Data Warehouse – Terminology Demystified

Data Warehouse
Creating a Star Schema Database is one of the most important steps in creating a data warehouse. Given how important this process is to building a data warehouse, it is important to understand how to move from a standard, on-line transaction processing (OLTP) system to a final star schema. Please note that a general term is relational data warehouse and may cover both star and snowflake schemas. Below we have looked at some of the terms that you may come across when working with data warehouses. We have tried to demystify the terminology and explain the reason for some of the techniques used when building a data warehouse.

OLTP
OLTP stands for Online Transaction Processing. This is a standard, normalized database structure. OLTP is designed for transactions, which means that inserts, updates, and deletes must be fast. Imagine a call centre that takes orders. Call takers are continually taking calls and entering orders that may contain numerous items. Each order and each item must be inserted into a database. Since the performance of the database is critical, database designers want to maximize the speed of inserts (and updates and deletes). To maximize performance, some businesses even limit the number of records in the database by frequently archiving data.

OLAP and Star Schema
OLAP stands for Online Analytical Processing. OLAP is a term that means many things to many people. Here, the term OLAP and Star Schema are basically interchangeable. The assumption is that a star schema database is an OLAP system. An OLAP system consists of a relational database designed for the speed of retrieval, not transactions, and holds read-only, historical, and possibly aggregated data. While an OLAP/Star Schema may be the actual data warehouse, most companies build cube structures from the relational data warehouse in order to provide faster, more powerful analysis on the data.
Data Warehouse and Data Mart

Data Warehouses and Data Marts differ in scope only. This means that they are built using the exact same methods and procedures, so the process is the same while only their intended scope varies.

A data warehouse (or mart) is a way of storing data for later retrieval. This retrieval is almost always used to support decision-making in the organization. That is why many data warehouses are considered to be DSS (Decision-Support Systems). While some data warehouses are merely archival copies of data, most are used to support some type of decision-making process. The primary benefit of taking the time to create a star schema, and then possibly cube structures, is to speed the retrieval of data and format that data in a way that it is easy to understand. This means that a star schema is built not for transactions but for queries.

Both a data warehouse and a data mart are storage mechanisms for read-only, consolidated, historical data. Read-only means that the person looking at the data cannot directly change it. "Consolidated" means that the data may have come from various sources. Many companies have purchased different vertical applications from various vendors to handle such tasks as human resources (HR), accounting/finance, inventory, and so forth. These systems may run on multiple operating systems and use different database engines. Each of these applications may store their own copy of an employee table, product table, and so on. A relational data warehouse must take data from all these systems and consolidate it so it is consistent, which means it is in a single format.

The "historical" part means the data may be only a few minutes old, but often it is at least a day old. A data warehouse usually holds data that goes back a certain period in time, such as five years. In contrast, most OLTP systems usually retain data for as long as it is "current" or active. An order table, for example, may move order data to an archive table once the order has been completed, shipped, and received by the customer.

The data in data warehouses and data marts may also be aggregated. While there are many different levels of aggregation possible in a typical data warehouse, a star schema may have a "base" level of aggregation, which is one in which all the data is aggregated to a certain point in time.

Aggregations

There is no magic to the term "aggregations." It simply means a summarized, typically additive value. The level of aggregation in a star schema depends on the scenario. Many star schemas are aggregated to some base level, called the grain, although this is becoming somewhat less common as developers rely on cube building engines to summarize to a base level of granularity.
Reasons to Denormalize

When database administrators are asked why they would ever denormalize, the first (and often only) answer is: speed. One of the key disadvantages to the OLTP structure is it is built for data inserts, updates, and deletes, but not data retrieval. Therefore, one method of squeezing some speed out of it is by denormalizing some of the tables and having queries pull data from fewer tables. These queries are faster because they perform fewer joins to retrieve the same record set. Joins are relatively slow and are also confusing to many end users. By denormalizing, users are presented with a view of the data that is far easier for them to understand. The second view is much easier for the end user to understand. While a normalized schema requires joins to create this view, putting all the data in a single table allows the user to perform this query without using joins.

Facts and Dimensions

When examining how people look at data, they usually want to see some sort of aggregated data. These data elements are called measures. These measures are numeric values that are measurable and usually additive. For example, sales value is a perfect measure. Every order that comes in generates a certain sales volume measured in some currency. If a company sells twenty products in one day, each for five pounds, they generate 100 pounds in total sales value. Therefore, sales value is one measure most companies track. Companies may also want to know how many customers they had that day. Did five customers buy an average of four products each, or did just one customer buy twenty products? Sales value and customer counts are two measures businesses may want to track. Just tracking measures isn't enough, however. People need to look at measures using those "by" conditions. The "by" conditions are called dimensions. In order to examine sales value, people almost always want to see them by day, or by quarter, or by year. There is almost always a time dimension on anything people ask for. They may also want to know sales by category or by product. These "by" conditions will map into dimensions: there is almost always a time dimension, and product and geography dimensions are very common as well. Therefore, in designing a star schema, the first order of business is usually to determine what people want to see (the measures) and how they want to see it (the dimensions).

Mapping Dimensions into Tables

Dimension tables answer the "why" portion of a question: how do people want to slice the data? For example, people almost always want to view data by time. Users often don't care what the grand total for all data happens to be. If the data happens to start on June 14, 1989, do users really care how much total sales have been since that date, or do they really care how one year compares to other years? Comparing one year to a previous year is a form of trend analysis and one of the most common things done with data in a star schema. Relational data warehouses may also have a location or geography dimension. This allows users to compare the sales in one region to those in another. They may see that sales are weaker in one region than any other region. This may indicate the
presence of a new competitor in that area, or a lack of advertising, or some other factor that bears investigation.

When designing dimension tables, there are a few rules to keep in mind. First, all dimension tables should have a single-field primary key. This key is typically a surrogate key and is often just an identity column, consisting of an automatically incrementing number. The value of the primary key is meaningless, hence the surrogate key; the real information is stored in the other fields. These other fields, called attributes, contain the full descriptions of the dimension record. For example, if there is a Product dimension (which is common) there are fields in it that contain the description, the category name, the sub-category name, the weight, and so forth. These fields do not contain codes that link to other tables. Because the fields contain full descriptions, the dimension tables are often fat; they contain many large fields.

Dimension tables are often short, however. A company may have many products, but even so, the dimension table cannot compare in size to a normal fact table. For example, even if a company has 30,000 products in the product table, the company may track sales for these products each day for several years. Assuming the company actually only sells 3,000 products in any given day, if they track these sales each day for ten years, they end up with this equation: 3,000 products sold X 365 day/year * 10 years equals almost 11,000,000 records! Therefore, in relative terms, a dimension table with 30,000 records will be short compared to the fact table.

Given that a dimension table is fat, it may be tempting to normalize the dimension table. Normalizing the dimension tables is called a snowflake schema.

**Dimensional Hierarchies**

Developers have been building hierarchical structures in OLTP systems for years. However, hierarchical structures in an OLAP system are different because the hierarchy for the dimension is actually stored in a single dimension table (unless snowflaked)

The product dimension, for example, contains individual products. Products are normally grouped into categories, and these categories may well contain sub-categories. For instance, a product with a product number of X12JC may actually be a refrigerator. Therefore, it falls into the category of major appliance, and the sub-category of refrigerator. There may have more levels of sub-categories, which would further classify this product. The key here is that all of this information is stored in fields in the dimension table.

**Snowflake Schemas**

Sometimes, the dimension tables have the hierarchies broken out into separate tables. This is a more normalized structure, but leads to more difficult queries and slower response times.

Adopting a snowflake approach increases the number of joins and can slow queries. Since the purpose of an OLAP system is to speed queries, snowflaking is usually not productive. Some people try to normalize the dimension tables to save space. However, in the overall scheme of the data warehouse, the dimension tables usually only account for about 1% of the total storage. Therefore, any space savings from normalizing, or snowflaking, are negligible.
Building the Fact Table

The Fact Table holds the measures, or facts. The measures are numeric and additive across some or all of the dimensions. For example, sales are numeric and users can look at total sales for a product, or category, or subcategory, and by any time period. The sales figures are valid no matter how the data is sliced.

While the dimension tables are short and fat, the fact tables are generally long and skinny. They are long because they can hold the number of records represented by the product of the counts in all the dimension tables.

Fact Granularity

One of the most important decisions in building a star schema is the granularity of the fact table. The granularity, or frequency, of the data is determined by the lowest level of granularity of each dimension table, although developers often discuss just the time dimension and say a table has a daily or monthly grain. For example, a fact table may store weekly or monthly totals for individual products. The lower the granularity, the more records that will exist in the fact table. The granularity also determines how far users can drill down without returning to the base, transaction-level data.

One of the major benefits of the star schema is that the low-level transactions may be summarized to the fact table grain. This greatly speeds the queries performed as part of the decision support process. The aggregation or summarization of the fact table is not always done if cubes are being built, however.

Changing Attributes

One of the greatest challenges in a star schema is the problem of changing attributes. As an example, imagine a store is located in a particular region, territory, and zone. Some companies realign their sales regions, territories, and zones occasionally to reflect changing business conditions. However, if the company simply updates the table to reflect the changes, and users then try to look at historical sales for a region, the numbers will not be accurate. By simply updating the region for a store, the total sales for that region will appear as if the current structure has always been true. The business has "lost" history.

In some cases, the loss of history is fine. In fact, the company might want to see what the sales would have been had this store been in that other region in prior years. More often, however, businesses do not want to change the historical data. In this case, the typical approach is to create a new record for the store. This new record contains the new region, but leaves the old store record, and therefore the old regional sales data, intact. This approach, however, prevents companies from comparing this stores current sales to its historical sales unless the previous Store ID is preserved. In most cases the answer it to keep the existing Store Name (the primary key from the source system) on both records but add start date and end date fields to indicate when each record is active. The Store ID is a surrogate key.
so each record has a different Store ID but the same Store Name, allowing data to be examined for the store across time regardless of its reporting structure. This particular problem is usually called a “slowly-changing dimension” and there are various methods for handling it. We won’t go into the detail of slowly changing dimensions here. There are no right and wrong answers. Each case will require a different solution to handle changing attributes.

Aggregations

The data in the fact table is already aggregated to the fact table’s grain. However, users often ask for aggregated values at higher levels. For example, they may want to sum sales to a monthly or quarterly number. In addition, users may be looking for a total at a product or category levels. These numbers can be calculated on the fly using a standard SQL statement. This calculation takes time, and therefore some people will want to decrease the time required to retrieve higher-level aggregations. Some people store higher-level aggregations in the database by pre-calculating them and storing them in the fact table. This requires that the lowest-level records have special values put in them. For example, a Time Dimension record that actually holds weekly totals might have a 9 in the Day Of Week field to indicate that this particular record holds the total for the week.

A second approach is to build another fact table but at the weekly grain. All data is summarized to the weekly level and stored there. This works well for storing data summarized at various levels, but the problem comes into play when examining the number of possible tables needed. To summarize at the weekly, monthly, quarterly, and yearly levels by product, four tables are needed in addition to the “real”, or daily, fact table. However, what about weekly totals by product subcategory? And monthly totals by store? and quarterly totals by product category and territory? Each combination would require its own table. This approach has been used in the past, but better alternatives exist. These alternatives usually consist of building a cube structure to hold pre-calculated values. Cubes were designed to address the issues of calculating aggregations at a variety of levels and respond to queries quickly.

Again there is not right or wrong way to approach aggregation and typically a mixture of all methods would be used to achieve the most efficient result.