**Legal Notice:** This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of Datamola.

U1M6.Star Schema Basics

#### Kiryl Bucha

Contents

[1. The Relational and Object Data Models 4](#_Toc43833503)

[1.1. The Relational Data Model 4](#_Toc43833504)

[1.2. The Object Data Model 4](#_Toc43833505)

[1.2.1. Data Warehouses—Why Not the Object Model? 5](#_Toc43833506)

[1.3. The Dimensional Data Model 6](#_Toc43833507)

[1.4. Star Schema 7](#_Toc43833508)

[1.5. Snowflake Schema 9](#_Toc43833509)

[2. Developing the Model 9](#_Toc43833510)

[2.1. Step 1: Select the Data of Interest 10](#_Toc43833511)

[2.1.1. Business Data Model 11](#_Toc43833512)

[2.1.2. The scope document 11](#_Toc43833513)

[2.1.3. Information Requirements 11](#_Toc43833514)

[2.1.4. Existing Reports and Queries 12](#_Toc43833515)

[2.1.5. Prototype 12](#_Toc43833516)

[2.1.6. Source Data 13](#_Toc43833517)

[2.2. Step 2: Selection Process 13](#_Toc43833518)

[2.2.1. Use of Data Element for a Derived Field 13](#_Toc43833519)

[2.2.2. Classification of Data as Transactional or Reference 14](#_Toc43833520)

[2.2.3. Source Data Structure 15](#_Toc43833521)

[2.3. Add Time to the Key 16](#_Toc43833522)

[2.3.1. Capturing Historical Data 16](#_Toc43833523)

[2.3.2. Capturing Historical Relationships 18](#_Toc43833524)

[2.4. Step 3: Add Derived Data 19](#_Toc43833525)

[2.5. Step 4: Determine Granularity Level 20](#_Toc43833526)

[2.6. Step 5: Summarize Data 23](#_Toc43833527)

[2.6.1. Summaries for Period of Time Data 23](#_Toc43833528)

[2.6.2. Summaries for Snapshot Data 24](#_Toc43833529)

[2.6.3. Vertical Summary 24](#_Toc43833530)

[2.7. Step 6: Merge Entities 25](#_Toc43833531)

[2.8. Step 7: Create Arrays 26](#_Toc43833532)

[2.9. Step 8: Segregate Data 27](#_Toc43833533)

[3. Summary 28](#_Toc43833534)

[4. Source Books and Articles 29](#_Toc43833535)

# 1. The Relational and Object Data Models

## 1.1. The Relational Data Model

The relational model uses a sequence of steps called normalization in order to break information into its smallest divisible parts, removing duplication and creating granularity.

**NOTE:** For more details about normalization please read next sources:

* MTN.NIX.07.Oracle DB.DWH\_courseware01\_Introduction to Data Warehouse.docx (Chapter 3)
* Oracle® Database Data Warehousing Guide 10g Release 1 (10.1) Part No. B10736-01 (Chapter 1)

## 1.2. The Object Data Model

The object model is even less appropriate for a data warehouse than the relational model. There is a simple reason for this. The object model is excellent at handling immense complexity. Why? The objective of an object-oriented design approach is to break every part of the model into its smallest self-contained part. Unlike the relational model where dependencies exist between the different parts, the object model allows each part to be autonomous. This is what is meant by self-contained. In other words, there is much less inter-object dependency in the object model than in the relational model. That is even worse than a relational model when considering a data warehouse data model.

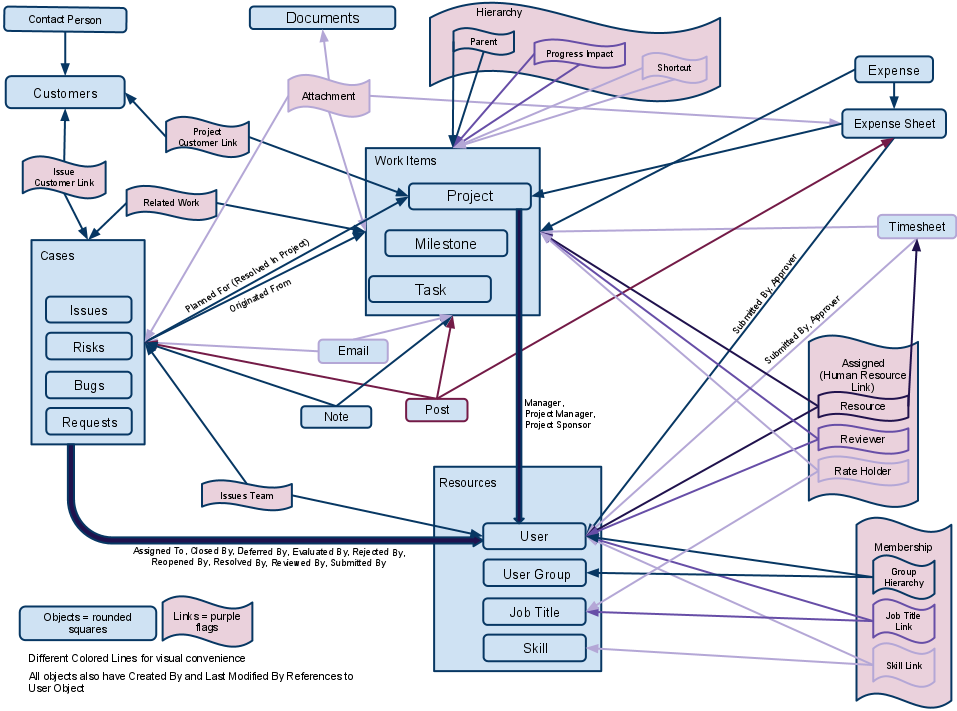
The object model is made up of the following structures, connections, and some of these commonly used terms, without getting too deeply specific about it:

* **Class** is the equivalent of a relational entity or table. It is important to understand that a class is not the same as an object.
* **Attribute** is equivalent to a column in a relational entity column or field.
* **Method** is a chunk of code or program executed exclusively on the contents of the object to which it is attached.
* **Inheritance,** Classes are linked together through an inheritance hierarchy.
* **Multiple Inheritance,** Multiple inheritance allows a class to inherit details from more than one class.
* **Specialization and Abstraction**, The result of inheritance is that classes can be both specialized and abstracted.
* **Collection** is the term applied to a repetition of elements of one object contained within another object.

### 1.2.1. Data Warehouses—Why Not the Object Model?

Why does the object model not cater for data warehouse requirements? The object model is inappropriate for a data warehouse because there is even more scope for granularity using the object model than with the relational

model. In a data warehouse logical granularity in the form of normalized relations or autonomous objects leads to inefficiencies due to a higher reliance on joins between entities, or in the case of the object model, entities become classes and objects.

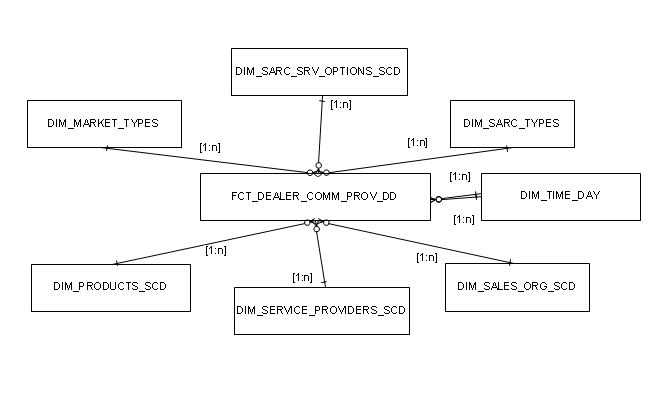


## 1.3. The Dimensional Data Model

An entity relationship model is inappropriate to the requirements of a data warehouse, even a denormalized one. Another modeling technique used for data warehouses is called dimensional modeling. In layman’s terms, a dimensional model consists of facts and dimensions. What is a fact and what is a dimension? A fact is a single iteration in a historical record. A dimension is something used to dig into, divide, and collate those facts into something useful. Facts are the equivalent of transactional entities and dimensions are the equivalent of static data.

Facts represent historical or archived data and dimensions represent smaller static data entities. It follows that dimension entities will generally be small and fact entities can become frighteningly huge. What does this tell us? Fact entities will always be appended to, and dimension entities can be changed, preferably not as often as the fact entities are appended to. The result is many very small entities related to data in groups from very large entities.

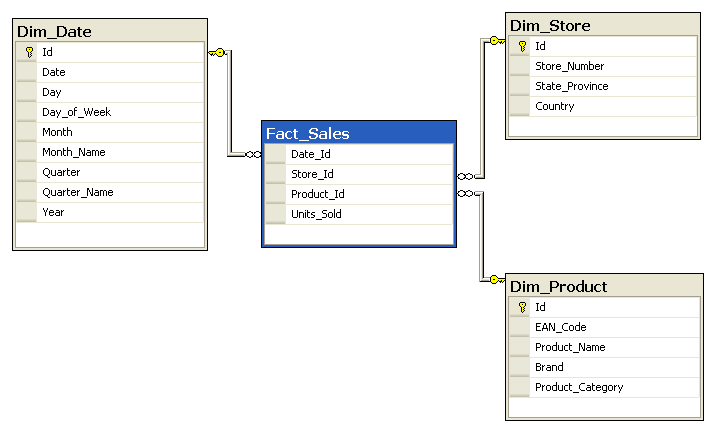
The most desirable result when modeling for a data warehouse using dimensions and facts is called a star schema. Show slightly modified pseudo-type star schema versions of the normalized entity relationship diagrams in figure below. Also we can see that all dimensions would be contained within a single fact entity, containing shipping history records of containers. Each row in the fact entity would have foreign key values to all related dimension entities.



## 1.4. Star Schema

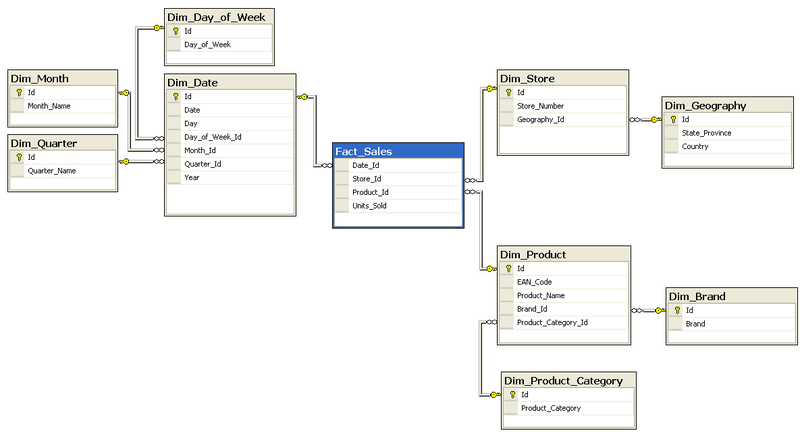
A star schema contains one, or at least very few, very large fact entities, plus a large number of small dimensional entities. As already stated, effectively fact entities contain transactional histories and dimension entities contain static data describing the fact entity archive entries. The objective for performance is to obtain joins on a single join level, where one fact entity is joined to multiple small dimension entities, or perhaps even a single dimension entity. Benefits from using Star scheme on DWH list below:

* Nontechnical users, such as sales people, forecasters, and sometimes even executives, often access data warehouse entities. A star schema is very simple to understand without bombarding the user with all the complex entities and intra-entity relationships of a relational data model and multiple dimensional hierarchies. Containers are meaningful to end users. Types of containers may not be as meaningful in a user’s mind, perhaps being understood better as refrigerated or flatbed. If users do not know what something is, it could potentially render the entire structure useless to some people.
* A star schema can easily be augmented by adding new dimensions, as long as they fit in with the fact entity.
* From a purely performance tuning perspective, a star schema rolls subset dimensions into single entities from a multiple dimensional hierarchy of a snowflake schema. The number of joins in queries will be reduced. Therefore, queries should execute faster.



## 1.5. Snowflake Schema

A snowflake schema is a normalized star schema such that dimension entities are normalized.



# 2. Developing the Model

The data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data to support strategic analysis. The major factors affecting the design of the data warehouse reflect its primary mission, which is to serve as a collection point for the needed data stored in various operational systems and as a distribution point for sending this data to the data marts. The major factors affecting the content of the data warehouse are the information needs of the people who will use the resultant data marts and the organization of the data in the source systems. Unlike the source systems that are built to support business processes, the data warehouse model needs to reflect the business relationships of the information, independent of the business processes and organization.

As explained earlier, the relational model, based on a third-normal form model that depicts the business relationships, best meets the needs for storage of data in the data warehouse. The third normal form model, in its pure form, however, is not the best structure for the data warehouse. Using the third normal form model for the data warehouse is analogous to selecting any screwdriver for the job. Just as the screwdriver should be based on the size of the screw being driven, the third normal form model needs to be adjusted to meet the data warehouse needs.

The data warehouse system data model is developed by applying an eight step transformation process to the business data model. The eight steps are:

1. Select the data of interest.
2. Add time to the key.
3. Add derived data.
4. Determine granularity level.
5. Summarize data.
6. Merge entities.
7. Create arrays.
8. Segregate data.

## 2.1. Step 1: Select the Data of Interest

The first step in developing the data warehouse is to select the data of interest. There are two major reasons for making this the first step. First, it places the purpose and business objectives of the data warehouse project in the foreground. All decisions made concerning the data warehouse model consider the business purpose and objectives. Second, this step confines the scope of the data warehouse model to just that needed in the project. Since this step is designed to serve as a funnel and eliminate consideration of data elements that are not needed in the data warehouse, it only makes sense to perform it before additional work is performed on those elements.

### 2.1.1. Business Data Model

A fully developed business data model provides an ideal inventory of the available data elements. The fully developed business data model usually contains hundreds, and possibly thousands, of data elements. The development team needs to quickly cull the elements to only those that must be included in this iteration of the data warehouse. When a business data model does not exist and the development team creates the business data model for the scope of the project, virtually all of the elements in that model are included in the data warehouse model. Since only the data elements considered necessary for the project were included in the business data model, the scope containment was performed by restricting the elements included in the business data model.

### 2.1.2. The scope document

The scope document sets the expectations for this iteration of the data warehouse. In addition to other information, it includes a section that identifies the data to be included, and may also include a section that delineates the data that is excluded from the iteration. In a way, this is a precursor to Step 1 in that the only data that needs to be considered in the model transformation is that data that could be considered to be within scope based on the level of detail contained in the scope document.

### 2.1.3. Information Requirements

Information requirements constitute the third set of inputs. There are several sources for the information requirements. Since the data warehouse should be aligned with the ultimate business goals, a review of available corporate planning documents and the annual report may prove useful. Facilitated sessions and interviews with business executives, analysts, and end users provide a major source of information requirements. These sessions are designed to get the business community to identify the specific business questions to be answered and the data elements that are needed to answer those questions. The data analyst should avoid asking the business person, “What do you want in the data warehouse?” Virtually all business users have a common answer to this question—everything!

### 2.1.4. Existing Reports and Queries

Existing reports and queries provide another source of information requirements, but these need to be used carefully. Just because a report is regularly produced and distributed does not mean that anyone uses it. Some reports were created to meet a specific need that no longer exists, but no one bothered to delete the report from the production schedule. Similarly, even if a report is used, it may contain some data that is not used. Data may be unused either because it was included “just in case,” but was never actually needed, or because it was needed when the report was created, but circumstances changed. So, even though an existing report contains extraneous data, its users may have deemed to be more expedient to leave the data in rather than try to justify the programming changes needed to generate a more streamlined report.

### 2.1.5. Prototype

Possibly the most effective way to identify the required data elements is by creating a prototype of the ultimate product. In the section dealing with information requirements, we indicated that a star schema may be drawn to help users visualize how they will receive the data. The prototype consists of deploying this design with actual data so that the users can verify its adequacy. A properly developed and managed prototype helps business users visualize the end result and enables them to better articulate the information needs. The prototype exercise consists of more than just providing the user with access to the mart and waiting for a reaction. A properly managed prototype exercise requires constant interaction between the data warehouse designer and the user and the incorporation of appropriate revisions to the design based on the feedback. Another critical aspect of a well-managed prototype is knowing when to stop. Remember, the objective of the prototype is to refine the requirements—not to provide an early production deliverable.

### 2.1.6. Source Data

Information about the anticipated data warehouse sources also provides useful information for selecting the data elements. The source system data structures provide information about how the data is physically stored within the systems used for day-to-day operation. These provide a checklist, and if a user has not requested data elements that are stored with other data elements of interest, the data analyst should consider asking additional questions to ensure that the additional elements are not needed. Once the elements that are needed are determined, the elements to be included can be selected. The next section introduces additional considerations that have an impact on this decision.

## 2.2. Step 2: Selection Process

### 2.2.1. Use of Data Element for a Derived Field

Often, users require a derived field, such as net sales amount, but do not require each of the elements that was used to calculate the derived field. As a general rule, we recommend that any element that is used in calculating a derived field be included in the data warehouse. (It need not be included in any data mart.) There are two major reasons for including the element. First, the algorithm used to calculate the needed element may change, and by retaining the individual components, the derived field can be recalculated for both current views and historical views if needed. Second, business users often drill down to the data used to calculate a needed field when they are analyzing results, particularly if the value of the derived element is unexpected. Even though this need is not identified initially, we should anticipate it.

### 2.2.2. Classification of Data as Transactional or Reference

Within the data warehouse, we are often interested in transactions reflect activity over a period of time, sometimes spanning several years. We need to recognize that users often can’t anticipate all of the data that they need from these transactions, and we’re then faced with deciding whether or not to bring additional elements into the data warehouse. As a general rule, with transactional data, if we are in doubt, we should bring it in. There are three major reasons for this, all of which deal with what it would take to retrieve the data should we later discover we need it:

1. The transaction may be purged from the source system. The data warehouse retains history beyond what is retained in the operational system. If the need for the data element is discovered after the data is purged from the source system, it can never be recovered.
2. The transactions occur over time. If the need for the data element is discovered months or years after the initial construction of the data warehouse, recovery of the data will require navigation through all the transactions that transpired during the intervening time. This can be a very significant effort, and complications may be introduced due to changes in reference data and the resultant complexities involved in ensuring referential integrity.
3. Transactional data integration is generally simple. Unlike reference data, such as customer data that may be gathered from multiple sources, individual transactions originate at only one place. Integration of transactional data entails adding transactions from multiple files and not merging of information within each transaction. This characteristic results in a simpler process, and hence the impact on the development project is minimized.

The major disadvantage of bringing in the questionable elements is the volume f data. If there are millions of transactions and dozens of questionable fields, the data warehouse tables will grow unnecessarily. An approach that some companies use is “triage,” in which all of the transactions from the source system are extracted and retained in an archived, offline, file.

### 2.2.3. Source Data Structure

The structure of the source data is another factor that should be considered. If most of the columns from a source table are required, then serious consideration should be given to including all of the elements. That approach simplifies the extraction, transformation, and load process. Mitigating factors to adding an element include perceptions concerning agreement on its definition and the quality of the actual data. (If there is significant disagreement on the definition or if the data quality is inadequate, then the analysis time and subsequent data acquisition process development are lengthened.) If only a few of the columns from a source table are needed, the tendency should be to exclude the remaining columns.

Keep in mind that the source data structure was designed based on operational needs, and tables often contain denormalized data. For example, the sales transaction in the operational system could contain the item description and the customer name. This data should not be obtained from the transaction table—it should be obtained from the reference table—unless the operational system provides users with the ability to override the standard reference data. The first step in creating the data warehouse model is complex, and it requires extensive business user involvement. The tendency of some data warehouse teams is to skip this step and just include all columns from any table that is used to provide data. Reasons for skipping this step include a perception that it will be easier to simply bring in all the data.

## 2.3. Add Time to the Key

The business data model is a “point-in-time” model. That is, the model portrays the business at the present. The data warehouse data model, on the other hand, is an “over-time” model. An “over-time” model portrays an enterprise with a historical perspective. Since the data warehouse is time variant (that is, it has a historical perspective or a series of snapshots), this is the type of model that is appropriate for the data warehouse.

### 2.3.1. Capturing Historical Data

Adding a time component to the key enables a distinct record to be created each time a new snapshot is taken and each time data for a new period is captured. Inclusion of this time component has a significant design implication. Since the entity may be the parent in some one-to-many relationships, the generated foreign key would now include the time component. It generates a new occurrence for every child, even if no data in the child entity changed. This happens because there is a change in the content of the time component of the foreign key. This condition is compounded if an entity is the child of several parent entities. Each of the parents generates a foreign key with its governing time period. This happens regardless of whether or not the change in the parent creates any change in the child. There are five potential approaches that can be chosen:

* Generate the dual foreign key, as previously described.
* Generate a serial key in the parent entity for each occurrence and store the identifier and time component as a nonkey attribute within the entity. This scenario reduces the number of attributes generated as foreign keys but still generates a new instance of the child each time there is a change in a parent entity.
* Programmatically enforce referential integrity rather than using the DBMS to enforce it, and only enforce referential integrity for the identifier. This approach requires additional programming effort to ensure referential integrity.
* Segregate the data into an entity that contains the history and another entity that only contains the current data. The relationships would only emanate from the table with the current data, and hence would not include the time component. The referential integrity for the historical data would be programmatically enforced by virtue of records being created as copies of records in the current data entity.
* Maintain the base entity with only data elements that can never change, and create an attributive entity with the attributes that could change over time. This approach enables the key of the base entity to consist solely of the identifier, and since the attributive entity has no children, the date component of the key does not cascade. (In the figure, the presumption is that the name never changes, but that the rank and square miles do.)

### 2.3.2. Capturing Historical Relationships

Relationships also change over time. The result of these changes is the creation of a many-to-many relationship. Within a third normal form model, predictable hierarchies are represented by a series of entities with successive oneto-many relationships While this may be true at a particular point in time, the historical perspective requires the model to handle situations in which a Sales Area may be moved from one Sales Territory to another and a Sales Territory may be moved from one Sales Region to another. This condition can be handled in one of two ways.

One way is to insert an associative entity to resolve the many-to-many relationship. This approach is useful if the historical information about the individual entities is handled by the fourth approach described in the previous section, and the fourth scenario described is expanded to include the associative entities on the right portion.

The approach to be taken for capturing historical relationships is dependent on two primary factors. One factor is the number of levels in the hierarchies. As the number of hierarchies increases, preference should be given to using the associative entity since a change in the highest level of the hierarchy generates a new entity for each subservient level. The second factor is the relative stability of the entity data versus the relationship data. If the entity data (for example, Sales Area description) is more stable than the relationship (for example, Sales Territory to which the Sales Area is assigned), then preference should be given to using the associative entity since few changes in the base entities would be needed. If the relationship data is more stable, then preference should be given to cascading the foreign key since the instances of cascading keys are minimized.

## 2.4. Step 3: Add Derived Data

The third step in developing the data warehouse model is to add derived data. Derived data is data that results from performing a mathematical operation on one or more other data elements. Derived data is incorporated into the data warehouse model for two major reasons—to ensure consistency, and to improve data delivery performance. The reason that this step is third is the business impact—to ensure consistency; performance benefits are secondary. One of the common objectives of a data warehouse is to provide data in a way so that everyone has the same facts—and the same understanding of those facts. A field such as “net sales amount” can have any number of meanings. Items that may be included or excluded in the definition include special discounts, employee discounts, and sales tax. If a sales representative is held accountable for meeting a sales goal, it is extremely important that everyone understands what is included and excluded in the calculation.

Creating a derived field does not usually save disk space since each of the components used in the calculation may still be stored, as noted in Step 1. Using derived data improves data delivery performance at the expense of load performance. When a derived field used in multiple data marts, calculating it during the load process reduces the burden on the data delivery process. Since most end-user access to data is done at the data mart level, another approach is to either calculate it during the data delivery process that builds the data marts or to calculate it in the end-user tool. If the derived field is needed to ensure consistency, preference should be given to storing it in the data warehouse. There are two major reasons for this:

* First, if the data is needed in several data marts, the derivation calculation is only performed once.
* The second reason is of great significance if end users can build their own data marts.

By including the derived data in the data warehouse, even when construction of the marts is distributed, all users retain the same definitions and derivation algorithms.

## 2.5. Step 4: Determine Granularity Level

The fourth step in developing the data warehouse model is to adjust the granularity, or level of detail, of the data warehouse. The granularity level is significant from a business, technical, and project perspective. From a business perspective, it dictates the potential capability and flexibility of the data warehouse, regardless of the initially deployed functions. Without a subsequent change to the granularity level, the warehouse will never be able to answer questions that require details below the adopted level. From a technical perspective, it is one of the major determinants of the data warehouse size and hence has a significant impact on its operating cost and performance. From a project perspective, the granularity level affects the amount of work that the project team will need to perform to create the data warehouse since as the granularity level gets into greater and greater levels of detail, the project team needs to deal with more data attributes and their relationships. Additionally, if the granularity level increases sufficiently, a relatively small data warehouse may become extremely large, and this requires additional technical considerations.

Some people have a tendency to establish the level of granularity based on the questions being asked. If this is done for a retail store for which the business users only requested information on hourly sales, then we would be collecting and summarizing data for each hour.

**Current business need.** The primary determining factor should be the business need. At a minimum, the level of granularity must be sufficient to provide answers to each and every business question being addressed within the scope of the data warehouse iteration. Providing a greater level of granularity adds to the cost of the warehouse and the development project and, if the business does not need the details, the increased costs add no business value.

**Anticipated business need.** The future business needs should also be considered. A common scenario is for the initial data warehouse implementation to focus on monthly data, with an intention to eventually obtain daily data. If only monthly data is captured, the company may never be able to obtain the daily granularity that is subsequently requested. Therefore, if the interview process reveals a need for daily data at some point in the future, it should be considered in the data warehouse design. The key word in the previous sentence is “considered” —before including the extra detail, the business representatives should be consulted to ensure that they perceive a future business value. As we described in Step 1, an alternate approach is to build the data warehouse for the data we know we need, but to build and extract data to accommodate future requirements.

**Extended business need.** Within any industry, there are many data warehouses already in production. Another determining factor for the level of granularity is to get information about the level of granularity that is typical for your industry. For example, in the retail industry, while there are a lot of questions that can be answered with data accumulated at an hourly interval, retailers often maintain data at the transactional level for other analyses. However, just because others in the industry capture a particular granularity level does not mean that it should be captured but the modeler and business representative should consider this in making the decision.

**Data mining need.** While the business people may not ask questions that require a display of detailed data, some data mining requests require significant details. For example, if the business would like to know which products sell with other products, analysis of individual transactions is needed.

**Derived data need.** Derived data uses other data elements in the calculation. Unless there is a substantial increase in cost and development time, the chosen granularity level should accommodate storing all of the elements used to derive other data elements.

**Data acquisition performance.** The level of granularity may (or may not) significantly impact the data acquisition performance. Even if the data warehouse granularity is summarized to a weekly level, the extract process may still need to include the individual transactions since that’s the way the data is stored in the source systems, and it may be easier to obtain data in that manner. During the data acquisition process, the appropriate granularity would be created for the data warehouse. If there is a significant difference in the data volume, the load process is impacted by the level of granularity, since that determines what needs to be brought into the data warehouse.

**Storage cost.** The level of granularity has a significant impact on cost. If a retailer has 1,000 stores and the average store has 1,500 sales transactions per day, each of which involves 10 items, a transaction-detail-level data warehouse would store 15,000,000 rows per day. If an average of 1,000 different products were sold in a store each day, a data warehouse that has a granularity level of store, product and day would have 1,000,000 rows per day.

**Administration.** The inclusion of additional detail in the data warehouse impacts the data warehouse administration activities as well. The production data warehouse needs to be periodically backed up and, if there is more detail, the backup routines require more time.

## 2.6. Step 5: Summarize Data

The fifth step in developing the data warehouse model is to create summarized data. The creation of the summarized data may not save disk space—it’s possible that the details that are used to create the summaries will continue to be maintained. It will, however, improve the performance of the data delivery process. The most common summarization criterion is time since data in the warehouse typically represents either a point in time (for example, the number of items in inventory at the end of the day) or a period of time (for example, the quantity of an item sold during a day). Some of the benefits that summarized data provides include reductions in the online storage requirements (details may be stored in alternate storage devices), standardization of analysis, and improved data delivery performance.

### 2.6.1. Summaries for Period of Time Data

Simple cumulations and rolling summaries apply to data that pertains to a period of time. Simple cumulations represent the summation of data over one of its attributes, such as time. For example, a daily sales summary provides a summary of all sales for the day across the common ways that people access it. If people often need to have sales quantity and amounts by day, sales person, store, and product, the summary table.

### 2.6.2. Summaries for Snapshot Data

The simple direct summary and continuous summary apply to snapshot data or data that is episodic, or pertains to a point in time. The simple direct file provides the value of the data of interest at regular time intervals. The continuous file generates a new record only when a value changes. Factors to consider for selecting between these two types of summaries are the data volatility and the usage pattern. For data that is destined to eventually migrate to a data mart that provides monthly information, the continuous file is a good candidate if the data is relatively stable. With the continuous file, there will be fewer records generated, but the data delivery algorithm will need to determine the month based on the effective (and possibly expiration) date. With the simple direct file, a new record is generated for each instance each and every month. For stable data, this creates extraneous records. If the data mart needs only a current view of the data in the dimension, then the continuous summary facilitates the data delivery process since the most current occurrence is used, and if the data is not very volatile and only the updated records are transferred, less data is delivered.

### 2.6.3. Vertical Summary

The last type of summarization—vertical summary—applies to both point in time and period of time data. For a dealer, point in time data would pertain to the inventory at the end of the month or the total number of customers, while period of time data applies to the sales during the month or the customers added during the month. In an E-R model, it would be a mistake to combine these into a single entity. If “month” is used as the key for the vertical summary and all of these elements are included in the entity, month has two meanings—a day in the month, and the entire month. If we separate the data into two tables, then the key for each table has only a single definition within its context.

Even though point-in-time and period-of-time data should not be mixed in a single vertical summary entity in the data warehouse, it is permissible to combine the data into a single fact table in the data mart. The data mart is built to provide ease of use and, since users often create calculations that combine the two types of data, (for example, sales revenue per customer for the month), it is appropriate to place them together. We combined sales information with inventory information into a single fact table. The meta data should clarify that, within the fact table, month is used to represent either the entire period for activity data such as sales, and the last day of the period (for example) for the snapshot information such as inventory level.

## 2.7. Step 6: Merge Entities

The sixth step in developing the data warehouse model is to merge entities by combining two or more entities into one. The original entities may still be retained. Merging the entities improves the data delivery process performance by reducing the number of joins, and also enhances consistency. Merging entities is a form of denormalizing data and, in its ultimate form, it entails the creation of conformed dimensions for subsequent use in the data mart.

The following criteria should exist before deciding to merge entities: The entities share a common key, data from the merged entities is often used together, and the insertion pattern is similar. The first condition is a prerequisite—if the data cannot be tied to the same key, it cannot be merged into a common entity since in an E-R model, all data within an entity depends on the key. The third condition addresses the load performance and storage. When the data is merged into a single entity, any time there is a change in any attribute, a new row is generated.

## 2.8. Step 7: Create Arrays

The seventh step in developing the data warehouse model is to create arrays. This step is rarely used but, when needed, it can significantly improve population of the data marts. Within the traditional business data model, repeating groups are represented by an attributive entity. For example, for accounts receivable information, if information is captured in each of five groupings (for example, current, 1–30 days past due, 31–60 days past due, 61–90 days past due, and over 90 days past due), this is an attributive entity. This could also be represented as an array, as shown in the right part of that figure. Since the objective of the data warehouse that the array is satisfying is to improve data delivery, this approach only makes sense if the data mart contains an array. In addition to the above example, another instance occurs when the business people want to look at data for the current week and data for each of the preceding 4 weeks in their analysis. The arrays are useful if all of the following conditions exist:

* The number of occurrences is relatively small. In the example cited above, there are five occurrences. Creating an array for sales at each of 50 regions would be inappropriate.
* The occurrences are frequently used together. In the example, when accounts receivable analysis is performed, people often look at the amount in each of the five categories together.
* The number of occurrences is predictable. In the example, there are always exactly five occurrences.
* The pattern of insertion and deletion is stable. In the example, all of the data is updated at the same time. Having an array of quarterly sales data would be inappropriate since the data for each of the quarters is inserted at a different time. In keeping with the data warehouse philosophy of inserting rows for data changes, there would actually be four rows by the end of the year, with null values in several of the rows for data that did not exist when the row was created.

## 2.9. Step 8: Segregate Data

The eighth step in developing the data warehouse model is to segregate data based on stability and usage. The operational systems and business data models do not generally maintain historical views of data, but the data warehouse does. This means that each time any attribute in an entity changes in value, a new row is generated. If different data elements change at different intervals, rows will be generated even if only one element changes, because all updates to the data warehouse are through row insertions.

This last transformation step recognizes that data in the operational environment changes at different times, and therefore groups data into sets based on insertion patterns. If taken to the extreme, a separate entity would be created for each piece of data. That approach will maximize the efficiency of the data acquisition process and result in some disk space savings. The first sentence of this section indicated that the segregation is based on two aspects—stability (or volatility) and usage. The second factor—usage—considers how the data is retrieved (that is, how it is delivered to the data mart) from the data warehouse. If data that is commonly used together is placed in separate tables, the data delivery process that accesses the data generates a join among the tables that contain the required elements, and this places a performance penalty on data retrieval. Therefore, in this last transformation step, the modeler needs to consider both the way data is received and the way it is subsequently delivered to data marts.

**NOTE:** The preceding steps define a methodology for creating the data warehouse data model. Like all methodologies, there are occasions under which it is appropriate to bend the rules. When this is being contemplated, the data modeler needs to carefully consider the risks and then take the appropriate action. For example, the second step entails adding a component of time to the key of every entity. Based on the business requirements, it may be more appropriate to fully refresh certain tablesif referential integrity can be met.

# 3. Summary

|  |  |  |  |
| --- | --- | --- | --- |
| STEP | ACTION | OBJECTIVE | ACTION DESCRIPTION |
| 1 | Select data of interest | Contain scope, reduce load time, reduce storage requirements | Determine data elements to be included in the model and consider archiving other data that might be needed in the future |
| 2 | Add time to the key | Accommodate history | Add time component to key and resolve resultant changes in the relationships due to conversion of the model from a “point-in time” model to an “overtime” model |
| 3 | Add derived data | Ensure business consistency and improve data delivery process performance | Calculate and store elements that are commonly used or that require consistent algorithms |
| 4 | Adjust granularity | Ensure that the data warehouse has the right level of detail | Determine the desired level of detail, balancing the business needs and the performance and cost implications |
| 5 | Summarize | Facilitate data delivery | Summarize based on use of the data in the data marts |
| 6 | Merge | Improve data delivery performance | Merge data that is frequently used together into a single table if it depends on the same key and has a common insertion pattern |
| 7 | Create arrays | Improve data delivery performance | Create arrays in lieu of attributive entities if the appropriate conditions are met |
| 8 | Segregate | Balance data acquisition and data delivery performance by splitting entities | Determine insertion patterns and segregate data accordingly if the query performance will not significantly degrade |

# 4. Source Books and Articles

1. Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming; Techniques and Solutions, Second Edition; Thomas Kyte ; 2010;
2. Pro Oracle SQL; Karen Morton, Kerry Osborne, Robyn Sands, Riyaj Shamsudeen, Jared Still ; 2010;
3. Oracle Data Warehouse Tuning for 10g; Gavin Powell; 2005;
4. Oracle® Database Data Warehousing Guide 10g Release 1 (10.1) Part No. B10736-01