Unit 06 Notes

Designing Dimensional Models

Process Overview

The design process iterates through three phrases:

1) Start with a few **preparation** steps, after which an **initial diagram** is derived from the bus matrix. The diagram clarifies the grain of the fact table, as well as the major dimensions, attributes and fact metrics.
2) This sketchy model is **iteratively developed** into the full-fledged and robust model, table by table, with increasing levels of detail.
3) The final model is **reviewed** and **validated** with business and IT representatives to make sure that it satisfies their requirements.

A dimensional model for a single business process can take four weeks to build. The recommended schedule includes two design sessions a day, one in the morning and the other in the afternoon, with 2-3 hours per session, 3-4 days a week. Figure 7-1 on pp 289 shows the process flow for one iteration (the deliverables are listed to the right). The actual design process iterates many times through the diagram. The next sections present the details of this process.

Preparation

The preparation phase for dimensional modeling includes the following steps:

- Identify the required team roles and participants. Table 7-1 on pp 290 lists the typical roles. Most of the work is done by the core team, which usually consists of a data modeler, a business analyst and a member of the ETL team.
- Review the business requirements document. Start the dimensional modeling from them rather than from the source data. The dimensional model must support the requirements.
- Review the source data. You can use simple SQL queries, data profiling tools, or profiling capabilities of ETL tools to enhance your understanding of the source data.
- Set up the modeling environment. Your initial sketch of the model should be stored in a spreadsheet like the one in Figure 7-2 on pp 294. This model is easy to improve iteratively. Switch to actual modeling tools only in later stages.
- Set the naming conventions. A common naming schema is Entity_Qualifiers_Class. For example, sales_dollar_amount is the field in the sales fact table that represents the amount.
- Obtain facilities and supplies. These include a room with a projector, whiteboards and markers.

At the completion of the preparation phase, we are ready to build the initial high-level model starting from the bus matrix.

Develop the Initial Model

In the first pass, we draw the first dimensional model diagram. This is a graphical representation of the fact and dimension tables for the chosen business process. It is a data model at the entity level. The diagram is sometimes called a **bubble chart**, and is best done using flipcharts. Each business process' fact table should occupy a separate page. Dimensions should be arranged in a predictable order, with the most important on top. Figure 7-3 on pp 304 presents an example.

The second pass creates an initial list of attributes for each dimension table and metrics for the fact table. This is a deliverable called the **initial attribute and metric list**. Store it in a spreadsheet like the one in Figure 7-2 on pp 294, creating one worksheet per table. When choosing some attributes over others, record concerns and controversies in an **issue list**, which is the third (and last) deliverable of these initial design sessions.

With the high level model diagram, the initial attribute and metric list, and the issue list in hand, we are ready to move to the design sessions for the iterative process that designs the final dimensional model.

Iterative Model Development Overview

The iterative phrase of dimensional modeling process has four steps:

1. **Choose the business process or measurement event** – use the enterprise data warehouse bus matrix, as each row corresponds to a business process. Iteratively build the DW one business process at a time.
2. **Declare the fact table grain** – the grain answers the question "when do we add a new row in the fact table?" The fact table should be as finely grained as possible. Skipping this step is the most common design error.
3. **Identify the dimensions** – the minimal set of dimensions follows from the fact table grain. Additional dimensions should take unique values at the same grain. Be ready to revise your intuitions that were based on the transaction system schema.
4. **Identify the facts** – they are physically measured or derived from such measurements. All facts must share the same grain.

Start with the dimension tables, preferably with an easy one, such as the date dimension. Do one or two tables per design session. The purpose of these detailed sessions is to add all the missing details to the high-level model.

Identify of the best data sources to populate the target design. Start by compiling a comprehensive list of candidate sources; some candidates may come from the business requirements document, others from the IT people. Then analyze and profile each candidate. Finally, choose the best sources to populate your model, and document the reasons for your choices.

Establish the conformed dimensions and then identify the base facts and the derived facts. Additive derived facts that can be calculated from the other facts in the same row can be shown in a user view as if they existed in the real data. Non-additive derived facts cannot be presented in a user view because

they violate the grain of the fact table. They will be calculated at query time by the BI tool – most BI tools offer a library to develop and maintain such facts. If you have many derived facts, document them in a separate document such as the **derived fact worksheet** shown in Figure 7-5 on pp 312.

The detailed design of each fact and dimension table should be documented in a **detailed design worksheet** as illustrated in Figure 7-6 on pp 313. We will refer again to this key deliverable in the section about the final documentation of the design.

The detailed design process enhances our understanding and can change some of our initial decisions regarding business processes, fact tables, and dimension tables. We should be ready to revise the bus matrix as needed. Figure 7-7 on pp 315 illustrates a bus matrix that has been updated at this time.

Model Review and Validation

The dimensional model is reviewed with various audiences, each with a different level of expertise and understanding. The first reviewers should be chosen from among DBAs and system developers from the IT department. They have a great understanding of the transaction system but don't usually know much about dimensional models. Therefore, a short tutorial is in order. They are good at spotting model errors and verifying that all required data is available.

The second review should involve business users not directly involved in the design process. This review is also an educative session for these users. They should be shown simple examples that illustrate the analytic capabilities of the dimensional model. The purpose is to verify that the model can answer all their questions about the business process.

Document and Finalize the Design

Once the design is reviewed and validated, we can start documenting it. The final design document includes several sections listed on pp 320. The detailed dimensional design worksheet (Figure 7-6 on pp 313) will be the basis for the project's **source-to-target mapping** document, which describes how each target table and column in the DW will be populated. This is an essential document for the ETL team.

Some important aspects of the dimensional modeling process are summarized in the BLUEPRINT FOR ACTION box on pp 323-325.

Alternative Modeling Techniques

Entity relationship modeling and dimensional modeling are not the only data modeling techniques. Alternatives include subject modeling, domain modeling, fact qualifier matrix/modeling, state transition modeling, and information flow modeling. Fact qualifier matrix/modeling is a useful tool to capture granularity/reporting requirements. Each model emphasizes different types of information. There is more than one way of modeling a business requirement, and every project has various types of requirements. It is a good practice to leverage the modeling techniques that best fit the project.

Designing the Physical Database

Process Overview

Kimball's Chapter 8 is addressed primarily to the Database Administrator (DBA) involved in the data warehouse development. It is not a complete course in physical model design, by any measure. Many concepts in this chapter were introduced in the recommended prerequisite database course. In this section of our notes, we will only mention some DB-design topics that are specific to data warehouses.

The top-level DB design process is illustrated in Figure 8-1 on pp 329. As shown in the figure, this is another iterative process. After developing the system-wide standards, we create an initial physical model to give the ETL development a target. As the ETL process starts populating the tables with data, we can design the OLAP DB and eventually complete the physical design process. Some elements of the physical design, such as indexing and aggregation, will be improved even in the late stages of the DW project. We'll give a bit more details on these design phases below.

Standards, Initial Model, and Size Estimates

The physical design process starts by developing standards regarding table and column naming conventions (these names should be the same as in the logical model), null values (they should be avoided in dimension tables), directory structure and file names (use a source control system), as well as primary and foreign keys. As we mentioned in the previous lecture, a dimension table should use an integer surrogate column as its primary key. The primary keys of all dimension tables become foreign keys in the fact table. The primary key of the fact table is a subset of these foreign keys.

After developing the necessary standards, we start developing the physical model itself. It is recommended to use the logical model as the starting point for the physical model, and depart from it as little as possible. In other words, we should keep the physical model similar to the logical model. One of the very first steps is to expand the dimensional design worksheet by adding the physical model information pieces listed on pp 336-337. We thus produce the definitive source-to-target mapping.

Our next task is to produce the initial estimate for the database size on disk. The key information to remember here is that the fact table stores 90% of the data in the data warehouse. Thus, we should start by estimating the size in bytes of a fact table row, and multiplying that number by the estimated number of rows. The second largest storage consumer is the fact table index. We also need to include in our estimates the storage space for ETL staging tables, ETL audit tables, access monitoring tables and security tables.

Indexes

After estimating the storage requirements, we are ready to build an initial index plan. This plan specifies which table columns will be indexed, and what type of index will be used in each case. The goal is to index the columns that are the most frequently used by the BI queries and reports, and select the index type that offers the shortest response time to frequent queries. The most common types used in RDBMs are the B-Tree index, the clustered index, and the bitmapped index.

A **B-Tree** (see Figure 1) is a tree data structure that keeps the data sorted in a way that enables searches, sequential access, insertions and deletions in logarithmic time. As shown in Figure 1, the data values 7 and 16 stored in the parent node are sorted in an ascending order. Between these two values, we have a pointer to a child node that contains values larger than 7 and smaller than 16. In addition, to the left of 7 we have a pointer to a node that stores data values smaller than 7. Similarly, to the right of 16 we have a pointer to a child node with data values larger than 16. All children nodes can be sub-trees that recursively follow the same rules. In a B-Tree, every data value occurs in a single node. In a B-Tree used for database indexing, every data value is accompanied by a list of row IDs that have that value on the column that is being indexed. B-Tree indexes work best for table columns with many different data values.

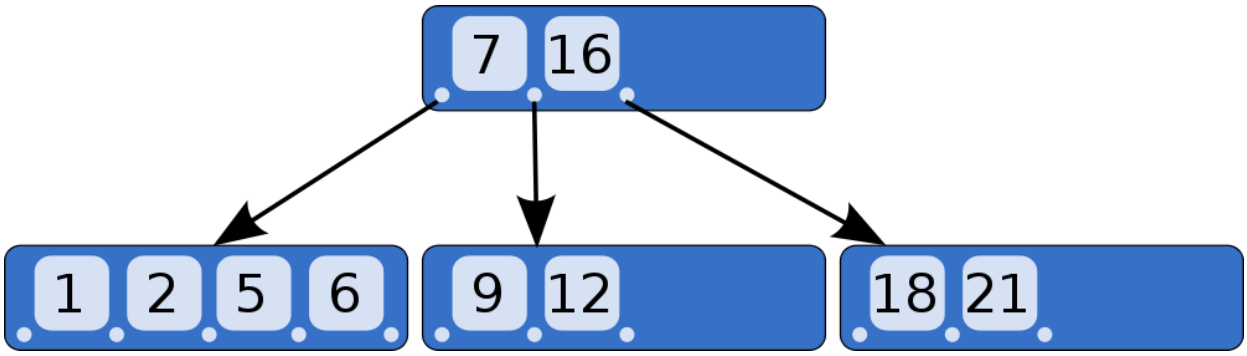

Figure 1 Example of a B-tree (Wikimedia Commons)

The next common database index type is the **clustered index**. This index sorts and stores the table's rows based on the values of their keys. The default index for a table's primary key is a clustered index, and there can be only one clustered index per table. These indexes also work best for columns with many different data values.

In contrast with the B-Tree and the clustered index, the **bitmap index** works best for columns with few different data values. The bitmapped index for a column whose cardinality is n consists of n strings of bits – one string for each data value. Such a bit string has one bit for each row in the table. In the bit string associated with the data value v, the bit for a particular is set to 1 if and only if that row contains value v. All the other bits of that bit string are set to 0.

Example: Assume that we have a table with 5 rows, and we want to index its gender column that has 3 possible values: "M", "F", and "?" Assume also that the values in the gender column for the 5 rows are, in order, "M", "F", "?", "F", and "F". The best index type for this low-cardinality column is a bitmapped index. Our index consists of 3 strings of bits (one string for each data value), each having 5 bits (one bit for each row). The string for "M" is 10000, the string for "F" is 01011, and the string for "?" is 00100.

Bitmap indexes are compact and be compressed further with a variety of efficient run-length encoding techniques. Even in compressed forms they can be used in bitwise operations, which computers can perform at great speeds. We can also index higher-cardinality columns with such an index, by first grouping data values into bins and then building the index.

Aggregations

The factor with the largest influence on the performance of a large data warehouses are the aggregate or summary records that must coexist with the atomic grain facts. A typical DW/BI system has multiple sets of aggregations, one for each grouping levels within the key dimensions. These aggregations can be built in several alternative ways.

- Store them in separate tables, each being are populated by executing a SQL query that performs a GROUP BY. This is the simplest solution.
- Store them in the materialized views offered by Oracle DB systems. A materialized view is a view that physically stores its result set, thus enabling indexing, partitioning, and controlled access.
- Use the aggregation management and capabilities of an OLAP engine. They are very robust and similar to the previous two solutions, but offer less control over storage and indexing.
- Use the aggregation navigation functionality of a commercial BI tool. This can be very efficient, but only the users of that tool will benefit from it.

The first key to good data aggregation is choosing the right aggregations. We can't build all possible aggregations, as this would be prohibitive in terms of storage space. On the other hand, too little aggregation increases query response time, since some aggregations will have to be built on the fly. Finding the sweet spot usually involves monitoring the query patterns and selecting the aggregations to match them. The OLAP DBs offer much more assistance in solving this very difficult problem.

After we chose the most appropriate aggregations and built them, we also need to maintain them over the life of the project. Basically, aggregations should be updated whenever new data is loaded into the fact table. Note that we can't allow users to query the DW after new data has been loaded but before the aggregations have been updated accordingly – that would produce inconsistent results. If our aggregations are simple summary tables populated by SQL queries, our ETL process must maintain them. If our aggregations are managed by the DB system (as materialized views or OLAP aggregations), the system does the maintenance work for us.

Data Modeling Tools

Data modeling tools can be very helpful in dimensional modeling and physical model design. They have capabilities to generate database object creation scripts based on the information captured in the data model. They can also automatically generate the objects in the database environment once connected to the database. Furthermore, these tools have capabilities to generate (or usually referred to as "reverse engineer") a data model based on existing database objects. They provide options to store useful metadata about the database objects. Popular data modeling tools in the industry include Embarcadero-ERStudio, ErWin, and Oracle Designer. It is always a good idea to learn how to use the data modeling tools available in your organization.

Some important aspects of physical model design are summarized in the BLUEPRINT FOR ACTION box on pp 364-367.