Data Warehouse Design Considerations

Dave Browning and Joy Mundy
Microsoft Corporation

December 2001

Applies to:
Microsoft® SQL Server™ 2000

Summary: Data warehousing is one of the more powerful tools available to support a business enterprise. Learn how to design and implement a data warehouse database with Microsoft SQL Server 2000. (25 printed pages)

Contents

Introduction

Data warehouses support business decisions by collecting, consolidating, and organizing data for reporting and analysis with tools such as online analytical processing (OLAP) and data mining. Although data warehouses are built on relational database technology, the design of a data warehouse database differs substantially from the design of an online transaction processing system (OLTP) database.

The topics in this paper address approaches and choices to be considered when designing and implementing a data warehouse. The paper begins by contrasting data warehouse databases with OLTP databases and introducing OLAP and data mining, and then adds information about design issues to be considered when developing a data warehouse with Microsoft® SQL Server™ 2000. This paper was first published as Chapter 17 of the *SQL Server 2000 Resource Kit*, which also includes further information about data warehousing with SQL Server 2000. Chapters that are pertinent to this paper are indicated in the text.

Data Warehouses, OLTP, OLAP, and Data Mining

A relational database is designed for a specific purpose. Because the purpose of a data warehouse differs from that of an OLTP, the design characteristics of a relational database that supports a data warehouse differ from the design characteristics of an OLTP database.

Data warehouse database	OLTP database
Designed for analysis of business measures by categories and attributes	Designed for real-time business operations
Optimized for bulk loads and large, complex, unpredictable queries that access many rows per table	Optimized for a common set of transactions, usually adding or retrieving a single row at a time per table
Loaded with consistent, valid data; requires no real time validation	Optimized for validation of incoming data during transactions; uses validation data tables

Supports few concurrent users relative to OLTP	Supports thousands of concurrent users

Back to top

A Data Warehouse Supports OLTP

A data warehouse supports an OLTP system by providing a place for the OLTP database to offload data as it accumulates, and by providing services that would complicate and degrade OLTP operations if they were performed in the OLTP database.

Without a data warehouse to hold historical information, data is archived to static media such as magnetic tape, or allowed to accumulate in the OLTP database.

If data is simply archived for preservation, it is not available or organized for use by analysts and decision makers. If data is allowed to accumulate in the OLTP so it can be used for analysis, the OLTP database continues to grow in size and requires more indexes to service analytical and report queries. These queries access and process large portions of the continually growing historical data and add a substantial load to the database. The large indexes needed to support these queries also tax the OLTP transactions with additional index maintenance. These queries can also be complicated to develop due to the typically complex OLTP database schema.

A data warehouse offloads the historical data from the OLTP, allowing the OLTP to operate at peak transaction efficiency. High volume analytical and reporting queries are handled by the data warehouse and do not load the OLTP, which does not need additional indexes for their support. As data is moved to the data warehouse, it is also reorganized and consolidated so that analytical queries are simpler and more efficient.

OLAP is a Data Warehouse Tool

Online analytical processing (OLAP) is a technology designed to provide superior performance for ad hoc business intelligence queries. OLAP is designed to operate efficiently with data organized in accordance with the common dimensional model used in data warehouses.

A data warehouse provides a multidimensional view of data in an intuitive model designed to match the types of queries posed by analysts and decision makers. OLAP organizes data warehouse data into multidimensional cubes based on this dimensional model, and then preprocesses these cubes to provide maximum performance for queries that summarize data in various ways. For example, a query that requests the total sales income and quantity sold for a range of products in a specific geographical region for a specific time period can typically be answered in a few seconds or less regardless of how many hundreds of millions of rows of data are stored in the data warehouse database.

OLAP is not designed to store large volumes of text or binary data, nor is it designed to support high volume update transactions. The inherent stability and consistency of historical data in a data warehouse enables OLAP to provide its remarkable performance in rapidly summarizing information for analytical queries.

In SQL Server 2000, Analysis Services provides tools for developing OLAP applications and a server specifically designed to service OLAP queries.

Data Mining is a Data Warehouse Tool

Data mining is a technology that applies sophisticated and complex algorithms to analyze data and expose interesting information for analysis by decision makers. Whereas OLAP organizes data in a model suited for exploration by analysts, data mining performs analysis on data and provides the results to decision makers. Thus, OLAP supports model-driven analysis and data mining supports data-driven analysis.

Data mining has traditionally operated only on raw data in the data warehouse database or, more commonly, text files of data extracted from the data warehouse database. In SQL Server 2000, Analysis Services provides data mining technology that can analyze data in OLAP cubes, as well as data in the relational data warehouse database. In addition, data mining results can be incorporated into OLAP cubes to further enhance model-driven analysis by providing an additional dimensional viewpoint into the OLAP model. For example, data mining can be used to analyze sales data against customer attributes and create a new cube dimension to assist the analyst in the discovery of the information embedded in the cube data.

For more information and details about data mining in SQL Server 2000, see Chapter 24, "Effective Strategies for Data Mining," in the *SQL Server 2000 Resource Kit*.

Back to top

Designing a Data Warehouse: Prerequisites

Before embarking on the design of a data warehouse, it is imperative that the architectural goals of the data warehouse be clear and well understood. Because the purpose of a data warehouse is to serve users, it is also critical to understand the various types of users, their needs, and the characteristics of their interactions with the data warehouse.

Data Warehouse Architecture Goals

A data warehouse exists to serve its users—analysts and decision makers. A data warehouse must be designed to satisfy the following requirements:

- Deliver a great user experience—user acceptance is the measure of success
- Function without interfering with OLTP systems
- Provide a central repository of consistent data
- Answer complex queries quickly
- Provide a variety of powerful analytical tools, such as OLAP and data mining

Most successful data warehouses that meet these requirements have these common characteristics:

- Are based on a dimensional model
- Contain historical data
- Include both detailed and summarized data
- Consolidate disparate data from multiple sources while retaining consistency
- Focus on a single subject, such as sales, inventory, or finance

Data warehouses are often quite large. However, size is not an architectural goal—it is a characteristic driven by the amount of data needed to serve the users.

Data Warehouse Users

The success of a data warehouse is measured solely by its acceptance by users. Without users, historical data might as well be archived to magnetic tape and stored in the basement. Successful data warehouse design starts with understanding the users and their needs.

Data warehouse users can be divided into four categories: Statisticians, Knowledge Workers, Information Consumers, and Executives. Each type makes up a portion of the user population as illustrated in this diagram.

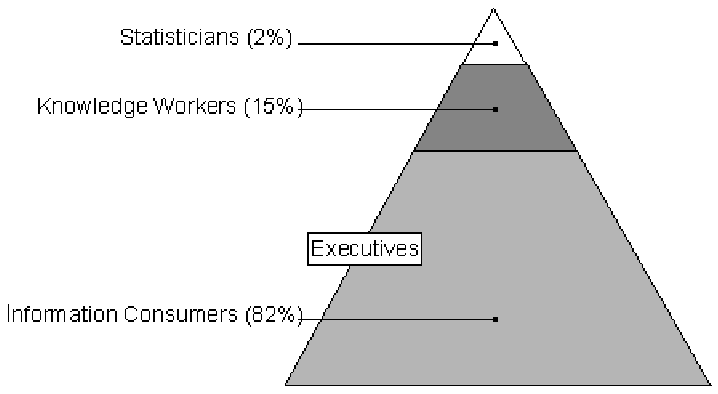

Figure 1. The User Pyramid

Statisticians: There are typically only a handful of sophisticated analysts—Statisticians and operations research types—in any organization. Though few in number, they are some of the best users of the data warehouse; those whose work can contribute to closed loop systems that deeply influence the operations and profitability of the company. It is vital that these users come to love the data warehouse. Usually that is not difficult; these people are often very self-sufficient and need only to be pointed to the database and given some simple instructions about how to get to the data and what times of the day are best for performing large queries to retrieve data to analyze using their own sophisticated tools. They can take it from there.

Knowledge Workers: A relatively small number of analysts perform the bulk of new queries and analyses against the data warehouse. These are the users who get the "Designer" or "Analyst" versions of user access tools. They will figure out how to quantify a subject area. After a few iterations, their queries and reports typically get published for the benefit of the Information Consumers. Knowledge Workers are often deeply engaged with the data warehouse design and place the greatest demands on the ongoing data warehouse operations team for training and support.

Information Consumers: Most users of the data warehouse are Information Consumers; they will probably never compose a true ad hoc query. They use static or simple interactive reports that others have developed. It is easy to forget about these users, because they usually interact with the data warehouse only through the work product of others. Do not neglect these users! This group includes a large number of people, and published reports are highly visible. Set up a great communication infrastructure for distributing information widely, and gather feedback from these users to improve the information sites over time.

Executives: Executives are a special case of the Information Consumers group. Few executives actually issue their own queries, but an executive's slightest musing can generate a flurry of activity among the other types of users. A wise data warehouse designer/implementer/owner will develop a very cool digital dashboard for executives, assuming it is easy and economical to do so. Usually this should follow other data warehouse work, but it never hurts to impress the bosses.

Back to top

How Users Query the Data Warehouse

Information for users can be extracted from the data warehouse relational database or from the output of analytical services such as OLAP or data mining. Direct queries to the data warehouse relational database should be limited to those that cannot be accomplished through existing tools, which are often more efficient than direct queries and impose less load on the relational database.

Reporting tools and custom applications often access the database directly. Statisticians frequently extract data for use by special analytical tools. Analysts may write complex queries to extract and compile specific information not readily accessible through existing tools. Information consumers do not interact directly with the relational database but may receive e-mail reports or access web pages that expose data from the relational database. Executives use standard reports or ask others to create specialized reports for them.

When using the Analysis Services tools in SQL Server 2000, Statisticians will often perform data mining, Analysts will write MDX queries against OLAP cubes and use data mining, and Information Consumers will use interactive reports designed by others.

Back to top

Developing a Data Warehouse: Details

The phases of a data warehouse project listed below are similar to those of most database projects, starting with identifying requirements and ending with deploying the system:

- Identify and gather requirements
- Design the dimensional model
- Develop the architecture, including the Operational Data Store (ODS)
- Design the relational database and OLAP cubes
- Develop the data maintenance applications
- Develop analysis applications
- Test and deploy the system

Back to top

Identify and Gather Requirements

Identify sponsors. A successful data warehouse project needs a sponsor in the business organization and usually a second sponsor in the Information Technology group. Sponsors must understand and support the business value of the project.

Understand the business before entering into discussions with users. Then interview and work with the users, not the data—learn the needs of the users and turn these needs into project requirements. Find out what information they need to be more successful at their jobs, not what data they think should be in the data warehouse; it is the data warehouse designer's job to determine what data is necessary to provide the information. Topics for discussion are the users' objectives and challenges and how they go about making business decisions. Business users should be closely tied to the design team during the logical design process; they are the people who understand the meaning of existing data. Many successful projects include several business users on the design team to act as data experts and "sounding boards" for design concepts. Whatever the structure of the team, it is important that business users feel ownership for the resulting system.

Interview data experts after interviewing several users. Find out from the experts what data exists and where it resides, but only after you understand the basic business needs of the end users. Information about available data is needed early in the process, before you complete the analysis of the business needs, but the physical design of existing data should not be allowed to have much influence on discussions about business needs.

Communicate with users often and thoroughly—continue discussions as requirements continue to solidify so that everyone participates in the progress of the requirements definition.

Back to top

Design the Dimensional Model

User requirements and data realities drive the design of the dimensional model, which must address business needs, grain of detail, and what dimensions and facts to include.

The dimensional model must suit the requirements of the users and support ease of use for direct access. The model must also be designed so that it is easy to maintain and can adapt to future changes. The model design must result in a relational database that supports OLAP cubes to provide "instantaneous" query results for analysts.

An OLTP system requires a normalized structure to minimize redundancy, provide validation of input data, and support a high volume of fast transactions. A transaction usually involves a single business event, such as placing an order or posting an invoice payment. An OLTP model often looks like a spider web of hundreds or even thousands of related tables.

In contrast, a typical dimensional model uses a star or snowflake design that is easy to understand and relate to business needs, supports simplified business queries, and provides superior query performance by minimizing table joins.

For example, contrast the very simplified OLTP data model in the first diagram below with the data warehouse dimensional model in the second diagram. Which one better supports the ease of developing reports and simple, efficient summarization queries?

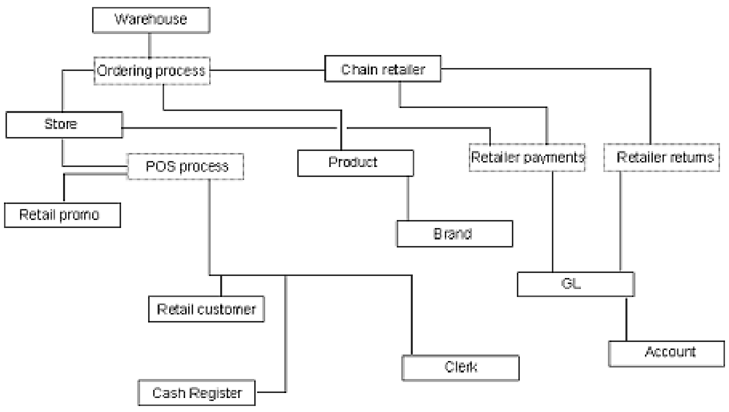

Figure 2. Flow Chart (click for larger image)

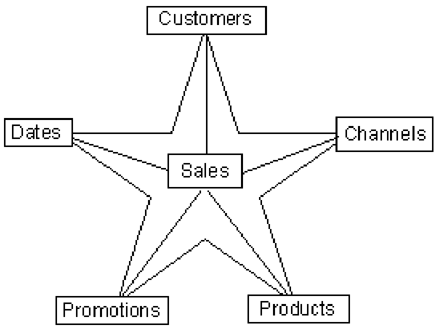

Figure 3. Star Diagram

Back to top

Dimensional Model Schemas

The principal characteristic of a dimensional model is a set of detailed business facts surrounded by multiple dimensions that describe those facts. When realized in a database, the schema for a dimensional model contains a central fact table and multiple dimension tables. A dimensional model may produce a *star schema* or a *snowflake schema*.

Star Schemas

A schema is called a *star schema* if all dimension tables can be joined directly to the fact table. The following diagram shows a classic star schema.

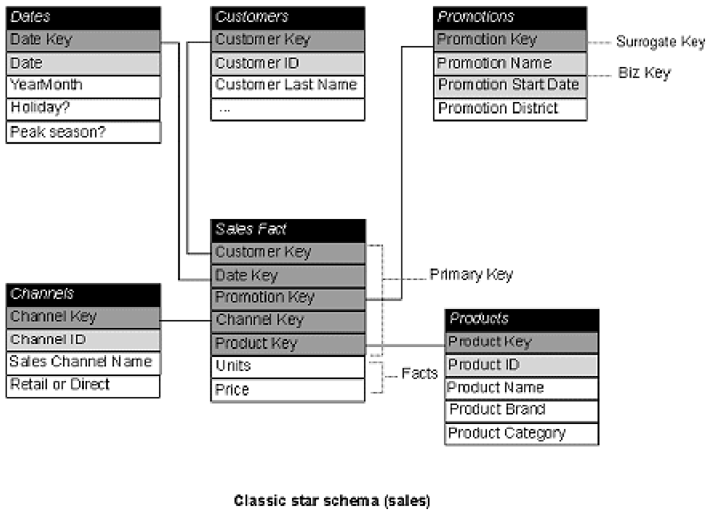

Figure 4. Classic star schema, sales (click for larger image)

The following diagram shows a clickstream star schema.

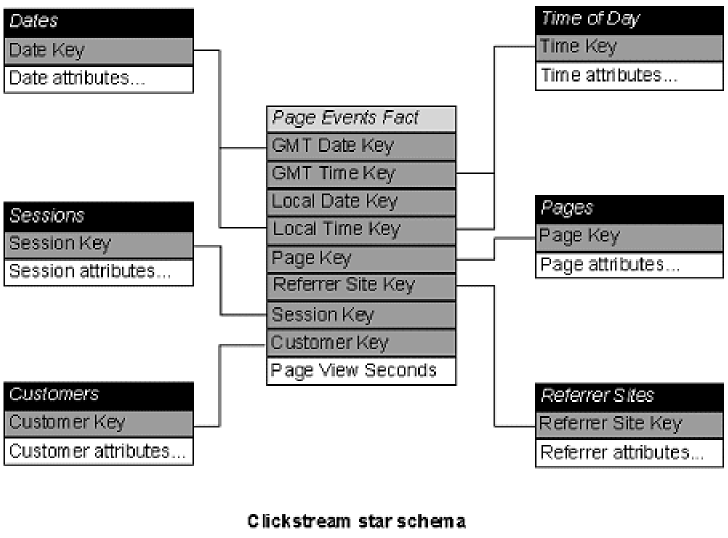

Figure 5. Clickstream star schema (click for larger image)

Snowflake Schemas

A schema is called a *snowflake schema* if one or more dimension tables do not join directly to the fact table but must join through other dimension tables. For example, a dimension that describes products may be separated into three tables (*snowflaked*) as illustrated in the following diagram.

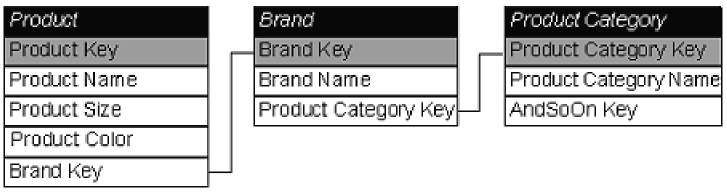

Figure 6. Snowflake, three tables (click for larger image)

A snowflake schema with multiple heavily snowflaked dimensions is illustrated in the following diagram.

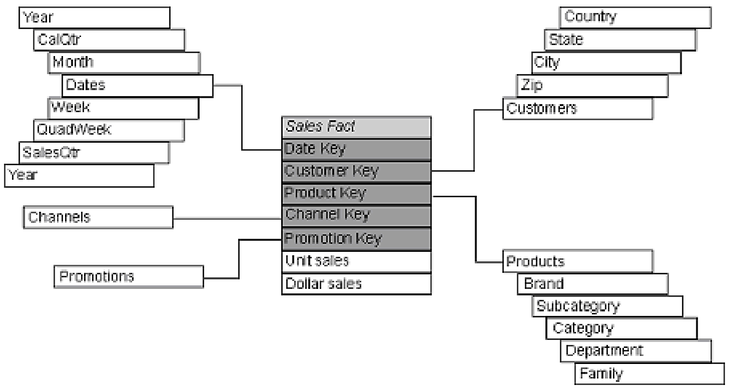

Figure 7. Many dimension snowflake (click for larger image)

Star or Snowflake

Both star and snowflake schemas are dimensional models; the difference is in their physical implementations. Snowflake schemas support ease of dimension maintenance because they are more normalized. Star schemas are easier for direct user access and often support simpler and more efficient queries. The decision to model a dimension as a star or snowflake depends on the nature of the dimension itself, such as how frequently it changes and which of its elements change, and often involves evaluating tradeoffs between ease of use and ease of maintenance. It is often easiest to maintain a complex dimension by snow flaking the dimension. By pulling hierarchical levels into separate tables, referential integrity between the levels of the hierarchy is guaranteed. Analysis Services reads from a snowflaked dimension as well as, or better than, from a star dimension. However, it is important to present a simple and appealing user interface to business users who are developing ad hoc queries on the dimensional database. It may be better to create a star version of the snowflaked dimension for presentation to the users. Often, this is best accomplished by creating an indexed view across the snowflaked dimension, collapsing it to a virtual star.

Back to top

Dimension Tables

Dimension tables encapsulate the attributes associated with facts and separate these attributes into logically distinct groupings, such as time, geography, products, customers, and so forth.

A dimension table may be used in multiple places if the data warehouse contains multiple fact tables or contributes data to data marts. For example, a product dimension may be used with a sales fact table and an inventory fact table in the data warehouse, and also in one or more departmental data marts. A dimension such as customer, time, or product that is used in multiple schemas is called a *conforming dimension* if all copies of the dimension are the same. Summarization data and reports will not correspond if different schemas use different versions of a dimension table. Using conforming dimensions is critical to successful data warehouse design.

User input and evaluation of existing business reports help define the dimensions to include in the data warehouse. A user who wants to see data "by sales region" and "by product" has just identified two dimensions (geography and product). Business reports that group sales by salesperson or sales by customer identify two more dimensions (salesforce and customer). Almost every data warehouse includes a time dimension.

In contrast to a fact table, dimension tables are usually small and change relatively slowly. Dimension tables are seldom keyed to date.

The records in a dimension table establish one-to-many relationships with the fact table. For example, there may be a number of sales to a single customer, or a number of sales of a single product. The dimension table contains attributes associated with the dimension entry; these attributes are rich and user-oriented textual details, such as product name or customer name and address. Attributes serve as report labels and query constraints. Attributes that are coded in an OLTP database should be decoded into descriptions. For example, product category may exist as a simple integer in the OLTP database, but the dimension table should contain the actual text for the category. The code may also be carried in the dimension table if needed for maintenance. This denormalization simplifies and improves the efficiency of queries and simplifies user query tools. However, if a dimension attribute changes frequently, maintenance may be easier if the attribute is assigned to its own table to create a snowflake dimension.

It is often useful to have a pre-established "no such member" or "unknown member" record in each dimension to which orphan fact records can be tied during the update process. Business needs and the reliability of consistent source data will drive the decision as to whether such placeholder dimension records are required.

Hierarchies

The data in a dimension is usually hierarchical in nature. Hierarchies are determined by the business need to group and summarize data into usable information. For example, a time dimension often contains the hierarchy elements: (all time), Year, Quarter, Month, Day, or (all time), Year Quarter, Week, Day. A dimension may contain multiple hierarchies—a time dimension often contains both calendar and fiscal year hierarchies. Geography is seldom a dimension of its own; it is usually a hierarchy that imposes a structure on sales points, customers, or other geographically distributed dimensions. An example geography hierarchy for sales points is: (all), Country or Region, Sales-region, State or Province, City, Store.

Note that each hierarchy example has an "(all)" entry such as (all time), (all stores), (all customers), and so forth. This top-level entry is an artificial category used for grouping the first-level categories of a dimension and permits summarization of fact data to a single number for a dimension. For example, if the first level of a product hierarchy includes product line categories for hardware, software, peripherals, and services, the question "What was the total amount for sales of all products last year?" is equivalent to "What was the total amount for the combined sales of hardware, software, peripherals, and services last year?" The concept of an "(all)" node at the top of each hierarchy helps reflect the way users want to phrase their questions. OLAP tools depend on hierarchies to categorize data—Analysis Services will create by default an "(all)" entry for a hierarchy used in a cube if none is specified.

A hierarchy may be balanced, unbalanced, ragged, or composed of parent-child relationships such as an organizational structure. For more information about hierarchies in OLAP cubes, see SQL Server Books Online.

Surrogate Keys

A critical part of data warehouse design is the creation and use of surrogate keys in dimension tables. A surrogate key is the primary key for a dimension table and is independent of any keys provided by source data systems. Surrogate keys are created and maintained in the data warehouse and should not encode any information about the contents of records; automatically increasing integers make good surrogate keys. The original key for each record is carried in the dimension table but is not used as the primary key. Surrogate keys provide the means to maintain data warehouse information when dimensions change. Special keys are used for date and time dimensions, but these keys differ from surrogate keys used for other dimension tables.

GUID and IDENTITY Keys

Avoid using GUIDs (globally unique identifiers) as keys in the data warehouse database. GUIDs may be used in data from distributed source systems, but they are difficult to use as table keys. GUIDs use a significant amount of storage (16 bytes each), cannot be efficiently sorted, and are difficult for humans to read. Indexes on GUID columns may be relatively slower than indexes on integer keys because GUIDs are four times larger. The Transact-SQL NEWID function can be used to create GUIDs for a column of **uniqueidentifier** data type, and the ROWGUIDCOL property can be set for such a column to indicate that the GUID values in the column uniquely identify rows in the table, but uniqueness is not enforced.

Because a **uniqueidentifier** data type cannot be sorted, the GUID cannot be used in a GROUP BY statement, nor can the occurrences of the **uniqueidentifier** GUID be distinctly counted—both GROUP BY and COUNT DISTINCT operations are very common in data warehouses. The **uniqueidentifier** GUID cannot be used as a measure in an Analysis Services cube.

The IDENTITY property and IDENTITY function can be used to create identity columns in tables and to manage series of generated numeric keys. IDENTITY functionality is more useful in surrogate key management than **uniqueidentifier** GUIDs.

Back to top

Date and Time Dimensions

Each event in a data warehouse occurs at a specific date and time; and data is often summarized by a specified time period for analysis. Although the date and time of a business fact is usually recorded in the source data, special date and time dimensions provide more effective and efficient mechanisms for time-oriented analysis than the raw event time stamp. Date and time dimensions are designed to meet the needs of the data warehouse users and are created within the data warehouse.

A date dimension often contains two hierarchies: one for calendar year and another for fiscal year.

Time Granularity

A date dimension with one record per day will suffice if users do not need time granularity finer than a single day. A date by day dimension table will contain 365 records per year (366 in leap years).

A separate time dimension table should be constructed if a fine time granularity, such as minute or second, is needed. A time dimension table of one-minute granularity will contain 1,440 rows for a day, and a table of seconds will contain 86,400 rows for a day. If exact event time is needed, it should be stored in the fact table.

When a separate time dimension is used, the fact table contains one foreign key for the date dimension and another for the time dimension. Separate date and time dimensions simplify many filtering operations. For example, summarizing data for a range of days requires joining only the date dimension table to the fact table. Analyzing cyclical data by time period within a day requires joining just the time dimension table. The date and time dimension tables can both be joined to the fact table when a specific time range is needed.

For hourly time granularity, the hour breakdown can be incorporated into the date dimension or placed in a separate dimension. Business needs influence this design decision. If the main use is to extract contiguous chunks of time that cross day boundaries (for example 11/24/2000 10 p.m. to 11/25/2000 6 a.m.), then it is easier if the hour and day are in the same dimension. However, it is easier to analyze cyclical and recurring daily events if they are in separate dimensions. Unless there is a clear reason to combine date and hour in a single dimension, it is generally better to keep them in separate dimensions.

Back to top

Date and Time Dimension Attributes

It is often useful to maintain attribute columns in a date dimension to provide additional convenience or business information that supports analysis. For example, one or more columns in the time-by-hour dimension table can indicate peak periods in a daily cycle, such as meal times for a restaurant chain or heavy usage hours for an Internet service provider. Peak period columns may be Boolean, but it is better to "decode" the Boolean yes/no into a brief description, such as "peak"/"offpeak". In a report, the decoded values will be easier for business users to read than multiple columns of "yes" and "no".

These are some possible attribute columns that may be used in a date table. Fiscal year versions are the same, although values such as quarter numbers may differ.

Column name	Data type	Format/Example	Comment
date_key	**int**	yyyymmdd	
day_date	**smalldatetime**		
day_of_week	**char**	Monday	
week_begin_date	**smalldatetime**		
week_num	**tinyint**	1 to 52 or 53	Week 1 defined by business rules
month_num	**tinyint**	1 to 12	
month_name	**char**	January	
month_short_name	**char**	Jan	
month_end_date	**smalldatetime**		Useful for days in the month
days_in_month	**tinyint**		Alternative for, or in addition to month_end_date
yearmo	**int**	yyyymm	
quarter_num	**tinyint**	1 to 4	
quarter_name	**char**	1Q2000	
year	**smallint**		
weekend_ind	**bit**		Indicates weekend
workday_ind	**bit**		Indicates work day
weekend_weekday	**char**	weekend	Alternative for **weekend_ind** and **weekday_ind**. Can be used to make reports more readable.
holiday_ind	**bit**		Indicates holiday
holiday_name	**char**	Thanksgiving	
peak_period_ind	**bit**		Meaning defined by business rules

Date and Time Dimension Keys

In contrast to surrogate keys used in other dimension tables, date and time dimension keys should be "smart." A suggested key for a date dimension is of the form "yyyymmdd". This format is easy for users to remember and incorporate into queries. It is also a recommended surrogate key format for fact tables that are partitioned into multiple tables by date.

Back to top

Slowly Changing Dimensions

A characteristic of dimensions is that dimension data is relatively stable—data may be added as new products are released or customers are acquired, but data, such as the names of existing products and customers, changes infrequently. However, business events do occur that cause dimension attributes to change, and the effects of these changes on the data warehouse must be managed. Of particular concern is the potential effect of a change to a dimension attribute on how historical data is tracked and summarized. "Slowly changing dimensions" is the customary term used for discussions of issues associated with the impact of changes to dimension attributes. Design approaches to dealing with the issues of slowly changing dimensions are commonly categorized into the following three change types:

- Type 1: Overwrite the dimension record.
- Type 2: Add a new dimension record.
- Type 3: Create new fields in the dimension record.

Type 1

Type 1 changes cause history to be rewritten, which may affect analysis results if an attribute is changed that is used to group data for summarization. Changes to a dimension attribute that is never used for analysis can be managed by simply changing the data to the new value. For example, if customer addresses are stored in the customer dimension table, a change to a customer's apartment number is unlikely to affect any summarized information, but a customer's move to a new city or state would affect summarization of data by customer location.

A Type 1 change is the easiest kind of slowly changing dimension to manage in the relational data warehouse. The maintenance procedure is simply to update an attribute column in the dimension table. However, the Type 1 slowly changing dimension presents complex management problems for aggregate tables and OLAP cubes. This is especially true if the updated attribute is a member of a hierarchy on which aggregates are precomputed, either in the relational database or the OLAP store.

For business users, a Type 1 change can hide valuable information. By updating the attribute in the dimension table, the prior value history of the attribute's value is lost. Consider the example where a customer has upgraded from the "Silver" to the "Gold" level of service. If the dimension table simply updates the attribute value, business users will not easily be able to explore differences of behavior before, during, and after the change of service level. In many cases, these questions are of tremendous importance to the business.

Type 2

Type 2 changes cause history to be partitioned at the event that triggered the change. Data prior to the event continues to be summarized and analyzed as before; new data is summarized and analyzed in accordance with the new value of the data.

Consider this example: In a sales organization, salespeople receive commissions on their sales. These commissions influence the commissions of sales managers and executives, and are summarized by sales group in standard reports. When a salesperson transfers from one group in the organization to another group, the historical information about commission amounts must remain applicable to the original group and new commissions must apply to the salesperson's new group. In addition, the total lifetime commission history for the employee must remain available regardless of the number of groups in which the person worked. A Type 1 change is not appropriate because it would move all of the salesperson's commission history to the new group.

The Type 2 solution is to retain the existing salesperson's dimension record and add a new record for the salesperson that contains the new reporting information. The original record still associates historical commission data with the previous sales group and the new record associates new commission data with the new group. It is customary to include fields in the dimension table to document the change. The following are examples of some common fields that can be used to document the change event:

- "Row Current", a Boolean field that identifies which record represents the current status
- "Row Start", a date field that identifies the date the record was added
- "Row Stop", a date field that identifies the date the record ceased to be current

Surrogate keys on the dimension table are required for Type 2 solutions. The salesperson's employee number is most likely used as the record key in OLTP systems. Even if some other key is used, it is unlikely that OLTP systems will need to create a new record for this individual. A second record for this individual cannot be created in the dimension table unless a different value is used for its primary key—surrogate keys avoid this restriction. In addition, because the salesperson's employee number is carried in the dimension table as an attribute, a summarization of the entire employee's commission history is possible, regardless of the number of sales groups to which the person has belonged.

Some queries or reports may be affected by Type 2 changes. In the salesperson example, existing reports that summarize by dimension records will now show two entries for the same salesperson. This may not be what is desired, and the report query will have to be modified to summarize by employee number instead of by the surrogate key.

Type 3

Type 3 solutions attempt to track changes horizontally in the dimension table by adding fields to contain the old data. Often only the original and current values are retained and intermediate values are discarded. The advantage of Type 3 solutions is the avoidance of multiple dimension records for a single entity. However, the disadvantages are history perturbation and complexity of queries that need to access the additional fields. Type 2 solutions can address all situations where Type 3 solutions can be used, and many more as well. If the data warehouse is designed to manage slowly changing dimensions using Type 1 and 2 solutions, there is no need to add the maintenance and user complexity inherent in Type 3 solutions.

Back to top

Rapidly Changing Dimensions, or Large Slowly Changing Dimensions

A dimension is considered to be a rapidly changing dimension if one or more of its attributes changes frequently in many rows. For a rapidly changing dimension, the dimension table can grow very large from the application of numerous Type 2 changes. The terms "rapid" and "large" are relative, of course. For example, a customer table with 50,000 rows and an average of 10 changes per customer per year will grow to about five million rows in 10 years, assuming the number of customers does not grow. This may be an acceptable growth rate. On the other hand, only one or two changes per customer per year for a ten million row customer table will cause it to grow to hundreds of millions of rows in ten years.

Tracking bands can be used to reduce the rate of change of many attributes that have continuously variable values such as age, size, weight, or income. For example, income can be categorized into ranges such as [0-14,999], [15,000-24,999], [25,000-39,999], and so on, which reduce the frequency of change to the attribute. Although Type 2 change records should not be needed to track age, age bands are often used for other purposes, such as analytical grouping. Birth date can be used to calculate exact age when needed. Business needs will determine which continuously variable attributes are suitable for converting to bands.

Often, the correct solution for a dimension with rapidly changing attributes is to break the offending attributes out of the dimension and create one or more new dimensions. Consider the following example.

An important attribute for customers might be their account status (good, late, very late, in arrears, suspended), and the history of their account status. Over time many customers will move from one of these states to another. If this attribute is kept in the customer dimension table and a Type 2 change is made each time a customer's status changes, an entire row is added only to track this one attribute. The solution is to create a separate account_status dimension with five members to represent the account states.

A foreign key in the customer table points to the record in the account_status dimension table that represents the current account status of that customer. A Type 1 change is made to the customer record when the customer's account status changes. The fact table also contains a foreign key for the account_status dimension. When a new fact record is loaded into the fact table, the customer id in the incoming fact record is used to look up the current account_table key in the customer record and populate it into the fact record. This captures a customer's account history in the fact table. In addition to the benefit of removing the rapidly changing item from the customer dimension, the separate account status dimension enables easy pivot analysis of customers by current account status in OLAP cubes. However, to see the entire account history for a customer, the fact table must be joined to the customer table and the account_status table and then filtered on customer id, which is not very efficient for frequent queries for a customer's account history.

This scenario works reasonably well for a single rapidly changing attribute. What if there are ten or more rapidly changing attributes? Should there be a separate dimension for each attribute? Maybe, but the number of dimensions can rapidly get out of hand and the fact table can end up with a large number of foreign keys. One approach is to combine several of these mini-dimensions into a single physical dimension. This is the same technique used to create what is often called a "junk" dimension that contains unrelated attributes and flags to get them out of the fact table. However, it is still difficult to query these customer attributes well because the fact table must be involved to relate customers to their attributes. Unfortunately, business users are often very interested in this kind of historical information, such as the movement of a customer through the various account status values.

If business users frequently need to query a dimension that has been broken apart like this, the best solution is to create a "factless" schema that focuses on attribute changes. For example, consider a primary data warehouse schema that keeps track of customers' purchases. The Customer dimension has been developed as a Type 2 slowly changing dimension, and account status has been pulled out into a separate dimension. Create a new fact table, CustomerChanges, that tracks only the changes to the customer and account status. A sample schema is illustrated in the following figure.

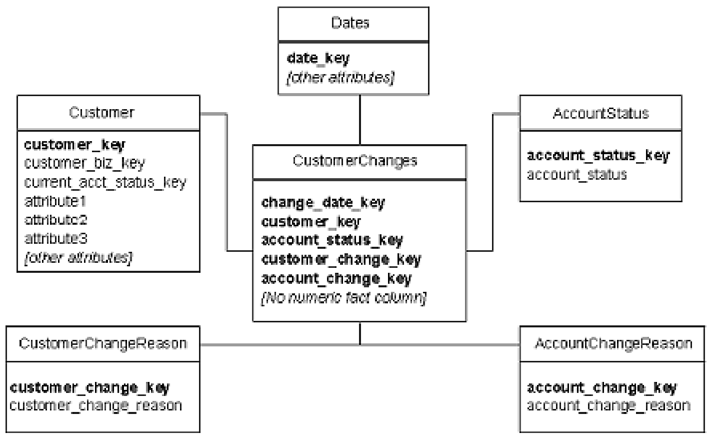

Figure 8. Customer changes schema (click for larger image)

The fact table, CustomerChanges, receives a new row only when a change is made to the Customer table that includes information about the customer's current account status. The fact table has no numeric measure or fact; an entry in the table signifies that an interesting change has occurred to the customer. Optionally, the CustomerChanges schema can track the reason for the change in the CustomerChangeReason and AccountChangeReason dimension tables. Sample values for the account_change_reason might include "Customer terminated account", "Account closed for non-payment", and "Outstanding balance paid in full".

Attribute history tables like this are neither dimension tables nor fact tables in the usual sense. The information in this kind of table is something like Quantity on Hand in an inventory fact table, which cannot be summarized by adding. However, unlike Quantity on Hand in an inventory table, these attributes do not change on a fixed periodic basis, so they cannot be numerically quantified and meaningfully averaged unless the average is weighted by the time between events.

Back to top

Multi-Use Dimensions

Sometimes data warehouse design can be simplified by combining a number of small, unrelated dimensions into a single physical dimension, often called a "junk" dimension. This can greatly reduce the size of the fact table by reducing the number of foreign keys in fact table records. Often the combined dimension will be pre-populated with the Cartesian product of all dimension values. If the number of discrete values creates a very large table of all possible value combinations, the table can be populated with value combinations as they are encountered during the load or update process.

A common example of a multi-use dimension is a dimension that contains customer demographics selected for reporting standardization. Another multiuse dimension might contain useful textual comments that occur infrequently in the source data records; collecting these comments in a single dimension removes a sparse text field from the fact table and replaces it with a compact foreign key.

Back to top

Fact Tables

A fact table must address the business problem, business process, and needs of the users. Without this information, a fact table design may overlook a critical piece of data or incorporate unused data that unnecessarily adds to complexity, storage space, and processing requirements.

Fact tables contain business event details for summarization. Fact tables are often very large, containing hundreds of millions of rows and consuming hundreds of gigabytes or multiple terabytes of storage. Because dimension tables contain records that describe facts, the fact table can be reduced to columns for dimension foreign keys and numeric fact values. Text, blobs, and de-normalized data are typically not stored in the fact table.

Multiple Fact Tables

Multiple fact tables are used in data warehouses that address multiple business functions, such as sales, inventory, and finance. Each business function should have its own fact table and will probably have some unique dimension tables. Any dimensions that are common across the business functions must represent the dimension information in the same way, as discussed earlier in "Dimension Tables." Each business function will typically have its own schema that contains a fact table, several conforming dimension tables, and some dimension tables unique to the specific business function. Such business-specific schemas may be part of the central data warehouse or implemented as data marts.

Very large fact tables may be physically partitioned for implementation and maintenance design considerations. The partition divisions are almost always along a single dimension, and the time dimension is the most common one to use because of the historical nature of most data warehouse data. If fact tables are partitioned, OLAP cubes are usually partitioned to match the partitioned fact table segments for ease of maintenance. Partitioned fact tables can be viewed as one table with an SQL UNION query as long as the number of tables involved does not exceed the limit for a single query. For more information about partitioning fact tables and OLAP cubes, see Chapter 18, "Using Partitions in a SQL Server 2000 Data Warehouse," in the *SQL Server 2000 Resource Kit*.

Back to top

Additive and Non-additive Measures

The values that quantify facts are usually numeric, and are often referred to as *measures*. Measures are typically additive along all dimensions, such as Quantity in a sales fact table. A sum of Quantity by customer, product, time, or any combination of these dimensions results in a meaningful value.

Some measures are not additive along one or more dimensions, such as Quantity-on-Hand in an inventory system or Price in a sales system. Some measures can be added along dimensions other than the time dimension; such measures are sometimes referred to as semiadditive. For example, Quantity-on-Hand can be added along the Warehouse dimension to achieve a meaningful total of the quantity of items on hand in all warehouses at a specific point in time. Along the time dimension, however, an aggregate function, such as Average, must be applied to provide meaningful information about the quantity of items on hand. Measures that cannot be added along any dimension are truly nonadditive. Queries, reports, and applications must evaluate measures properly according to their summarization constraints.

Nonadditive measures can often be combined with additive measures to create new additive measures. For example, Quantity times Price produces Extended Price or Sale Amount, an additive value.

Back to top

Calculated Measures

A calculated measure is a measure that results from applying a function to one or more measures, for example, the computed Extended Price value resulting from multiplying Quantity times Price. Other calculated measures may be more complex, such as profit, contribution to margin, allocation of sales tax, and so forth.

Calculated measures may be precomputed during the load process and stored in the fact table, or they may be computed on the fly as they are used. Determination of which measures should be precomputed is a design consideration. There are other considerations in addition to the usual tradeoff between storage space and computational time. The ability of SQL to state complex relationships and expressions is not as powerful as that of MDX, so complex calculated measures are more likely to be candidates for pre-computing if they are accessed using SQL than if they are accessed through Analysis Services using MDX.

Fact Table Keys

The logical model for a fact table contains a foreign key column for the primary keys of each dimension. The combination of these foreign keys defines the primary key for the fact table. Physical design considerations, such as fact table partitioning, load performance, and query performance, may indicate a different structure for the fact table primary key than the composite key that is in the logical model. These considerations are discussed in the section "Design the Relational Database and OLAP Cubes." For more information about partitioning fact tables, see Chapter 18, "Using Partitions in a SQL Server 2000 Data Warehouse," in the *SQL Server 2000 Resource Kit*.

A fact table resolves many-to-many relationships between dimensions because dimension tables join through the fact table. For examples of fact tables, see the illustrations in "Dimensional Model Schemas" earlier in this paper.

Back to top

Granularity

The grain of the fact table is determined after the fact content columns have been identified. Granularity is a measure of the level of detail addressed by an individual entry in the fact table. Examples of grain include "at the Transaction level", "at the Line Item level", "Sales to each customer, by product, by month." As you can see, the grain for a fact table is closely related to the dimensions to which it links. Including only summarized records of individual facts will reduce the grain and size of a fact table, but the resulting level of detail must remain sufficient for the business needs.

Business needs, rather than physical implementation considerations, must determine the minimum granularity of the fact table. However, it is better to keep the data as granular as possible, even if current business needs do not require it—the additional detail might be critical for tomorrow's business analysis. Analysis Services is designed to rapidly and efficiently summarize detailed facts into OLAP cubes so highly granular fact tables impose no performance burden on user response time. Alternatively, the OLAP cubes can be designed to include a higher level of aggregation than the relational database. Fine-grained data allows the data mining functionality of Analysis Services to discover more interesting nuggets of information.

Do not mix granularities in the fact table. Do not add summary records to the fact table that include detail facts already in the fact table. Aggregation summary records, if used, must be stored in separate tables, one table for each level of granularity. Aggregation tables are automatically created by Analysis Services for OLAP cubes so there is no need to design, create, and manage them manually.

Care must also be taken to properly handle records from source systems that may contain summarized data, such as records for product orders, bills of lading, or invoices. An order typically contains totals of line items for products, shipping charges, taxes, and discounts. Line items are facts. Order totals are summarized facts—do not include both in the fact table. The order number should be carried as a field in the line item fact records to allow summarization by order, but a separate record for the order totals is not only unnecessary, including it will make the fact table almost unusable.

The most successful way to handle summary data like taxes and shipping charges is to get business users to define rules for allocating those amounts down to the detailed level. For taxes, the rule already exists and is easy to implement. By contrast, shipping charges may be allocated by weight, product value, or some more arcane formula. It is common for business users to resist providing an allocation scheme that they may view as arbitrary. It is important for the data warehouse designers to push on this point, as the resulting schema is generally much more useful and usable.

Business needs, rather than physical implementation considerations, must determine the minimum granularity of the fact table.

However, it is better to keep the data as granular as possible, even if current business needs do not require it—the additional detail might be critical for tomorrow's business analysis. Analysis Services is designed to rapidly and efficiently summarize detailed facts into OLAP cubes so highly granular fact tables impose no performance burden on user response time. More detail also allows the data mining functionality of Analysis Services to discover more interesting nuggets of information.

Back to top

Develop the Architecture

The data warehouse architecture reflects the dimensional model developed to meet the business requirements. Dimension design largely determines dimension table design, and fact definitions determine fact table design.

Whether to create a star or snowflake schema depends more on implementation and maintenance considerations than on business needs. Information can be presented to the user in the same way regardless of whether a dimension is snowflaked. Data warehouse schemas are quite simple and straightforward, in contrast to OLTP database schemas with their hundreds or thousands of tables and relationships. However, the quantity of data in data warehouses requires attention to performance and efficiency in their design.

Design for Update and Expansion

Data warehouse architectures must be designed to accommodate ongoing data updates, and to allow for future expansion with minimum impact on existing design. Fortunately, the dimensional model and its straightforward schemas simplify these activities. Records are added to the fact table in periodic batches, often with little effect on most dimensions. For example, a sale of an existing product to an existing customer at an existing store will not affect the product, customer, or store dimensions at all. If the customer is new, a new record is added to the customer dimension table when the fact record is added to the fact table. The historical nature of data warehouses means that records almost never have to be deleted from tables except to correct errors. Errors in source data are often detected in the extraction and transformation processes in the staging area and are corrected before the data is loaded into the data warehouse database.

The date and time dimensions are created and maintained in the data warehouse independent of the other dimension tables or fact tables—updating date and time dimensions may involve only a simple annual task to mechanically add the records for the next year.

The dimensional model also lends itself to easy expansion. New dimension attributes and new dimensions can be added, usually without affecting existing schemas other than by extension. Existing historical data should remain unchanged. Data warehouse maintenance applications will need to be extended, but well-designed user applications should still function although some may need to be updated to make use of the new information.

An entirely new schema can be added to a data warehouse without affecting existing functionality. A new business subject area can be added by designing and creating a fact table and any dimensions specific to the subject area. Existing dimensions can be reused without modification to maintain conformity throughout the entire warehouse. If a different, more aggregated, grain is used in a new subject area, dimensions may be reduced in size by eliminating fine-grained members, but the resulting dimension must still conform to the master dimension and must be maintained in conformance with it.

Analysis Services OLAP cubes can be extended to accommodate new dimensions by extending their schemas and reprocessing, or by creating new virtual cubes that contain the new dimensions and incorporate existing cubes without modification to them.

Back to top

Design the Relational Database and OLAP Cubes

In this phase, the star or snowflake schema is created in the relational database, surrogate keys are defined and primary and foreign key relationships are established. Views, indexes, and fact table partitions are also defined. OLAP cubes are designed that support the needs of the users.

Keys and Relationships

Tables are implemented in the relational database after surrogate keys for dimension tables have been defined and primary and foreign keys and their relationships have been identified. Primary/foreign key relationships should be established in the database schema. For an illustration of these relationships, see the sample star schema, Classic Sales, in "Dimension Model Schema," earlier in this paper.

The composite primary key in the fact table is an expensive key to maintain:

- The index alone is almost as large as the fact table.
- The index on the primary key is often created as a clustered index. In many scenarios a clustered primary key provides excellent query performance. However, all other indexes on the fact table use the large clustered index key. All indexes on the fact table will be large, the system will require significant additional storage space, and query performance may degrade.

As a result, many star schemas are defined with an integer surrogate primary key, or no primary key at all. We recommend that the fact table be defined using the composite primary key. Also create an IDENTITY column in the fact table that could be used as a unique clustered index, should the database administrator determine this structure would provide better performance.

Back to top

Indexes

Dimension tables must be indexed on their primary keys, which are the surrogate keys created for the data warehouse tables. The fact table must have a unique index on the primary key. There are scenarios where the primary key index should be clustered, and other scenarios where it should not. The larger the number of dimensions in the schema, the less beneficial it is to cluster the primary key index. With a large number of dimensions, it is usually more effective to create a unique clustered index on a meaningless IDENTITY column.

Elaborate initial design and development of index plans for end-user queries is not necessary with SQL Server 2000, which has sophisticated index techniques and an easy to use Index Tuning Wizard tool to tune indexes to the query workload. The SQL Server 2000 Index Tuning Wizard allows you to select and create an optimal set of indexes and statistics for a database without requiring an expert understanding of the structure of the database, the workload, or the internals of SQL Server. The wizard analyzes a query workload captured in a SQL Profiler trace or provided by an SQL script, and recommends an index configuration to improve the performance of the database.

The Index Tuning Wizard provides the following features and functionality:

- It can use the query optimizer to analyze the queries in the provided workload and recommend the best combination of indexes to support the query mix in the workload.
- It analyzes the effects of the proposed changes, including index usage, distribution of queries among tables, and performance of queries in the workload.
- It can recommend ways to tune the database for a small set of problem queries.
- It allows you to customize its recommendations by specifying advanced options, such as disk space constraints.

A recommendation from the wizard consists of SQL statements that can be executed to create new, more effective indexes and, if wanted, drop existing indexes that are ineffective. Indexed views are recommended on platforms that support their use. After the Index Tuning Wizard has suggested a recommendation, it can then be implemented immediately, scheduled as a SQL Server job, or executed manually at a later time.

The empirical tuning approach provided by the Index Tuning Wizard can be used frequently when the data warehouse is first implemented to develop the initial index set, and then employed periodically during ongoing operation to maintain indexes in tune with the user query workload.

SQL Server Books Online provides detailed discussions of indexes and the Index Tuning Wizard, and procedures for using the wizard to tune database indexes.

Back to top

Views

Views should be created for users who need direct access to data in the data warehouse relational database. Users can be granted access to views without having access to the underlying data. Indexed views can be used to improve performance of user queries that access data through views. Indexed views are discussed in depth in SQL Server Books Online.

View definitions should create column and table names that will make sense to business users. If Analysis Services will be the primary query engine to the data warehouse, it will be easier to create clear and consistent cubes from views with readable column names.

Design OLAP Cubes

OLAP cube design requirements will be a natural outcome of the dimensional model if the data warehouse is designed to support the way users want to query data. Effective cube design is addressed in depth in "Getting the Most Out of Analysis Services" in Chapter 22, "Cubes in the Real World," of the *SQL Server 2000 Resource Kit*.

Back to top

Develop the Operational Data Store

Some business problems are best addressed by creating a database designed to support tactical decision-making. The Operational Data Store (ODS) is an operational construct that has elements of both data warehouse and a transaction system. Like a data warehouse, the ODS typically contains data consolidated from multiple systems and grouped by subject area. Like a transaction system, the ODS may be updated by business users, and contains relatively little historical data.

A classic business case for an operational data store is to support the Customer Call Center. Call center operators have little need for broad analytical queries that reveal trends in customer behavior. Rather, their needs are more immediate: the operator should have up-to-date information about all transactions involving the complaining customer. This data may come from multiple source systems, but should be presented to the call center operator in a simplified and consolidated way.

Implementations of the ODS vary widely depending on business requirements. There are no strict rules for how the ODS must be implemented. A successful ODS for one business problem may be a replicated mirror of the transaction system; for another business problem a star schema will be most effective. Most effective operational data stores fall between those two extremes, and include some level of transformation and integration of data. It is possible to architect the ODS so that it serves its primary operational need, and also functions as the proximate source for the data warehouse staging process.

A detailed discussion of operational data store design and its implications for data warehouse staging, is beyond the scope of this paper.

Back to top

Develop the Data Maintenance Applications

The data maintenance applications, including extraction, transformation, and loading processes, must be automated, often by specialized custom applications. Data Transformation Services (DTS) in SQL Server 2000 is a powerful tool for defining many transformations. Other tools are Transact-SQL and applications developed using scripting such as Microsoft Visual Basic® Scripting Edition (VBScript) or Microsoft JScript®, or languages such as Visual Basic.

An extensive discussion of the extraction, transformation, and loading processes is provided in Chapter 19, "Data Extraction, Transformation, and Loading Techniques," in the *SQL Server 2000 Resource Kit*.

Develop Analysis Applications

The applications that support data analysis by the data warehouse users are constructed in this phase of data warehouse development.

OLAP cubes and data mining models are constructed using Analysis Services tools, and client access to analysis data is supported by the Analysis Server. Techniques for cube design, MDX, data mining, and client data access to Analysis Services data are covered in depth in the section "Getting the Most Out of Analysis Services" in Chapter 22 of the *SQL Server 2000 Resource Kit*.

Other analysis applications, such as Microsoft PivotTables®, predefined reports, Web sites, and digital dashboards, are also developed in this phase, as are natural language applications using English Query. Specialized third-party analysis tools are also acquired and implemented or installed. Details of these specialized applications are determined directly by user needs. Digital dashboards are discussed in Chapter 27, "Creating an Interactive Digital Dashboard" and Chapter 28, "A Digital Dashboard Browser for Analysis Services Meta Data," in the *SQL Server 2000 Resource Kit*.

Back to top

Test and Deploy the System

It is important to involve users in the testing phase. After initial testing by development and test groups, users should load the system with queries and use it the way they intend to after the system is brought on line. Substantial user involvement in testing will provide a significant number of benefits. Among the benefits are:

- Discrepancies can be found and corrected.
- Users become familiar with the system.
- Index tuning can be performed.

It is important that users exercise the system during the test phase with the kinds of queries they will be using in production. This can enable a considerable amount of empirical index tuning to take place before the system comes online. Additional tuning needs to take place after deployment, but starting with satisfactory performance is a key to success. Users who have participated in the testing and have seen performance continually improve as the system is exercised will be inclined to be supportive during the initial deployment phase as early issues are discovered and addressed.

Back to top

Conclusion

Businesses have collected operational data for years, and continue to accumulate ever-larger amounts of data at ever-increasing rates as transaction databases become more powerful, communication networks grow, and the flow of commerce expands. Data warehouses collect, consolidate, organize, and summarize this data so it can be used for business decisions.

Data warehouses have been used for years to support business decision makers. Data warehousing approaches and techniques are well established, widely adopted, successful, and not controversial. Dimensional modeling, the foundation of data warehouse design, is not an arcane art or science; it is a mature methodology that organizes data in a straightforward, simple, and intuitive representation of the way business decision makers want to view and analyze their data.

The key to data warehousing is data design. The business users know what data they need and how they want to use it. Focus on the users, determine what data is needed, locate sources for the data, and organize the data in a dimensional model that represents the business needs. The remaining tasks flow naturally from a well-designed model—extracting, transforming, and loading the data into the data warehouse, creating the OLAP and data mining analytical applications, developing or acquiring end-user tools, deploying the system, and tuning the system design as users gain experience.

Microsoft SQL Server 2000 provides a wide range of powerful and easy to use tools you can use to create a data warehouse and analyze the data it contains. The ability to design and create data warehouses is no longer isolated to experts working with primitive implements.

Back to top