The Benefits of Data Modeling in Data Warehousing

NOVEMBER 2008

Table of Contents

Copyright © 2008 CA. All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies. This document is for your informational purposes only. To the extent permitted by applicable law, CA provides this document “As Is” without warranty of any kind, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose, or noninfringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, business interruption, goodwill or lost data, even if CA is expressly advised of such damages.

Executive Summary

CHALLENGES

Organizations today have vast quantities of data. Although this data contains information that is useful to the business, it can be extremely difficult to gather and report on this information. There are several key challenges that need to be addressed:

- Discovering, collecting, and transforming data into a single source of record.
- Ensuring that data is relevant and accurate for business reporting.
- Storing historical data in a format that enables fast searches across large amounts of data.

OPPORTUNITIES

If the data that a business holds can be unlocked and provide meaningful information to business users, there are several opportunities:

- Business users can have access to relevant information to enable them to make informed decisions.
- Fast queries make data more accessible and historical data enables trends to be identified.
- Data can be assessed at every level -- from an individual purchase to the total sales of a multinational corporation.

BENEFITS

Designing data warehouses correctly by using a data model will help meet many of today's data challenges. Key benefits include:

- Designing structures specifically to enable fast querying for business-centric reporting.
- Ensuring that business requirements are met, and reports are accurate and meaningful.
- Documenting source and target systems correctly to aid development, ensure effective version control, and enhance understanding of the systems.

SECTION 1

Introduction

Most organizations have vast quantities of data. Data is constantly collected as every transaction is made, every employee review is completed, and every sales lead is chased. This data is at the core of the systems that run an organization, and the databases that power these systems have been designed in a way to make an organization's business processes as efficient as possible. The problem arises when business users need to report on this information, to determine how many transactions were made this year, for example.

There are several challenges to using transactional systems for business reporting:

- The database design that is required for reporting is very different from the design that is required to optimize the performance of transactional systems.
- Running reports against mission-critical transactional systems slows their performance, negatively affecting the systems that run the organization.
- Data that is stored in a transactional database system is not centralized; there is no single source of information against which reports can be generated.

To solve these challenges, we should create a data warehouse that is designed specifically with business reporting in mind, and contains all of the relevant information for reporting.

The data warehouse is an important supplier of information to the business, so it is essential that we model both its physical and logical designs. The physical design determines the performance and functionality of the data warehouse, and the logical design is the view that we present to developers and users to capture business requirements.

SECTION 2

Designing the Data Warehouse

Data warehouses store all of our data in a form that prioritizes query performance, rather than transactional performance or storage volumes. We can access information about any part of our business, and how these entities relate to performance metrics or other entities. This business insight can provide an insight into the workings of the organization, help with planning, and provide a competitive advantage.

A true data warehouse contains all available data, although often, the database stores a focused subset of the data and is then technically called a data mart.[1]

Data warehouses typically store historical data, which enables us to query previous events accurately. For example, an online transaction processing (OLTP) system would store the current importer of a product and, if queried, would return this importer, regardless of whether the current importer was the same as when the transaction occurred. A data warehouse, on the other hand, would typically store all importers of the product in a form that enables us to link each purchase to its importer accurately.

It is essential to model the data storage design so that the questions that the business asks can be answered effectively. This section covers some of the design choices for data warehouses, which we create in a specific type of data model called the dimensional data model.

Fact Table

The central table of a data warehouse design is called the fact table. This table has one row for every fact, or event. Each fact has one or more numerical, quantifiable measures, for example, price. The fact table also contains multiple dimension values. The dimension

[1] Every concept in this paper refers equally to data warehouses and data marts. The only difference between the two is scope.

values describe the fact and could include values such as time, employee, customer, and location.

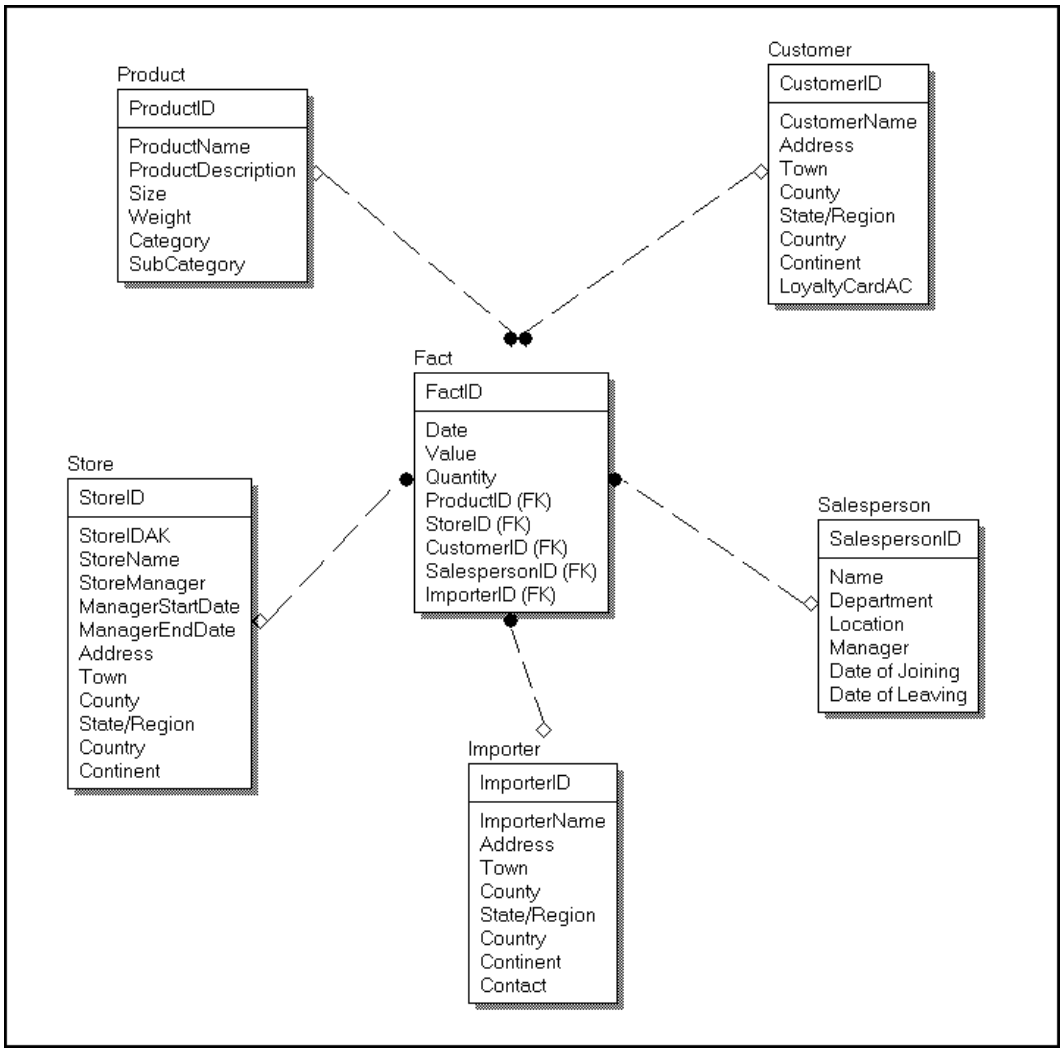
Dimensions

Dimension values are typically stored as foreign keys in the fact table. These keys relate to the primary keys of dimension tables. The dimension tables describe each dimension member. For example, a fact table includes the employee ID of a salesperson and the Employee dimension includes the employee's ID, name, location, and manager. Not all dimensions require a dimension table. For example, a Time dimension might exist entirely in the fact table because there are no further properties other than the time itself. If other properties exist, such as public holidays, a dimension table is required.

STAR SCHEMA

If a fact table has one level of dimension tables, the database design is known as a star schema (see Figure 1).

FIGURE 1: STAR SCHEMA



DATA MODEL SHOWING A STAR SCHEMA

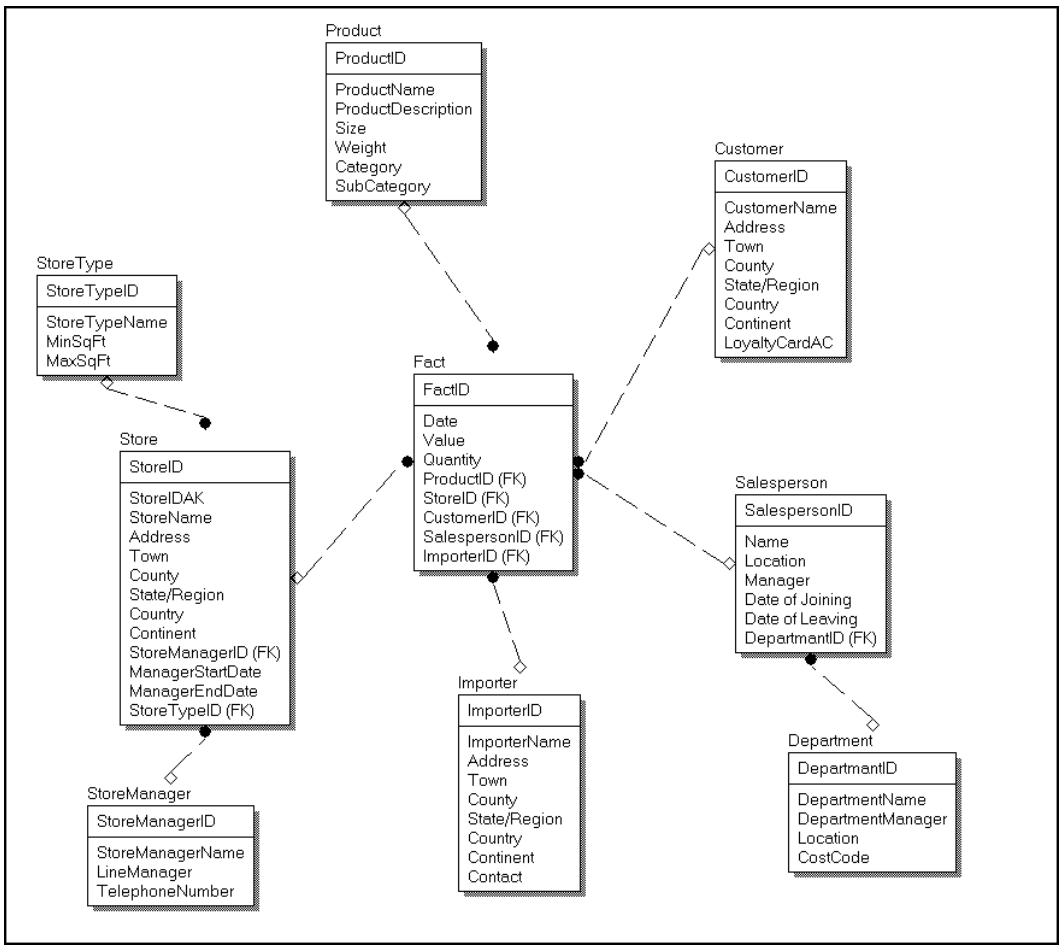
Star schemas are useful because every property of a dimension can be retrieved with one join from the fact table to the relevant dimension table. This improves query performance, but increases data volumes.

SNOWFLAKE SCHEMA

Some details are used infrequently in queries, and should be modeled differently. For example, we seldom query information about an employee's department. Every salesperson is in the Sales department, and, therefore, there is little benefit in analyzing data that pertains to the Sales department in a query. Nevertheless, we want to store this data. We can create a further dimension table that relates to the Employee dimension. This is a snowflake schema (see Figure 2) and is useful because it removes the duplication that would occur if we duplicated departmental information for every employee. We should always consider how often a query will use the snowflake data, however, because it requires an extra join and is therefore slower.

Due to the performance issues, we should typically avoid snowflake schemas in our model.

FIGURE 2: SNOWFLAKE SCHEMA



DATA MODEL INCLUDUING SNOWFLAKE SCHEMAS

Star and snowflake schemas are modeled at the dimension level and do not apply to the design of the whole data warehouse. In Figure 2, the Store and Salesperson dimensions have a snowflake schema, but the Product, Customer, and Importer dimensions have a star schema.

We can see from the design of the fact and dimension tables that data warehouse design is heavily denormalized. Normalization seeks to improve the efficiency of OLTP systems by removing duplication, but, in a system that is purely designed for query speed normalization, it is detrimental to performance.

SLOWLY CHANGING DIMENSIONS

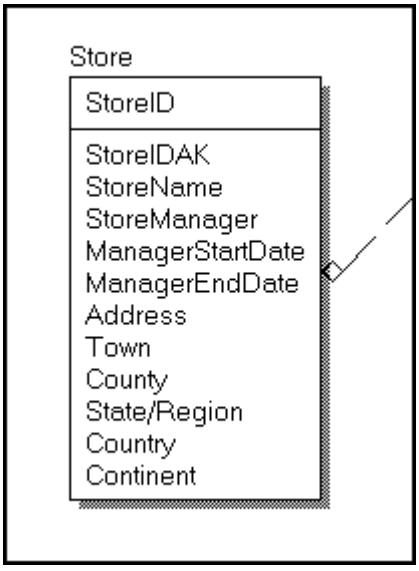
So far, we have taken each dimension as a snapshot in time, but, in reality, dimension attributes change. For example, a customer might be listed as living in Canada, but has only recently moved there from France. We should not assign any facts to Canada that occurred in France. Similarly, employees change department, and stores change manager.

To effectively store the historical changes to dimension members, we need to create slowly changing dimensions (SCDs) in our model. This is an area where modeling is essential because it requires planning to achieve the correct results and an ad-hoc approach is unlikely to succeed. There are three types of SCD in use that cover most historical requirements.

Type 1 SCDs are effectively standard dimensions that allow changes. If an attribute needs to be changed, we simply change it and take no further action. This keeps our records up to date, but does not solve any of the historical problems that were previously mentioned.

Type 2 SCDs (see Figure 3) solve the problem of historical changes by creating multiple records for each dimension member when there is a change. For example, if a store manager changes, we would create another record that lists the new manager. This creates a duplicate key for the record. Therefore, we can store the original key of the store as an alternate key and create a new, unique, primary key. The other problem with a Type 2 SCD is that we need to know which record applies to each fact in the fact table. This is achieved by storing the start and end dates of each record.

FIGURE 3: TYPE 2 SCD



TYPE 2 SCD WITH ALTERNATE PRIMARY KEY AND START AND END DATES

Figure 3 shows the alternate key, StoreIDAK, and the start and end dates of the manager. It is straightforward to find the current manager because this is the only record with a NULL ManagerEndDate field.

Type 3 SCDs are a compromise between Type 1 SCDs and Type 2 SCDs. We need only know the original and current values of the dimension member. For example, we might want a starting price and a current price. We do not require any other prices, so we do not require the complexity of a Type 2 SCD. We can simply store the original and current values as separate attributes of the dimension and only one row is required for each member. Although this simplifies storage, the limited functionality means that most models use Type 1 or Type 2 SCDs.

As you can see, a data warehouse has a substantially different design from an OLTP system, and we need to implement special design techniques to optimize a data warehouse for business reporting.

SECTION 3

Extract, Transform, and Load

Although a data warehouse has a substantially different design from an OLTP system, OLTP systems are the main source of data for a data warehouse. We must carefully plan how to move data from the OLTP systems to the data warehouse. This process is known as extract, transform, and load (ETL), and this section looks at some of the decisions that we must make to model an ETL system.

The first task is to find the available data. This is straightforward in a small, recently started, IT-focused organization. However, a large multinational corporation that has been trading since before computers existed, and has had multiple mergers, is much more complex. It is necessary to investigate each of the systems in place and either check existing documentation or, if this is nonexistent, document the data stores. This could be a very time-consuming and labor-intensive process, and highlights the need for modeling and documentation. We can reverse-engineer many systems by using data-modeling software to reduce the time that it takes to document the data store.

After we have identified the data to be loaded into the data warehouse, we should plan the extract operation. Most modern systems use recognized standards for connectivity such as Open Database Connectivity (ODBC), which make data extraction straightforward. However, legacy systems may require additional analysis such as data profiling before these systems can be modeled effectively and loaded into the data warehouse.

It is highly unlikely that data is already in the correct format for the data warehouse. As we have discussed previously, data warehouses and OLTP systems have fundamentally different designs, so we must transform the data.

Data transformation can be as simple as a Structured Query Language (SQL) query that generates the correct format in the results; however, most systems require a much more complex transformation phase. There are often many inconsistencies in the source data, as the following examples demonstrate:

- The city of Mumbai was previously known as Bombay and the city of Saint Petersburg was previously known as Leningrad.
- Many different currencies might be used.
- Required attributes might be omitted in some data stores.
- Different codes might be used to represent the same value.

We need to standardize the attributes of entities to enable proper analysis of the data to occur. Using a data modeling tool to analyze the source and target systems, especially a repository-based system, assists in the creation and implementation of these standards.

After the source and target data have been documented and designed, we can use an ETL tool to perform most transform operations, but the most complex transforms may benefit from, or require, programming.

Compared to extraction and transformation, loading is relatively straightforward. After extraction and transformation, the data should be in the correct format for loading into the data warehouse. The most important consideration is timing. This is a resource-intensive process and should be performed when the system is barely used, or is not used at all. This is typically at night or during the weekend.

ETL operations require that we have a thorough knowledge of the source and destination database systems. Data modeling provides documentation of these designs and helps to ensure a correct ETL process design.

SECTION 4

The Importance of Data Modeling as a Data-Warehousing Best Practice

Most data warehouse designers use a data modeling tool to create the logical and physical design of the data warehouse. The logical design ensures that all business requirements, definitions, and rules are supported. The physical design ensures optimal performance in the planning of indexes, relationships, data types, and properties. To support developers of OLAP, data-mining, and reporting systems, the data model also acts as documentation for the final data warehouse.

Gathering Business Requirements

It is particularly useful to create logical models as well as physical models for the data warehouse. Typically, we would start a data warehouse design with business users and data architects deciding which entities are required in the data warehouse and which facts should be recorded. This initial design presents the vision of the data warehouse and often has many iterations before all parties are happy with the design. At this stage, we should strive to avoid the common pitfalls of data warehouse design. For example, because we are loading a data warehouse from existing systems, it is very easy to leave behind elements of these systems in the finished design, or even use large parts of the existing model design to save time. By using data modeling, we can see these problems at a very early stage.

We should not think of the logical model purely as a building block for the physical model. Although logical models are a crucial stage in creating a physical model, they also have many uses after we have created the physical model and created the data warehouse. The logical model captures business requirements. It uses naming conventions that closely match the business terms that an organization uses and is the design that is presented to the outside world. Developers of other systems use this design to create interfaces into the data warehouse. We can create several logical models to match the needs of data consumers, while underneath the physical model remains the same. By continuing to develop and maintain the logical model, we avoid the risk of a physical model that attempts to perform both physical and logical modeling tasks, and suffers as a result.

Optimizing Database Performance

Query performance is critical for a data warehouse. We sacrifice data volumes and transactional performance to ensure the highest level of performance for our queries, but the queries only perform optimally if the correct database design is used.

Huge data volumes are involved, so it is very difficult to use a trial-and-error approach to data warehouse design. Therefore, it is beneficial to use data modeling products to automate this process and make it easy to manage all of the metadata that is associated with data warehouses and the data that is consumed by business intelligence (BI) systems. After we have designed the dimension entities and fact table, we can decide on the relationships between tables. The wider BI team can then review and assess this design, and we can incorporate any changes. Making changes at this stage is very straightforward, especially when we compare it to the complexity of modifying a completed data warehouse. After the logical design is approved, we can then begin work on the physical design.

The physical design adds details such as data types and indexing, but may also change the basic entity design for performance reasons. Physical design is particularly important for performance. In an OLTP system, it is very common to see data types that are slightly too large for the data that they store. For example, an eight-byte integer is often used where a four-byte integer would suffice. In a system that contains millions of records, these small inefficiencies are magnified. The indexing of the data warehouse is even more important. We design data warehouses for query performance; the most important aspects of this are the schema, such as star or snowflake, and index design. We may also decide to change the schema of the data warehouse, perhaps from a star schema to a snowflake schema.

Providing Source and Target System Documentation

When modeling an ETL system, it is essential to verify the logical and physical models of the source system as well as the models of the destination. It is often necessary to create an intermediate model for a staging area because many operations cannot be performed in one step. Also, often, extract, transform, and load operations cannot happen at the same time due to the requirements of the source and destination systems.

SECTION 5

Conclusion

If we can harness the vast quantities of data that are available in our organizations, we can derive huge business benefits. We can accurately analyze past results, supply this information to BI systems to find correlations in our data, and present information in a user-friendly way to business users.

To achieve these goals, we must model the data warehouse very carefully. The source systems are often varied and might lack accuracy. The destination systems often have different requirements and, equally, each business user might have different demands. We should spend a substantial amount of time modeling the data warehouse system. Data warehouses are an expensive and time-consuming undertaking, but, when designed correctly, they can give enormous business benefits.

To achieve our goals, we must carefully model a data warehouse system that we have designed to meet our goals; we must create logical models for the many consumers of our data and physical models to ensure proper database performance; and most importantly, we must strive to meet the needs of the business.


Transforming
IT Management