, and attribute and measure definitions are established.

Partitioning is a key tool employed by both DBAs and database developers when working with very large tables and is covered in more detail in Chapters 4 and 5.

**Striping**

Striping is associated with the type of RAID levels implemented as a part of the data warehouse architecture. RAID levels 0, 1, 5, and 10 are typically implemented with SQL Server. This is an important aspect of database architecture because striping improves read performance by spreading operations across disks. The following SQL Server Customer Advisory Team (SQLCAT) article contains more information about SQL Server striping and RAID levels for SQL Server:

* [Storage Top 10 Best Practices](http://sqlcat.com/top10lists/archive/2007/11/21/storage-top-10-best-practices.aspx)

**Data Compression**

Data compression can help reduce the size of the database as well as improve the performance of I/O-intensive workloads. However, CPU cycles are required on the database server to compress and decompress the data while data is exchanged with the application.

SQL Server provides two levels of data compression: row compression and page compression.

* Row compression helps store data more efficiently in a row by storing fixed-length data types in variable-length storage format. A compressed row uses 4 bits per compressed column to store the length of the data in the column. NULL and 0 values across all data types take no additional space other than these 4 bits.
* Page compression is a superset of row compression. In addition to storing data efficiently inside a row, page compression optimizes storage of multiple rows in a page by minimizing data redundancy.

It is important to estimate space savings and apply compression only to those tables and indexes that will yield reduced I/O and memory consumption due to the reduced size. The [sp\_estimate\_data\_compression\_savings](http://msdn.microsoft.com/en-us/library/cc280574.aspx) stored procedure can be used for SQL Server 2008 R2 databases.

Data compression for SQL Server is explained in detail in the SQLCAT article [Data Compression: Strategy, Capacity Planning and Best Practices](http://msdn.microsoft.com/en-us/library/dd894051(SQL.100).aspx).

## Data Modeling

Success of a data warehouse implementation initiative is greatly affected by what data modeling technique is used and how it is implemented. Data modeling begins the process of structuring the data elements that make up a data warehouse.

In this process, requirements are analyzed and defined to support the objectives of the data warehouse. The results of this analysis are contained within conceptual, logical, and physical data models. Before we dig into the different types of data models, let’s review some key aspects of data modeling for a data warehouse.

**Inherent Definition**

Data modeling identifies definitions of entities in regard to their inheritance properties. This provides clear relationships and dependencies. For example, a result of inherent definition modeling would be a definition that a product can belong to multiple categories in a manufacturing data warehouse implementation.

**Alignment through Logical Decomposition**

Entities, relationships, and dependencies as well as metrics are modeled across subject areas of a system. Commonalities in modeling are identified and appropriated throughout systems. Modeling of each component of a data warehouse is aligned with a business requirement relevant to a particular subject. Thus, all models in a data warehouse are included in an overall data warehouse strategy parsed through initiatives and relevant projects.

**Identifying Complexity, Priorities, and Needs for Simplification**

Data modeling efforts become more complex as you define entities and the relationships between them. Incomplete data models lead to inadequate business rules, resulting in a sub-optimal solution. However, it’s also important to prioritize data modeling efforts in accordance with project milestones and deliverables.

Decisions about the importance of defining all possible attributes of a single entity versus defining just desired outputs and metrics of a data warehouse system will affect the overall result of data warehouse initiatives.

**Eliminating Non-Critical, Isolated, and Unrelated Data**

An overarching objective for a data warehouse is often to include all data that exists in an enterprise. However, it’s important to keep the focus on data as it relates to the overall scope of requirements and priorities. Failure to do so can result in extending timelines for deliverables, analysis paralysis, and eventual failure of a data warehouse.

**Data Modeling Variables and Inputs**

A sub-optimal data model affects the ROI of a data warehousing effort, including higher costs for systems, infrastructure, and business consumer discovery and access as well as additional resources for maintenance and modification. Business rules and requirements are inputs into data models, and this dependency translates into data model changes when business rules and requirements change.

### **Data Modeling Overview**

A data warehouse data model includes definitions of entity types, which are typically grouped into dimensions, reference tables, hierarchies, fact tables, and bridge tables. This section will cover each of these elements. In addition, data modeling includes the classifications of attributes, measures, relationships, and integrity rules.

The process of data modeling begins with the translation of business requirements into *conceptual models,* which include data definitions.

Following the conceptual model is a *logical model* that outlines the actual implementation. The logical data model identifies both data objects and the relationships between them. One conceptual model often yields multiple logical models, which are referred to as *subject* or *context* models.

The *physical* model rounds out the data modeling process and includes additional physical information, such as physical data types, referential integrity, constraints, and partitions.

Data warehouse teams are faced with several options when implementing data models:

* The team can work in a serial order, where the data modeling team completes the entire model before moving to the next phase (i.e., conceptual model first, then logical, and finally physical).
* Or the data modeling team can work by subject area, iterating through the conceptual, logical, and physical models for one subject area before moving on to the next one.
* Alternatively, the team can perform a parallel effort, in which different areas within the data model are in different development stages (i.e., conceptual, logical, or physical).

The parallel effort typically involves enterprise data modeling tools and requires the team to traverse between subject areas as well as to move between the conceptual, logical, and physical data model within a subject area until the overall model is complete. However, the reality of data warehouses is that data models change and subject areas are continuously added. This reality frequently translates into a parallel or continuous data modeling effort.

For example, in a recent data warehouse project, an organization realized it needed to add support for analyzing data warehouse measures in respect to multiple time zones. The data warehouse had been in production for over a year when the request came in from business stakeholders, but this added requirement was part of a never-ending need to support changes in how the business operated.

This is a very common scenario, and it should come as no surprise to data warehouse developers that they will have to be agile and creative in combining conceptual, logical, and physical data modeling efforts to support new and changing business needs.

The following sections provide more detail on the purpose of the different data models and the activities that take place during each stage.

### **Conceptual Model**

Major characteristics of a conceptual data model relate to the following points:

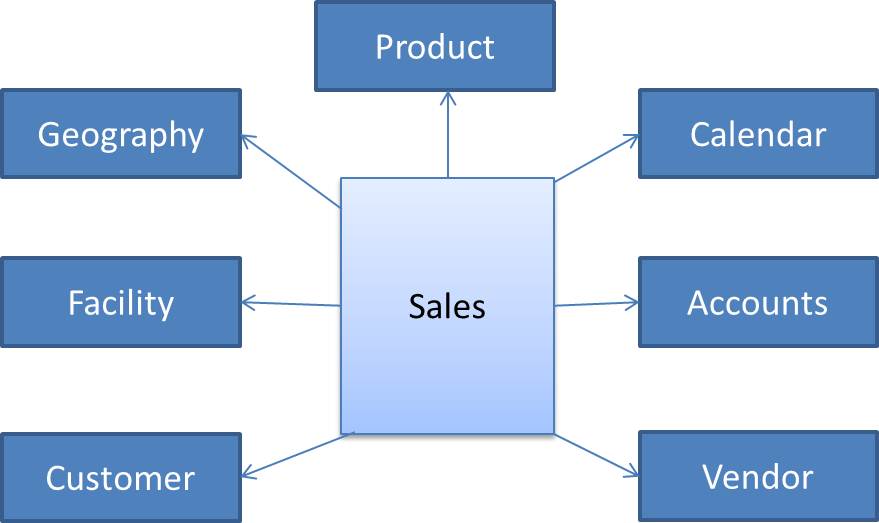
**Reduce the Enterprise Highest-Level Nouns**

Creating one common definition for common entities reduces ambiguity when comparing data and results across different business and organizational units. Examples of these efforts are consolidated definitions for Customer, Product, and Inventory entities.

These nouns may be referred to as domains, master objects, or another abstract term. For example, the Product domain in physical terms will include numerous tables, but in the conceptual phase, it is identified as a single abstract term.

**Limit the Conceptual Model to 1 Page, or Less than 8 Objects**

The purpose of the conceptual model is to provide a high-level understanding of the major entities and business processes in scope for the data warehouse. The general rule is that if the conceptual model can’t fit on a single page, it’s too granular. Figure 2-38 is an example of a high-level conceptual model.



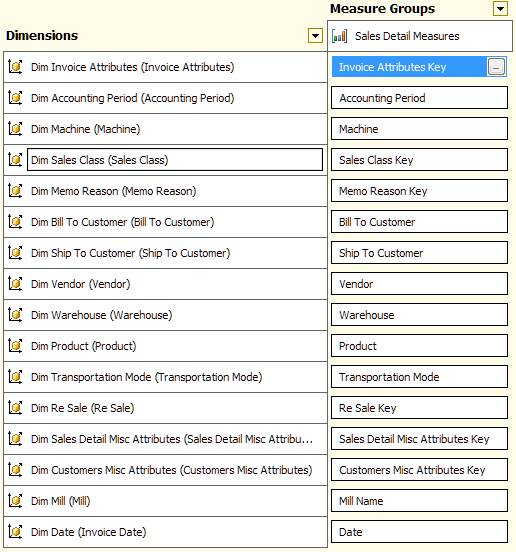
**Figure 2-38**: Conceptual data model

**Subject or Context Model**

The context or subject data model defines intersections of conceptual data elements with subject areas in a data warehouse. Here are the main characteristics of a subject data model:

* The subject model presents the highest-level functions of the enterprise (Sales, Marketing, Finance, IT).
* Some high-level concepts are also subjects (Finance).
* The grid of concepts by subject should show that most subjects need most concepts.
* The subject model should anticipate the design of data marts or reporting groups.

Figure 2-39 shows an example of a subject model represented via a reverse-engineered bus matrix of dimension usage in SSAS.

****

**Figure 2-39**: Subject data model

### **Logical Model**

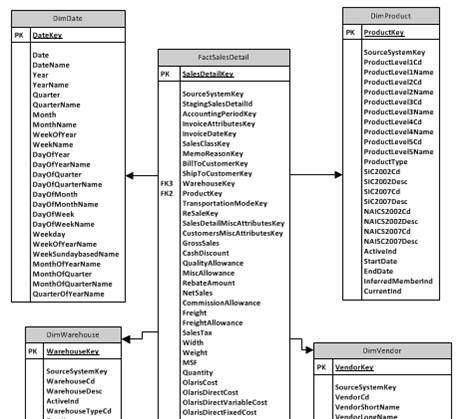
The logical data model deepens the analysis of data elements in scope of a data warehouse effort and includes more detail about the data.

Included in the logical model are:

* **Entities, attributes, and domains** – The logical model includes more granular information than the conceptual model, defining logical attribute names and domains in which entities belong.
* **Normalization/denormalization and relationships** – Primary and foreign key relationships are identified as well as objects containing various levels of hierarchies and entity relationships.
* **Advanced concepts** – The following concepts are also addressed in the logical model: sub-typing (inheritance), one-to-many relationships (rather than 0-to-many), null meaning and operations, and row interdependence (chaining in history).

For example, attributes for product type, product category, product sub-category, and product are defined as well as relationships between these hierarchies. In transactional systems, data is normalized and definitions of primary and foreign keys as well as relevant relationships are included in the model. In analytical systems, data can be denormalized, and in this case, logical models will also include definitions of business keys on dimensions and composites for surrogate keys on facts.

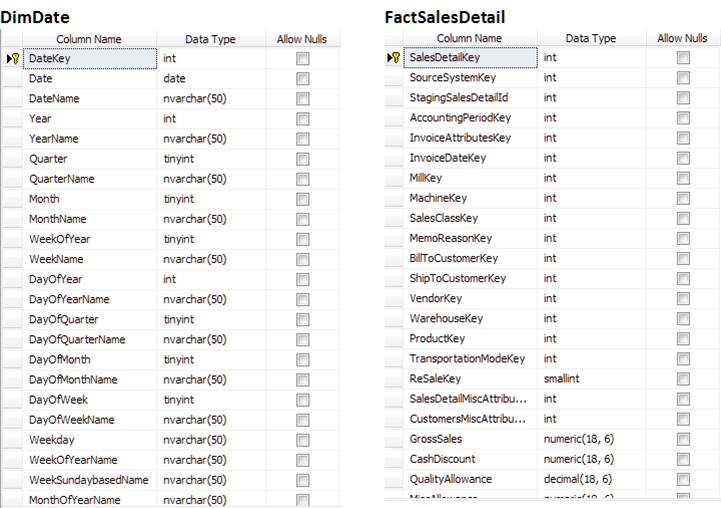
Figure 2-40 is an example of a partial logical model for the larger denormalized model.



**Figure 2-40**: Partial logical data model

### **Physical Model**

The physical model adds final detail to the modeling effort in respect to column data types, nullability, primary keys, indexes, statistics, and other relevant table properties. The diagram in Figure 2-41 expands on the logical model from above, introducing information related to the physical modeling level of detail.



**Figure 2-41**: Partial physical data model

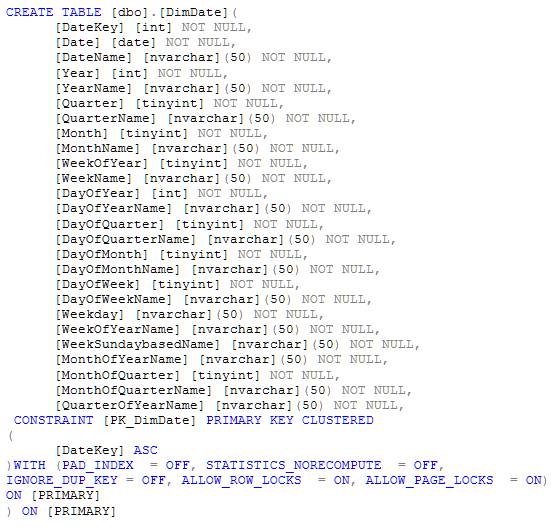
In physical modeling, it is important to properly manage data definition language (DDL) statements for all tables, functions, stored procedures, and other relevant database objects.

When changes to database objects are properly managed, switching between versions of changes to metadata becomes more feasible. This is sometimes required in the normal course of development and introduction of new features.

Having DDL properly managed also provides for more encompassing disaster recovery procedures and enables recreating data warehouse structures in additional development, testing, or user acceptance testing (UAT) environments.

An effective method for managing DDL is to have scripts versioned and controlled via version control software such as Microsoft Team Foundation Server.

Figure 2-42 shows the DDL for the DimDate table used in the previous diagram.



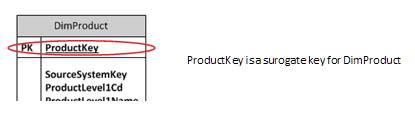
**Figure 2-42**: DimDate DDL

### **Data Modeling – Column Types**

So far, our focus has been at the table level. However, this section covers different column types defined and used during data modeling.

**Surrogate Keys**

A surrogate key, shown in Figure 2-43, is a unique identifier for records within a table. Surrogate keys are either generated by the database when the record is inserted into a table or by a data integration application prior to the record insertion. SQL Server automatically generates the surrogate key when the column is created with the IDENTITY attribute. Note that PDW does not support the IDENTITY attribute in its initial release.

  
**Figure 2-43**: Surrogate key - AdventureWorksDW2008 DimProduct table

Surrogate keys are at the core of every data warehouse model and are typically integer columns that provide a pointer to a unique instance of a dimension member defined by its natural key. Surrogate keys are used to keep fact tables as narrow as possible, to create effective indexes on dimension tables, and to support Type 2 dimensions. In addition, surrogate keys replace the need for including a source system identifier (along with the natural key) for each table that merges data from multiple source systems.

Surrogate keys are typically designed as sequentially incrementing integers and have no logical correlation to a natural key (defined below). In special cases, such as in the Date dimension, the surrogate key is an integer representation of the canonical date value (e.g., May 1, 2009, has a surrogate key of 20090501). This type of surrogate key is also referred to as an intelligent key. Although intelligent keys are not generally recommended, they are acceptable in this case because the Julian calendar is a stable entity not subject to change.

The SQL Server data type for a surrogate key is typically an int because this data type’s maximum value (2,147,483,647) is larger than most tables’ projected cardinality. However, use of tinyint and smallint for small dimensions, as well as use of bigint for very large dimensions, is relevant depending on the projected table cardinality over the life of the data warehouse. The SQL Server data type section below contains more information about integer data types, including their minimum and maximum values.

Data modeling also includes creating a record containing a default surrogate key used to represent null or not available records. Data integration processes will use the default surrogate key (e.g., 0) instead of a NULL value when a lookup doesn’t produce a match.

**Natural Keys**

The natural key, also known as a business ID, is one or more attributes (columns) within a source system used to uniquely identify an entity. Figure 2-44 shows the natural key (ProductLevel1Cd) for an AdventureWorks product.



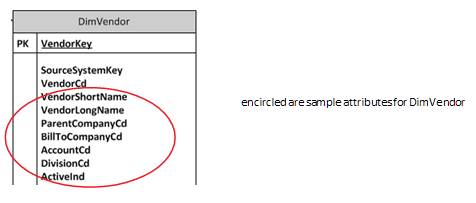
**Figure 2-44**: Natural key

Note that a natural key is used to uniquely identify one product, while the surrogate key is used to uniquely identify one instance of that product over time. Also note in the above example that the SourceSystemKey may also be part of the natural key when there’s the risk of duplicated product codes from different source systems.

The data warehouse data model needs to be flexible to account for changes in uniqueness of dimension records as business rules change over time. However, the natural key is one value that must never change over time, and when it does, it’s considered a new item (e.g., a new product SKU).

**Attributes**

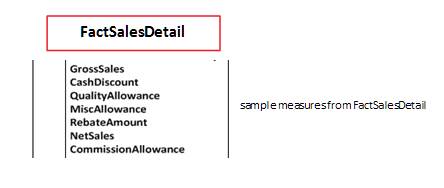
An attribute (or column) adds descriptive characteristics to a table. Attributes exist for all types of tables within a data model. Figure 2-45 shows some attributes for the AdventureWorks DimVendor table.



**Figure 2-45**: Attributes for DimVendor table

**Measures or Metrics**

A measure, or metric, is a numeric column typically used to store transaction amounts, counts, quantities, and ratios. Figure 2-46 shows some measures from the AdventureWorksDW2008 FactSalesDetail table.



**Figure 2-46**: Measures from the FactSalesDetail table

Measures are classified as either base or calculated. One or more base measures are used when resolving the value of a calculated measure. Decisions about which layer of a data warehouse architecture hosts measure calculations are defined during data modeling sessions. These decisions are determined based on whether calculations are relevant to all subject areas in a data warehouse or not.

Measures can also be categorized by how they are calculated:

* **Additive** – These measures aggregate across all dimensions with no special provisions. Examples of additive measures are the OrderQuantity and SalesAmount measures found in the AdventureWorksDW2008 fact tables.
* **Semi-additive** – These measures can be aggregated only over specific dimensions. Examples of semi-additive measures are account balances and inventory levels. These measures can be aggregated by some of the dimensions. But if account balances, for example, are summed up over 1 year for one customer, the resulting sum of account balance snapshots would be inaccurate.
* **Non-aggregated measures** – Ratios and percentages are examples of non-aggregated measures. These measures can’t be aggregated over any dimension.

**Dates and Time**

Dates play a central role within a data warehouse. Time does as well, but usually to a lesser degree. Date and time are special entities in a data warehouse and, as such, require particular attention in data modeling efforts. Dates and time are typically stored in the SQL Server datetime or smalldatetime data types and can represent:

* Time of day (e.g., May 10, 2010, 10:30am EST)
* Date (e.g., May 10, 2010)
* Time within a day (e.g., 10:30am EST)

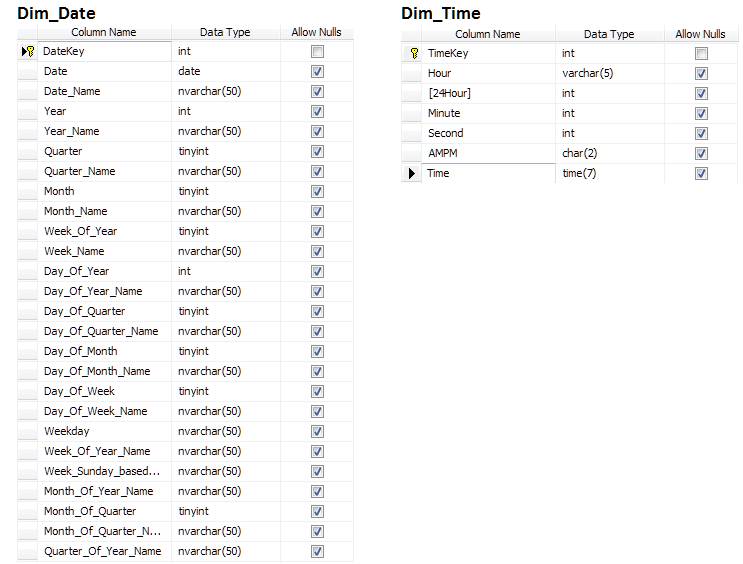
Besides including intelligent surrogate keys for these dimensions, as described in the section on surrogate keys above, organizations often use date and time for “to date” calculations.

Whether to keep date and time in a single dimension or separate them is one of the questions data modeling touches on. Separating these two dimensions provides for greater manageability and usability in a data warehouse. In this approach, two surrogate keys are derived from one source column. For example, a source column for a transaction date of ‘2010-01-01 14:01:00’ would yield a surrogate key for the Date dimension based on the ‘2010-01-01’ segment, while a surrogate key for the Time dimension would be derived from the ’14:01:00’ part of this source column.

Date and time are often used as role-playing dimensions, so one physical date dimension will logically be modeled to reference invoice date and transaction date, as well as create date and other dates in a data warehouse.

Also note that date dimensions differ across vertical industries—for example, a financial date dimension will differ from a retail date dimension.

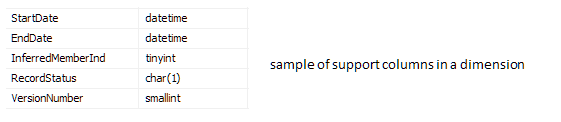
Figure 2-47 shows an example of both a Date and a Time dimension.



**Figure 2-47**: Date and Time dimensions

**Support Columns**

Support columns are included in data models in support of tracking values over time. These columns allow for more efficient management of state changes for dimensions and for more effective auditing of data loads. Figure 2-48 shows some instrumentation support columns.



**Figure 2-48**: Support columns

The following columns are used to track different historical versions:

* **StartDate** – The date when this record version become active.
* **EndDate** – The date when this record became inactive. The current record will either have a NULL value or a value representing the maximum date (e.g., 12/31/9999).
* **RecordStatus** – The status of this record. Typical values are Active, InActive and Pending.
* **VersionId** – This value represents the version number of the record and is incremented every time a version of the record is created.
* **InferredMemberInd** – This value indicates if a dimension member was loaded during fact load as an inferred member. Inferred members are dimension members that don’t have attributes other than business ID available during the time facts are being loaded (a lookup for surrogate key during fact load yields no results).

The other support column types are in support of data integration instrumentation. LineageId is an example of this kind of column and contains a value that lets data stewards track the process that loaded the record. Another example is InferredMemberInd, which as we just saw is a flag indicating whether a record is an inferred member or not.

All of these columns are covered in more detail in Chapter 3 – Data integration.

**Other Columns**

Data modelers also address other types of columns as they are relevant to business requirements. These columns include spatial data, XML, large text, and images.

### **Keys**

Data models include definitions on keys as well as changes to keys as data changes custody in data warehouse layers. Typically, data models account for introducing new keys for each change in data custody. These keys are implemented by using either the natural key from the source system or the surrogate key created during the data integration process.

The blanket recommendation is for data warehouse data models to use surrogate keys for all primary/foreign key activity. The reasons for doing so include:

* **More efficient representation** – Natural keys in source systems are often defined using character data types, which are less efficient than integer data types in SQL Server. Additionally natural keys may be represented as different values in different source systems, making it challenging to consolidate. Instead, use a surrogate key.
* **Tracking changes across history** – The need to report across history requires the data model to store the instance of that record at a particular point in time.

Natural keys are often used in data warehouses to aggregate multiple versions of one entity across all instances of this entity (i.e., one natural key for more than one surrogate key).

Additional benefits of proper key management in data warehouses include:

* **Enabling the conceptual and enterprise model** – Failing to properly identify natural keys or to efficiently manage lookups of natural keys for surrogates negates the basic principles of data modeling. Key management is one of the essential factors for enabling accurate and effective conceptual and logical models.
* **Preparing for the future** – Managing natural and surrogate keys prepares the data warehouse for downstream data consumption. OLAP structures and BI semantic layer components perform more efficiently and provide for greater data analysis when data models include proper handling of keys.

Once a decision has been made to use surrogate keys, the next question becomes:

*How do we generate surrogate keys?*

The short answer is that the SQL Server data modeler and developer can use globally unique identifiers (GUIDs), IDENTITY columns, or a key generation utility. GUIDs usually are not used due to their size (16 bytes) and their randomly generated nature. For more information about surrogate key generation, see the Surrogate Key section in Chapter 3 – Data Integration.

### **Dimensions**

*Dimension* is a term most commonly associated with Ralph Kimball. In his white paper [Facts and Fables about Dimensional Modeling](http://www.kimballgroup.com/html/PDFs/Fables_Facts.pdf), Kimball attributes the first reference of facts and dimensions to a joint research project conducted by General Mills and Dartmouth University in the 1960s.

Wikipedia states that:

*Dimensional modeling always uses the concepts of facts (measures), and dimensions (context). Facts are typically (but not always) numeric values that can be aggregated, and dimensions are groups of hierarchies and descriptors that define the facts.*

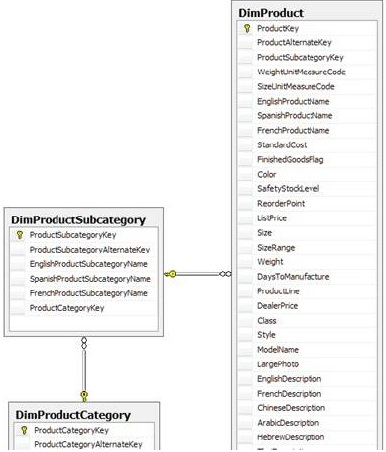
Dimensions are often modeled based on multiple entities from one or more source systems. Depending on reporting needs, consumers of a data warehouse system will be accessing information in entities for operational reporting, while analytical reporting will rely on dimensions.

Traditionally, flattened, denormalized structures are the most efficient data model technique because they require the least amount of joins to produce the requested result. However, in a few cases, a dimension can become too wide, and data modelers need to opt for a more normalized structure.

For more information about dimensions, a good starting point is Kimball’s articles at [his Web site](http://www.ralphkimball.com/html/articles.html).

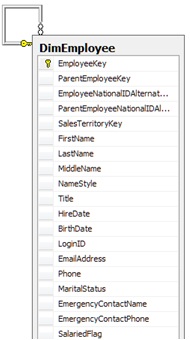
Let’s spend the rest of this section getting an overview of the most common dimension types and concepts:

* **Star dimension** – A star dimension is a fully denormalized version of a dimension, where all hierarchies and reference values are merged into the dimension. Star dimension tables are directly linked to the fact table.
* **Snowflake dimension** – Snowflake dimensions contain more than one dimensional table to include references to all relevant attributes. The AdventureWorks Product Category hierarchy, shown in Figure 2-49, is an example of a snowflake dimension. In a snowflake dimension, some of the tables may be indirectly linked to the fact table.



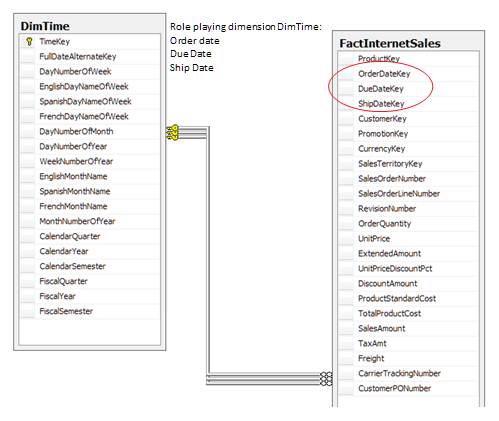
**Figure 2-49**: Snowflake schema example

* **Parent-child dimension** – This dimension type is used to model hierarchies such as an organizational chart or a chart of accounts. These hierarchies are unbalanced and ragged, which makes them difficult to model using a snowflake technique. The AdventureWorks2008DW DimEmployee table, shown in Figure 2-50, is an example of a parent-child hierarchy.

****

**Figure 2-50**: Parent-child dimension example

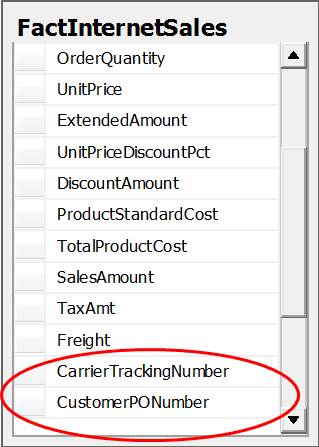
* **Role-playing dimension** – A role-playing dimension occurs when one dimension is linked to multiple times within one table. The most common example of this is kind of dimension is Date, where one dimension is used to provide information on order date, invoice date, create date, and other dates, as shown in Figure 2-51.



**Figure 2-51**: Role-playing dimensions

* **Junk dimension** – Junk dimensions represent a collection of low-cardinality, non-related attributes contained within one dimension, where each possible combination of attributes is represented with a single surrogate key. This design decision is purely for optimization—one 2-byte junk dimension key is more efficient than five 1-byte keys, which results in significant savings in storage space for fact tables with billions of rows or more.
* **Degenerate dimension**—A degenerate dimension does not have its own table; it is represented by its value within the fact table. This typically occurs in transaction tables, and examples are order number and invoice number. Degenerate dimensions are useful to capture the transaction number or natural primary key of the fact table. Because it does not make sense to create a dimension with no attributes, the attributes instead may be directly stored in the fact table.

Figure 2-52 shows two examples of degenerate dimensions within the AdventureWorksDW2008 FactInternetSales table.



**Figure 2-52**: Degenerate dimensions

**Tracking History**

Accurately reporting historical results often requires that the state of a dimension at a particular point in time be recorded and saved, as opposed to being overwritten. There are basically three methods for tracking history:

* **Update one record** – There’s one version of a record, and all changes are applied to this one record. This approach is often referred to as a Type I Slowly Changing Dimension (or SCD I).
* **Track record changes** – Every change in a record will result in a new version of that record with a unique surrogate key. This is often referred to as a Type II SCD (or SCD II).
* **Add new columns** – Every change in a key value results in a new column added to the table. This is often referred to as a SCD III.

It’s often not practical to add columns, so the options come down to two data integration patterns: versioned inserts and versioned updates. Each of these is covered in more detail in Chapter 3 – Data Integration.

Now that dimensions have been covered, the next topic discusses fact tables, which typically model a business transaction or business process.

### **Fact Tables**

The largest tables within source systems are the transaction tables. These tables are often orders of magnitude larger than dimensions. These tables are modeled as fact tables within a dimensional data model. Another common source for fact tables is business processes.

Fact tables are optimized structures that are typically comprised of numbers, dates, and very small character columns. The numbers are divided into numeric and monetary values and foreign key references to dimensions. Classification of facts can be done by fact table types and categories.

Fact table categories were introduced in the Column Types section above and include additive, semi-additive, and non-additive measures. Another category is a custom rollup fact table, where the aggregation rules are specific to a dimensional value or values. The most common example of this is a financial chart of accounts.

Fact table types include:

* **Transactional** – This is the most common type of fact table. The classic example is entering one product sale at a store. Transaction facts are usually additive—that is, SQL aggregate functions such as SUM, MIN, MAX, and COUNT can be applied to the measures.
* **Snapshot** – The facts within a snapshot fact table are not additive across one or more dimensions, typically the Date dimension. The classic example is inventory, where the measures represent values at a point in time. Inventory is not additive across time, but is additive across other dimensions referenced by the snapshot fact table. Other examples include event booking levels and chart of account balance levels.
* **Fact less** – This table has no measured facts—row counts are the only measure. This is typically used to describe events, such as something that has or has not happened, or many-to-many relationships like coverage models or to measure that something has or has not happened. It contains only dimensions or one fact with a value of 0 or 1. Common examples include class attendance, event tracking, coverage tables, promotions, or campaign facts. Ralph Kimball used Voyages as the example in his seminal book, *The Data Warehousing Toolkit*.
* **Aggregate** – Aggregate fact tables include information rolled up at a certain hierarchy level. These tables are typically created as a performance optimization in support of a business reporting requirement. Deriving an aggregation table once in a data integration process is much more efficient than aggregating every time the information is requested. Aggregations need to account for the additive nature of the measures, created on the fly or by pre-aggregation.

### **Reference Data**

Reference data was described earlier in this chapter as a simple classification of a noun or transaction. These classifications play a key role in data warehouse reporting and analytics by providing the ability to filter and/or aggregate results. Reference data is considered a category of master data.

The DimScenario reference table, shown in Table 2-1, is one example of reference data within the AdventureWorks samples.



**Table 2-1**: AdventureWorksDW2008 DimScenario reference table

This table is a common reference table found in budgeting and forecasting software and allows the filtering of fact tables. Note that the above example isn’t truly representative because many reference data tables consist of a character code and description. For example, a State reference table would have state code values and description columns, as shown in Table 2-2.



**Table 2-2**: Example State reference table

In the above example, the data modeler decides whether to add a surrogate key, as shown in Table 2-3, or to use the natural key.



**Table 2-3**: Example State reference table with surrogate key

The general data modeling rule is that adding a surrogate key rarely has a negative impact. The more relevant question is whether to use the surrogate key (StateKey) or the natural key (StateCode) as the foreign key. The general rule of thumb is to always use the surrogate key, but the downside is that:

* There will always be a need to join to the reference table.
* Reference tables with versioned inserts reference an instance of a reference table entry.

Data modelers often include the natural key and description in a denormalized table when modeling hierarchies, as shown in Table 2-4.



**Table 2-4**: Denormalized City table

**Merging Reference Data**

The data warehouse team will often need to merge multiple reference data tables into one master reference table. This is required when there are multiple source systems, each with their own set of reference tables and codes. In addition, organization-specific reference data frequently needs to be merged with industry-standard data because the standard either does not fully meet the organization’s needs or the reference data existed prior to the standard. This is a common occurrence in the health care industry, where codes used in subject areas such as clinical analysis are organization-specific.

Creating a single version of the truth for reference tables and hierarchies is a significant portion of MDM efforts. Merging different clinical codes may require the creation of a new natural key, which is represented by two or more different natural keys in their respective source systems.

Note that using a surrogate key as the natural key could result in an issue if the reference table supports versioned inserts. What seems like a very simple issue can turn into something much more complex.

Although adding a surrogate key doesn’t hurt, determining whether to use the surrogate key or the natural key for foreign key references is a function of:

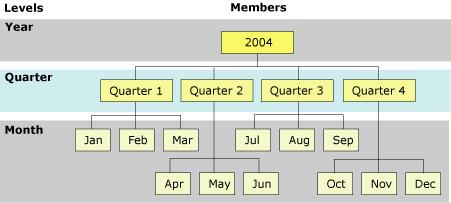
* Whether the natural key will ever change
* Whether the objective is to reference a versioned instance of the record or the record regardless of version

This decision becomes more important when modeling hierarchies, which are often comprised of multiple levels, each with their own reference table.

### **Hierarchies**

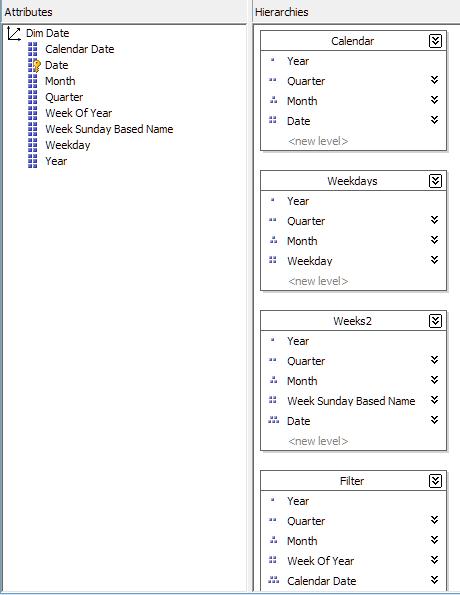
A hierarchy is a multi-level structure in which each member is at one level within the hierarchy. It is a logical structure of dimension attributes that uses order levels to define data aggregations and end user navigational drill paths.

Each hierarchy member has zero or one parent and zero or more children. Figure 2-53 shows an example of a date hierarchy.

  
**Figure 2-53**: Date hierarchy

The leaf level is the lowest level in the hierarchy, and each member within this leaf is connected to one parent. This structure is repeated for each level above the leaf level, until the top level is reached.

Multiple hierarchies can be created from one source. Figure 2-54 shows an example of this: The AdventureWorks2008DW DimDate table is displayed on the left, and the multiple hierarchies that can be created from this one table are displayed on the right. (Note that this SSAS example was used to visually show multiple hierarchies; SSAS is not required when creating multiple hierarchies within a data warehouse.)



**Figure 2-54**: DimDate table hierarchies

Hierarchies are a large topic, and this chapter covers balanced, ragged, unbalanced, and network hierarchies.

**Balanced Hierarchies**

Figure 2-54 above is an example of a balanced hierarchy. All branches of a balanced hierarchy descend to the same level, with each member’s parent being at the level immediately above the member. Balanced hierarchies can be collapsed in one table, as in the above example, or exist in multiple tables, as with the Product Category hierarchy shown in Figure 2-49. In a standard balanced hierarchy, each level of the hierarchy is stored in one and only one column of the dimension table.

**Ragged Hierarchies**

In a ragged hierarchy, the parent of a member can come from any level above the level of the member, not just from the level immediately above. This type of hierarchy can also be referred to as ragged-balanced because levels still exist. A Geography dimension is a typical example of such a hierarchy. In some countries, the province/state level may not exist, as with the Republic of Singapore or Vatican City State, for example.

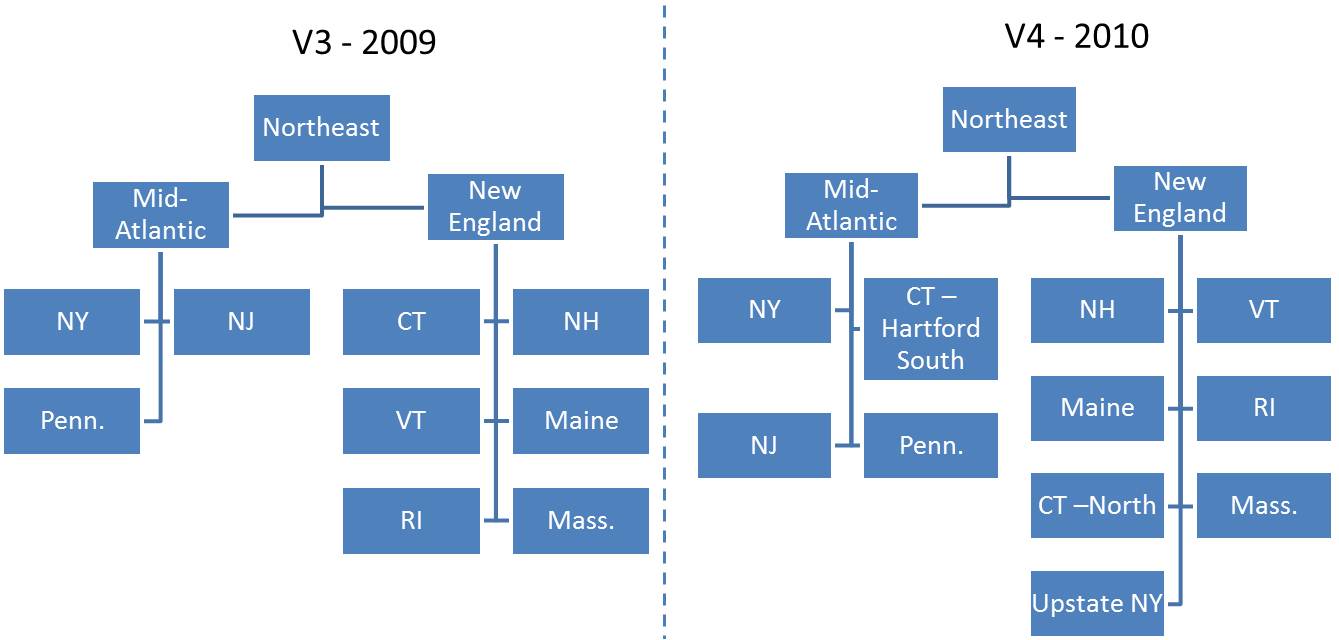
**Unbalanced Hierarchies**

Unbalanced hierarchies include levels that have a consistent parent-child relationship but have logically inconsistent levels. The hierarchy branches can also have inconsistent depths. An example of an unbalanced hierarchy is an organization chart, which shows reporting relationships among employees in an organization. The levels within the organizational structure are unbalanced, with some branches in the hierarchy having more levels than others. The AdventureWorksDW2008 DimEmployee dimension table is an example of an unbalanced hierarchy.

**Hierarchies and History**

Hierarchies change over time, which requires the data warehouse to display the correct hierarchy based on the particular date range or period in time. One example is reporting on sales based on the Sales Territory hierarchy. Figure 2-55 shows the Northeast sales territory in 2009 and 2010. This hierarchy has three levels:

* Division (Northeast)
* Region( Mid-Atlantic, New England)
* District (State)



**Figure 2-55**: Northeast sales territory

As you can see, sales management changed the sales territories in 2010 and subdivided New York and Connecticut as follows:

* Created a new district, Upstate NY, and assigned it to the New England region. The remainder of New York stays within the New York district.
* Divided Connecticut into two districts:
  + North of Hartford is now part of the New England region.
  + Hartford and South is part of the Mid-Atlantic region.

However, the business consumer may still want to see 2009 Northeast sales totals based on:

* How the sales territory was organized in 2009
* The current sales territory structure (i.e., the 2010 sales territory)

**Modeling a Versioned Balanced Hierarchy**

A hierarchy can be modeled in several ways, depending on whether the hierarchy is balanced or not. When the hierarchy is balanced, it can be represented by multiple reference tables, where each reference table is used to populate one level in the hierarchy.

This can be modeled either by each level containing a foreign key reference to its parent or in a flattened structure, as shown in Table 2-5. This table shows how the changed sales territory information presented above is stored within a denormalized Sales Territory table.

Note that this table could also be modeled in a normalized structure, using multiple reference tables with foreign key references to their parent. However, each of the reference tables would need to be versioned to account for the sales territory versioned changes over time.

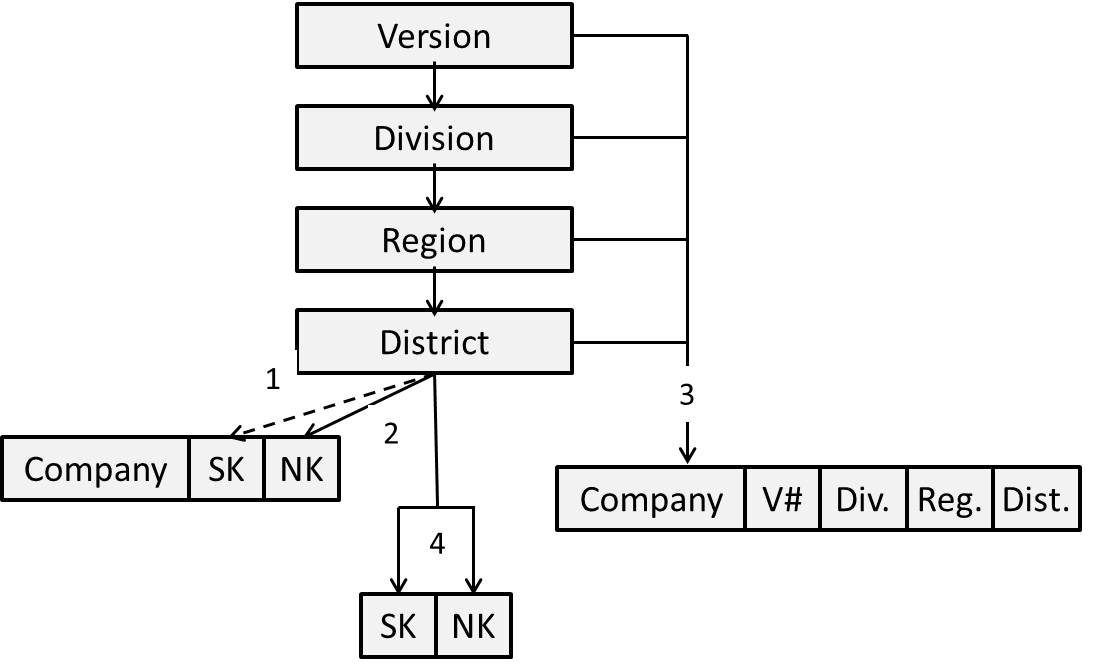


**Table 2-5:** Sales territory structure

Note the following about the Sales Territory table:

* Sales Division, Region, and District all are separate reference tables. Their natural key is used to model the sales territory.
* Using the natural key, as opposed to the surrogate key, avoids an issue of returning different results from reference tables that support versioned inserts.
* The version of the hierarchy is a key attribute in this example. Many hierarchical structures such as a financial chart of accounts or a sales territory are created as one instance or version, as opposed to treating each separate change as its own version.

The next consideration is how dimension members that are children of the Sales Territory table (such as Sales Rep and Company of both Suppliers and Customers) should be modeled. Figure 2-56 shows the options available to data modelers.



**Figure 2-56:** Hierarchy modeling options

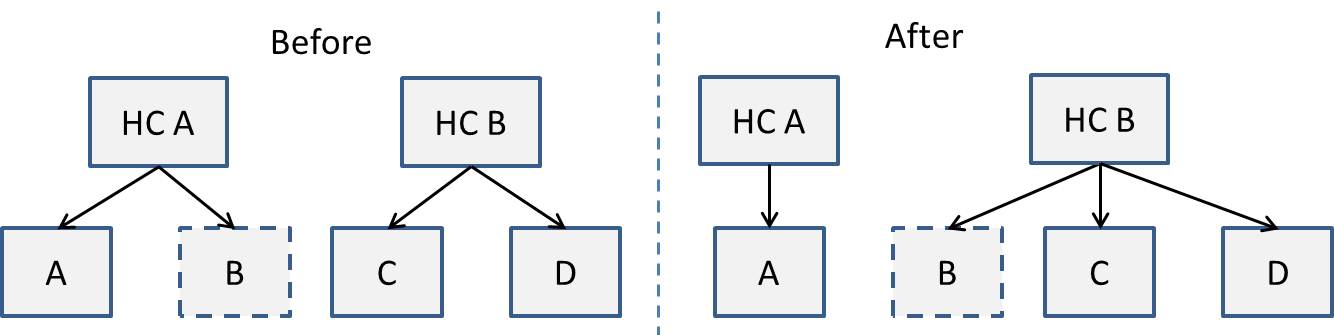
Data modeling options include:

1. Store the *surrogate key* of the Sales Territory instance.
2. Store the *natural key* of the sales territory.
3. *Denormalize* the Sales Territory into the Company table.
4. Add to the hierarchy a *Company level* that contains the Company surrogate and natural keys. Every version of the hierarchy will include all companies.

Options #1 and #3 would both require a different version for every Sales Rep record each time the sales territory changed. Option #2 would not require an extra version of the Sales Rep record, but would require the query to have a version filter to return the desired version of the sales territory. Option #4 would require the creation of one Sales Rep record for every version of the sales territory.

**Modeling a Time-Dependent Balanced Hierarchy**

Other hierarchies change based on events as opposed to a versioned release. One example of this scenario is a company linking structure that links subsidiaries and legal entities to their parent company. Figure 2-57 shows a simple example with two parent companies each consisting of two companies. The time-dependent event would be Holding Company B’s acquisition of Company A.



**Figure 2-57:** Company linking example

Table 2-6 shows how this two-level hierarchy could be modeled.



**Table 2-6:** Modeling a company linking hierarchy

Notice that in this example, the changes are in response to an independent event instead of a scheduled version release. Any query that aggregates or filters on the parent company could return different results when you apply an “as of” date to the Start and End date ranges.

**Ragged and Unbalanced Hierarchies**

As discussed earlier, this class of hierarchy will have different levels and will be unbalanced. Organizational charts and a financial chart of accounts are typical examples of ragged, unbalanced hierarchies. These hierarchies are often modeled as parent-child structures. The DimEmployee dimension illustrated in Figure 2-43 above is a good example of this.

Although the parent-child structure makes it easy to store these values, reporting becomes more difficult because the table is not level-centric—meaning there isn’t one table for each level in the hierarchy. When reporting, the hierarchy levels become more important because aggregation is frequently based on a level number.

The options in this case are:

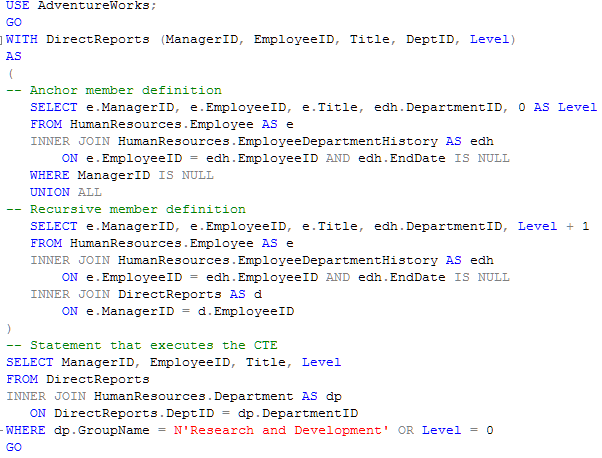
* Create a balanced hierarchy
* Transform to a balanced hierarchy within SQL Server

Creating a balanced hierarchy from a ragged, unbalanced hierarchy is typically done by following these steps:

1. Create a balanced hierarchy table, with the code and description columns for the maximum level within the ragged hierarchy.
2. Populate this balanced hierarchy table starting with the parent and descending down through the hierarchy.
3. Repeat parent values in the child level if the number of levels within a branch is less than the maximum number of levels.

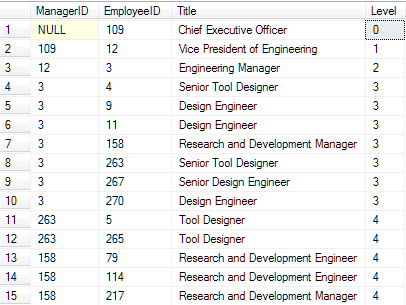
**Using Common Table Expressions**

In SQL Server, Common Table Expressions (CTEs) are a powerful tool for querying hierarchies. For example, Figure 2-58 shows a CTE for querying the HumanResources.Employee hierarchy in the AdventureWorks sample database.



**Figure 2-58:** Common Table Expression example

Results of this CTE include manager and employee IDs, relevant titles, and levels within the organization, as Figure 2-59 illustrates.

  
**Figure 2-59:** Common Table Expression results

For more information about SQL Server CTEs, see [WITH common\_table\_expression (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms175972.aspx). Note that CTEs are not supported in the initial release of PDW.

Also note that in a network hierarchy, nodes can contain more than one parent. A common example of a network hierarchy is a family tree.

**Which Hierarchy-Handling Option Is Most Effective?**

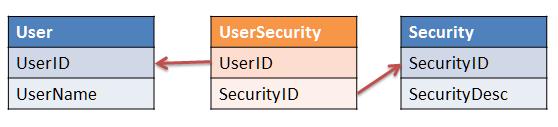
In data warehouse architecture, the most effective design option to handle hierarchies has proven to be flattening hierarchies into a single table. Also, if there is more than one hierarchy defined for a dimension, all hierarchies should be included in the one table. This approach eliminates joins between the main dimension table and lookup tables, improving data retrieval, which is what data warehouse systems are built for.

Finally, in parent-child hierarchies, for smaller dimensions, you can use a typical ID-ParentID recursive approach. But for larger dimensions, this technique can have significant performance issues. A recommended strategy in these cases is to introduce many-to-many fact(less) tables. This approach works well with extending data models to OLAP, especially with SSAS.

### **Bridge Tables**

Bridge tables hold values for multiple instances of relationships between entities. These containers for storing many-to-many relationships are also referred to as junction or cross-reference tables.

One of the examples for using bridge tables is in handling security privileges for users in a data warehouse. Figure 2-60 depicts this scenario, with the UserSecurity table serving as a bridge table.



**Figure 2-60:** Bridge table example

### **Nulls and Missing Values**

This section outlines guidelines for handling nulls and missing values in respect to architecting effective data warehouse systems.

**Null Values**

Nulls are not recommended for attribute values because they provide no meaning for analysis. The existence of nulls also translates into more complex queries (i.e., the existence of nulls must be checked in addition to comparing values). In addition, in respect to business ID, nulls need to be handled before they reach data warehouse tables.

Business rules implemented in data integration processes need to include handlers for nulls and process them based on what type of attributes they are assigned to. Typical rules for handling null values for dimension attributes include replacing code values with ‘N/A’ and replacing descriptive values with ‘Unknown’. Of course, data integration processing of null values must be consistent across the data warehouse; otherwise, data will be subjected to various interpretations.

Finally, if business ID columns include nulls, depending on relevant business rules, corresponding records can be prevented from loading to the data warehouse by loading them into exception tables. Alternatively, they can be loaded under an Unknown dimension member.

For measures, the decision to persist null values is sometimes a function of the downstream consumers. For example, OLAP engines such as SSAS that support sparse cubes typically perform much better when null values are used for fact table measures used to populate the OLAP database.

**Missing Values (Empty Strings)**

Empty strings for attributes usually don’t translate into an asset for a data warehouse. Preserving empty strings can lead to inconsistent reporting and confusion for information workers if an empty string in a source system is equivalent to a null value. Business consumers should have input into whether null values and empty strings can both be replaced with the same default. If so, data integration business rules can replace missing values and null values with the same default value.

### **Referential Integrity**

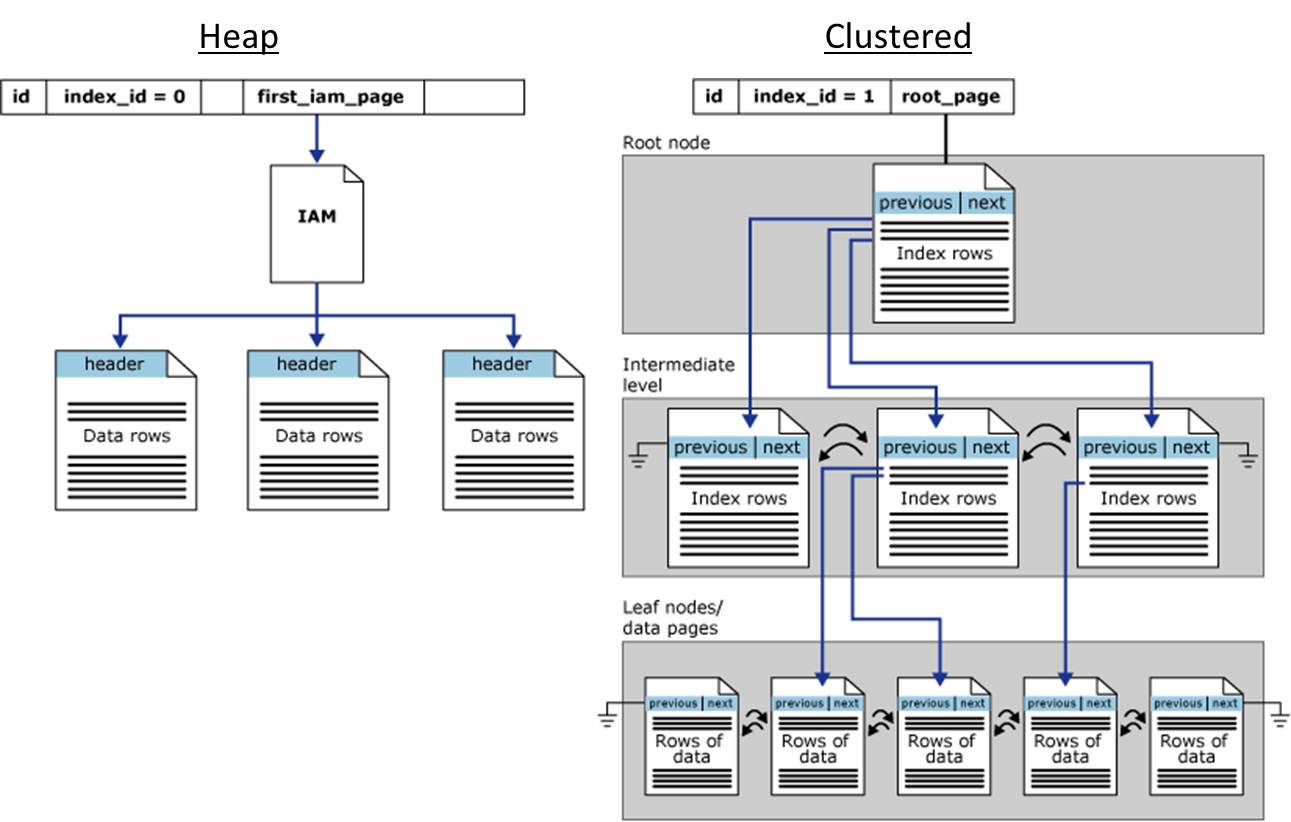
Referential integrity defines relationships between entities as stipulated by business rules in the logical data warehouse model. Enforcing referential integrity is a core element of data quality. Traditionally, the best practice in data modeling for preserving referential integrity was to create a foreign key constraint.

However, this best practice can cause performance issues when applied to large tables with a significant number of foreign keys. The reason is that SQL Server enforces referential integrity by first verifying that a record exists in the referenced table prior to the insert. This is a consideration for most SQL Server data warehouses because many fact tables have a lot of foreign keys and very large table sizes.

The alternative is to enforce referential integrity within the data integration processes rather than creating foreign key constraints. This is a best practice in large SQL Server data warehouse implementations and applies to both the staging and data warehouse data stores.

### **Clustered vs. Heap**

One of the important physical design decisions for SQL Server tables is whether a table should be modeled with a clustered index or as a heap. SQL Server clustered index tables have one index that stores and sorts the data rows based on the index key values. Tables without this feature are referred to as heap tables. Figure 2-61, from SQL Server 2008 R2 Books Online, illustrates the difference between heaps and clustered tables at a physical level.

****

**Figure 2-61:** Heap vs. clustered table on disk structure

The leaf level of a clustered table index is the data page, which is organized by the clustered index key. Heap tables are organized by their insert order—each insert appends the record to the last data page.

Table 2-7 compares clustered and heap tables with no indexes across storage and access options.

|  |  |  |
| --- | --- | --- |
| **Category** | **Heap** | **Clustered** |
| Storage | Data is stored in order insert | Data is stored by the clustered key |
| Reads | Table scan | Index if request is by clustered key |
| Inserts | Inserts occur at the end of table | Inserts occur by the clustered key and can result in fragmentation |
| Updates | Updates can result in page splits | Updates can result in page splits |
| Overhead | No overhead | Additional overhead for both disk space and time to manage the index |
| Maintenance | Less need for defragmentation | Defragmentation required due to clustered key inserts |

**Table 2-7:** Heap vs. clustered table comparison

The general rule for SQL Server data modelers is that clustered tables provide better performance when the clustered key(s) is commonly used for query operations. This is typically the case for dimensional data models.

The MSDN SQL Server best practices article [Clustered Indexes and Heaps](http://msdn.microsoft.com/en-us/library/cc917672.aspx) recommends that a clustered index always be created for a table. However, data modelers should recognize that creating a clustered index on every table will result in additional load times for very large fact tables. That additional load time may be unacceptable if it results in the data integration processes’ execution times overlapping with the time the data warehouse is available to business consumers.

Heap tables with no indexes are necessary in VLDW scenarios. In this scenario, data is optimized for loading data. Data is inserted into heap tables, and updates are never applied due to performance reasons. Updates are expensive for very large tables.

Heap tables are a consideration when large tables are not directly accessed by business consumers, such as with the data warehouse database within a hub-and-spoke architecture. In this scenario, the fact tables within the data mart spokes should be modeled as clustered unless VLDW performance considerations prohibit this.

Here are some implementation notes to keep in mind about clustered indexes:

* For clustered tables, the data integration process should sort the source data by the clustered index key, which in most cases will translate into records being inserted at the end of the table to reduce fragmentation.
* These indexes may need to be maintained on a regular basis because clustered indexes can suffer from fragmentation depending on the extent of insert and update activity. Fragmentation in clustered tables can easily be resolved by rebuilding or reorganizing the clustered index. However, this can be a very lengthy operation, especially for fact tables with cardinality in the billions of rows.
* It’s recommended that the clustered index column be an integer value. For dimensional data models, it’s common for the Date dimension’s primary key to be a 4-byte integer (YYYYMMDD). For example, the date Aug 4, 2010, is converted to 20100804. The clustered index key for the fact table is also a date represented in this format.

Chapter 5 includes details about performance and query plans when accessing clustered vs. heap tables.

### **SQL Server Data Type Considerations**

As a general rule, data modelers should choose the smallest data types when possible to reduce the total size of a record. This in turn reduces the amount of storage, disk I/O, and network bandwidth required to load and retrieve the data. This section reviews the data type options available to data modelers.

**Character Data Types**

The general rules of thumb for choosing character data types are:

* **Only use nchar and nvarchar when the universe of values spans or will span multiple languages.** This is because the nchar and nvarchar data types are twice as large (2 bytes vs. 1 byte) as their char and varchar equivalents.
* **Use varchar or nvarchar for columns with descriptive values, names, or addresses.** These varying data types are represented by a length and value, which make them slower to locate within a data page but smaller to store when there’s any variability in the length of the values.
  + Source system char data types often are space-padded, so a name column defined as a char(8) would represent the name “Smith” as “Smith “.
* **Use char for columns containing coded values.** SQL Server char columns are more efficient because they are stored on disk in fixed locations within a record. Columns containing codes and abbreviations, such as StateCode, should be a char(2) not varchar(2).

For more information about SQL Server character data types, see the following links:

* [char and varchar](http://msdn.microsoft.com/en-us/library/aa258242(SQL.80).aspx)
* [nchar and nvarchar (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms186939.aspx)

**Integers**

Exact integers are the most common data type found in data warehouses and are used for measures, counts, and surrogate keys. The SQL Server data modeler can choose to model exact integers as:

* 1 byte (tinyint), 0-255
* 2 bytes (smallint) -32,768 to 32,767
* 4 bytes (int) -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
* 8 bytes (bigint) -2^63 to -2^63

Keep these notes in mind about SQL Server integer data types:

* There’s no unsigned small integer, integer, or big integer data types.
* Date dimensions are an example where smallints are used, provided that the date range is less than 89 years (i.e., 32767/Days in a year).
* Int is the most common data type used in SQL Server data warehouse models.
* Bigint should only be used when the maximum value has a chance of exceeding 2.14 billion, the maximum value for an int.
* GUIDs are 16-byte integers and are a convenient way to generate unique surrogate key values. However, they are not recommended if other options such as IDENTITY columns or a key generator are available. This recommendation is discussed in more detail in Chapter 3 – Data Integration.

For more about SQL Server integer data types, see [int, bigint, smallint, and tinyint (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187745.aspx).

**Numeric Data Types**

SQL Server includes decimal, numeric, real, float, money, and smallmoney data types to store numbers. For data warehousing implementations, the use of precise data types—decimal, numeric, money, and smallmoney is recommended. The money data type can be used instead of decimal as long as requirements don’t include definitions for more than four decimal digits.

The use of approximate data types such as real and float are not recommended because they are approximate numbers and, as such, can lose precision when aggregated.

If source data is stored in the real data type, there can be a slight loss of data precision when converting to decimal, but decimal data types are more accurate for querying in WHERE conditions and are typically more compatible with applications consuming data from a data warehouse.

See [Using decimal, float, and real Data](http://msdn.microsoft.com/en-us/library/ms187912.aspx) for more information about SQL Server real data types.

**Date Data Types**

Date and time values are relevant to data warehouse implementations from the perspective of designing calendar dimensions. Depending on the grain of the data in a data warehouse, modelers can decide to have a single calendar dimension or one dimension for date values and another for time values.

The datetime2 data type should be considered for columns that store full date values as long as this date detail includes hours, minutes, seconds, and milliseconds. The date data type should be considered if the data contains only year, month, and day values due to the storage savings.

If a dimension holding values for time is modeled, you should use the time data type.

In case of two calendar-type dimensions, fact tables will have both surrogate keys, providing for more efficient data analysis.

You can find more information about SQL Server date data types at [datetime (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187819.aspx).

**Other Data Types**

SQL Server supports many other data types. For a complete list, see [Data Types (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187752.aspx).

Data warehouses, however, mostly use the data types we covered in this section. This is predominately because data warehouses are architected for most efficient data analysis.

Note that the emergence of XML as a standard for data integration shouldn’t translate into XML being used as a data type for data warehouses. Data modelers should review the XML data and decide whether to shred it into a table structure more conducive to efficient querying.

### **Very Large Data Sets**

When you are architecting data warehouses with very large data sets, you can run into potential data consumption and data management difficulties.

While there are number of considerations to keep in mind when architecting for VLDBs, major points related to the data warehouse architecture revolve around removing contention between read and write operations and maintaining the effectiveness of data retrieval and indexes.

**Read and Write Contention**

Organizing data and log files and tempdb onto separate disks is the focus of the effort to remove conflicts between read and write processes. This topic is covered in the Database Architecture section earlier in this chapter.

**Effective Data Retrieval and Indexes**

The size of a VLDB often introduces difficulties with quick retrieval of data needed for reports and analysis. Data consumption also becomes more complicated because indexes tend to grow to the point that their maintenance becomes impractical. You can address these issues with SQL Server by incorporating a partitioning strategy for both data and indexes.

Table partitioning, which we referenced in the Database Architecture section above, is an important tool for getting the most value out of your data warehouse data. See Chapter 4 for details about table partitioning, and see Chapter 5 for how to set up partitioned tables for efficient query access.

## Conclusion and Resources

A successful data warehouse requires a solid data architecture. Data architecture is a broad topic, and this chapter has focused on the data architect and data developer responsibilities and deliverables, including database architecture, platform architecture, and data models.

First, the database architecture depends on the data warehouse implementation pattern, which is typically a centralized EDW, a federated data mart, or a hub-and-spoke approach. In all cases, the data warehouse team is faced with a series of challenges, including scope (maintaining a single version of the truth throughout the implementation), scale (handling huge data volumes), and quality (delivering results that business users trust).

Providing a single version of the truth presents the data warehouse team with significant technical challenges. However, there are larger business and process issues that have resulted in the emergence of data governance as a key corporate activity.

Master data and MDM are crucial to arriving at a single version of the truth and are old problems in data warehousing that have spawned an emerging class of software products. The decision about whether to purchase and use an MDM software product is not a forgone conclusion and depends on a variety of factors.

Once you’ve selected an appropriate data warehouse implementation pattern, a robust platform architecture is required to support the data warehouse volumes, including loading data into the data warehouse and obtaining results from the data warehouse. The SQL Server platform provides customers with a variety of options, including reference hardware solutions (Fast Track Data Warehouse), data warehouse appliances (SQL Server 2008 R2 PDW), and data virtualization (SQL Server 2008 R2 Enterprise Edition or Data Center).

As stated above, physical best practices and guidance within this chapter are for the symmetric multi-processing (SMP) versions of SQL Server 2008 R2 due to some differences in functionality between the SQL Server 2008 R2 SMP release and the initial release of PDW.

The next key deliverable is developing the correct data models for the different databases within the data warehouse. The level of denormalization your database development team chooses depends on whether the database’s user community is business consumers (denormalized) or data integration developers and data stewards (more normalized).

As we can see in this chapter, the data architecture deliverables provide the foundation for the core data warehouse activities, which we cover in the remaining chapters of this toolkit:

* Loading data into the data warehouse (Chapter 3 – Data Integration)
* Managing the data warehouse (Chapter 4 – Database Administration)
* Retrieving results from the data warehouse (Chapter 5 – Querying, Monitoring, and Performance)

### **Resources**

To learn more about data warehouse data architecture considerations and best practices, see the following links:

* Best Practices for Data Warehousing with SQL Server 2008  
  <http://msdn.microsoft.com/en-us/library/cc719165(v=SQL.100).aspx>
* Clustered Indexes and Heaps  
  <http://msdn.microsoft.com/en-us/library/cc917672.aspx>
* Data Compression: Strategy, Capacity Planning and Best Practices  
  <http://msdn.microsoft.com/en-us/library/dd894051(v=SQL.100).aspx>
* Data Governance Institute’s Web site:  
  <http://www.datagovernance.com/>
* **The Data Governance & Stewardship Community of Practice**<http://www.datastewardship.com/>
* The Data Loading Performance Guide  
  <http://msdn.microsoft.com/en-us/library/dd425070(v=SQL.100).aspx>
* Data Warehousing 2.0 and SQL Server: Architecture and Vision  
  <http://msdn.microsoft.com/en-us/library/ee730351.aspx>
* Data Warehouse Design Considerations  
  <http://msdn.microsoft.com/en-us/library/aa902672(SQL.80).aspx>
* Fast Track Data Warehouse 2.0 Architecture  
  <http://msdn.microsoft.com/en-us/library/dd459178(v=SQL.100).aspx>
* Hub-And-Spoke: Building an EDW with SQL Server and Strategies of Implementation  
  <http://msdn.microsoft.com/en-us/library/dd459147(v=SQL.100).aspx>
* Introduction to New Data Warehouse Scalability Features in SQL Server 2008  
  <http://msdn.microsoft.com/en-us/library/cc278097(v=SQL.100).aspx>
* An Introduction to Fast Track Data Warehouse Architectures  
  <http://technet.microsoft.com/en-us/library/dd459146(SQL.100).aspx>
* Introduction to the Unified Dimensional Model (UDM)  
  <http://msdn.microsoft.com/en-US/library/ms345143(v=SQL.90).aspx>
* Kimball University: Data Stewardship 101: First Step to Quality and Consistency  
  <http://www.intelligententerprise.com/showArticle.jhtml?articleID=188101650>
* Partitioned Tables and Indexes in SQL Server 2005  
  <http://msdn.microsoft.com/en-US/library/ms345146(v=SQL.90).aspx>
* Partitioned Table and Index Strategies Using SQL Server 2008  
  <http://download.microsoft.com/download/D/B/D/DBDE7972-1EB9-470A-BA18-58849DB3EB3B/PartTableAndIndexStrat.docx>
* Scaling Up Your Data Warehouse with SQL Server 2008  
  <http://msdn.microsoft.com/en-us/library/cc719182(v=SQL.100).aspx>
* Storage Top 10 Best Practices   
  <http://msdn.microsoft.com/en-US/library/cc966534.aspx>
* Strategies for Partitioning Relational Data Warehouses in Microsoft SQL Server  
  <http://msdn.microsoft.com/en-US/library/cc966457.aspx>
* Thinking Global BI: Data-Warehouse Principles for Supporting Enterprise-Enabled Business-Intelligence Applications  
  <http://msdn.microsoft.com/en-us/architecture/aa699414.aspx>
* Top 10 SQL Server 2005 Performance Issues for Data Warehouse and Reporting Applications  
  <http://msdn.microsoft.com/en-US/library/cc917690.aspx>
* Using SQL Server to Build a Hub-and-Spoke Enterprise Data Warehouse Architecture  
  <http://msdn.microsoft.com/en-us/library/dd458815(v=SQL.100).aspx>