DB2 Information Management Software

IBM

Dimensional Modeling: In a Business Intelligence Environment

- Dimensional modeling for easier data access and analysis
- Maintaining flexibility for growth and change
- Optimizing for query performance

Chuck Ballard
Daniel M. Farrell
Amit Gupta
Carlos Mazuela
Stanislav Vohnik

ibm.com/redbooks

Redbooks

International Technical Support Organization

Dimensional Modeling: In a Business Intelligence Environment

March 2006

SG24-7138-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (March 2006)

This edition applies to DB2 Universal Database Enterprise Server Edition, Version 8.2, and the Rational Data Architect, Version 6.1.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo)  ®	DB2 OLAP Server™	MQSeries®
pSeries®	DB2®	Rational®
AIX®	DRDA®	Red Brick™
Cube Views™	Informix®	Redbooks®
Database 2™	Intelligent Miner™	RS/6000®
Distributed Relational Database Architecture™	IBM®	WebSphere®
	IMS™	

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

MetaStage, MetaBroker, MetaArchitect, DataStage, Ascential, are trademarks or registered trademarks of Ascential Software Corporation in the United States, other countries, or both.

Co-Author, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

EJB, Java, JDBC, JDK, JRE, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, Visual Basic, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Business intelligence (BI) is a key driver in the business world today. We are now deep into the information age, and things have changed dramatically. It has long been said that information is power, and we can now understand that statement ever more clearly.

Business is moving at a much faster pace. Management is looking for answers to their questions, and they need these answers much more quickly. Time is money, and real-time information is fast becoming a requirement. These directions, movement, and initiatives force major changes in all the business processes, and have put a sharper focus on the whole area of data management.

To support all this requires integrated, accurate, current, and understandable sources of data that can be accessed very quickly, transformed to information, and used for decision-making. Do you have the data infrastructure in place to support those requirements? There can be a good number of components involved, such as:

- A well-architected data environment
- Access to, and integration of, heterogeneous data sources
- Data warehousing to accumulate, organize, and store the data
- Business processes to support the data flow
- Data federation for heterogeneous data sources
- Dashboarding for proactive process management
- Analytic applications for dynamic problem recognition and resolution

It should be quite clear that data is the enabler. And the first point in the above list is about having a good data architecture. What would you say is the cornerstore building block for creating such an architecture? You are correct! It is the data model. Everything emanates from the data model. More precisely, in a data warehousing and business intelligence environment, the dimensional model. Therefore, the subject of this IBM® Redbook.

Are you ready for these changes? Planning for them? Already started? Get yourself positioned for success, set goals, and quickly move to reach them.

Need help? That is one of the objectives of this IBM Redbook. Read on.

Acknowledgement

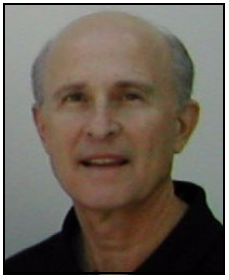
Before proceeding, we would like to acknowledge Dr. Ralph Kimball for his work in data warehousing and dimensional data modeling. He is well known in the industry, is a leading proponent of the technology, and is generally acknowledged

to be the originator of many of the core concepts in this subject area. When you think of subjects such as data warehousing, data marts, and dimensional modeling, one of the first names that comes to mind is Dr. Kimball. He has written extensively about these and related subjects, and provides education and consulting offerings to help clients as they design, develop, and implement their data warehousing environments. We have listed a few of his publications about dimensional modeling and data warehousing, published by John Wiley & Sons, in “Related publications” on page 637. We consider these to be required reading for anyone who is interested in, and particularly for those who are implementing, data warehousing.

The team that wrote this redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

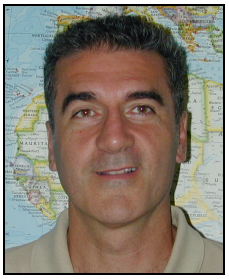
Some team members worked locally at the International Technical Support Organization - San Jose Center, while others worked from remote locations. The team members are depicted below, along with a short biographical sketch of each:



Chuck Ballard is a Project Manager at the International Technical Support organization, in San Jose, California. He has over 35 years experience, holding positions in the areas of Product Engineering, Sales, Marketing, Technical Support, and Management. His expertise is in the areas of database, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and seminars worldwide. Chuck has both a Bachelors degree and a Masters degree in Industrial Engineering from Purdue University.

Daniel M. Farrell is an IBM certified Professional and Pre-sales Engineer, from Denver, Colorado. In 1985, Daniel began working with Informix® software products at a national retail and catalog company, a job that would set his direction for the next twenty or so years. He joined IBM as a result of the acquisition of Informix. Daniel has a Masters in Computer Science from Regis University and is currently pursuing a Ph.D. from Clemson in Adult Education and Human Resource Studies.

Amit Gupta is a Data Warehousing Consultant from IBM, India. He is a Microsoft® Certified Trainer, MCDBA, and a Certified OLAP Specialist. He has about seven years of experience in the areas of databases, data management, data warehousing, and business intelligence. He teaches extensively on dimensional modeling, data warehousing, and BI courses in IBM India. Amit has also been a Co-Author® for a previous IBM Redbook on Data Mart Consolidation. He holds a degree in Electronics and Communications from Delhi Institute of Technology, Delhi University, New Delhi, India.



Carlos Mazuela is a data warehousing specialist, data modeler, and data architect from Zurich, Switzerland. He has been with IBM for eight years, and is currently working on data warehousing projects in the insurance and banking sectors. Carlos is also a DB2/UDB specialist, primarily in the areas of security, performance, and data recovery. He has skills and experience in the business intelligence environment, and has used tools such as Business Objects, Brio Enterprise, and Crystal Reports. Carlos holds a degree in Computer Science from the Computer Technical School of the Politechnical University of Madrid, Spain.

Stanislav Vohnik is an IT Architect from Prague, Czech Republic. He holds a Masters Degree in Mathematics and Physics from the Charles University, in Prague, Czech Republic. He has more than 10 years of experience in data management, and has been a programmer, IT specialist, IT adviser, and researcher in Physics. His areas of expertise include database, data warehousing, business intelligence, database performance, client/server technologies, and internet computing.

Other Contributors:

Thanks to the following people for their contributions to this project:

Sreeram Potukuchi: We would like to give special acknowledgement to Sreeram, who is a data warehouse architect and manager. He works for Werner Enterprises, an IBM Business Partner. Sreeram contributed content, and worked along with the team in determining the objectives and topics of the redbook.

A special thanks also to Helena, Marketa and Veronika.

From IBM Locations Worldwide

Melissa Montoya - DB2® Information Management Skills Segment Manager, Menlo Park, CA.
Christine Shaw - Information Management Senior IT Specialist, Denver, CO.
Stepan Bem - BI Solutions Expert, IBM Global Services, Prague, Czech Republic.
Milan Rafaj - Data Management Senior IT Specialist, IBM Global Services, Prague, Czech Republic.

From the International Technical Support Organization, San Jose Center
Mary Comianos - Operations and Communications
Deanna Polm - Residency Administration
Emma Jacobs - Graphics
Leslie Parham - Editor

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our Redbooks® to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbook@us.ibm.com`

1

Introduction

Within the scope of business computing and information technology (IT), it has been said that the 1980s were about *performing your business*, the 1990s were about *analyzing your business*, and the late 1990s and beyond are about new routes to market and *bringing your business to the World Wide Web.* To build further upon these categorizations, we offer the following points:

- *Performing your business*
  - Online Transaction Processing (OLTP) was the supporting technology.
  - Companies could increase net revenue by:
    - Lowering cost of sales through computerized automation, and less intensive labor. The internal cost and percent of (human) labor is lowered.
    - Better utilization of (capital) inventory, and increased visibility of on hand inventory. A customer new order call center can view and sell products from numerous and remote **company-owned distribution centers.**
    - Faster turnaround on order processing, faster order assembly and shipment, and faster bill generation and collection. The *number of days outstanding*, an important accounting metric, is lowered.
  - Companies could grow in size beyond previously known limits.
  - OLTP data is modeled using Entity Relationship (E/R) modeling, sometimes referred to as *Third Normal Form (3NF).*

- Why business or IT cares: OLTP business systems are designed and delivered to support the business and operational goals. While IT is a cost center, IT support systems deliver cost savings on operational activities.

- *Analyzing your business*
  - Business Intelligence (BI) systems provide the information that management needs to make good business decisions.
  - Increase company net revenue and decrease operating margins (internal cost) by:
    - Lowering customer service. BI aids in identifying high value customers, delivering customer reward programs, and identifying causes of customer loss through data analysis.
    - Analysis of markets (product and customer demographic data) enables more efficient application of (target) marketing programs. BI systems support increases in market share by enabling better understanding and execution of the business plan to enable increased sales.
    - Better operational efficiencies through better understanding of operational data.
  - Allows companies to compete with the most efficient operating margins.
  - BI data is modeled using a small amount of E/R (Entity/Relationship), such as OLTP systems, but a larger percentage of businesses uses dimensional modeling.
  - Why business or IT cares: At this point in history, most companies are expected to deliver and execute competent OLTP/operational type systems for efficiency. However, BI systems, when executed properly, can improve their effectiveness and offer a distinct and strategic competitive advantage. While technically still a cost center, IT moves more into the strategic side of business planning and execution. IT is now, or should be, viewed as moving from simply being *necessary* for operations to being a *strategic requirement for success.*

- *Bringing your business to the World Wide Web*
  - While OLTP and BI are types, or categorizations, of business application systems, the Web is a *standard computing platform.* OLTP and BI systems could be delivered via the Web, two-tier client/server systems, or possibly even simple ASCII (green screen) terminals. In this context, the Web is merely a delivery platform for OLTP and BI systems.
  - You can increase company gross revenue and net revenue by the following, for example:
    - Allow for new routes to market (new sales channels). For example, a local retailer can offer to sell products nationally and even

internationally via their commercial Web site. Larger and global markets offer the opportunity to increase sales revenue.

- Customers can self-service their account through a Web site or Web Portal, make inquiries, and place, manage and track orders at any time of day they choose. Customer self-service lowers the cost of sales. The ability for customer self-service at the time of their choosing raises customer satisfaction and lowers customer churn.
- Shared inventory visibility between retailer and manufacturer (extranet application) lowers cost of sales, and can lower, or even eliminate, inventory levels. A customer new order call center, or Web-based retail site, can sell and deliver products from numerous and remote *manufacturer-maintained* distribution centers.

– Allow companies to sell and compete globally, and at lower costs.

– While the Web is a standard computing platform, meaning standard communication protocols, there are a few (at least) standard computing infrastructure platform choices. Microsoft's [dot].Net and the open source Java/J2EE are two examples. This IBM Redbook gives attention and preference to open source, cooperative computing, implying Java/J2EE™.

– Why business or IT cares: IT can become a profit center. That is, it can become the virtual, automated sales agent, or customer self-service agent. OLTP and BI systems are delivered remotely or via the Web, and should be designed and constructed to leverage this and future platforms.

This is not to say that these categorizations imply an ending to one and beginning of another. They are all ongoing and in a constant state of improvement. For example, BI is all about analyzing your business. And, it is still not a mature category. The fact that we are now in the process of bringing business to the Web does not mean we have finished with data analysis and performing our business. It is just a continuation of the evolution of business computing and information technology.

1.1 Scope of this redbook

Business intelligence (BI) environments are changing, and quite dramatically. BI is basically comprised of a data warehousing infrastructure, and a query, analysis, and reporting environment. In this redbook, we focus on the data warehousing infrastructure, but primarily a specific element of it termed the data model. Or, more precisely in a data warehousing and business intelligence environment, the dimensional model. We consider this the base building block of the data warehouse. The focus then is on the *data model*. Or, more precisely, the topic of *data modeling and its impact on the business and business applications*.

We discuss data modeling techniques and how to use them to develop flexible and highly performant data models. We refer to the two primary techniques businesses use, as EAR or E/R (Entity Attribute Relationship or sometimes as simply Entity Relationship) data modeling and dimensional modeling.

In this redbook, we take a specific focus on dimensional modeling. There is a detailed overview of dimensional modeling, along with examples to aid in understanding. We also provide best practices for implementing and maintaining a dimensional model, for converting existing data models, and for combining multiple models.

Acknowledgement

Before proceeding, we would like to acknowledge Dr. Ralph Kimball for his work in data warehousing and dimensional data modeling. He is well known in the industry, is a leading proponent of the technology, and is generally acknowledged to be the originator of many of the core concepts in this subject area. When you think of subjects such as data warehousing, data marts, and dimensional modeling, one of the first names that comes to mind is Dr. Kimball. He has written extensively about these and related subjects, and provides education and consulting offerings to help clients as they design, develop, and implement their data warehousing environments. We have listed a few of his publications about dimensional modeling and data warehousing, published by John Wiley & Sons, in “Related publications” on page 637. We consider these to be required reading for anyone who is interested in, and particularly for those who are implementing, data warehousing.

Objective

Once again, the objective is not to make this redbook a treatise on dimensional modeling techniques, but to focus at a more practical level. That is to relate the implementation and maintenance of a dimensional model to business intelligence. The primary purpose of business intelligence is to provide answers to your business questions, and that requires a robust data warehousing infrastructure to house your data and information objects. But, it also requires a

query, analysis, and reporting environment to get the information out of the data warehouse and to the users. And to get it to those users of the system with acceptable performance.

We also provide a discussion of three current business intelligence initiatives, which are business performance management, real-time business intelligence, and data mart consolidation. These are all initiatives that can help you meet your business goals and objectives, as well as your performance measurements. For more detailed information about these specific initiatives, refer to the following IBM Redbooks:

- *Business Performance Management...Meets Business Intelligence*, SG24-6340
- *Preparing for DB2 Near-Realtime Business Intelligence*, SG24-6071
- *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653

To get these, and other IBM Redbooks, see “How to get IBM Redbooks” on page 638.

Programming and the data model

This IBM Redbook also has a focus on business application programming. More specifically, we dedicated it to business application programming for business intelligence systems, including such elements as data marts, data warehouses, and operational data stores, and related technologies using relational database servers that utilize industry standard Structured Query Language (SQL).

In general, there is systems programming and there is business application programming. Systems programming involves creating the next great operating system, spreadsheet, or word processor. Business application programming would involve creating such things as an employee time and attendance system, a customer new order entry system, a market productivity analysis and reporting system, or something similar. While both systems programming and business application programming make use of data modeling, systems programming does so sparingly. Systems programming is typically more about algorithms, program function, the user interface, and other similar things. Systems programming uses data modeling merely as a means to accomplish its task, which is to deliver such things as word processing functionality or a spreadsheet program. Business application programming is typically all about processing data, and is almost entirely dependent on data modeling. Regardless of the architecture, a business application systems always sits on top of a persistent data model of some type.

1.2 What this redbook includes

It seems that the informational technology industry is always chasing the *next great thing*. At this time, the current next great thing seems to be service-oriented architecture (SOA); a specific subtopic or sub-capability within a Web-based architecture. So why do we want to provide a new publication on business intelligence and dimensional modeling? Well, consider the following:

- We focus specifically on the combined topics of dimensional modeling and business intelligence. This redbook is not intended to be an academic treatise, but a practical guide for implementing dimensional models oriented specifically to business intelligence systems.
- Business intelligence is a strategic type of information technology that can deliver a significant contribution to the net and operating revenues of a company. While the Web and initiatives such as SOA are in high demand, they are merely the architectures with which business intelligence may be delivered.
- This redbook is a strategic and comprehensive dimensional modeling publication.
- We included best practices, as well as specific procedures to deliver systems of this type more quickly, and with measurable and increased success rates.
- Because we want to see you succeed, and we believe this redbook has information that can help you do that.

Further, this IBM Redbook includes:

- Detailed discussion of a dimensional model life cycle (DMDL). This was developed to help you create functional and highly performant dimensional models for your BI environment.
- An extensive case study about developing a dimensional model by following the processes and steps in the DMDL.
- A detailed analysis of an existing sample dimensional model, along with a discussion of techniques that you can use to improve it.
- Practical and understandable examples. We present business intelligence concepts by using examples taken from the business environment.
- Application of current, practical technologies. We present examples that are demonstrated using technology in currently available software products, where applicable.
- A common base of knowledge. We assume that you are already familiar with information technology, and even online transaction processing (OLTP). This IBM Redbook bridges the transformation and delivery of OLTP systems data into business intelligence.

1.3 Data modeling and business intelligence

We have discussed the evolution of information technology, and the directions it has taken over the years. It has served many needs, and particularly those associated with business intelligence. It is, after all, information that enables business intelligence. And if we continue looking at the information structure, we see that at the base level there is data. That data is collected from many sources and integrated with technology to enhance its usefulness and meaning.

However, to finish our investigations, we must go to one more level. That is the level that defines and maintains the structure of the data, and is the key enabler of the usefulness of the data. That level is the data model.

While there are many technologies, techniques, and design patterns, presented in the pages and chapters that follow, this section demonstrates, by example, a single and simple online transaction processing (OLTP) business application system that is then *migrated* to a business intelligence (BI) system.

1.3.1 SQL, OLTP, and E/R modeling

Structured Query Language (SQL) is a data access command and control language associated exclusively with *relational databases.* Databases are simply sizeable collections of related data; such as facts, transactions, names, dates, and places. Relational databases are those databases of a given operational style or design pattern. As computer software languages go, SQL is best referred to as a *declarative language*. Declarative languages declare; they tell some other entity what to do without telling them how to do it. Hyper Text Markup Language (HTML) is another declarative computer software language. HTML tells the Web browser what the markup instructions are, but it is the Web browser that determines how to render the given text. For example, HTML may request a bold emphasis on a text string, but the Web browser determines that a given installation desires a heavyweight character font and 12 point type face when bold is requested. SQL is the command language to read, write, and define the data structures that reside within a database.

The technology for relational database was unveiled in 1970 in a publication by IBM researcher, E.F. Codd. However, it was not until the early to mid-1980s that relational databases began to prove their commercial viability. At that time, independent software vendors, such as Informix Software (originally named Relational Database Systems), Oracle® Software (originally named Relational Database Technologies), and Ingres Software began shipping software systems that would serve data via receipt and processing of SQL commands. A business application program handled the user interface. Both keyboard input and terminal output execute some amount of business logic, but then rely on the database server to read and write data to a shared repository, the *relational database*.

A project plan to develop an E/R data model

Rather than project plan, perhaps it would be better to say we are reviewing the software development life cycle in which an online transaction processing business application system is developed using E/R data modeling. A few more points to consider here:

- As a phrase, *software development life cycle* represents an abstract concept, a life cycle.
- The Waterfall Method is one implementation of an SDLC. In a Waterfall Method SDLC, there are generally five to seven development and analysis stages through which you create business application software. Figure 1-1 on page 9 displays a Waterfall Method.
  - Stage 1 of Figure 1-1 on page 9 is entitled *discovery*. During this phase, you determine the requirements of the business application. For an OLTP business application system, this includes gathering the layouts of all printed reports and specific data to be collected during data entry.
  - Stage 2 of Figure 1-1 on page 9 is entitled *data model*. During this phase, you create the data model. How you create an entity relationship (E/R) data model, and, more specifically, how you create a dimensional model, is the topic of this IBM Redbook.
  - There is a very specific and intentional dashed line drawn between Stage 2, and the remaining stages of this Waterfall Method SDLC. Stage 2 is normally completed by a data modeler, where Stages 3, 4, and 5 are normally performed by programmers and programmer management. This can raise numerous project management and performance issues, which we address throughout this redbook.
- OLTP is a type, or categorization, of a business application, and generally has the following characteristics:
  - Data reads (SQL SELECT statements) return very few data records, generally in the range of five to one hundred and certainly fewer than hundreds or thousands of records.
  - The filter criteria for a given SQL SELECT is generally well known in advance of execution; such as to return a customer order by order number, report revenue by company division, and then by product stock keeping identifier.
  - The database and database server run-time environments are easily optimized for this workload.

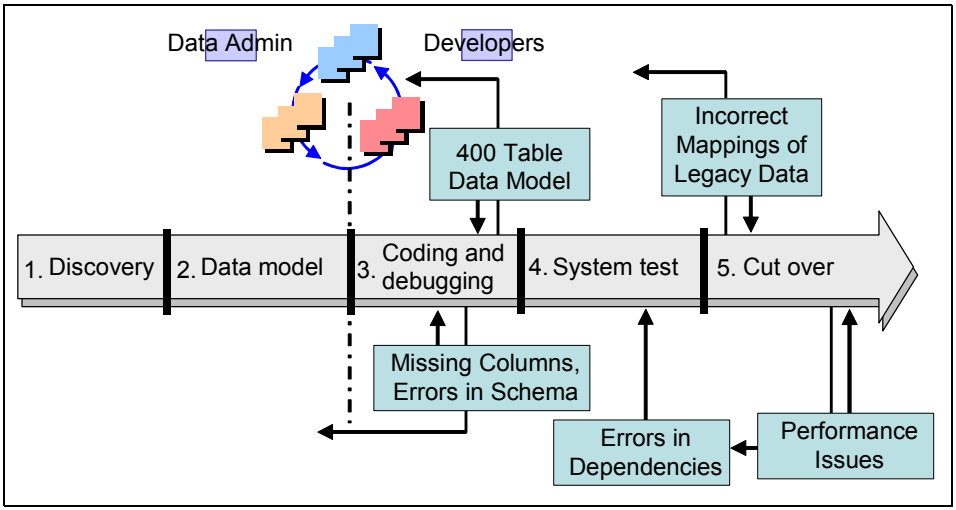


Figure 1-1 Example of Waterfall Method, Software Development Life Cycle

Project plan issues

A number of problems display themselves in a typical project plan, as shown in Figure 1-1. The issues include:

- The people who gather all of the application expertise and business knowledge to create the data model, are not the same people who author the programming to access this data model. Optimizations and efficiencies would be gained if the software development life cycle being employed somehow encouraged this knowledge to propagate from the data modeling phase to the programming and testing phases.
- The application programmers are those people best enabled to find missing columns and other errors in the data model as they begin and then complete their work. This can create a cycle of dependency, and then greater errors as the programmers rely on the modelers to make changes, which then often introduces errors in other application programs which had already been completed.
- In the system test phase, further errors are uncovered, as it becomes known that the programmers misinterpreted the large data model, incorrectly joining tables, and other similar activities.
- Often the first migration of data from the existing system to the new system occurs in whole only towards the end of the project. This leads to discovery of errors in the data model, missing attributes, or incorrect data type mappings.

- And then last, but certainly not all inclusive, application performance too is one of the last elements to be tested following data migration from the existing system.

In addition to data modeling for business intelligence, the redbook addresses the process of data modeling, not only as a standalone topic, but as a topic to be integrated into a software development life cycle.

An example OLTP business system with E/R data model

Figure 1-2 on page 12 displays an OLTP data model. Again, an OLTP data model is a bit of a misnomer; it is actually a style or categorization of business application system. OLTP systems are almost exclusively associated with E/R data modeling. As a means to detail dimensional (data) modeling, which is associated with BI business application systems, both E/R modeling and dimensional modeling are detailed in the redbook. For now, see the following:

- As shown in Figure 1-2 on page 12, E/R data models tend to *cascade*; meaning one table flows to the next, picking up volume as it goes down the data hierarchy. A customer places one or more orders, and an order has one or more order line items. A single customer could have purchased hundreds of order line items, as brokered through the dozens of orders placed.
- From the example in Figure 1-2 on page 12, a printed copy of a Customer Order would have to extract (assemble) data from the Customer table, the Customer Order table, the Order Line Item table, and perhaps others. E/R data models optimize for writing, and as a result have little or no data redundancy. The customer contact data, a single record in the customer table, is listed in one location. If this customer contact data is needed to print a given customer order, data must be read from both the Customer table and Customer Order (and perhaps even Order Line Item) table.
- E/R data models, and therefore OLTP business applications, are optimized for writing. E/R data models do not suffer for reading data, because the data access methods are generally well known; and data is generally located by a well known record key.
- If E/R data models are reasonably performant for writing and reading, why then do we need dimensional modeling?
  - E/R data models represent the data as it currently exists. From the example in Figure 1-2 on page 12, the current state of the Customer Account, the current Inventory level (Stock table), the current Item Price (Stock table), are all things that are known. However, E/R data models do not adequately represent *temporal data*. That is, a history of data as it changes over time. What was the inventory level by product each morning at 9 AM, and by what percentage did sale revenue change and net profit rise or fall with the last retail price change. These are examples of the

types of questions that E/R models have trouble answering, but that dimensional models can easily answer.

– Although E/R models perform well for reading data, there are many assumptions for that behavior. E/R models typically answer simple questions which were well anticipated, such as reading a few data records, and only then via previously created record keys. Dimensional models are able to read vast amounts of data, in unanticipated manners, with support for huge aggregate calculations. Where an E/R model might not perform well, such as with sweeping ad hoc reads, dimensional models likely can. Where dimensional models might not perform well, such as when supporting intensive data writes, E/R models likely can. Dimensional models rely upon a great amount of redundant data, unlike E/R models, which have little or no redundant data. Redundant data supports the sweeping ad hoc reads, but would not perform as well in support of the OLTP writes.

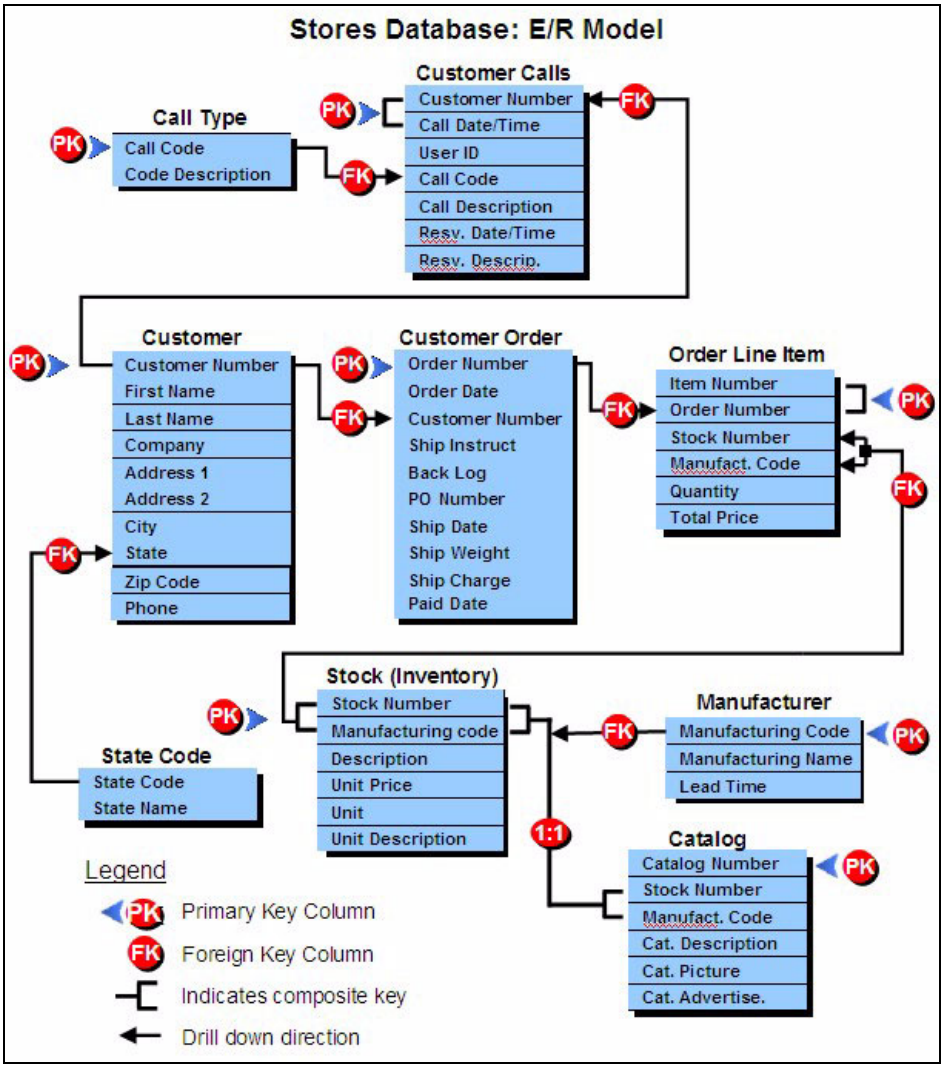


Figure 1-2 Sample OLTP schema

1.3.2 Dimensional modeling

The history of E/R modeling is tied to the birth of relational database technology. E/R models did perform well and serve OLTP business application systems well. However the promise of relational database to provide easy access for all to the corporate database came into dispute because of the E/R model.

An E/R data model for even a corporate division level application can have 100-200 significant data tables and thousands of columns. And all with possibly cryptic column names, such as “mnth_end_amt_on_hnd”. This was thought to be too complex an environment for non-IT users. And the required task of joining three, five, nine, or more tables of data to produce a useful report was generally considered too complex a task for an everyday user. A new approach was required, and that approach was using dimensional modeling rather than E/R modeling.

This new categorization of business application was called Decision Support, or Business Intelligence, or a number of other names. Specific subtypes of these business applications were called data warehouses, data marts, and operational data stores.

Data warehouses and data marts still reside within relational database servers. They still use the Structured Query Language (SQL) data access command and control language to register their requests for service. They still place data in tables, rows, and columns.

How data warehouses and data marts differ from OLTP systems with their E/R data model, is also detailed, and expanded upon, throughout this redbook. As a means to set current scope, expectations, and assumptions, in the next section we take the E/R data model from Figure 1-2 on page 12 and convert it to a dimensional model.

An example dimensional model

E/R data models are associated with OLTP business application systems. There is a process you go through to accurately design and validate an E/R data model. A business intelligence business application system may use an E/R data model, but most commonly uses a dimensional model. A dimensional model also goes through a design process so that it can be accurately designed and validated. Figure 1-3 on page 14 displays a dimensional model that was created from the E/R data model displayed in Figure 1-2 on page 12.

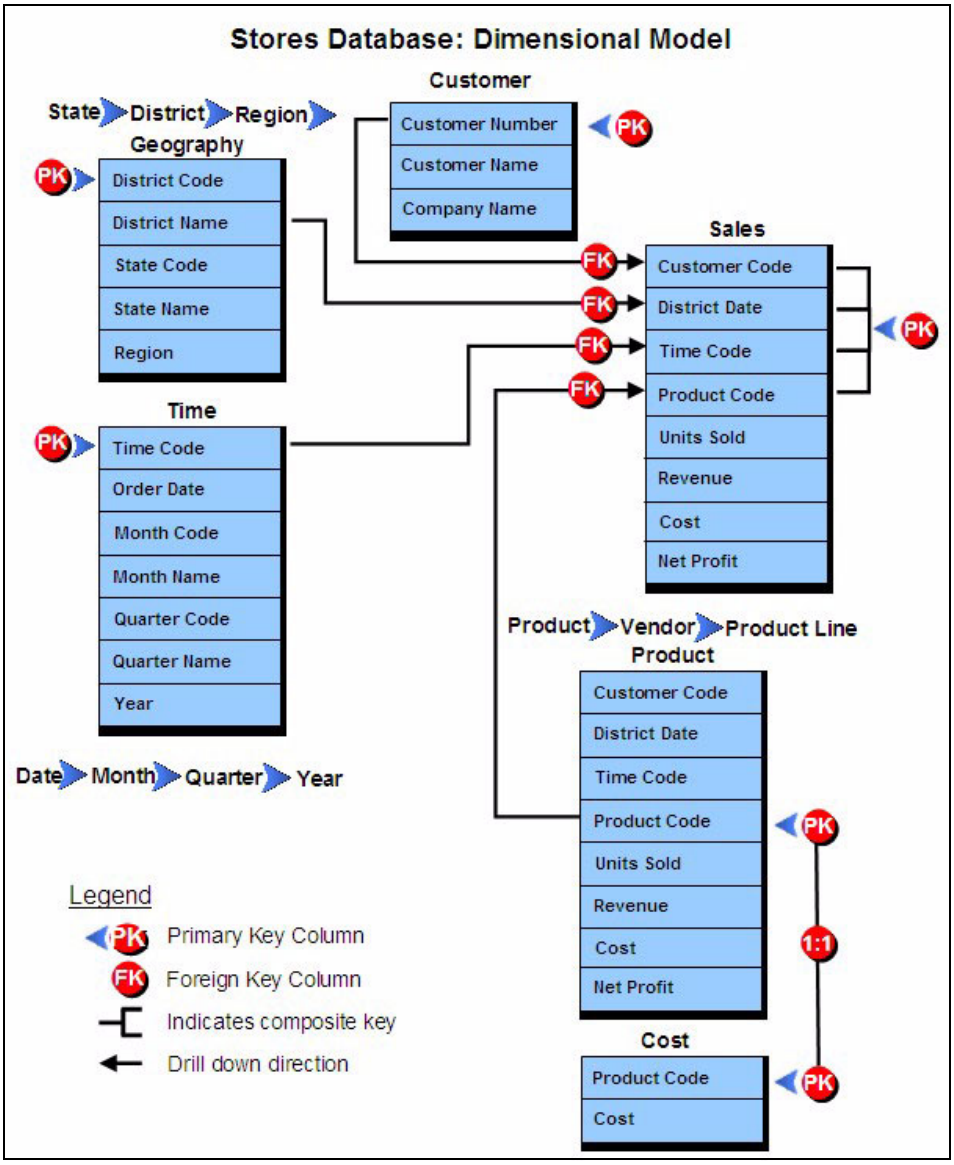


Figure 1-3 Dimensional model created from that shown in Figure 1-2 on page 12

This data model was created to support a customer new order entry OLTP business application system. As mentioned, there is a process to create and then validate a data model.

The following points address the creation of a dimensional model.

- Creation of an E/R data model has a semi-rigid and well defined procedure to follow (there are rules called Normal Forms to which you must adhere); dimensional modeling is somewhat less formal and less rigid. Certainly there are design patterns and goals when creating a dimensional model, but dimensional modeling involves a percent of style. Perhaps all of this originates from the target audiences that E/R data models and dimensional models each have. E/R data models serve the IT architects and programmers, and indirectly serve users, where dimensional models directly serve users.
- The central purpose of the E/R data model in Figure 1-2 on page 12 was to serve a sales application in a customer new order entry business application system. A customer order is, by one viewpoint, the central event or whole point of that application. You could also imagine it to centrally be an inventory management system, revenue recognition system, or other system. Taking the viewpoint of the central event being a customer placing a new order, we see that most of the tables in Figure 1-3 on page 14 point to a table titled Sales.
- A dimensional model may have one or more subjects, but simple dimensional models have only one subject. In the case of a single subject dimensional model, this subject is the primary topic, fact, or event of the model, and is joined by numerous dimensions. A synonym for dimension in this context might be demographic trait, index into (a given fact), or temporal state.
- The (central) *fact* table in the dimensional model shown in Figure 1-3 on page 14 is
  Sales. Sales is joined by the *dimensions*; Customer, Geography, Time, and then Product and Cost.
- Where E/R data models have little or no redundant data, dimensional models typically have a large amount of redundant data. Where E/R data models have to assemble data from numerous tables to produce anything of great use, dimensional models store the bulk of their data in the single *Fact table,* or a small number of them.
- Dimensional models can read voluminous data with little or no advanced notice, because most of the data is resident (pre-joined) in the central Fact table. E/R data models have to assemble data and best serve small well anticipated reads. Dimensional models would not perform as well if they were to support OLTP writes. This is because it would take time to locate and access redundant data, combined with generally few if any indexes. Why are there few indexes in a dimensional model? So often the majority of the fact table data is read, which would negate the basic reason for the index. This is referred to as index negation, a topic discussed further in Chapter 10, “SQL query optimizer: A primer” on page 497.

These are but brief examples so that you may gather an introductory understanding of the topic of this IBM Redbook. The following is a detailed list of the remaining contents.

1.4 Redbook contents abstract

In this section, we give a brief description of the topics we present in this IBM Redbook and how we organize them. We cover a wide spectrum of information, because that is what is demanded for business intelligence solutions. However, keep in mind that the more specific focus of this redbook is on dimensional modeling and how it impacts business intelligence solutions.

The information presented includes topics such as OLTP, business intelligence, dimensional modeling, program architectures, and database server architectures. Depending on your specific interest, level of detail, and job focus, certain chapters may be more relevant to your needs than others. So, we have organized this redbook to enable selectivity in reading.

Let us get started with a brief overview of the IBM Redbook contents:

- Chapter 1, “Introduction” includes:
  - A description of the objectives and scope of the redbook, and summarizes the value to be gained from reading this redbook.
  - A summary of the evolution of information technology and describes how it relates to dimensional modeling.
  - Positioning and relationship of data modeling and business intelligence.
  - An introduction to dimensional modeling, that includes descriptions and examples of the types of data modeling.
- Chapter 2, “Business Intelligence: The destination”, includes:
  - A focus on business intelligence, beginning with an overview.
  - A discussion of the IBM Information Pyramid, an important structure to understand as you develop your data, and data warehousing, architecture.
  - A brief discussion of the impact of the World Wide Web on data modeling.
  - An overview of three of the key BI initiatives and how data modeling impacts them.
- Chapter 3, “Data Model: The organizing structure”, includes:
  - More detailed descriptions of data modeling, along with detailed discussions about the types of data modeling.

  - An description of several data warehousing architectures, discussions about the implementation of data warehousing architectures, and their reliance on the data model.
  - An introduction to the data modeling life cycle for data warehousing.
- Chapter 4, “Data analysis techniques”, includes:
  - A description of data analysis techniques, other than data warehousing and BI. This includes a discussion of the various enterprise information layers in an organization.
  - An overview of BI reporting tool architectures, which includes a classification of BI users based on their analytical needs. This includes query and reporting, and multidimensional analysis techniques.
  - An overview of querying and reporting tool capabilities that you can use in your data analysis environment.
- Chapter 5, “Dimensional model design life cycle”, includes:
  - An introduction to a dimensional modeling design life cycle. Included are the structure, phases, and a description of the business process requirements.
  - A description of the life cycle phases that can guide you through the design of a complete dimensional model.
- Chapter 6, “Modeling considerations”, includes:
  - A discussion of considerations and issues you may encounter in the development of a dimensional model. It provides you with examples and alternatives for resolving these issues and challenges.
  - Guidelines for converting an E/R model to a dimensional model.
- Chapter 7, “Case Study: Dimensional model development”, includes:
  - A case study in the development of a dimensional model. It includes a description of the project and case study company. The company data model requirements are described, and the dimensional design life cycle is used to develop a model to satisfy the requirements. The objective is to demonstrate how you can start with requirements and use the life cycle to design dimensional models for your company.
- Chapter 8, “Case Study: Analyzing a dimensional model”, includes:
  - Another case study. However, this is a study analyzing an existing dimensional model. The requirements and existing model are analyzed for compliance. Where the model design does not meet the requirements, you are provided with suggestions and methods to change the model.
  - A case study that can help prepare you to analyze your existing dimensional models to verify that they satisfy requirements.

- Chapter 9, “Managing the meta data”, includes:
  - Information on meta data. Having emphasized the importance of the dimensional model, now take a step back. Meta data is the base building block for all data. It defines and describes the data, and gives it structure and content.
  - Definitions and descriptions of meta data types and formats. It also provides a discussion about meta data strategy, standards, and design.
  - An overview of tools used to work with meta data.
- Chapter 10, “SQL query optimizer: A primer”, includes:
  - A review of relational database server disk, memory, and process architectures, which includes the relational database server multistage back-end.
  - A detailed and real world query optimizer case study. In this section, a query is moved from executing in five minutes to subsecond. In addition to the technology used to improve the performance of this example, a methodology of SQL statement tuning and problem solving is introduced.
  - A full review of the algorithms of rules and cost-based query optimizers. This includes reading query plans, table access methods, table join methods, table join order, b-tree+ and hash indexes, and gathering optimizer statistics (including data distributions, temporary objects, and others).
  - A detailed review of other query optimizer technologies, to include; query rewrite, optimizing SQL command parser use, index negation, query optimizer directives, table partitioning, and multidimensional clustering.
- Chapter 11, “Query optimizer applied”, includes:
  - An overview of the Development Life Cycle, specifically the Waterfall method.
  - A discussion of the issues encountered with the life cycle, and the subject of process modeling within the life cycle.
  - A discussion of artifacts created by a process model, such as the SQL API, and query plan documents.
  - An example of process modeling with explanations of the steps and structure. It also includes an SQL DML example, along with an explanation of those steps and the structure.

That is a brief description of the contents of the redbook, but it hopefully will help guide you with your reading priorities.

2

Business Intelligence: The destination

In Chapter 1, we gave an overview introduction to the scope, objectives, and content of the redbook. We also introduced the two primary techniques used in data modeling (E/R and Dimensional), positioned them, and then stated that in this redbook we have a specific focus on dimensional modeling.

However, it is good to keep in mind the bigger picture and understand why we are so interested in the topic of dimensional modeling. Simply put, it is because the data model is the base building block for any data structure. In our case, the data structure is a data warehouse. And, the data warehouse is the base building block that supports business intelligence solutions - which is really the goal we are trying to achieve. It is our destination.

Since it is our destination, we need to have an understanding of what it is and why we want to go there.

Here we introduce the topic of business intelligence, discuss BI activities, describe key business intelligence initiatives, show you several data warehouse architectures, and then describe the impact that data modeling has on them.

The specific topics we cover are:

- An overview of business intelligence
- Business intelligence initiatives

  - Business Performance Management
  - Real-time Business Intelligence
  - Data Mart Consolidation
- The impact of data modeling on the key BI initiatives

2.1 Business intelligence overview

In the competitive world of business, the survival of a company depends on how fast they are able to recognize changing business dynamics and challenges, and respond correctly and quickly. Companies must also anticipate trends, identify new opportunities, transform their strategy, and reorient resources to stay ahead of the competition. The key to succeeding is information.

Companies collect significant volumes of data, and have access to even more data from outside their business. They need the ability to transform this raw data to actionable information by capturing, consolidating, organizing, storing, distributing, analyzing, and providing quick and easy access to it. This is the competitive advantage, but also the challenge. All of this is the goal of business intelligence (BI). BI helps a company create knowledge from that information to enable better decision making and to convert those decisions into action.

BI can help with the critical issues of a company, such as finding areas with the best growth opportunities, understanding competition, discovering the major profit and loss areas, recognizing trends in customer behavior, determining their key performance indicators, and changing business processes to increase productivity. BI analyzes historical business data that is created by business or derived from external sources, such as climatic conditions and demographic data, to study a particular function or line of business. Information is used to understand business trends, strengths, weaknesses, and to analyze competitors and the market situation. Prior to the existence of BI technologies, many companies used standard conventional methods to transform data into actionable information. This was certainly a laborious process that consumed enormous amounts of resources and time, not to mention the human error factor.

2.1.1 Information environment

It should be clear that BI is all about information, and the home for that information is an enterprise data warehouse. It is no longer something that is built for a particular advantage, it should now be seen as a business requirement. That requirement is to have a structured and organized information environment. Such a structure contains a number of different types and organizations of data. And, those can be organized into a structured environment. We depict this environment as an information pyramid in Figure 2-1 on page 24.

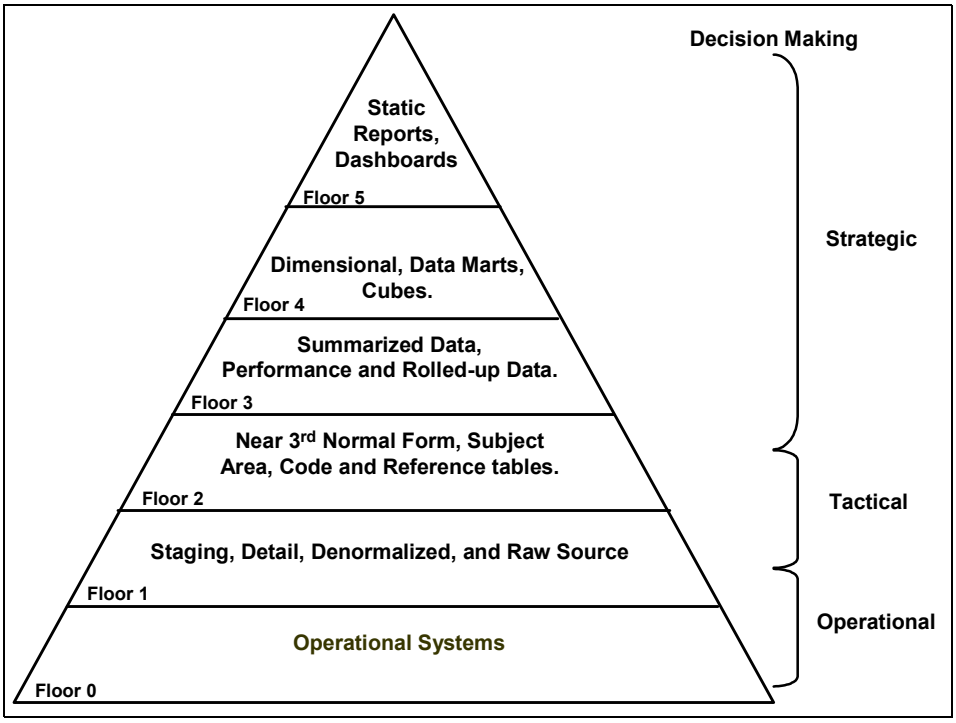


Figure 2-1 Information pyramid: an example

The information technology organization (IT) has traditionally seen different types of data in the information pyramid as separate layers, and required that data to be copied from one layer to another. However, you should look at different types of data as different views of the same data, with different characteristics, required to do a specific job. To emphasize that, we have labeled them as floors, rather than layers, of information.

To move and copy the data between the floors (and typically from the lower to the higher floors) is no longer the only option available. There are a number of approaches that enable integration of the data in the enterprise, and there are tools that enable those approaches. At IBM, information integration implies the result, which is integrated information, not the approach.

We have stated that the data on each floor has different characteristics, such as volume, structure, and access method. Now we can choose how best to physically store and organize the data on the different floors. For example, we can decide whether the best technology solution is to build the floors separately or to build the floors together in a single environment. An exception is floor zero, which, for some time, will remain separate. For example, an OLTP system may

reside in another enterprise, or another country. Though separate, we still can have access to the data and can move it into our data warehouse environment.

Floors one to five of the information pyramid can be mapped to the layers in an existing data warehouse architecture. However, this should only be used to supplement understanding and subsequent migration of the data. The preferred view is one of an integrated enterprise source of data for decision making — and a view that is current, or real-time.

2.1.2 Web services

At the start of Chapter 1, we discussed major eras in IT. One of those was about bringing your business to the World Wide Web. The Web has revolutionized the information industry, and dramatically changed the way the world does business.

There are many things we could discuss regarding the advantages and advances made and supported by the Web. However, although the Web has not significantly impacted the technology of dimensional modeling, it has certainly resulted in a number of changes. For example, there are many new data definitions to be included in the meta data and data model. And there has been a significant increase in the volumes of data captured and transmitted, but this does not directly impact the dimensional model.

Web services are a key advance that has significantly impacted applications. The word *Web* in Web services means that all operations are performed using the technology and infrastructure of the World Wide Web. The word *service* represents an activity or processing performed on behalf of a requestor, such as a person or application. Web services have existed ever since the Web was invented. The ability of a Web browser to access e-mail and the ability to order a product on the Internet are examples of Web services. More recently, however, Web services increasingly make use of XML-based protocols and standards, and it is better to think in terms of XML Web services. In this book, for simplicity, we use the term Web services to signify XML Web services.

The promise of Web services

Web services technology is essentially a new programming paradigm to aid in the development and deployment of loosely-coupled applications both within and across enterprises. In the past, developers have developed most of their applications *from the ground up.* The term *code reuse* was used, but this was often not put into practice because developers typically only trust the code they develop. As software development has progressed as a discipline, and as programming languages have also advanced, the ability to reuse application code has increased. The Java™ programming language, for example, has many built-in class libraries that developers use.

As applications grow, they must execute in a distributed environment. Distributed applications provide unlimited scalability and other benefits. Defining an interface for distributed applications has been a challenge over the years. Language-independent technologies, such as CORBA (Common Object Request Broker Architecture), provide a comprehensive and powerful programming model. Other distributed technologies work well within a single language environment, such as Java RMI (Remote Method Invocation) and Microsoft DCOM (Distributed Common Object Model), but are not useful in a heterogeneous systems environment.

In contrast, Web services provide a simple-to-understand interface between the provider and the consumer of application resources using a Web Service Description Language (WSDL). Web services also provide the following technologies to help simplify the implementation of distributed applications:

- Application interface discovery using Universal Description, Discovery, and Integration (UDDI)
- Application interface description, again using UDDI
- A standard message format using Simple Object Access Protocol (SOAP), which is being developed as the XML Protocol specification by W3C

Web services enable any form of distributed processing to be performed using a set of standard Web- and XML-based protocols and technologies. For more detailed information about Web services and related topics, see the work effort by the World Wide Web Consortium at:

`http://www.w3c.org`

In theory, the only requirements for implementing a Web service are:

- A technique to *format* service requests and responses
- A way to *describe* the service
- A method to *discover* the existence of the service
- The ability to *transmit* requests and responses to and from services across a network

The primary technologies used to implement these requirements in Web services are XML (format), WSDL (describe), UDDI (discover), and SOAP (transmit). There are, however, many more capabilities (authentication, security, transaction processing, for example) required to make Web services viable in the enterprise, and there are numerous protocols in development to provide those capabilities.

It is important to emphasize that one key characteristic of Web services is that they are platform neutral and vendor independent. They are also somewhat easier to understand and implement than earlier distributed processing efforts, such as CORBA. Of course, Web services still need to be implemented in vendor

specific environments, and this is the focus of facilities such as IBM Web Services.

Web services architecture

The Web services architecture is defined in several layers. These layers are illustrated in Figure 2-2.

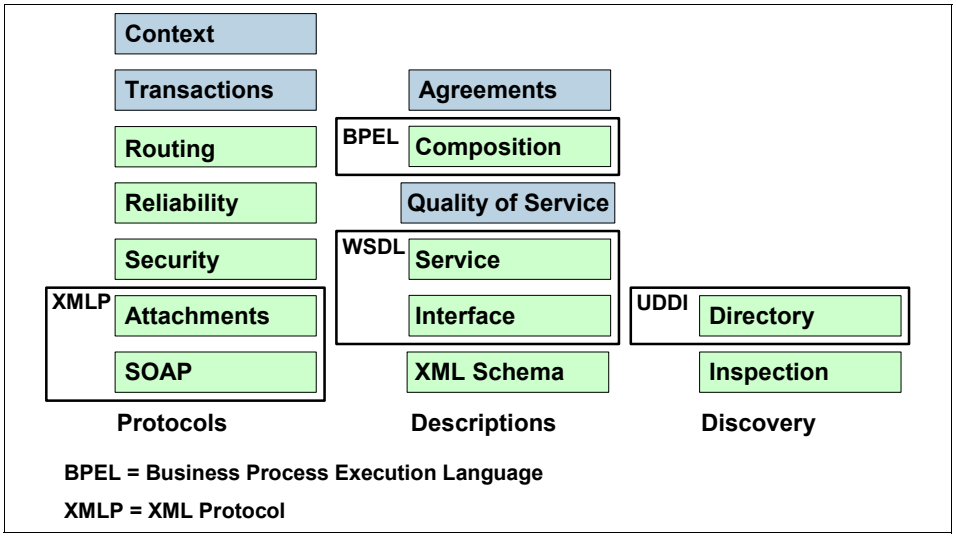


Figure 2-2 Web services layered architecture

The underpinnings of the Web services architecture are WSDL and SOAP. WSDL is an XML vocabulary used to describe the interface of a Web service, its protocol binding and encoding, and the endpoint of the service. SOAP is a lightweight protocol for the exchange of information in a distributed environment, and is used to access a Web service. It is transport-protocol independent. SOAP messages can be transported over HyperText Transfer Protocol (HTTP), for example, but other protocols are also supported. Examples include:

- SOAP over WebSphere® MQ (Message Queuing)
- RMI (Remote Method Invocation) over IIOP (Internet Inter-ORB [Object Request Broker] Protocol)

At present, the current SOAP standard only defines bindings for HTTP. SOAP is rightfully seen as the base for Web application-to-application interoperability. The fast availability of SOAP implementations, combined with wide industry backing, has contributed to its quick adoption.

SOAP employs a XML-based RPC (Remote Procedure Call) mechanism with a request/response message-exchange pattern. It is used by a service requestor

to send a request envelope to a service provider. The SOAP request envelope contains either an RPC method call or a structured XML document. Input and output parameters, and structured XML documents are described in XML schema. The service provider acts on a request and then sends back a SOAP response envelope.

The existence of a Web service can be published and advertised in a public UDDI registry. Publishing Web services in a public registry allows client applications to *discover* and *dynamically* bind to Web services. UDDI helps distributed application developers solve the maintenance problem caused by constantly changing application interfaces. Developers can use internal private registries, and public UDDI registries (hosted on the Internet by companies, such as IBM and Microsoft) to publicize their application interfaces (as specified by WSDL) and to discover other Web services. When a WSDL interface changes, a developer can republish the new interface to the registry, and subsequent access to the Web service will bind dynamically to the new interface.

IBM Web services

Two key IBM products for supporting Web services are WebSphere Studio and the WebSphere Application Server.

WebSphere Studio contains a set of development tools for creating and maintaining Java applications that use Web services, and it is based on an open development framework known as Eclipse. For more details, see:

`http://www.eclipse.org`

WebSphere Studio provides tools for creating WSDL interfaces to Java applications and DB2 data. You can publish Web services defined using WebSphere Studio to a UDDI registry directly from the WebSphere Studio environment. WebSphere Studio provides a UDDI browser.

IBM WebSphere Application Server is a J2EE-compliant Java Web Application Server. It is an ideal platform for hosting DB2 Web service provider applications. WebSphere Application Server includes the Apache SOAP server. For details, see:

`http://ws.apache.org/soap/`

2.1.3 Activity examples

Typically, BI solutions use different technologies for lines of business and departments, and you can implement them in several ways. The focus of this redbook is to explore dimensional modeling and its impact on BI implementations.

Business Intelligence activities include such activities as:

- Multidimensional Cube Analysis
- Business analysis
- Clickstream analysis
- Information visualization
- Forecasting
- Data mining (text, video, and voice)
- Trending analysis
- Query and reporting
- Geo-spatial analysis
- Enterprise portal implementation
- Digital dashboards

The following are examples of BI usage in a company:

- **Operations:** BI has an active role in helping management meet their operational performance measurements. For example, each manager in a retail company can receive a digital dashboard with summaries of key performance indicators in their particular area of responsibility. This can be areas such as a store or department, specific merchandise, loss prevention, risk, or cash flow. Whenever the actual performance falls below a preset threshold, BI can send alerts to the managers indicating a potential problem. This gives the manager the ability to monitor performance metrics, analyze information, make proactive decisions, and act on those decisions. The manager can visualize the impact of the changes made as a result of the changing performance indicators on the dashboard.
- **Finance:** BI provides immediate access to financial budgeting and forecasting data. That enables business decisions to be made based on current and accurate financial data, such as personalized views of revenue information by product, customer, merchandise, region, store, and time period. It also enables business decision makers to develop trend revenue forecasts with accuracy and speed, to compare and contrast revenues with the goals, and identify the areas in which the company is performing better or worse. Management at that point is well-armed with accurate information so they can act effectively and in a timely fashion.
- **Customer Service:** BI helps organizations assess various market segments and customers, identify potential new business, and retain existing business. For example, you can directly correlate customer information with the sales information. Companies that have focused on customer service and customer relationships have typically realized significant competitive advantage.
- **Human Resources:** BI supports activities such as recruitment, employee retention, and career development. Organizations can align staffing needs with strategic goals by mapping those goals to the skills needed to achieve them. They can then identify the recruitment practices required to bring high

quality individuals on board and identify new potential candidates for management. BI also provides information critical for areas, such as compensation planning, employee benefit planning, productivity planning, and skills rating.

- **Marketing:** One of the most important beneficiaries of BI in any company is the marketing department. BI helps them identify various trends in business, analyze revenues on a daily basis, identify high performers, quantify the impact of price changes, and identify opportunities for growth. For example, a trucking company can analyze fuel rate increases to determine such things as the impact on revenue and profitability, identify heavy/low impact segments, obtain optimized trucking routes, and identify multi-modal routing opportunities.

2.1.4 Drivers

Business intelligence permeates every area of a business enterprise. It is the need for more and more information that drives it. Those who have access to more and accurate information are in a position to enable better decision-making. Information is a significant business and competitive advantage.

Business measurements

There is an ongoing and ever demanding need for businesses to increase revenues, reduce costs, and compete more effectively. Companies today are under extreme pressure to deploy applications rapidly, and provide business users with easy and faster access to business information that will help them make the best possible decisions in a timely manner.

Business complexity

Companies today are offering and supporting an ever growing range of products and services to a larger and more diverse set of clients than ever before. BI provides the sophisticated information analysis capabilities required to handle those business complexities, such as mergers, acquisitions, and deregulation.

Business costs

There is a growing need to reduce the overhead of IT expenditures in every company. There is increased pressure on IT to leverage their existing systems to maximum benefit for the business. BI widens the scope of that ability by enabling access not only to operational and data warehouse data, but also to external data.

2.2 Key business initiatives

For example, we provide a brief summary of three of the key initiatives. They are quickly becoming the differentiators as competition becomes more and more intense. This not only applies to revenues from products and services, but also to the internal processes and operations that can minimize production delivery schedules and impact cost. These internal goals are significantly increasing in importance, because they can dramatically impact the ability of the company to meet their business performance measurements.

Business measurements and goals have come under more close scrutiny by management and stakeholders, and the measurement time frames are much shorter. For example, management must now be in a position to track and report performance against quarterly goals. Missing those goals can result in a significant response from the marketplace - and can typically be directly reflected in a changing stakeholder value assessment.

2.2.1 Business performance management

In the dynamic business environment, increased stakeholder value has become the main means by which business executives are measured. The ability to improve business performance is therefore a critical requirement for organizations. Failure to improve business performance is under close scrutiny by stakeholders. Their voices are heard through the buying or selling of company stock. One result of this is increased volatility of stock prices, which creates a tense roller-coaster ride for executives. Bringing more pressure to bear, is the fact that business performance measurement time frames are becoming ever shorter. Quarterly targets have replaced annual ones, and the expectation of growth and success is there at every quarter end.

To help smooth out the roller-coaster ride, businesses must react quickly to accommodate changing marketplace demands and needs. Flexibility and business agility are key to remaining competitive and viable. We need a holistic approach that enables companies to align strategic and operational objectives in order to fully manage achievement of their business performance measurements.

The objective of BPM is to help companies improve and optimize their operations across all aspects of the business. Business requirements, therefore, determine what type of BPM environment is necessary. Implementing BPM, however, is more than just about installing new technology, it also requires organizations to review the business environment to determine if changes are required to existing business processes to take advantage of the benefits that BPM can provide.

To become more proactive and responsive, businesses need the information that gives them a single view of their enterprise. With that, they can:

- Make more informed and effective decisions
- Manage business operations and minimize disruptions
- Align strategic objectives and priorities both vertically and horizontally throughout the business
- Establish a business environment that fosters continuous innovation and improvement

The need to continuously refine business goals and strategies, however, requires an IT system that can absorb these changes and help business users optimize business processes to satisfy business objectives. BPM assists here by providing performance metrics, or key performance indicators (KPIs), which businesses can employ to evaluate business performance. A KPI is a performance metric for a specific business activity that has an associated business goal or threshold. The goal or threshold is used to determine whether or not the business activity is performing within accepted limits. The tracking and analysis of KPIs provides business users with the insight required for business performance optimization.

BPM also becomes a great guide for IT departments that are asked to do more with less. It helps them focus their resources in areas that provide the most support to enable management to meet their business goals. They can now prioritize their tasks and focus on those aligned with meeting business measurements and achieving the business goals.

BPM requires a common business and technical environment that can support the many tasks associated with performance management. These tasks include planning, budgeting, forecasting, modeling, monitoring, analysis, and so forth.

Business integration and business intelligence applications and tools work together to provide the information required to develop and monitor business KPIs. When an activity is outside the KPI limits, alerts can be generated to notify business users that corrective action needs to be taken. Business intelligence tools are used to display KPIs and alerts, and guide business users in taking appropriate action to correct business problems. To enable a BPM environment, organizations may need to improve their business integration and business intelligence systems to provide proactive and personalized analytics and reports.

Simply stated, BPM is a process that enables you to meet your business performance measurements and objectives. A BPM solution enables that process. It enables you to proactively monitor and manage your business processes, and take the appropriate actions that result in you meeting your objectives.

There are a few words in the previous statement that require you to take action. Here are a few examples:

- Monitor your processes. This means you have well-defined processes, and a way to monitor them. And the monitoring should be performed on a continuous basis.
- Manage your processes. You must be aware of their status, on a continuous basis. That means you must be notified when the process is not performing satisfactorily. In your well-defined process, you need to define when performance becomes unsatisfactory. Then you must get notified so you can take action.
- Appropriate action. You need the knowledge, flexibility, and capability to take the appropriate action to correct problems that arise.

BPM places enormous demands on BI applications and their associated data warehouses to deliver the right information at the right time to the right people. To support this, companies are evolving their BI environments to provide (in addition to historical data warehouse functionality) a high-performance and enterprise-wide analytical platform with real-time capabilities. For many organizations this is a sizable challenge, but BPM can be the incentive to move the BI environment to a new level of capability.

IBM has also developed a BPM Framework that enables the assembly of components encompassing business partner products and IBM foundation technologies. The framework includes a wide range of capabilities for modeling, integrating, connecting, monitoring, and managing business operations within an enterprise and across a value chain of trading partners and customers. The unifying framework that accommodates the IBM framework is illustrated in Figure 2-3 on page 34. This framework identifies the functional components required for the real-time monitoring, analysis, and optimization of business operations, and their underlying IT infrastructure.

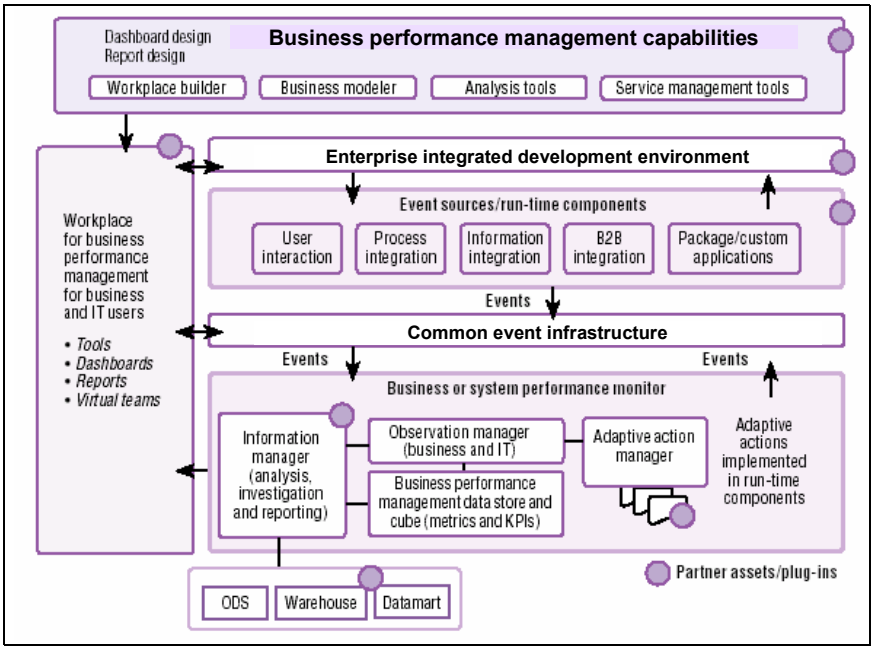


Figure 2-3 IBM BPM framework

Creating a unified framework is critical to the success of a BPM implementation. BPM is a paradigm that succeeds from proactively managing the environment, rather than reactively resolving individual business issues. There is a need, therefore, for an architected solution that consolidates and integrates data related to monitoring, events, alarms, and situation-related information across the enterprise. Fracturing that enterprise view with isolated data silos defeats the purpose of BPM. That is, it actually inhibits the ability to monitor and manage enterprise-wide business performance.

You can satisfy the need for a unified solution by integrating the required data in a DB2 data warehouse and using the data warehouse to feed the BPM environment. If this is not feasible, use information integration technologies, such as the IBM WebSphere Information Integrator, to create an integrated business view of disparate source data. Using dependent data marts, that leverage data from a data warehouse, is another possible solution. We do not recommend, however, building a BPM solution using data from independent data marts, or sourcing the data directly out of the operational systems. These approaches involve isolated data silos that can lead to inconsistencies and inaccurate results.

IBM BPM solutions are based on DB2 relational databases, as well as a combination of DB2 relational databases and DB2 OLAP databases. In most cases, the BPM environment involves a multi-tier approach, consisting of existing

applications, data warehouses that feed the BPM solution, BPM tools, and packaged applications.

Implementing a BPM system results in making business performance data available to everyone that needs it. Usually, most of this performance data has not been available to business users prior to the BPM implementation. A BPM solution is typically facilitated by providing the proactive distribution of the data through graphical dashboards, rather than relying on users to search for data they require. Most users respond positively to graphical dashboards embedded in enterprise portals, and require little if any training in how to use them.

BPM and BI

Any BI implementation is aimed at turning available data into information and putting it into the hands of decision makers. It might be easy to conclude therefore that BI and BPM are the same thing. BPM is focused, however, on a subset of the information delivered by a BI system. BPM is concerned with information that shows business performance and indicates business success or failure. This information subset enables organizations to focus on the important task of optimizing business performance.

BI enables businesses to access, analyze, and use their data for decision making purposes. It is used for long-term strategic planning, short-term tactical analysis, and for managing the daily operational business activities. Key developments in the use of BI include:

- Tactical and strategic BI are moving closer together. This is because strategic time frames (budgeting and forecasting cycles, for example) are shrinking to enable companies to become more responsive to business needs and customer requirements.
- Analytic applications are used more and more for *proactively* delivering business intelligence to users, rather than requiring them to discover it for themselves. In many cases, these applications not only deliver information about business operations, but also put actual business performance into *context* by comparing it against business plans, budgets, and forecasts.
- Dashboards are the becoming the preferred method for delivering and displaying business intelligence to users. Dashboards are more visual and intuitive, and typically provide linkages that can enable people to take immediate action.
- Business rules are a key requirement as companies implement so-called *closed-loop* processing to use the results of business intelligence processing to optimize business operations. This is particularly true when there is a requirement to support automated decisions, recommendations, and actions.

- Close to real-time (near real-time), or *low-latency,* business information is becoming an important requirement as organizations increasingly make use of business intelligence for managing and driving daily business operations.

BPM forms the underpinnings for many of the BI developments we outline. Specifically, BPM provides the BI to improve operational decision making, become more proactive and timely, and support a wide range of business functions.

BPM enables a BI system to tap into business events flowing through business processes, and to measure and monitor business performance. BPM extends traditional operational transaction processing by relating measures of business performance to specific business goals and objectives. User alerts and application messages can then inform business analysts and applications about any situations that require attention. The integration of business process monitoring with operational BI analytics enables a *closed-loop solution*.

2.2.2 Real-time business intelligence

As business and technology continue to change and improve, a new phenomena is occurring. The two originally opposite ends of the business intelligence spectrum, namely tactical and strategic decision-making, are becoming much more closely aligned.

There is a merging of requirements and a need for current information by everyone. It is fueling and enabling the acceleration toward another key capability that is called *closed-loop analytics*. That means the results of data warehousing, or business intelligence, analytics are being fed back to the operational environment. Events can now be acted upon almost immediately, avoiding costly problems rather than simply trying to minimize their impact. This is a significant leap forward, and can now be realized with the capability to support real time.

The Internet is also playing a key role in this movement. For example, investors now have access to information as never before. And, they can move their money in and out of investments quickly. If one company is not performing, they can quickly and easily move their investment to another. Now everyone seems to have the *need for speed*. They want results, and they want them now! They want information, and they want it in *real time*!

To meet the requirements and support this change in thinking and measurement, companies are moving towards real time. Change, flexibility, and speed are now key requirements for staying in business. Typically, tactical (or short-term) decisions are made from the operational systems. However, that data usually requires a good deal of analysis and time to provide benefit. And strategic (or

long-term) decisions are made from historical data - which may exist in a data warehouse. This process must change to support real time.

The direction is to make data for both types of decision-making available, or accessible, from the data warehousing environment. Then you can combine, analyze, and present it in a consistent manner to enable more informed management decision-making. The data warehousing environment is now fulfilling its destiny as the enterprise data repository. It is the base for all business intelligence systems. However, that means you must get the data into the data warehouse in a much more timely manner. Can you do it in real time?

The answer is yes, depending on your definition of real time. Regardless, the movement to real time has be an evolution, because it requires a gradual re-architecting of the company infrastructure and business process re-engineering. The evolution needs to be based on business requirements and priorities, and a sound business case. Companies will move a step at a time along this evolution.

The various processes that supply data can work independently or in concert with one another. One differentiator that impacts the list of processes for *Getting Data Into* the data warehouse is identified in the term *Continuous Loading* that is depicted in front of these processes in Figure 2-4 on page 38. That is a key requirement for enabling real time, and brings with it a number of issues and required product or implementation capabilities.

Getting the data into the data warehouse is, of course, of utmost importance. Without it, there is no business intelligence. However, the real value of data warehousing and BI lies on the right side of Figure 2-4 on page 38. That is the category of *Getting Data Out.* That data provides the information that enables management to make more informed decisions. And, the key now is that management works with data that is genuinely current, that is, real-time, or near real-time, data.

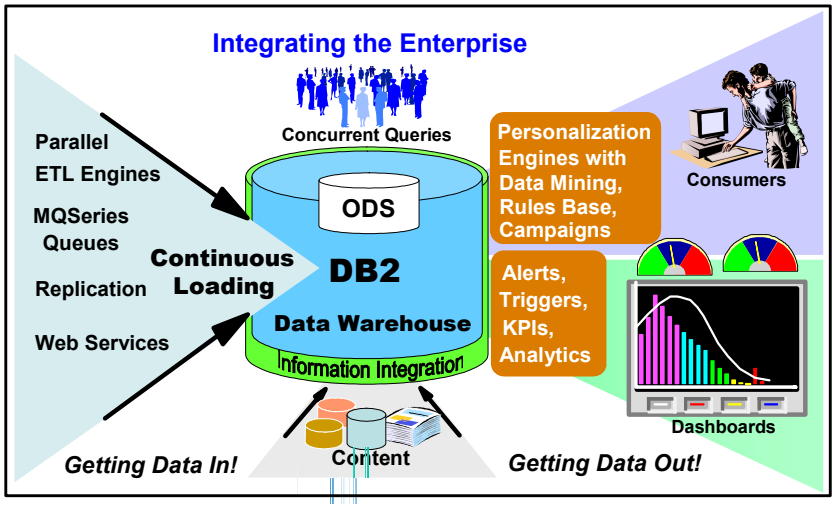


Figure 2-4 Real-time BI

To make use of the current data, users must be aware of it. Many implementations create the environment, and then simply enable users with query capability. To increase the value received, the environment needs to be proactive. Enable the environment to recognize important events and alert users automatically so they can take corrective action. Or, the environment must display current status as changes occur rather than requiring a user to ask for current status. To enable this capability, you can define key performance indicators (KPI) with threshold values as one way to trigger such alerts. When a defined threshold is exceeded, the system could automatically trigger an alert.

This capability will typically require a change in your business processes. Logic must be included in the process and imbedded in your applications. It is not sufficient for a user to run queries or look at reports to try to identify when processes need attention. That decision process should be added to your application software so it can automatically be ever watchful for those processes that exceed specific thresholds. And notify you in real time! That is a vital and very beneficial value of real-time processing. The objective is to avoid problems rather than just reacting to them. To avoid costly problems and issues rather than simply trying to minimize their impact.

To implement these added capabilities requires new types of applications, such as analytic applications and interactive information dashboards. They can monitor processes, detect values and compare to thresholds, and display or deliver alerts and associated real-time data to the users for immediate action. Better still, put logic into the analytic applications to enable them to do much, or all, of the decision processing and action initiation. This capability would require

a good rules base for automated action taking. For situations requiring manual intervention, provide guidance. That guidance (or guided analysis) could be problem solving logic embedded in the analytic application. These are capabilities that can enable, initiate, and/or facilitate closed-loop processing. And it is a significant competitive advantage.

Implementing real-time BI

The implementation of near real-time BI involves the integration of a number of activities. These activities are required in any data warehousing or BI implementation, but now we have elevated the importance of the element of time. The traditional activity categories of *getting data in* and *getting data out* are still valid. But, now they are ongoing continuous processes rather than a set of steps performed independently.

Figure 2-4 on page 38 depicts an overall view of real-time BI. On one side, it shows the various techniques for getting data into the data warehouse from the various data sources and integrating it. We see the following examples:

- Parallel ETL Engines: Data must be extracted from the operational environment, cleansed, and transformed before it can be placed in the data warehouse in order to be usable for query processing and analysis. These extract, load, transform (ETL) processes have historically been batch-oriented. Now they must be altered to run on a continuous or near-continuous basis. Running multiples of these processes in parallel is another alternative for getting data into the data warehouse as quickly as possible. Another approach is to extract and load the data, then perform any required transformations after the data is in the data warehousing environment. This extract, load, transform (ELT) process can result in updating the data warehouse in a shorter time frame.
- MQSeries® queues: These queues are key in a real-time environment. They are part of a messaging-oriented infrastructure, in this case delivered by the WebSphere family of products. Messaging systems assume the responsibility for delivery of the messages (data) put into their queues. They guarantee data delivery. This is extremely important because it relieves application programmers of that responsibility, which can significantly improve the application programmers' productivity and translate into reduced development costs. As a general rule, it is always better to have systems software do as much as possible in the application development process. The result can be faster application delivery and at a lower cost.
- Replication: WebSphere Information Integrator - Replication Edition delivers this capability. The capture component of replication takes the required data from system logs, which eliminates the need for application development to satisfy the requirement. It is another example of having the system provide the capability you need rather than developing, and maintaining, it yourself.

- Web services: This is a strategy where you acquire and use specific services by downloading them from the Web. This makes them generally available to everyone for use in the application development process. It is another example of providing tested, reusable application modules when and where they are needed. And, it represents a potential reduction in application development costs.
- Information Integration: Data must also be available from external data sources, many of which are housed in a heterogeneous data environment. WebSphere Information Integrator is a product that can more easily enable you to access and acquire the data from these sources and use them for updating the data warehouse.
- The Operational Data Store (ODS) is another source of input to the data warehouse, but is unique in that it can concurrently be used by the operational systems. It typically contains data that comes from the operational transaction environment, but has been cleansed and prepared for inclusion in the data warehouse. The ODS is typically a separate sub-schema, or set of tables, and itself a driver for continuous loading of the real-time data warehouse. As such it is then considered as part of the data warehousing environment.

To summarize, real-time business intelligence is having access to information about business actions as soon after the fact as is justifiable based on the requirements. This enables access to the data for analysis and its input to the management business decision-making process soon enough to satisfy the requirement.

2.2.3 Data mart consolidation

A *data mart* is a construct that evolved from the concepts of data warehousing. The implementation of a data warehousing environment can be a significant undertaking, and is typically developed over a period of time. Many departments and business areas were anxious to get the benefits of data warehousing, and reluctant to wait for the natural evolution.

The concept of a data mart exists to satisfy the needs of these departments. Simplistically, a data mart is a small data warehouse built to satisfy the needs of a particular department or business area. Often the data mart was developed by resources external to IT, and paid for by the implementing department or business area to enable a faster implementation.

The data mart typically contains a subset of corporate data that is valuable to a specific business unit, department, or set of users. This subset consists of historical, summarized, and possibly detailed data captured from transaction processing systems (called *independent data marts*), or from an existing enterprise data warehouse (called *dependent data marts*). It is important to

realize that the *functional scope of the data mart's users* defines the data mart, not the size of the data mart database.

Figure 2-5 depicts a data warehouse architecture, with both dependent and independent data marts.

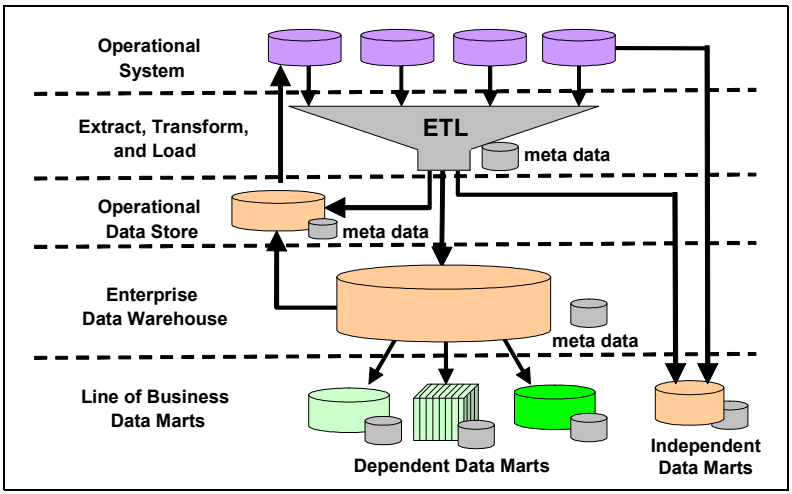


Figure 2-5 Data warehouse architecture with data marts

As you can see, there are a number of options for architecting a data mart. For example:

- Data can come directly from one or more of the databases in the operational systems, with few or no changes to the data in format or structure. This limits the types and scope of analysis that can be performed. For example, you can see that in this option, there may be no interaction with the data warehouse meta data. This can result in data consistency issues.
- Data can be extracted from the operational systems and transformed to provide a cleansed and enhanced set of data to be loaded into the data mart by passing through an ETL process. Although the data is enhanced, it is not consistent with, or in sync with, data from the data warehouse.
- Bypassing the data warehouse leads to the creation of an independent data mart. It is not consistent, at any level, with the data in the data warehouse. This is another issue impacting the credibility of reporting.
- Cleansed and transformed operational data flows into the data warehouse. From there, dependent data marts can be created, or updated. It is key that updates to the data marts are made during the update cycle of the data warehouse to maintain consistency between them. This is also a major consideration and design point, as you move to a real-time environment. At

that time, it is good to revisit the requirements for the data mart, to see if they are still valid.

However, there are also many other data structures that can be part of the data warehousing environment and used for data analysis, and they use differing implementation techniques. These fall in a category we are simply calling *analytic structures*. However, based on their purpose, they could be thought of as data marts. They include structures and techniques, such as:

- Materialized query tables (MQT)
- Multidimensional clustering (MDC)
- Summary tables
- Spreadsheets
- OLAP (Online Analytical Processing) databases
- Operational data stores
- Federated databases

Although data marts can be of great value, there are also issues of currency and consistency. This has resulted in recent initiatives designed to minimize the number of data marts in a company. This is referred to as *data mart consolidation (DMC)*.

Data mart consolidation may sound simple at first, but there are many things to consider. A critical requirement, as with almost any project, is executive sponsorship, because you will be changing many existing systems on which people have come to rely, even though the systems may be inadequate or outmoded. To do this requires serious support from senior management. They will be able to focus on the bigger picture and bottom-line benefits, and exercise the authority that will enable making changes.

To help with this initiative, we constructed a data mart consolidation life cycle. I Figure 2-6 on page 43 depicts a data mart consolidation life cycle.

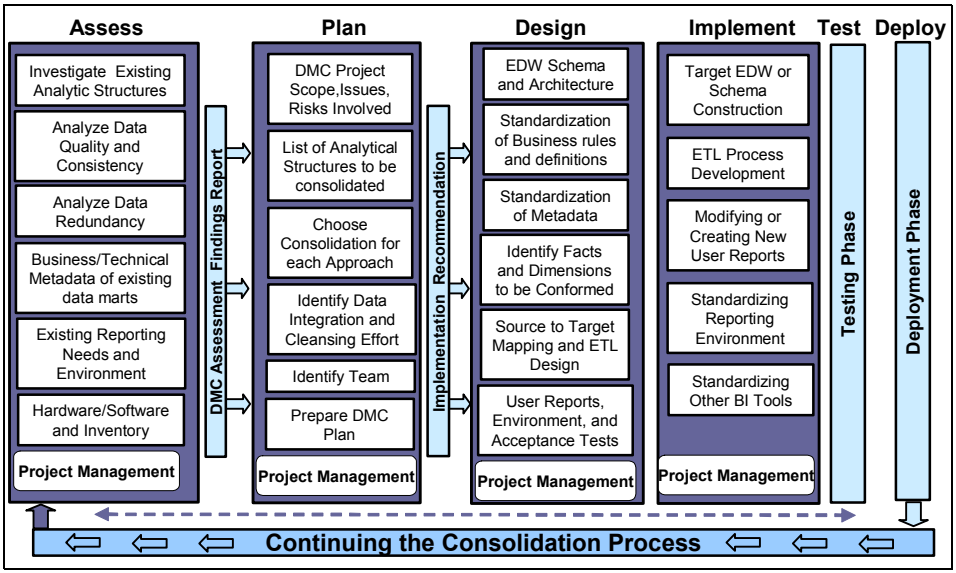


Figure 2-6 Data mart consolidation life cycle

In the following section, we give you a brief overview of the DMC process. The data mart consolidation life cycle consists of the following activities:

- **Assessment:** During this phase, we assess the following topics:
  - Existing analytical structures
  - Data quality and consistency
  - Data redundancy
  - Source systems involved
  - Business and technical meta data
  - Existing reporting needs
  - Reporting tools and environment
  - Other BI tools
  - Hardware, software, and other inventory

Note: Based on the assessment phase, create the “DMC Assessment Findings” report.

- **Planning:** Key activities in the planning phase include:
  - Identifying business sponsor
  - Identifying analytical structures to be consolidated
  - Selecting the consolidation approach
  - Defining the DMC project purpose and objectives
  - Defining the scope

  - Identifying risks, constraints, and concerns
  - In the planning phase above, based on the DMC Assessment Findings report, create the Implementation Recommendation report.
- **Design:** Key activities involved in this phase are:
  - Target EDW schema design
  - Standardization of business rules and definitions
  - Meta data standardization
  - Identify dimensions and facts to be conformed
  - Source to target mapping
  - ETL design
  - User reports
- **Implementation:** The implementation phase includes the following activities:
  - Target schema construction
  - ETL process development
  - Modifying or adding user reports
  - Standardizing reporting environment
  - Standardizing other BI tools
- **Testing:** This may include running in parallel with production.
- **Deployment:** This will include user acceptance testing.
- **Loopback:** Continuing the consolidation process, which loops you back to start through some, or all, of the process again.

When you understand all the variables that enter into a data mart consolidation project, you can see how it can become a rather daunting task. There are not only numerous data sources with which to deal, there is the heterogeneity. That is, data and data storage products from numerous different vendors. Consider that they all have underlying data models that must be consolidated and integrated. It can be a significant integration (consolidation) project.

2.2.4 The impact of dimensional modeling

We have discussed three key BI initiatives. They are all powerful and can provide significant benefits to the implementing companies. The common thread among all of them is the need for data. Well, the actual need is information, but that comes from the data. And each of those initiatives not only use data, but they also provide it.

That data typically comes from, and goes into, the enterprise data warehouse. That is the one common and consistent source of data in an enterprise. And as we all know, BI is built on, integrated with, and dependent on, the data warehouse. Therefore, it should not be a surprise when we state that all

companies need a robust and well designed data warehouse if they want to be successful, and survive.

And one thing you should understand from reading this redbook, is that the unifying structure of the data warehouse is the data model. It is the glue that holds everything together, and so is deserving of significant attention. And to get the information from the data warehouse, you need business intelligence solutions.

You can begin to get a better appreciation for the impact of data modeling as you look closer at data mart consolidation. This initiative involves merging data (and the associated data models) from multiple data marts. And these data marts can exist on many heterogeneous platforms and reside in many different databases and data structures from many different vendors. As you look at the formats, data types, and data definitions from these heterogeneous environments, you quickly see the beginning of a complex task, and that is the integration of all these heterogeneous components and elements, along with all the associated applications and queries.

From all this it should become clear that the role of the data modeler is highly significant and of utmost importance in the quest for the integrated real-time enterprise.

3

Data modeling: The organizing structure

Data modeling is important because it specifies the data structure, which can impact all aspects of data usage. For example, it can have a significant impact on performance. This is particularly true with data warehousing. And, the data warehouse is the primary structural element in business intelligence.

Key to business intelligence is the ability to analyze huge volumes of data, typically by means of query processing and analytic applications. And for this, performance is critical. We provide more detail on this topic in this chapter.

In addition, we discuss the topics of:

- The primary data modeling techniques (E/R and dimensional modeling)
- Data warehouse (DW) architecture choices and the data models involved:
  - Enterprise data warehouse
  - Independent data marts
  - Dependent data marts
- The data modeling life cycle for the data warehouse, which includes both logical and physical modeling

3.1 The importance of data modeling

Generally speaking, a model is an abstraction and reflection of the real world. Modeling gives us the ability to visualize what we cannot yet realize. It is the same with data modeling. The primary aim of a data model is to make sure that all data objects required by the business are accurately and fully represented.

From the business perspective, a data model can be easily verified because the model is built by using notations and language which are easy to understand and decipher.

However, from a technical perspective the data model is also detailed enough to serve as a blueprint for the DBA when building the physical database. For example, the model can easily be used to define the key elements, such as the primary keys, foreign keys, and tables that will be used in the design of the data structure.

Different approaches of data modeling

Data models are about capturing and presenting information. Every organization has information that is typically either in the operational form (such as OLTP applications) or the informational form (such as the data warehouse).

Traditionally, data modelers have made use of the E/R diagram, developed as part of the data modeling process, as a communication media with the business analysts. The focus of the E/R model is to capture the relationships between various entities of the organization or process for which we design the model. The E/R diagram is a tool that can help in the analysis of business requirements and in the design of the resulting data structure.

However, the focus of the dimensional model is on the business. Dimensional modeling gives us an improved capability to visualize the very abstract questions that the business analysts are required to answer. Utilizing dimensional modeling, analysts can easily understand and navigate the data structure and fully exploit the data. Actually, data is simply a record of all business activities, resources, and results of the organization. The data model is a well-organized abstraction of that data. So, it is quite natural that the data model has become the best method for understanding and managing the business of the organization. Without a data model, it would be very difficult to organize the structure and contents of the data in the data warehouse.

E/R and dimensional modeling, although related, are extremely different. Of course, all dimensional models are also really E/R models. However, when we refer to E/R models in this book, we mean *normalized* E/R models. Dimensional models are *denormalized*.

There is much debate about which method is best and the conditions under which you should select a particular technique. People use E/R modeling primarily when designing for highly transaction-oriented OLTP applications. When working with data warehousing applications, E/R modeling may be good for reporting and fixed queries, but dimensional modeling is typically better for ad hoc query and analysis.

For the OLTP applications, the goal of a well-designed E/R data model is to efficiently and quickly get the data inside (Insert, Update, Delete) the database. However, on the data warehousing side, the goal of the data model (dimensional) is to get the data out (select) of the warehouse.

In the following sections, we review and define the modeling techniques and provide selection guidelines. We also define how to use the data modeling techniques (E/R and Dimensional Modeling) together or independently for various data warehouse architectures. We discuss those architectures in 3.3, “Data warehouse architecture choices” on page 57.

3.2 Data modeling techniques

In Chapter 1, “Introduction” on page 1, we briefly discussed E/R and dimensional data modeling techniques. In this section, we discuss these important data modeling techniques in more detail, to understand the advantages and disadvantages of each.

3.2.1 E/R modeling

E/R modeling is a design technique in which we store the data in highly normalized form inside a relational database. Figure 3-1 on page 50 shows a *visualization* of a normalized E/R model. It is simply to depict how the various tables in an E/R model connect and interrelate. It is called a *normalized* structure. Normalization basically involves splitting large tables of data into smaller and smaller tables, until you have tables where no column is functionally dependent on any other column, each row consists of a single primary key and a set of totally independent attributes of the object that are identified by the primary key. This type of structure is said to be in third normal form (3NF).

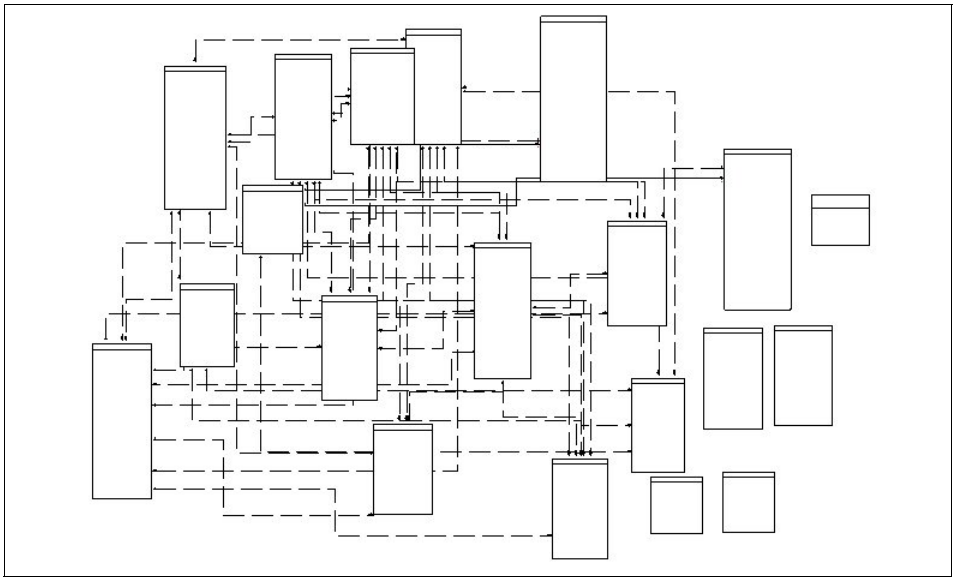


Figure 3-1 A typical E/R model

The objective of normalization is to minimize redundancy by not having the same data stored in multiple tables. As a result, normalization can minimize any integrity issues because SQL updates then only need to be applied to a single table. However, queries, particularly those involving very large tables, that include a join of the data stored in multiple normalized tables may require additional effort and programming to achieve acceptable performance.

Although data in normalized tables is a very pure form of data and minimizes redundancy, it can be a challenge for analysts to navigate. For example, if an analyst must navigate a data model that requires a join of 15 tables, it may likely be difficult and not very intuitive. This is one issue that is mitigated with a dimensional model, because it has standard and independent dimensions.

We strongly recommend third normal form for OLTP applications since data integrity requirements are stringent, and joins involving large numbers of rows are minimal. Data warehousing applications, on the other hand, are mostly read only, and therefore typically can benefit from *denormalization*. Denormalization is a technique that involves duplicating data in one or more tables to minimize or eliminate time consuming joins. In these cases, adequate controls must be put in place to ensure that the duplicated data is always consistent in all tables to avoid data integrity issues.

Note: A table is in third normal form (3NF) if each non-key column is independent of other non-key columns, and is dependent only on the key. Another much-used shorthand way of defining third normal form is to say, *it is the key, the whole key, and nothing but the key.*

The E/R model basically focuses on three things, entities, attributes, and relationships. An entity is any category of an object in which the business is interested. Each entity has a corresponding business definition, which is used to define the boundaries of the entity — allowing you to decide whether a particular object belongs to that category or entity. Figure 3-2 depicts an entity called *product*.

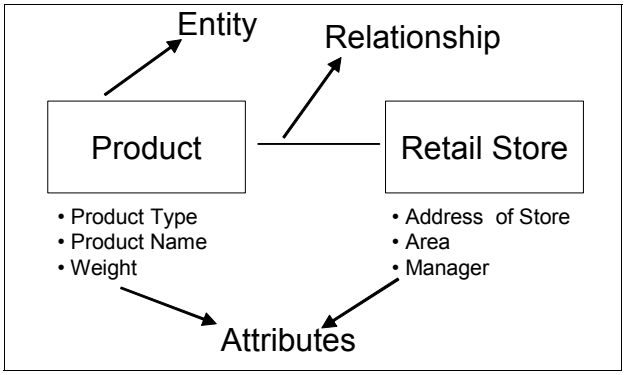


Figure 3-2 Example E/R model showing relationships

Product is defined as any physical item that may be stocked in one or more of the retail stores in the company. Whether this definition is appropriate or not depends on the use to which the model is put. In this sense, an entity may be quite specific at one extreme, or very generic at the other extreme. Each entity has a number of attributes associated with it. An *attribute* is any characteristic of an entity that describes it and is of interest to the business.

A *relationship* that exists between the entities in a model describes how the entities interact. This interaction is usually expressed as a verb. In the example, the relationship between *Product* and *Retail Store* is defined as *the retail store stocks product*.

In summary, here are advantages of the E/R modeling technique:

- It eliminates redundant data, which saves storage space, and better enables enforcement of integrity constraints.

- The INSERT, UPDATE, and DELETE commands executed on a normalized E/R model are much faster than on a denormalized model because there are fewer redundant sources of the data, resulting in fewer executions.
- The E/R modeling technique helps capture the interrelationships among various entities for which you are designing the database. In other words, an E/R model is very good at representing relationships.

A disadvantage of an E/R model is that it is not as efficient when performing very large queries involving multiple tables. In other words, an E/R model is good at INSERT, UPDATE, or DELETE processing, but not as good for SELECT processing.

3.2.2 Dimensional modeling

To overcome performance issues for large queries in the data warehouse, we use dimensional models. The dimensional modeling approach provides a way to improve query performance for summary reports without affecting data integrity. However, that performance comes with a cost for extra storage space. A dimensional database generally requires much more space than its relational counterpart. However, with the ever decreasing costs of storage space, that cost is becoming less significant.

What is a dimensional model?

A dimensional model is also commonly called a *star schema*. This type of model is very popular in data warehousing because it can provide much better query performance, especially on very large queries, than an E/R model. However, it also has the major benefit of being easier to understand. It consists, typically, of a large table of facts (known as a *fact table*), with a number of other tables surrounding it that contain descriptive data, called *dimensions*. When it is drawn, it resembles the shape of a star, therefore the name. Figure 3-3 on page 53 depicts an example star schema.

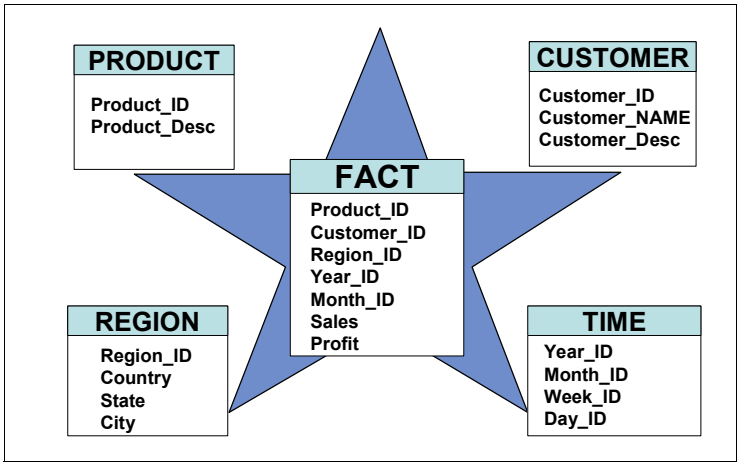


Figure 3-3 A star schema or a dimensional model

The dimensional model consists of two types of tables having different characteristics. They are:

- Fact table
- Dimension table

The following sections provide more detail for understanding the two types of tables. Figure 3-4 on page 54 depicts an example of the fact table structure.

Fact table characteristics

- The fact table contains numerical values of what you measure. For example, a fact value of 20 might mean that 20 widgets have been sold.
- Each fact table contains the keys to associated dimension tables. These are called *foreign keys* in the fact table.
- Fact tables typically contain a small number of columns.
- Compared to dimension tables, fact tables have a large number of rows.
- The information in a fact table has characteristics, such as:
  - It is numerical and used to generate aggregates and summaries.
  - Data values need to be additive, or semi-additive, to enable summarization of a large number of values.
  - All facts in Segment 2 must refer directly to the dimension keys in Segment 1 of the structure, as you see in Figure 3-4 on page 54. This enables access to additional information from the dimension tables.

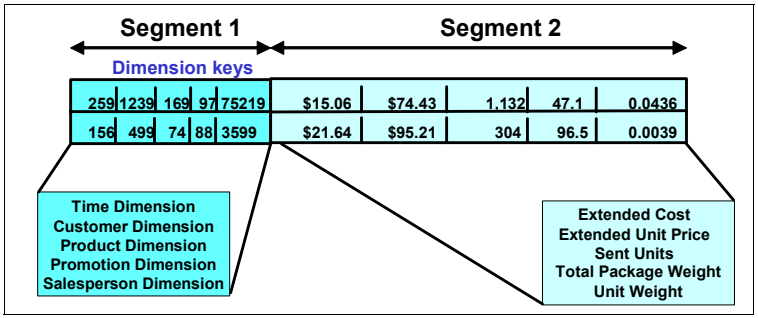


Figure 3-4 Fact table structure

– We have depicted examples of a *good* fact table design and a *bad* fact table design in Figure 3-5. The bad fact table contains data that does not follow the basic rules for fact table design. For example, the data elements in this table contain values that are:

  - Not numeric. Therefore, the data cannot be summarized.
  - Not additive. For example, the discounts and rebates are hidden in the unit price.
  - Not directly related to the given key structure, which means they cannot be not additive.

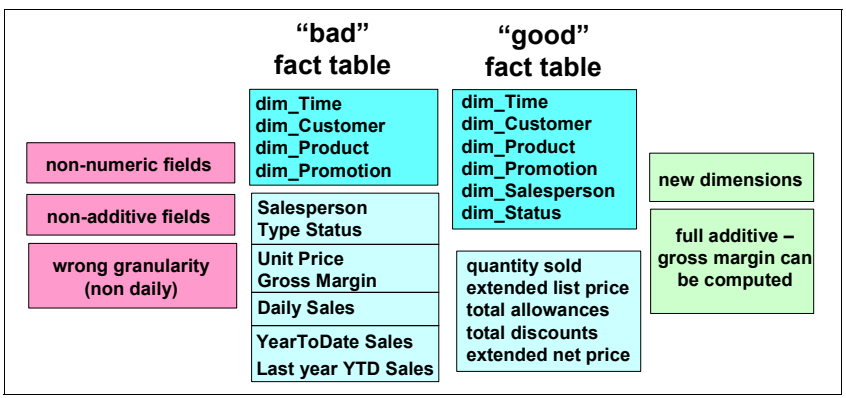


Figure 3-5 Good and bad fact table

We provide a more detailed discussion about fact table design in Chapter 5, "Dimensional Model Design Life Cycle" on page 103. We now look at the second component, the dimension table characteristics.

Dimension table characteristics

- Dimension tables contain the details about the facts. That, as an example, enables the business analysts to better understand the data and their reports.
- The dimension tables contain descriptive information about the numerical values in the fact table. That is, they contain the attributes of the facts. For example, the dimension tables for a marketing analysis application might include attributes such as time period, marketing region, and product type.
- Since the data in a dimension table is denormalized, it typically has a large number of columns.
- The dimension tables typically contain significantly fewer rows of data than the fact table.
- The attributes in a dimension table are typically used as row and column headings in a report or query results display. For example, the textual descriptions on a report come from dimension attributes. Figure 3-6 depicts an example of this.

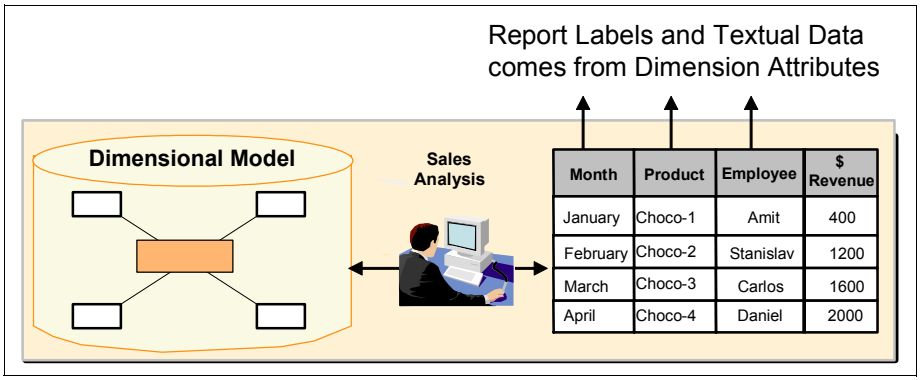


Figure 3-6 The textual data in the report comes from dimension attributes

Types of dimensional models

There are three basic types of dimensional models, and they are:

- Star model
- Snowflake model
- Multi-star model

Figure 3-7 on page 56 depicts these models:

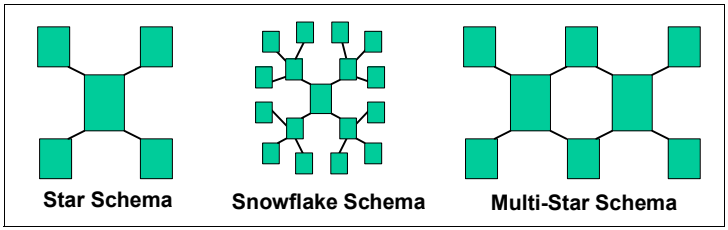


Figure 3-7 Types of dimensional models

In the following section, we give a brief summary of the three types of dimensional models:

- **Star model**: Star schemas have one fact table and several dimension tables. The dimension tables are not denormalized.
- **Snowflake model:** Further normalization and expansion of the dimension tables in a star schema result in the implementation of a snowflake design. A dimension is said to be snowflaked when the low-cardinality columns in the dimension have been removed to separate normalized tables that then link back into the original dimension table. Figure 3-8 depicts this.

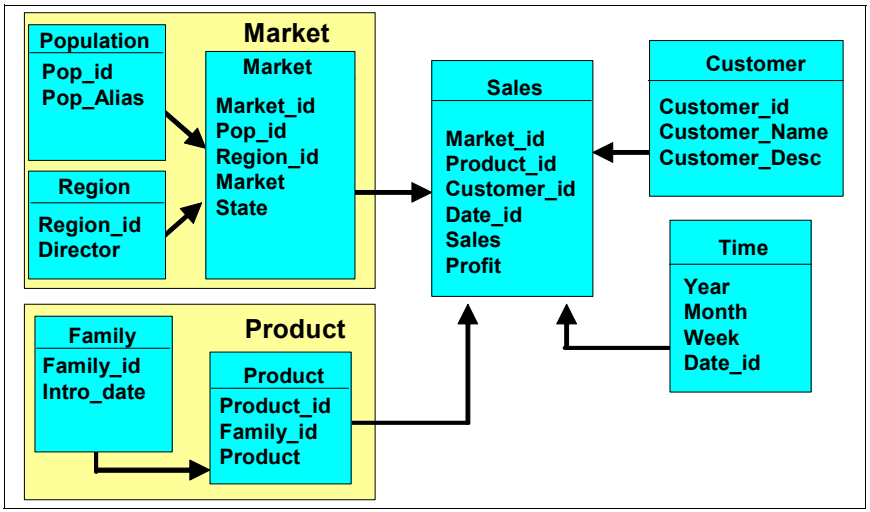


Figure 3-8 Snowflake schema

In this example, we expanded (snowflaked) the Product dimension by removing the low-cardinality elements pertaining to Family, and putting them in a separate Family dimension table. The Family table is linked to the Product dimension table by an index entry (Family_id) in both tables. From the Product dimension table, the Family attributes are extracted by, in this example, the Family Intro_date. The keys of the hierarchy (Family_Family_id) are also

included in the Family table. In a similar fashion, the Market dimension was snowflaked.

For a more detailed discussion of the snowflake schema, refer to 6.3.7, “Identifying dimensions that need to be snowflaked” on page 277.

- **Multi-star model:** A multi-star model is a dimensional model that consists of multiple fact tables, joined together through dimensions. Figure 3-9 depicts this, showing the two fact tables that were joined, which are EDW_Sales_Fact and EDW_Inventory_Fact.

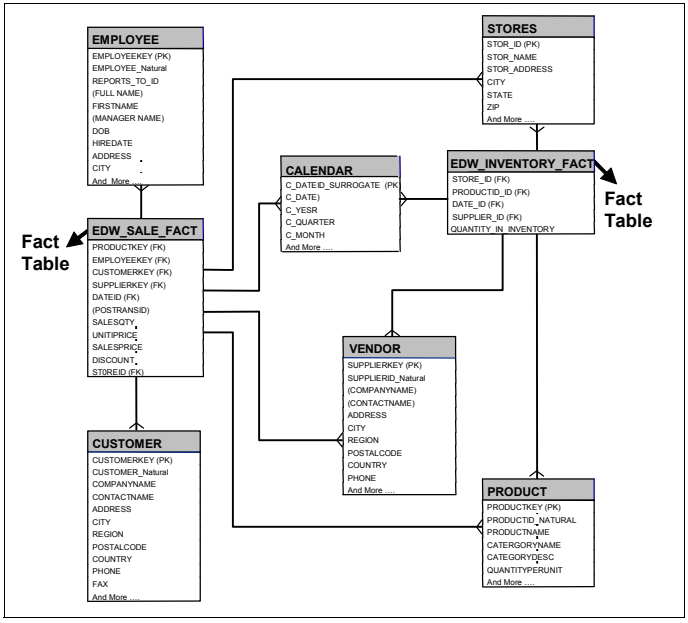


Figure 3-9 Multi-star model

3.3 Data warehouse architecture choices

In this section, we describe three architectural approaches for data warehousing. The approach, or combination of approaches, you select impact the data modeling requirement.

The data warehouse architecture will determine, or be determined by, the locations of the data warehouses and data marts themselves, and where the control resides. For example, the data can reside in a central location that is managed centrally. Alternatively, the data can reside in distributed local and/or remote locations that are either managed centrally or independently.

Here we describe three architectural approaches, listed below and depicted in Figure 3-10:

- Enterprise data warehouse (EDW)
- Independent data marts
- Dependent data marts

You can also use these architectural choices in combinations. The most typical example of which is combining the EDW and dependent data marts. In this instance, the data marts receive data from the EDW, rather than directly from the staging area.

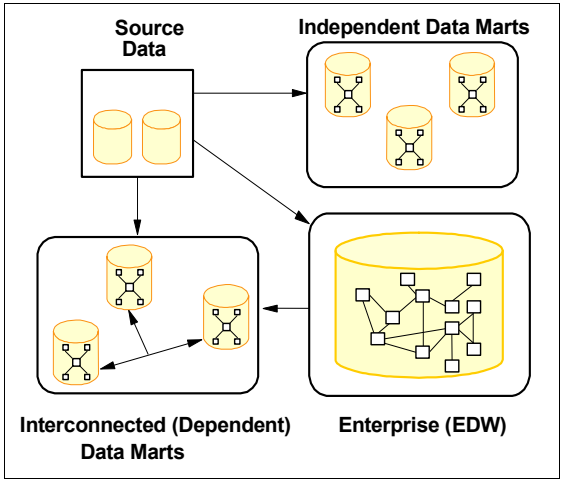


Figure 3-10 Various data warehouse architectures

3.3.1 Enterprise data warehouse

An *enterprise data warehouse* is one that will support all, or a large part, of the business requirement for a more fully integrated data warehousing environment that has a high degree of data access and usage across departments or lines of business. That is, the data warehouse is designed and constructed based on the needs of the business as a whole. Consider it a common repository for decision-support data that is available across the entire organization, or a large subset of that data. We use the term *Enterprise* here to reflect the scope of data access and usage, not the physical structure.

Figure 3-11 on page 59 shows an architecture diagram for an enterprise data warehouse.

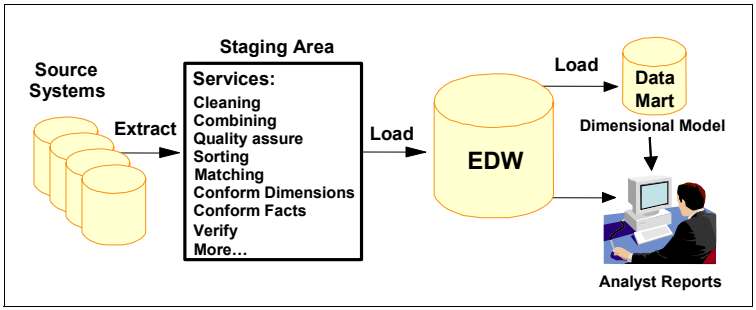


Figure 3-11 Enterprise data warehouse architecture

This type of data warehouse is characterized as having all the data under central management. However, centralization does not necessarily imply that all the data is in one location or in one common systems environment. That is, it is centralized, but logically centralized rather than physically centralized. When this is the case, by design, it then may also be referred to as a *hub and spoke* implementation. The key point is that the environment is managed as a single integrated entity.

Note: The enterprise data warehouse may also be called a *Hub and Spoke* data warehouse implementation if the control is logically centralized even if the data is spread out and physically distributed, such as the EDW and data marts, as shown in Figure 3-11.

3.3.2 Independent data mart architecture

An independent data mart architecture, as the name implies, is comprised of standalone data marts that are controlled by particular workgroups, departments, or lines of business. They are typically built solely to meet the particular needs of that particular workgroup, department, or line of business. Although there could be, there typically is no connectivity with data marts in other workgroups, departments, or lines of business. Therefore, these data marts do not share any conformed dimensions and conformed facts between them.

This is one of the concerns when using an independent data mart. The data in each may be at a different level of currency, and the data definitions may not be consistent - even for data elements with the same name.

For example, let us assume that data mart#1 and data mart#2, as shown in Figure 3-12 on page 60, have a customer dimension. However, since they do not share conformed dimensions, it means that these two data marts must each implement their own version of a customer dimension. It is these types of decisions that can lead, for example, to inconsistent and non-current sources of

data on independent data marts in an enterprise. And this can result in inconsistent and non-current sources of data that can lead to inaccurate decision making

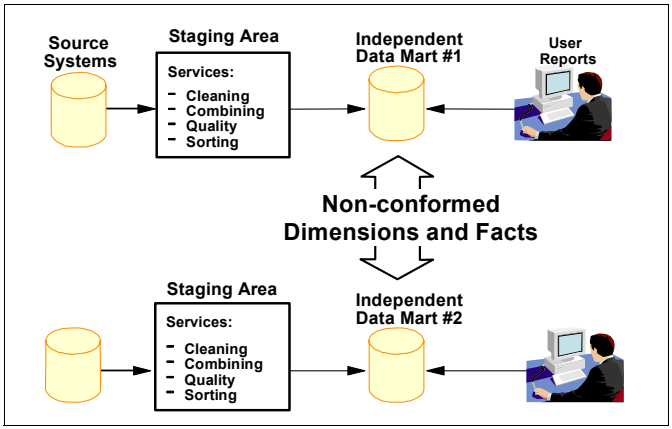


Figure 3-12 Independent data warehouse architecture

Independent data marts are primary candidates for data mart consolidation for companies around the world today. The proliferation of such independent data marts has resulted in issues such as:

- Increased hardware and software costs for the numerous data marts
- Increased resource requirements for support and maintenance
- Development of many extract, transform, and load (ETL) processes
- Many redundant and inconsistent implementations of the same data
- Lack of a common data model, and common data definitions, leading to inconsistent and inaccurate analyses and reports
- Time spent, and delays encountered, while deciding what data can be used, and for what purpose
- Concern and risk of making decisions based on data that may not be accurate, consistent, or current
- No data integration or consistency across the data marts
- Inconsistent reports due to the different levels of data currency stemming from differing update cycles; and worse yet, data from differing data sources
- Many heterogeneous hardware platforms and software environments that were implemented, because of cost, available applications, or personal preference, resulting in even more inconsistency and lack of integration

For more detailed information about data mart consolidation specific initiatives, refer to the following IBM Redbook:

- *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653

3.3.3 Dependent data mart architecture

An interconnected data mart architecture is basically a distributed implementation. Although separate data marts are implemented in a particular workgroup, department, or line of business, they are integrated, or interconnected, to provide a more global view of the data. These data marts are connected to each other using, for example, conformed dimensions and conformed facts. For example, look at Figure 3-13. Assume that data mart#1 and data mart#2 both use a customer dimension. When we say that these data marts share conformed dimensions, it means that the two data marts implement the same common version of a customer dimension. Also, each of these data marts typically has a common staging area. At the highest level of integration, the combination of all dependent data marts could be thought of as a distributed enterprise data warehouse.

Figure 3-13 shows the architecture for dependent data marts. In the implementation previously mentioned, where the EDW and dependent data mart architectures are combined, the Staging Area is basically replaced by the EDW.

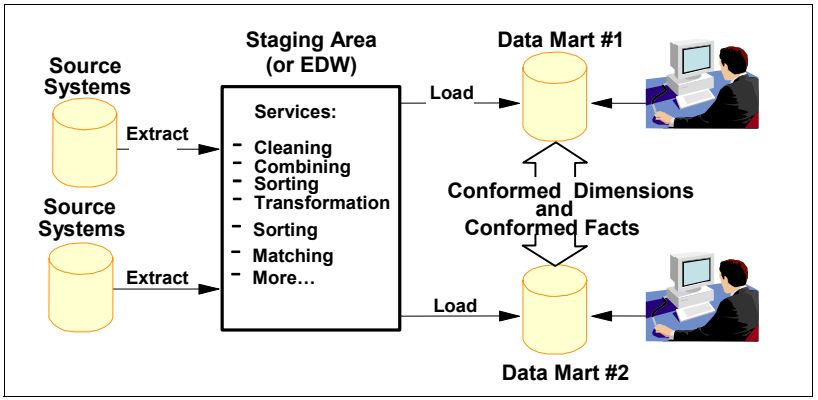


Figure 3-13 Dependent data mart architecture

3.4 Data models and data warehousing architectures

Now that we have an understanding of the various data warehouse architectures, let us spend time investigating the components of each of the implementations.

The primary focus here is to understand how the data models are used jointly or independently to design these data warehouse architectures.

3.4.1 Enterprise data warehouse

Figure 3-14 shows the primary components of the enterprise data warehousing architecture. It also shows the type of data model (E/R or Dimensional) on which each of the components is based.

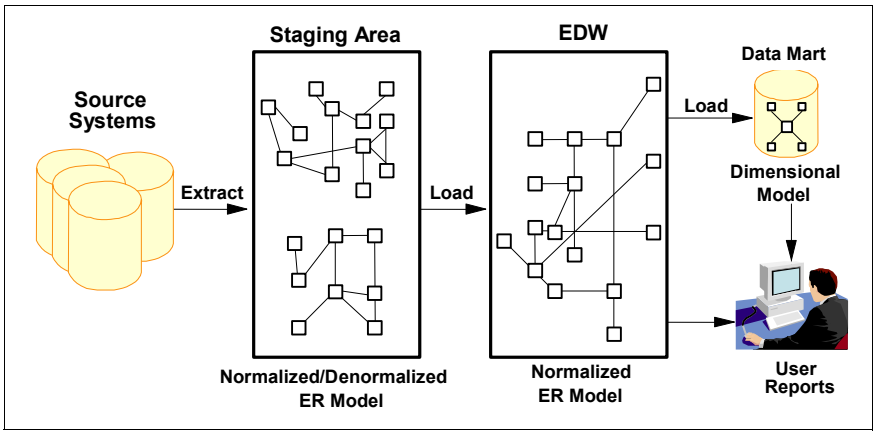


Figure 3-14 Data models in the enterprise data warehousing environment

Source systems

The source systems are the operational databases of the enterprise. As such, they are typically in 3NF and represented by E/R models. However, they can also be dimensional models or other file structures. They are typically created and used by the applications that capture the transactions of the business. In most cases, they are highly normalized for fast performance on transactions, using the typical insert, update, and delete processing. The priorities of a transaction-oriented source system are performance and high availability.

Data staging area

The staging area is the place where the extracted and transformed data is placed in preparation for being loaded into the data warehouse. As you see in Figure 3-14, the data staging area is primarily in 3NF, and represented by an E/R model. However, the staging area may also contain denormalized models.

Enterprise data warehouse

As shown in Figure 3-14, the EDW is primarily in 3NF and based on an E/R data model. Key here is that we differentiate between the data warehouse and the data warehousing environment. That is, within the data warehousing

environment, there can also be dimensional models - in the form of dependent data marts. The analysts can query the data warehouse directly, or through a data mart which is populated from the data warehouse. The latter may improve the ad hoc query performance and availability depending on the configuration.

Data mart

The data marts in the enterprise data warehousing environment are based on a dimensional model, and could be any of the following types of schema:

- Star
- Snowflake
- Multi-Star

And, those data marts employ conformed dimensions and conformed facts.

3.4.2 Independent data mart architecture

Figure 3-15 shows the primary components in the independent data mart architecture. It also shows the type of data model (E/R or Dimensional) upon which each component is based.

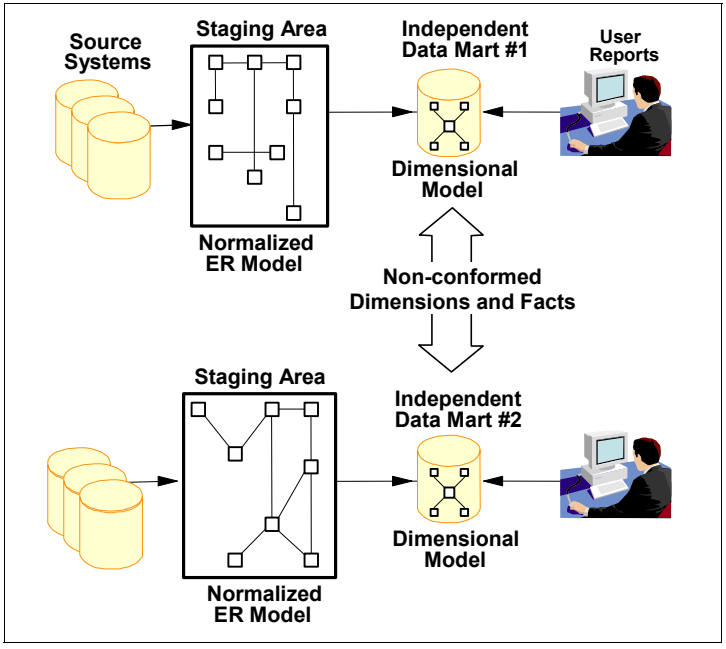


Figure 3-15 Data models in an independent data mart architecture

The descriptions of the components of the independent data mart architecture are as follows:

Source systems

The source systems are typically in 3NF and based on an E/R model. However, they can include dimensional models and other file structures. They are the data stores used by the applications that capture the transactions of the business. The source systems are highly normalized for fast performance on transactions, using the typical insert, update, and delete processing. The priorities of a transaction-oriented source system are performance and high availability.

Data staging area

The staging area is the place where the extracted and transformed data is placed in preparation for being loaded into the data warehouse. As shown in Figure 3-15 on page 63, the data staging area is primarily in 3NF, and represented by an E/R model. However, the staging area may also contain denormalized models. In case of independent data warehouse architecture, there are separate staging areas for each data mart and therefore no sharing of data between these disparate staging areas.

Independent data marts

As shown in Figure 3-15 on page 63, the data marts are based on a dimensional model which can be any of the following types of schemas:

- Star
- Snowflake
- Multi-Star

The data marts in the independent data warehouse are not connected to each other because they are not designed using conformed dimensions and conformed facts. Here again, this can result in inconsistent and old, out of date sources of data that can lead to inaccurate decision making.

3.4.3 Dependent data mart architecture

The dependent data mart, as the name implies, is dependent on something. And, that something is the enterprise data warehouse.

That means that the data for the dependent data mart comes from the EDW. Therefore, the data loaded into the dependent data mart is already transformed, cleansed, and consistent. The primary concern when using a data mart is the currency of the data, or how fresh it is. That is, what is the date and time of the last extract of data from the EDW that was loaded into the dependent data mart.

And, when data is used from multiple data marts, care must be taken to assure that the freshness of the data is consistent across them.

Figure 3-16 on page 65 shows the components of a dependent data mart architecture, and the type of data model upon which it is based.

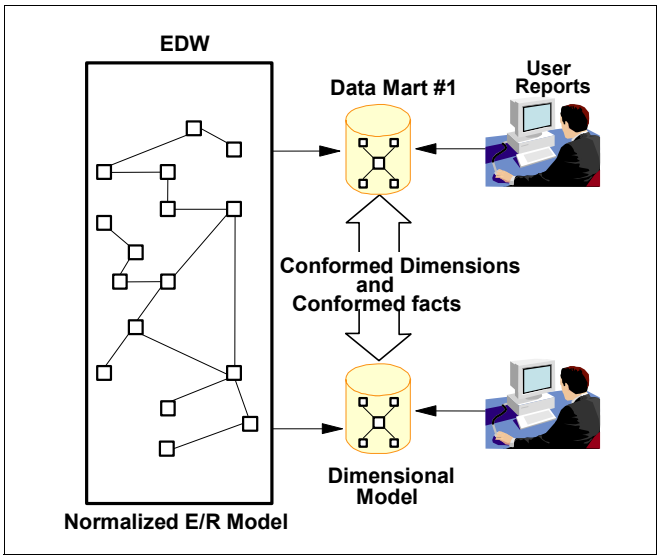


Figure 3-16 Data models in a dependent data mart architecture

The descriptions of the components of a dependent data mart architecture are as follows:

Source systems

The source system for the dependent data mart is the EDW.

Data staging area

Since data is coming from the EDW, there is no requirement for a data staging area.

Dependent data marts

The data marts are based on a dimensional model. The dimensional model can be any of the following schemas:

- Star
- Snowflake
- Multi-Star

The data marts are based on conformed dimensions and conformed facts.

3.5 Data modeling life cycle

In this section, we describe a data modeling life cycle. It is a straight forward process of transforming the business requirements to fulfill the objectives for storing, maintaining, and accessing the data within IT systems. The result is a logical and physical data model for an enterprise data warehouse.

We describe a specific life cycle to design a dimensional model (star schema) in Chapter 5, “Dimensional Model Design Life Cycle” on page 103.

3.5.1 Modeling components

The goal of the data modeling life cycle is primarily the creation of a storage area for the business data. That area comes from the logical and physical data modeling stages, as depicted in Figure 3-17.

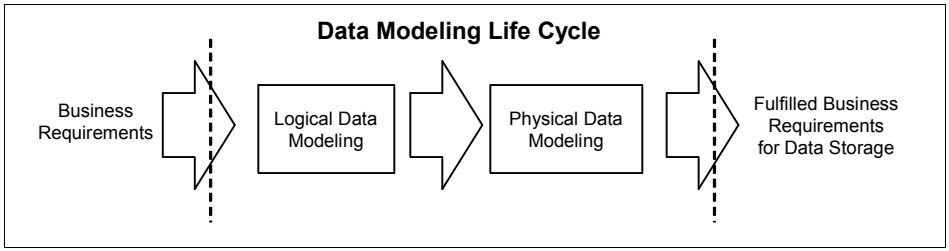


Figure 3-17 A generic data modeling life cycle

In the context of data warehousing, data modeling is a critical activity. The complexity of the business and its needs for business intelligence are a real challenge for data modelers. We start now with a brief overview of the life cycle components:

- **Logical data modeling:** This component defines a network of entities and relationships representing the business information structures and rules. The entities are representations of business terms of relevance to the business, such as: involved party, location, product, transaction, and event. The relationships are representations of associations between entities. An *entity* characterizes attributes, such as name, description, cost, sale price, and business code. Figure 3-18 on page 67 is an example of a logical model.

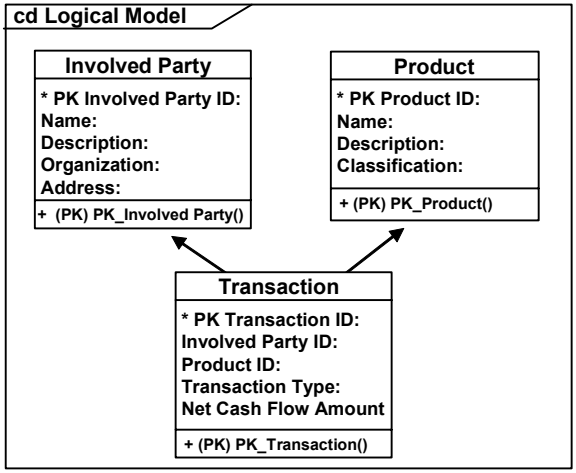


Figure 3-18 A logical model representation

- **Physical data modeling**: This component maps the logical data model to the target database management system (DBMS) in a manner that meets the system performance and storage volume requirements. The physical database design converts the logical data entities to physical storage (such as tables and proprietary storage) of the target DBMS or secondary storage medium. The physical database design is composed of the Data Definition Language (DDL) that implements the database, an information model representing the physical structures and data elements making up the database, and entries in the data dictionary documenting the structures and elements of the design. An example of the DDL for the table named *transaction (TXN)* is in Example 3-1.

Example 3-1 Sample DDL

```
create table TXN
( TXN_ID              INTEGER              not null,
PPN_DTM              TIMESTAMP,
SRC_STM_ID           INTEGER,
UNQ_ID_SRC_STM       CHAR(64),
MSR_PRD_ID           INTEGER,
TXN_TP_ID            INTEGER,
ENVT_TP_ID           INTEGER,
UOM_ID               INTEGER,
TXN_VAL_DT           DATE,
TXN_BOOK_DT          DATE,
NET_CASH_FLOW_AMT    NUMERIC(14,2),
```

```
constraint P_TXNXPK primary key
(TXN_ID)
```

3.5.2 Data warehousing

In the 3.3, “Data warehouse architecture choices” on page 57, we describe data warehousing concepts and possible architectures. In Figure 3-19, we depict an example enterprise data warehouse, where the arrows show the data flow among components.

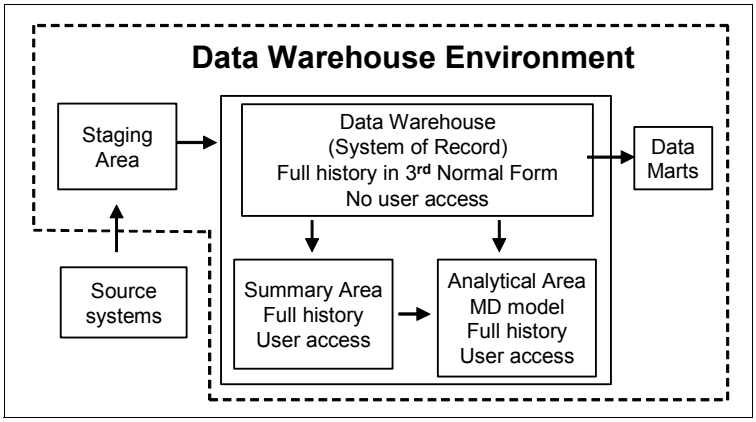


Figure 3-19 Enterprise data warehouse environment

The DW components differ not only by content of data but also by the way they store the data and by whom it can be accessed.

- *Staging area:* For handling data extracted from source systems. There can be data transformations at this point and/or as the data is loaded into the data warehouse. The structure of the staging area depends on the approach and tools used for the extract, transform, and load (ETL) processes. The data model design affects not only performance, but also scalability and ability to process new data without recreating the entire model.
- *Data warehouse*: This is the area, also called the system of record (SOR), that contains the history data in 3NF and is typically not accessed for query and analysis. Use it for populating the summary area, analytical areas, and the dependent data marts.
- *Summary area:* This area contains aggregations. Structures are usually derived from the data warehouse where one or more attributes are at the higher grain (less detail) than in the data warehouse. These are constructed for high performance data analysis where low level detail is not required.

- *Analytical area*: Contains multidimensional (MD) structures, such as the star schema, snowflakes, or multi-star schemas, constructed for high performance data analysis.

3.5.3 Conceptual design

The conceptual design represents an early basis for design reviews, including confirmation that the business requirements are sufficiently described and that there is an available solution. From this point starts the logical data modeling which transforms the business requirements into the context of the data/information necessary to be stored, accessed, and maintained.

3.5.4 Logical data modeling

In this section, we focus on the modeling of data warehouse. First we look at the logical data model. The primary purpose of logical data modeling is to document the business information structures, processes, rules, and relationships by a single view - the logical data model.

The logical data model helps to address the following:

- Validation of the functional application model against business requirements
- The product and implementation independent requirements for the physical database design (Physical Data Modeling)
- Clear and unique identification of all business entities in the system along with their relations, by the logical model

Note: What happens if we do not have the logical data model?

Without the logical data model, the stored business information is described by a functional model or application (such as ETL or OLAP). Without the logical data model, there is no single view of all data, and data normalization is impossible. In this case, the physical data model has to be designed from a functional model. This will potentially cause performance problems, and data inconsistency and redundancies, which can result in an inefficient physical design.

Design steps in logical modeling

The design activity of logical data modeling of the data warehouse consists of the following steps:

1. Identification of entities, attributes, and relationships
2. Normalization and identification of entities
3. Merging the ETL functional model with the logical data model

4. Validation of the logical model against business requirements

We now discuss each of the tasks in detail.

One - Identification of entities, attributes, and relationships

This step consists of the following activities:

1. Review available documentation for the project, including the scope of the project and information about the source systems from where data is loaded. Look for business requirements, process models, profiles, architectural design, and data models.
2. Create a list of nouns representing general categories of information required to be stored in the data warehouse. Review this entity list with subject matter experts. The nouns should represent standalone concepts rather than attributes or subsets of something larger.
3. From the noun list, identify the entities. An *entity* is a generalization of the concepts, involved parties, products, arrangements, locations, or events about which the system store informations. Entities may occur directly in the list or may be names for collections of nouns in the list. Each entity name must be unique and meaningful. Nouns representing out-of-scope concepts and implementation concepts are not logical data model entities.
4. Once each entity has been defined, determine its relationships with other entities. Each entity may have multiple relationships with other entities. However, a single relationship is only held between two entities. When analyzing the relationships between entities, it is important that the relationships are analyzed from the context of the business view. Each relationship is considered bidirectional and granted names for each side of the relationship.
5. Identify the associated cardinalities (numbers of occurrences of one entity relative to another entity) and describe it. Cardinality refers to the minimum and maximum numbers of occurrences implied by the existence of two entities participating in a relationship.
6. Identify the attributes or characteristics of the entities that are relevant to the business, and the primary key for each entity. The primary key is made up of is the subset of attributes that uniquely identify each entity. Attributes must be *atomic*. That is, they cannot be further decomposed to simpler elements.
7. Relationships, cardinalities, and inter-attribute dependencies are usually derived from business rules.
8. For each entity and attribute record, define a text description in the data dictionary that clearly represents the element from a business point of view.

Two - Normalization and identification of entities

Normalizing the model means removing structural redundancies and inconsistencies. The recommended approach is to proceed in taking the model to: *first normal form,* then *second normal form*, and finally to *third normal form*. The third normal form is used for the data warehouse. This is where necessary detail and historical data is kept, typically in an E/R model.

The primary reason for a data warehouse is to enable data query and analysis to provide information for decision making. However, as the volume of data grows in an E/R model, performance issues may arise. Thus, one of the reasons for the dimensional model.

Dimensional modeling is a technique that can provide the required performance for query and analysis on huge volumes of data. It has since become the defacto technique for data warehousing analytics, and that is why it is the primary subject of this redbook. There is more detailed information about dimensional modeling in Chapter 5, “Dimensional Model Design Life Cycle” on page 103.

To prepare for dimensional modeling and analytics, the data in the data warehouse must be formatted properly. The following list briefly describes activities involved in this formatting:

1. Repeating Groups: The repeating attribute groups must be removed because they are indicative of the existence of another entity. Each set is an instance of the new entity and must contain the primary key of the original entity as a foreign key. After these sets are removed and made separate entities, the model is said to be in *first normal form*.
2. Functional dependencies: Any partial functional dependencies among attributes in the entities must be removed. In entities that have primary keys comprised of more than one attribute, non-key attributes may be a function of the entire key or of part of the key. In the former case, the attribute is said to be fully functionally dependent. In the latter case, the attribute is said to be partially functionally dependent. Partially functionally dependent attributes are indicative of the existence of another entity and should be removed from the original entity. The primary key of the new entity is that part of the primary key of the original entity that was involved in the partial dependence. At this point, the model is said to be in *second normal form*.
3. Transitive dependencies: Transitive dependencies among attributes in the entities must be removed. Mutually dependent non-key attributes are indicative of the existence of another entity. One of the dependent attributes is said to be dependent on the primary key of the original entity. The other mutually dependent attributes are said to be transitively dependent on the primary key. The dependent attribute is left in the original entity as a foreign key. The transitively dependent attributes are removed. A new entity is formed whose primary key is the dependent attribute in the original entity and

whose other attributes are the transitively dependent attributes of the original entity. The model is now said to be in *third normal form*.

4. Primary key resolution: Instances where multiple entities have the same primary key and these instances must be resolved. During the normalization process, it may be found that multiple entities have the same primary keys. These entities should only be merged if all the joined attributes are valid for all instances of every entity. If this is not the case, a super/subtype structure should be developed where the shared attributes are placed in the super-type and the unique attributes are in separate subtypes.

Three - Merging the ETL functional model with the logical model

In this step, we do the following:

1. Entities in the logical data model should be mappable to source systems.
2. Attributes in the entities are to be created, read, updated, and deleted by data flows in the ETL functional model and analytic applications.
3. The processes of the ETL functional model should maintain the relationships and cardinalities of the E/R model and the values of the attributes.

Four - Validation of the logical model against business requirements

Consider the following points for validation and verification of the data model:

1. Each entity should represent an involved party, location, product, transaction, or event relevant to the business.
2. The logical data model should include all information the data warehouse needs to store about the business.
3. Each entity should have a name, a primary key, and one or more attributes, and enter into one or more relationships with other entities.
4. Each relationship should have correct cardinalities that reflect the needs of the business.
5. Each entity should be properly normalized.
6. Each entity and attribute should be accounted for in the data warehouse, and related to functions or processes such as ETL, real-time DW, and DW housekeeping.

Note: Important guidelines for developing the model for the DW are:

- Focus on staying within the scope of the data warehouse system being developed.
- Use any existing data models that you have as a starting point. However, do not just accept it *as is*. If required, modify it and enhance it, so that it provides an accurate representation of business data requirements.
- The high-level data model should include representation for potential future data and relationships. This requires good knowledge of company plans for the future.
- It is often useful to first assemble a draft entity relationship diagram early after creation of the noun list and then review it with client staff through successive interviews or group sessions. By doing so, review sessions will have more focus. Following the initial first draft, continuous client involvement in data model development is critical for success.

3.5.5 Physical data modeling

In this section, we focus on physical data modeling of the main storage of the data warehouse. The data warehouse provides a reliable single view of the data at the required level of detail, along with the necessary history data for the enterprise. Data modeling for the analytical area (dimensional model) or the summary area (see Figure 3-19 on page 68) is discussed in more detail in Chapter 5, “Dimensional Model Design Life Cycle” on page 103.

Introduction to physical modeling

The objective of physical data modeling is the mapping of the logical data model to the physical structures of the RDBMS system hosting the data warehouse. This involves defining physical RDBMS structures, such as tables and data types to use when storing the data. It may also involve the definition of new data structures for enhancing query performance. However, you must do it without changing the meaning of the logical data model schema.

Note: What happens if there is no physical database design?

- We generate DDL from the logical data model.
- The final database is fully normalized without additional tables or attributes for improving performance.

Other important factors to consider:

There are important factors to consider while designing the physical model. For example:

- You need tools for physical data modeling, such as:
  - Case tools for maintaining the physical data model (as well as the logical data model)
  - Database performance tools
  - Meta data management tools
- Scalability of the design, and the physical RDBMS
- Queries, ETL, and other applications that use the data warehouse
- Use of an abstracted data model for performance
- Operation/Maintenance of DW (housekeeping)

Note: How much does physical data modeling in OLTP differ from physical modeling for the data warehouse? The answer is, not much. At the conceptual model level, it differs primarily in performance design. The key difference is that in OLTP, we are primarily concerned with data and transaction volumes, where with the DW we must focus on load performance, for population of the analytical areas and the summary tables by batch/real-time applications, and on performance of the analytical queries.

Physical design activities for the data warehouse

We divide the physical design activity for the data warehouse into the following steps:

One - Physical modeling of entities and attributes

In this phase, we do the following:

1. For each entity in the logical data model, we define an RDBMS table. We document this activity and assign the name compatible with the common DBMS and according to your company naming convention.
2. For each attribute in the entity, we identify a column and assign the name compatible with the common DBMS naming syntax. We define RDBMS specific data types, such as character, varchar, integer, float, and decimal.
3. We define Primary and Foreign keys of the entities to the tables.

Note: Before starting any data warehousing project, it is a good practice to establish within the company a common naming convention for business and technical objects, along with recommended data types in the data directory.

Two - Build the DDL

The following is a list of activities for this step:

1. Create target database.
2. Define the target database vendor.
3. Connect to target database.
4. Generate DDL by a preferred Case tool.
5. Implement the DDL code using Case tool or a script.

Three - Performance design and tuning

In this step, we comment briefly about how to proceed with the performance design and tuning for the data warehouse. We discuss tuning the population of the E/R, rather than optimizing query performance against the data warehouse. Query performance against the data warehouse is typically supported by using summary tables, analytical data marts, and derived data marts, as we have seen in Figure 3-19 on page 68.

The two primary processes for populating the E/R model of the warehouse are batch and real time. In batch mode, the E/R model is typically populated using custom applications, ETL tools, or native database utilities that deliver good performance. It is much the same with real time, but with modifications.

In a real-time environment, you typically must change process models. The objective is to move to more of a continuous load/update scenario. It cannot be achieved by simply moving data faster. The processes must be changed to enable the data to be made available faster. New techniques should also be considered. For example, rather than the typical ETL process many are moving to an ELT process. That is, the data is extracted and loaded. Then, the required transformations are performed. This can enable a performance improvement.

Note: Performance is dependent on the physical data structures of RDBMS. Altering or adding more appropriate physical structures may possibly improve the performance of query/extraction/replication. However, it may also increase the load time of the data warehouse. Performance tuning is a cost minimization issue. For example, performance can always be improved by adding more CPU and I/O resources. But the objective is to find a compromise between acceptable performance and total cost of the system.

Four - Verification of physical data modeling process

At the end of physical design, check the following list for completed activities:

1. Generated physical model DDL script should properly define the physical structures along with performance enhancements.
2. Good documentation of the physical design should exist in the case tool used.

3. Each entity of logical design should represent a physical table with the appropriate attributes and relations.
4. Each relationship should describe correct cardinalities (one to one, one to *n*, and *n* to *n*).
5. Each entity and attribute should be properly described in the data dictionary.
6. All capacity estimates should be validated.

In this section, we have discussed the data modeling techniques for data warehousing. In Chapter 5, “Dimensional Model Design Life Cycle” on page 103, we discuss in detail the technique for designing a dimensional model.

4

Data analysis techniques

In this chapter, we focus on data analysis techniques used in data warehousing. This is, after all, the primary reason for a data warehouse. We do not intend this to be an all-inclusive treatise on the subject, but an overview so you have a good general understanding.

We discuss the following topics:

- The information pyramid and associated reporting that you can perform
- BI reporting tool architectures
- Classification of BI users based on analytical needs
- Query and reporting
- Multidimensional analysis techniques:
  - Slice and dice
  - Pivoting
  - Drill-up, drill-down, and drill-across
  - Roll-up and roll-down
- Query and reporting tools

4.1 Information pyramid

Every enterprise produces regulatory, statutory, and internal reports on a regular basis. The reports are predefined documents composed from tables, summaries, or charts, containing business information consisting of measures and tables with a description of columns and rows. The target users are auditors, governmental regulatory institutions, shareholders, internal users from areas such as finance, or from the top management of a company. This says that there is a need for information in the form of analytics, dashboards, queries, and reports by all levels of people in an organization. The focus of the IT department is to provide for the analysis and business reporting needs of all company decision makers.

Gone are the days when you could plan and manage business operations effectively by using monthly batch reports, and when IT organizations had months to implement new applications. Today companies need to deploy informational applications rapidly, and provide business users with easy and fast access to business information that reflects the rapidly changing business environment. In short, the pressure on the IT department to deliver the information to the business continues to dramatically increase.

In this section, we focus on different information environments in the organization, and discuss how to accomplish data analysis and reporting from each of them.

4.1.1 The information environment

Consider the information pyramid in Figure 4-1 on page 79. The information pyramid depicts several environments or levels within an organization where the information may reside, in what form, and for what duration. Traditionally, IT has seen these levels of data as separate, with the requirement of copying data from one level to another for appropriate usage.

However, these different levels should be thought of simply as different views of the same data, although individual users may only focus on a particular level to perform their specific job. But, to emphasize this difference in perspective, we have named these levels as *floors* of data. While data copying may continue between the floors, this approach is no longer the only one possible.

The data on the different floors of the pyramid does have different characteristics, such as volumes, structures, and access methods. Because of that we can choose the best way to physically instantiate the data. And the pyramid emphasizes that today the requirement is to access data from all floors within a single activity or process.

We now describe each floor in a bit more detail, along with the part it plays in the reporting environment. From this discussion, we can see the advantages, disadvantages, and how the decision-making processes should be architected.

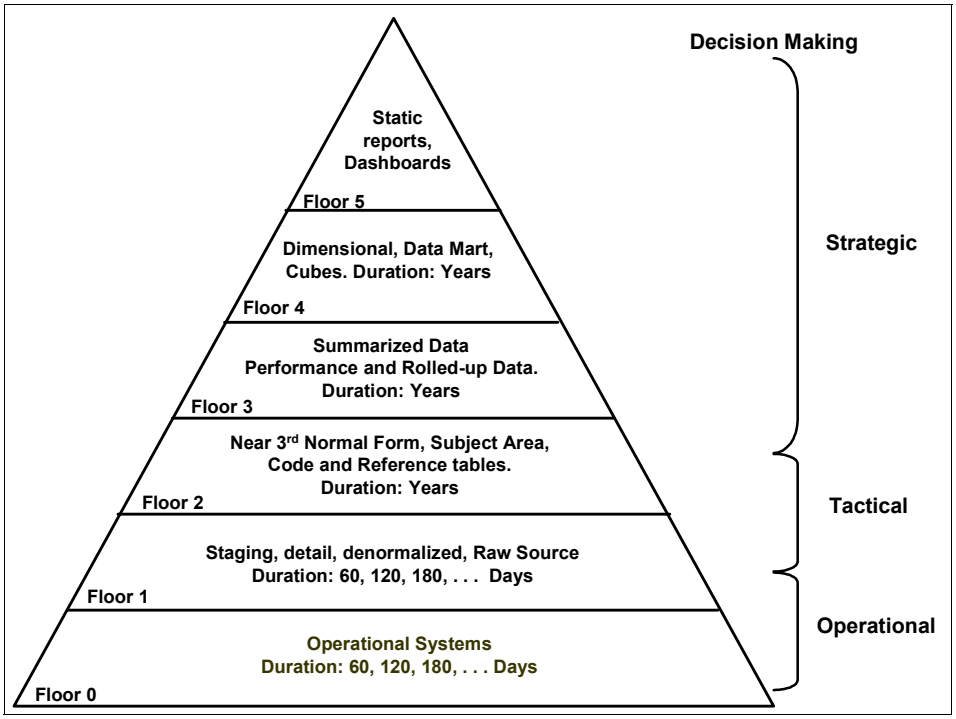


Figure 4-1 Information pyramid

The concept behind the information pyramid is to map the delivery of data needs to your applications and organization needs. This is to help determine the data flow and delivery requirements. We call this *right time* data delivery. The lower floors of the architecture offer the freshest, and most detailed data. This is the source of data for making operational decisions on a day to day basis. As you move up the floors, the data becomes more summarized and focused for specific types of analysis and applications. For example in floors 3, 4, and 5, the data is used for more strategic analysis.

Now we take a look at each of the floors and discuss them in detail.

Floor 0

This floor represents data from the operational or source systems. The data in the source systems is typically in 3NF and represented in an E/R model. The source systems are the applications that capture the transactions of the business. The data is highly normalized so that the transactions performing

inserts, updates, and deletes can be executed quickly. The primary requirements for these source systems are transactional performance and high availability. The source systems are periodically purged, so they typically store relatively little history data. With source systems, you can make operational, day to day decisions, but they are not formatted or architected for long-term reporting solutions.

The advantages of querying from the source systems (floor 0) are:

- No need for additional hardware.
- Quick start.
- Less difficult to accomplish.

The disadvantages of querying from the source systems (floor 0) are:

- They are not optimized for query processing.
- Executing queries or reports against a source system can negatively impact performance of the operational transactions.
- Insufficient history data for general query and reporting.
- They need new data structures (such as summary tables) created and maintained.

Note: Floors 1-5 can broadly be mapped to the layers in existing data warehouse architectures. These layers are based on the same fundamental data characteristics that provided the basis for separating the different types of data when defining the architecture of the data warehouse.

Floor 1

This floor represents the staging area and the denormalized source systems. The staging area is not appropriate for reporting because the data in the staging area is prepared for consumption by the business users. On the other hand, a denormalized source system, which is also often part of the floor 1, may be used for limited reporting.

The advantages of querying from floor 1 are:

- Reporting can be obtained from the denormalized source systems.

The disadvantages of querying from floor 1 are:

- Data in the staging area cannot be used for reporting purposes because it may not have been cleansed or transformed, and therefore, it is not ready to be used.

Floor 2

As shown in Figure 4-1 on page 79, floor 2 represents the 3NF data warehouse and associated reference tables. In Figure 4-2, we show an architecture where users are querying a data warehouse designed in a 3NF E/R model. A normalized database is typically difficult for business users to understand because of the number of tables and the large number of relationships among them.

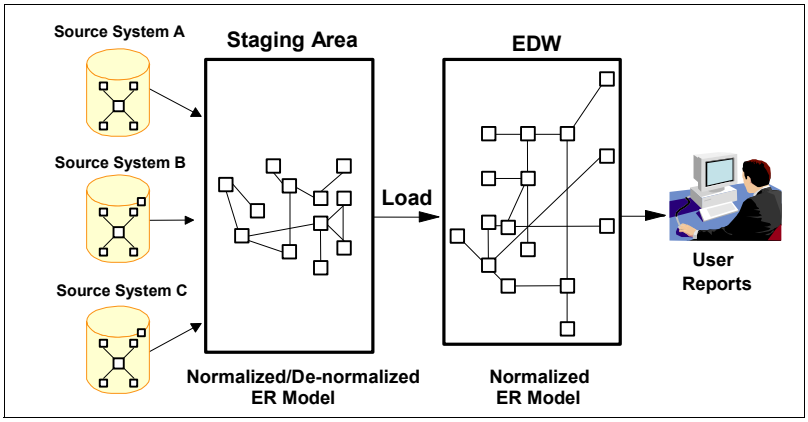


Figure 4-2 Querying a data warehouse

The advantages of querying from floor 2 are:

- There are single sources of data.
- Floor 2 contains history and detail data.
- Floor 2 is optimized for storage of large volumes of data.

The disadvantages of querying from floor 2 (the data warehouse) are :

- There is no structure for dimensional analysis, making it difficult to understand the data relationships.
- Complex project, difficult to deliver.

Floor 3

Floor 3 represents summarized data, which is created directly from the warehouse.

The advantages of querying from the summarized warehouse are:

- The history data is easily available.
- It is optimized for a set of data.

The disadvantages of querying from the data warehouse are:

- You may need to create summaries for several business processes, and maintaining such summaries becomes a maintenance issue.

Floor 4

On floor 4, there are dimensional data marts stored in relational databases, and data cubes.

The advantages of querying from the dimensional data mart or cubes are:

- History data is easily available.
- Highly optimized data is available for each business process.
- Queries are easy to understand by the business users.

The disadvantages of querying from the dimensional data mart or cubes are:

- You may need to create and maintain several dimensional models for each business process. This adds to the overall cost of the data warehouse.
- For querying cubes, you may need to purchase specialized software.
- For creating and storing cubes, you need additional software.
- There will be additional storage requirements.
- The additional effort to populate data marts.

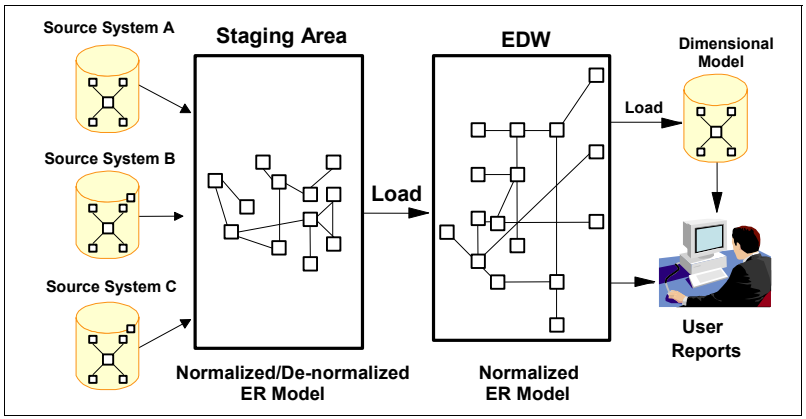


Figure 4-3 Querying from a dimensional model

Floor 5

Floor 5 represents the reporting environment and infrastructure that is used for making static reports, dynamic ad hoc reports, and dashboards. It is the delivery platform that supports all the reporting needs.

4.2 BI reporting tool architectures

The different data analysis tools for reporting typically fall into two main architectures as shown in Figure 4-4. The architectures are:

- 2-tier: In this architecture, the BI reporting tool is installed on the client machines and these tools (clients) directly access the data warehouse or the data marts.
- 3-tier: In this architecture, the BI reporting server software is installed on a server machine. All the clients access the data warehouse or the data marts by using a browser. No special tool must be installed on the clients.

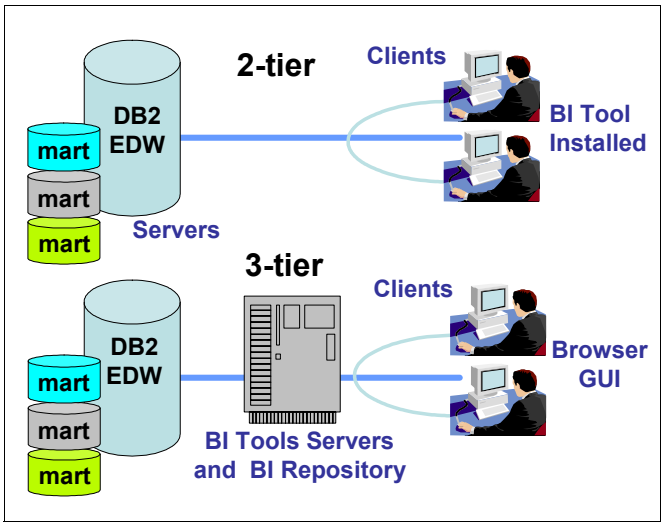


Figure 4-4 BI reporting tool architectures

4.3 Types of BI users

When buying BI reporting tools, it is important to match them with the type of user who uses them. Each product tends to excel at certain tasks, but may lag in others. This typically means that the client ends up with 3-4 BI tools to meet the requirements. Then there is a need for additional skills, education, maintenance, and licenses, which are all additional expenses. Soon there is a movement to consolidate those tools to save money. In this case, you must take care to make sure the business users still have the specific capabilities they need to do their jobs. Figure 4-5 on page 84 shows classifications of users according to their BI reporting needs.

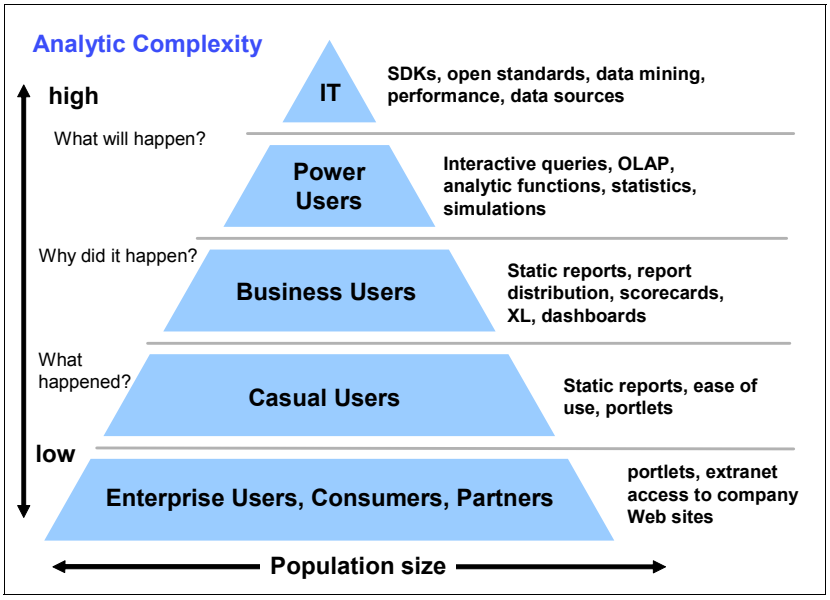


Figure 4-5 BI user types and their requirements

Based on the analytical needs, we classify the BI users within an organization as follows:

- **Enterprise users, consumers, and partners:** These users have the lowest reporting needs. These users typically access data in the form of portlets or company Web sites which have been given extranet access.
- **Casual users:** Casual users mainly use static reports and portlets. They require ease of use, and typically prefer to work with standalone applications, such as spreadsheets.
- **Business users:** Now we are getting to users who typically belong to middle and executive management. They are more interested in dashboards and other static types of reports. Business users are involved with both tactical and strategic decision making processes.
- **Power users:** These users require higher functionality compared with other user types. They understand the environment, are familiar with the data analysis tools, and are comfortable with complex applications.
- **IT users:** IT users are responsible for creating the reports for the business. IT users work with advanced reporting applications, such as software development kits (SDK) and data mining software. They understand the IT environment and are comfortable interacting directly with several heterogeneous data sources.

4.4 Query and reporting

Query analysis and reporting are the processes for posing questions to answer, retrieving relevant data from the data warehouse, transforming it into the appropriate context, and displaying it in a readable format. Query analysis and reporting are primarily driven by analysts who are quite familiar with posing such queries to determine the answers to their questions.

Traditionally, queries have dealt with two dimensions, or two factors, at a time. For example, you might ask, "How much of that product has been sold this week?" Subsequent queries would then be posed to perhaps determine how much of the product was sold by a particular store. Figure 4-6 depicts the process flow in query and reporting.

Query definition is the process of taking a business question or hypothesis and translating it into a query format that can be used by a particular decision support tool. When the query is executed, the tool generates the appropriate language commands to access and retrieve the requested data, which is returned in what is typically called an *answer set*. The data analyst then performs the required calculations and manipulations on the answer set to achieve the desired results. Those results are then formatted to fit into a display or report template that has been selected for ease of understanding by the user. This template could consist of combinations of text, graphic images, video, and audio. Finally, the report is delivered to the user on the desired output medium, which could be printed on paper, visualized on a computer display device, or presented audibly.

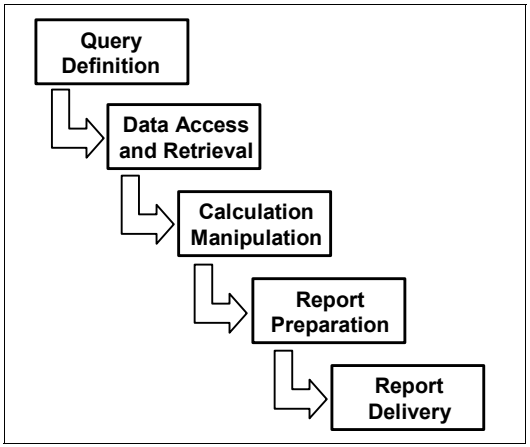


Figure 4-6 Process of querying and reporting

Users are primarily interested in processing numeric values, which they use to analyze the behavior of business processes, such as sales revenue and shipment quantities. They may also calculate, or investigate, quality measures

such as customer satisfaction rates, delays in the business processes, and late or wrong shipments. They might also analyze the effects of business transactions or events, analyze trends, or extrapolate their predictions for the future. Often the data displayed will cause the user to formulate another query to clarify the answer set or gather more detailed information. This process continues until the desired results are reached.

A more detailed discussion on different types of reporting and querying tools in available in 4.6, “Query and reporting tools” on page 94.

4.5 Multidimensional analysis techniques

Multidimensional analysis has become a popular way to extend the capabilities of query and reporting. That is, rather than submitting multiple queries, data is structured to enable fast and easy access to answers to the questions that users typically ask. For example, the data would be structured to include answers to the question, "How much of each of our products was sold on a particular day, by a particular salesperson, in a particular store?" Each separate part of that query is called a *dimension*. By precalculating answers to each subquery within the larger context, many answers can be readily available because the results have been precalculated for each query; they are simply accessed and displayed. For example, by having the results to the above query, one would automatically have the answer to any of the subqueries. That is, we would already know the answer to the subquery, "How much of a particular product was sold by a particular salesperson?" Having the data categorized by these different factors, or dimensions, makes it easier to understand, particularly by business-oriented users of the data. Dimensions can have individual entities, or a hierarchy of entities, such as region, store, and department.

Multidimensional analysis enables users to look at a large number of interdependent factors involved in a business problem and to view the data in complex relationships. Users are interested in exploring the data at different levels of detail, which is determined dynamically. The complex relationships can be analyzed through an iterative process that includes drilling down to lower levels of detail or rolling up to higher levels of summarization and aggregation.

Figure 4-7 on page 87 demonstrates that the user can start by viewing the total sales for the organization, then *drill-down* to view the sales by continent, region, country, and finally by customer. Or, the user could start at customer and *roll-up* through the different levels to finally reach total sales. *Pivoting* in the data can also be used. This is a data analysis operation where the user takes a different viewpoint than is typical on the results of the analysis, changing the way the dimensions are arranged in the result. Like query and reporting, multidimensional analysis continues until no more drilling down or rolling up is performed.

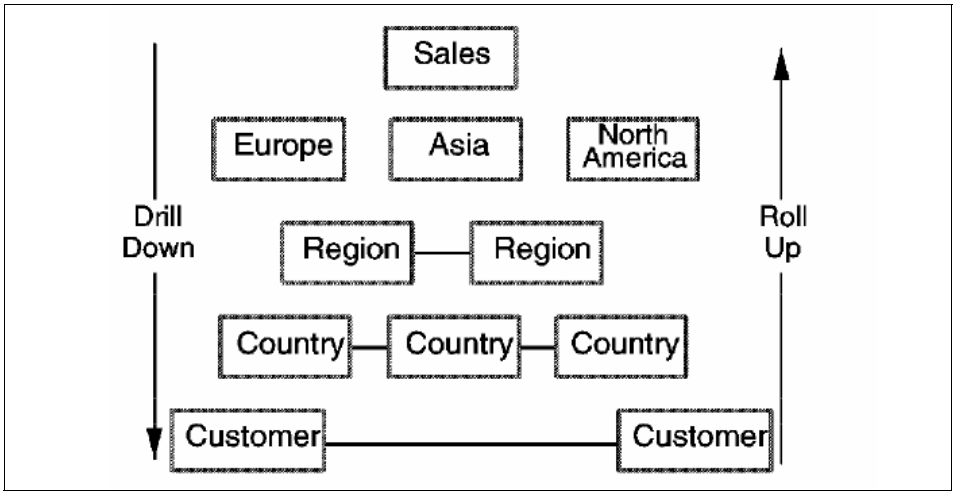


Figure 4-7 Drill-down and roll-up analysis

Multidimensional analysis enables you to look at the business problem by large number of interdependent factors describing the matter. In other words, multidimensional analysis enables you to view the information at different levels of detail or to analyze complex relationships.

The following are multidimensional techniques that we discuss in more detail:

- Slice and dice
- Pivoting
- Drill-down, drill-up, and drill-across
- Roll-down and roll-up

4.5.1 Slice and dice

We start by discussing slice and dice analysis as individual activities.

Slice

The term slice in multidimensional terminology is used to define a member or a group of members that are separated (from ALL other dimensions) and then evaluated across all the dimensions. A *member of a dimension* means a value inside a column. Slicing is slightly difficult to understand on a two-dimensional paper. In order to understand the slicing concept, consider a dimensional model example. Assume that we have only three dimensions named product, store, and date in a simple dimensional model. In this simple dimensional model, we just have one fact table with a fact called sales.

Assume that we isolate three members from the product dimension. The three members we isolated for the product dimension are soda, milk, and juice. This is shown in Figure 4-8. If we measure the SUM of sales quantity for ALL stores and for ALL dates across one or more members of one dimension (product in our case), then this concept is called *slicing*. The arrow in Figure 4-8 shows that the sum is across all dates and all stores.

This slice of the product dimension lets us to select our concerned members (soda, milk, and juice) from the product dimension. The slicing of the members allows us to focus only on these three members across all other dimensions. This concept is called *slicing*.

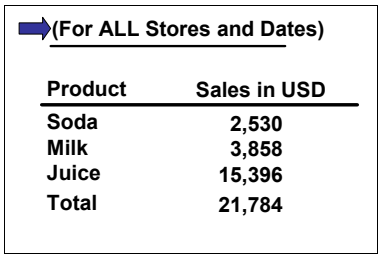


Figure 4-8 Slice for product

The slice in Figure 4-8 shows that soda generates the smallest sales amount, milk second, and juice third.

Note: When you *slice*, you choose one or more members of a dimension and consolidate (or summarize) across all other dimensions (in our example, the other dimensions were store and date.)

Dice

The *dicing concept* means that you put multiple members from a dimension on an axis and then put multiple members from a different dimension on another axis. This allows you to view the interrelationship of members from different dimensions.

Dicing is analysis of interrelationships among different dimensions or their members. Figure 4-9 on page 89 and Figure 4-10 on page 89 show examples of dicing.

In Figure 4-9 on page 89, we see multiple members listed vertically for the store dimension in one axis. These members are CA, OR, and LA. Similarly, we have multiple members for the date dimension which are listed horizontally. We are able to view the interrelationship of members from different dimensions. In other

words, we are able to see the relationship between CA and dates 1/1/2005, 1/2/2005, 1/3/2005, and vice versa.

DATE	1/1/2005	1/2/2005	1/3/2005	Total
Metrics	Sales in USD	Sales in USD	Sales in USD	Sales in USD
STORE				
CA	40	50	90	**180**
OR	3,115	3,340	1,267	**7,722**
LA	1,583	7,418	4,881	**13,882**
Total	**4,738**	**10,808**	**6,238**	**21,784**

Figure 4-9 Dice for store and date

Another example of dicing is shown in Figure 4-10.

PRODUCT	Milk	Coke	Juice	Total
Metrics	Sales in USD	Sales in USD	Sales in USD	Sales in USD
STORE				
CA	40	60	80	**180**
OR	60	1,452	6,210	**7,722**
LA	2,430	2,346	9,106	**13,882**
Total	**2,530**	**3,858**	**15,396**	**21,784**

Figure 4-10 Dice of store and product dimension

In this example, we can see the interrelationship between the members of the store and product dimensions. Here we analyze:

- How each store contributes to total sales amounts for each product (Soda, Milk, and Juice).
- How a particular product contributes to total sales for each store location.

Note: You *dice* when you choose one or more members of same dimension on one axis and on the other axis you choose a member or members from another dimension. Now you can analyze interrelationships of those dimensions.

4.5.2 Pivoting

Pivoting in multidimensional modeling means exchanging rows with columns and vice versa. Figure 4-11 on page 90 shows an example of pivoting. We exchange the store rows with columns of the product dimension members. It is simply a quick way to view the same data from a different perspective.

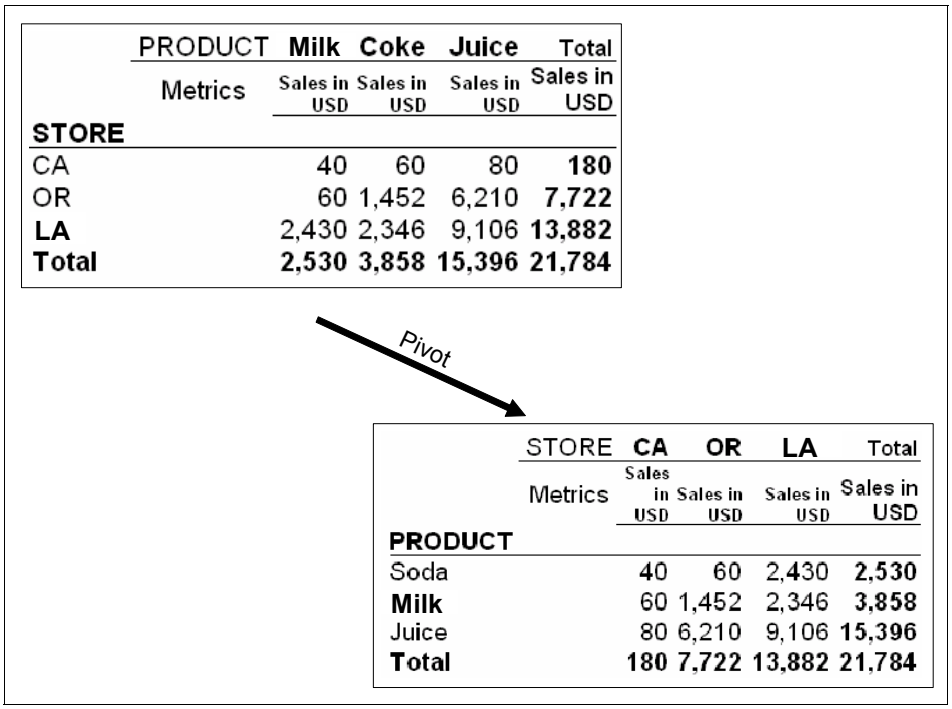


Figure 4-11 Pivoting

Note: You *pivot* when you exchange the axes of the report.

4.5.3 Drill-down and drill-up

Drilling in multidimensional terminology means going from one hierarchy level to another. In other words, drill-down can be defined as the capability to browse through information, following a hierarchical structure.

In the example shown in Figure 4-12 on page 91, we show drilling down through a simple three level hierarchy present in the product dimension. The hierarchy is

'Group Class' → 'Group' → 'Product'. When we drill-down the *Group Class* attribute, we reach the *Group*. Finally by drilling down on the *Group* attribute, we reach the lowest detail present inside the product dimension (which is the individual product) as shown in Figure 4-12 on page 91.

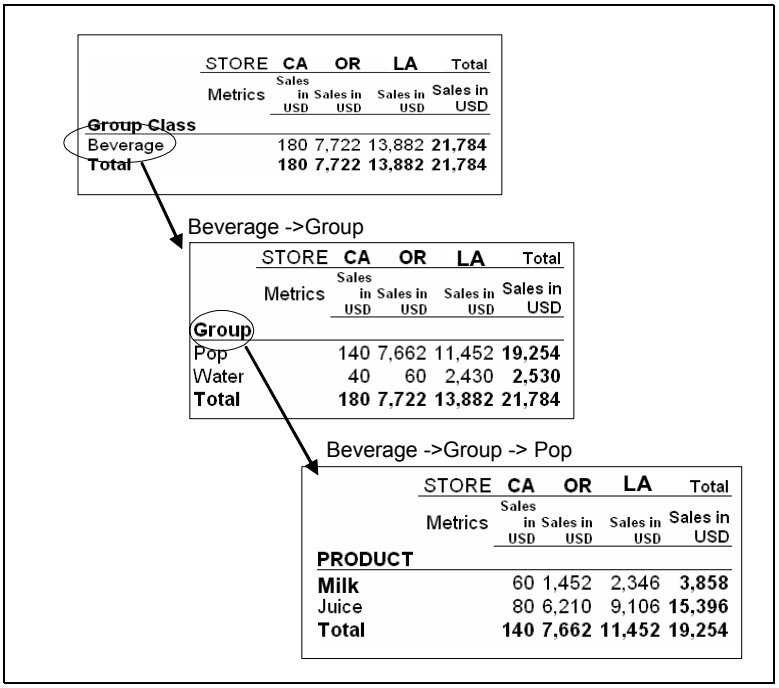


Figure 4-12 Drill-down on product dimension

Note: We consider drilling up and drilling down when we want to analyze the subject at different levels of detail. Drilling is possible if a dimension contains a multiple level hierarchy.

Another example of drill-down is shown in Figure 4-13 on page 92. Here we drill down from total sales in the US to sales at a particular store in Buffalo.

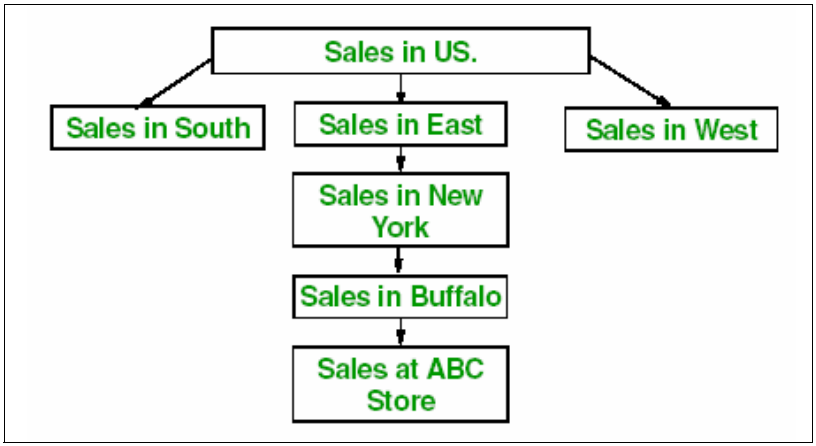


Figure 4-13 Drill-down example

Drill-up is exactly the opposite of drill-down.

4.5.4 Drill-across

Drill-across is a method where you drill from one dimension to another. You must define the drill-across path. This function is often used in ROLAP. In Figure 4-14, you see the result of drill-across from store CA to the product dimension. The first chart depicts the sales in stores in three different states. And, in particular, we have focused on CA (California).

DATE	1/1/2005	1/2/2005	1/3/2005	Total
Metrics	Sales in USD	Sales in USD	Sales in USD	Sales in USD
STORE				
CA	40	50	90	**180**
OR	3,115	3,340	1,267	**7,722**
LA	1,583	7,418	4,881	**13,882**
Total	**4,738**	**10,808**	**6,238**	**21,784**

CA→Product

DATE	1/1/2005	1/2/2005	1/3/2005	Total
Metrics	Sales in USD	Sales in USD	Sales in USD	Sales in USD
PRODUCT				
Soda	10	10	20	**40**
Milk	20	10	30	**60**
Juice	10	30	40	**80**
Total	**40**	**50**	**90**	**180**

Figure 4-14 Drill-across result

By drilling across to the product dimension, we can see the details about which products comprised the sales for the store *CA*.

4.5.5 Roll-down and Roll-up

Roll-down and roll-up are OLAP functions that give the higher or lower aggregate over whole dimension at a given hierarchy level.

In the example in Figure 4-15, we roll-down the product dimension from level 3, to level 2, and to level 1. This is done through the product hierarchy level: “Group class” → “Group” → “Product”.

The roll-down concept is the same as a drill-down.

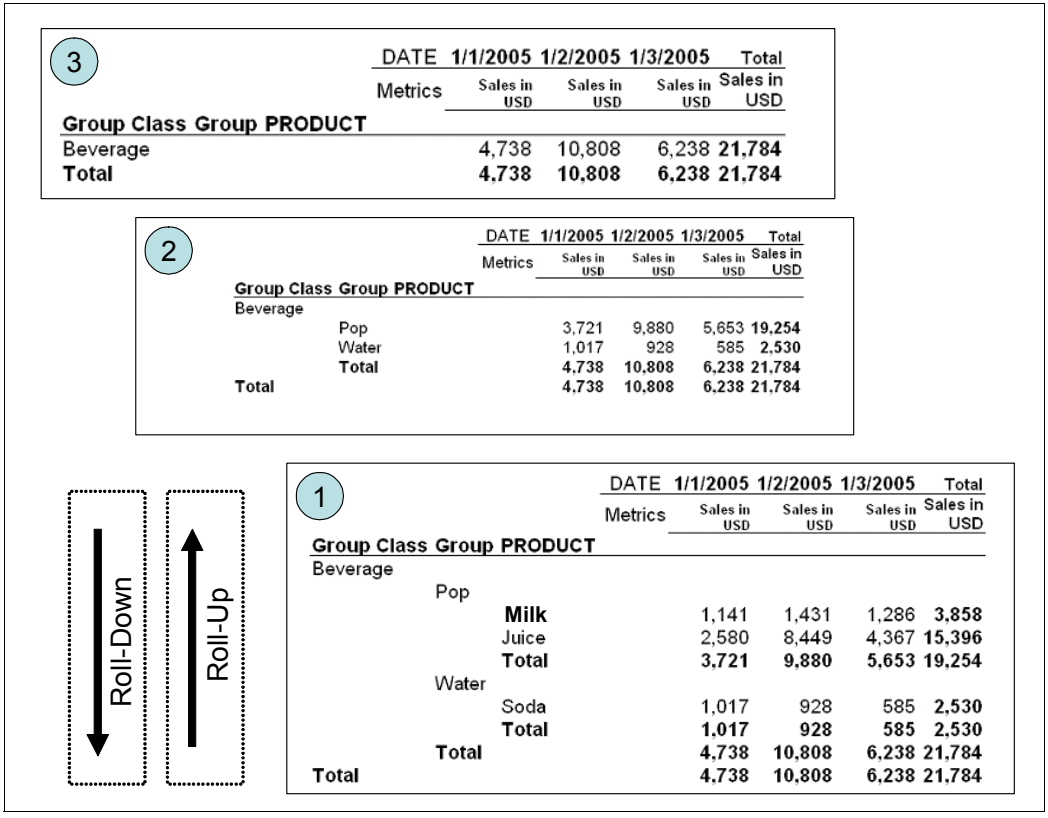


Figure 4-15 Roll-down, Roll-up

Roll-up is exactly opposite to roll-down. The arrows in Figure 4-15 show the roll directions. The roll-up concept is the same as drill-up.

4.6 Query and reporting tools

In this section, we discuss various query and reporting tools you can use to generate reports. We categorize various tools that can help in creating reports as follows:

- SQL query language using select statement, views, or stored procedures
- Spreadsheets
- Reporting Applications (Client-Server and Web-based)
- Dashboard and scorecard applications
- Data Mining tools

4.6.1 SQL query language

Select

In the RDBMS environment, the basic query reporting language is the structured query language (SQL) which enables you to query data in a database. In Example 4-1, we show an SQL select returning daily sales values for Product with subtotals and totals grouped by Date, Product Group, and Products.

Example 4-1 SQL select

```
select   D.DAY_AD as DAY,P.SUB_PRODUCT_GROUP as GROUP,P.PRODUCTS,
         sum(S.VALUE)  Sale_Amount
from     SALE       S
   join  PRODUCT    P
     on  (S.P_ID = P.P_ID)
   join  DATE       D
     on  (S.D_ID = D.D_ID)
where P.SUB_PRODUCT_GROUP in ('Pop')
group by rollup(D.DAY_AD,P.SUB_PRODUCT_GROUP,P.PRODUCTS)
order by 1,2,3,4;
```

The output of the Select statement is shown in Example 4-2.

Example 4-2 Output of select

```
DAY        GROUP    PRODUCTS    SALE_AMOUNT
---------- -------- --------- -------------
2005-01-01 Pop      Milk               1141
2005-01-01 Pop      Juice              2580
2005-01-01 Pop      -                  3721
2005-01-01 -        -                  3721
2005-01-02 Pop      Milk               1431
2005-01-02 Pop      Juice              8449
2005-01-02 Pop      -                  9880
2005-01-02 -        -                  9880
```

```
2005-01-03 Pop      Milk              1286
2005-01-03 Pop      Juice             4367
2005-01-03 Pop      -                 5653
2005-01-03 -        -                 5653
-          -        -                19254
  13 record(s) selected.
```

Views

When an SQL select is complex and used very often in users' queries or by many applications, consider embedding it in an SQL view, as shown in Example 4-3.

Example 4-3 SQL view

```
create view sales_by_date_Pop as (
select   D.DAY_AD as DAY,P.SUB_PRODUCT_GROUP as GROUP,P.PRODUCTS,
         sum(S.VALUE) as Sale_Amount
from     SALE      S
   join  PRODUCT   P
     on  (S.P_ID = P.P_ID)
   join  DATE      D
     on  (S.D_ID = D.D_ID)
where P.SUB_PRODUCT_GROUP in ('Pop')
group by rollup(D.DAY_AD,P.SUB_PRODUCT_GROUP,P.PRODUCTS)
)
```

Basically the views are helpful in hiding the complexity of the SQL statement. The user only needs to select the output from the view (select * from sales_by_date_pop) instead of writing the long SQL statement, such as the one in Example 4-3.

RDBMS stored procedures

Stored procedures are programs that use data from RDBMS and can be executed from the RDMS environment. There are generally two types of RDBMS procedures, PL SQL-based and External (written in a higher programing language, such as C, Cobol, or BASIC). In Example 4-4, we show a sample DB2 PL SQL-stored procedure using an SQL select code with parameter "Group". In Example 4-4, we use the **`create procedure`** command and pass the procedure with an input parameter specifying a product group. The procedure returns sales amount for all dates for the specific product group defined by parameter. The syntax of command is: **`call sales_by_date('Pop')`**.

Example 4-4 Stored procedure

```
CREATE PROCEDURE sales_by_date (IN i_GROUP char(35))
RESULT SETS 1
LANGUAGE SQL
```

```
SPECIFIC SELECT_LIST
READS SQL DATA
DETERMINISTIC
BEGIN
DECLARE c1 CURSOR WITH RETURN FOR
select D.DAY_AD as DAY, P.SUB_PRODUCT_GROUP as GROUP, P.PRODUCTS, sum(S.VALUE)
as Sale_Amount
from SALE S
   join PRODUCT P on (S.P_ID = P.P_ID)
   join DATE D on (S.D_ID = D.D_ID)
where P.SUB_PRODUCT_GROUP in (i_GROUP)
group by rollup(D.DAY_AD, P.SUB_PRODUCT_GROUP,P.PRODUCTS) order by 1,2,3,4;
open c1;
END@
```

And the result we get is shown in Example 4-5.

Example 4-5 PL SQL stored procedure output for parameter "Pop"

```
  Result set 1
  --------------
 DAY        GROUP    PRODUCTS    SALE_AMOUNT
  ---------- -------- --------- -------------
  2005-01-01 Pop      Milk               1141
  2005-01-01 Pop      Juice              2580
  2005-01-01 Pop      -                  3721
  2005-01-01 -        -                  3721
  2005-01-02 Pop      Milk               1431
  2005-01-02 Pop      Juice              8449
  2005-01-02 Pop      -                  9880
  2005-01-02 -        -                  9880
  2005-01-03 Pop      Milk               1286
  2005-01-03 Pop      Juice              4367
  2005-01-03 Pop      -                  5653
  2005-01-03 -        -                  5653
  -          -        -                 19254
  13 record(s) selected.
Return Status = 0
```

A different SQL command **`call sales_by_date('Water')`** gives us a different result set for Product group *Water* as shown in Example 4-6.

Example 4-6 PL SQL stored procedure output for parameter Water

```
  Result set 1
  --------------

 DAY        GROUP    PRODUCTS  SALE_AMOUNT
 ---------- -------- --------- -----------
```

```
2005-01-01 Water    Soda             1017
2005-01-01 Water    -                1017
2005-01-01 -        -                1017
2005-01-02 Water    Soda              928
2005-01-02 Water    -                 928
2005-01-02 -        -                 928
2005-01-03 Water    Soda              585
2005-01-03 Water    -                 585
2005-01-03 -        -                 585
-          -        -                2530

  10 record(s) selected.

 Return Status = 0
```

Note: Views and stored procedures are also used in the query and reporting environment to control data access for tables and their rows.

4.6.2 Spreadsheets

One of the most widely used tools for analysis is the spreadsheet. It is a very flexible and powerful tool; and therefore, you can find it in almost every enterprise around the world. This is a good-news and bad-news situation. It is good, because it empowers users to be more self-sufficient. It is bad, because it can result in a multitude of independent (non-integrated and non-shared) data sources that exist in any enterprise.

Here are a few examples of spreadsheet use:

- Finance reports, such as a price list or inventory
- Analytic and mathematical functions
- Statistical process control, which is often used in manufacturing to monitor and control quality

4.6.3 Reporting applications

There are many analytical query and reporting applications in the market which work with different relational databases or with special multidimensional structures, such as *cubes*. These applications include:

- IBM Alfablox
- DB2 OLAP Server™
- Microstrategy

- Hyperion Essbase
- Brio
- BusinessObjects
- Cognos Impromptu
- Crystal Reports

In Table 4-1, we list important criteria for you to consider when you are selecting reporting applications for your company.

Table 4-1 Selection criteria for reporting applications

Report authoring and formatting	Drag and drop creation, multiple sources, mixed tables, graphs, tabular formats, style sheets, WYSIWYG, print control, sorting, invoices, and labels
Report distribution	Time or event scheduled; table of contents navigation; formats, such as PDF, HTML, and Excel®; alerts
Analytical functions	Running totals, percent of total, Euro conversion, ranking, highlight exceptions, mining, and binning
Query interaction	Ease of use, shield user from SQL and database navigation complexity, and modify and reuse existing queries
OLAP functionality	Hierarchical summaries, drill-down, drill-up, and view pivoting
SDKs and APIs	Data sources, language libraries, embedded reporting, MDX support, and performance
Security	Report element security
Aggregates	Aggregate awareness

4.6.4 Dashboard and scorecard applications

The most common mechanisms for viewing performance data are *dashboards* and *scorecards*. Top and middle management are not the target users for analytical tools. Their requirements are for viewing performance data at high levels. Dashboards provide the management with a high level view of the data.

A *dashboard* provides a graphical user interface that can be personalized to suit the needs of the user. A dashboard graphically displays *scorecards* that show performance measurements, together with a comparison of these measurements against business goals and objectives.

Note: Top executives are the target users for dashboard applications.

In Figure 4-16 we show a business process management (BPM) dashboard example from the insurance industry. It is not critical to read the values on the dashboard, but to understand the concept. It gives management critical data on a number of strategic elements that require special focus and monitoring. For example, it shows new business growth by category. Management can monitor these elements to make sure they are in line with the business goals and strategy. If not, management can take immediate action.

The dashboard also gives a current status on a number of projects with appropriate alerts.

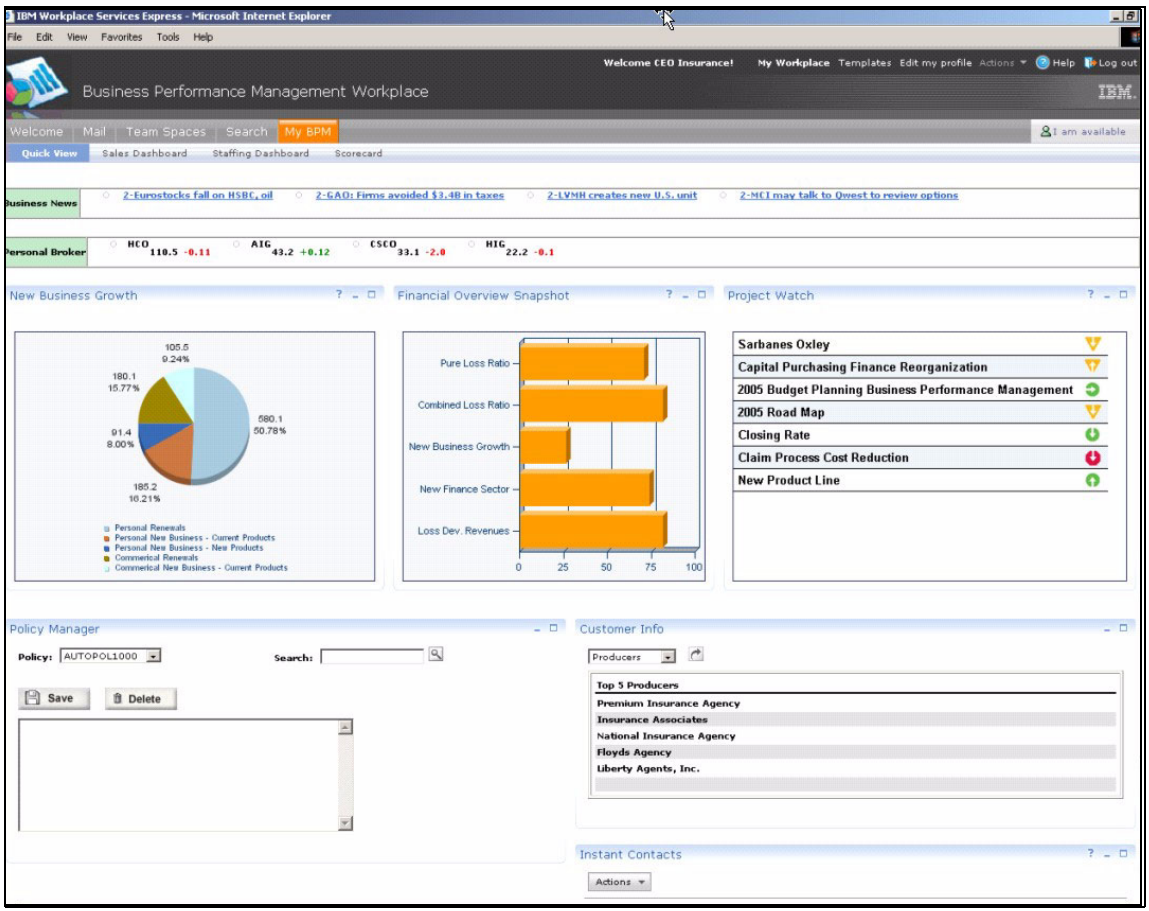


Figure 4-16 Insurance dashboard

In Figure 4-17 on page 100, we show an example retail dashboard. It shows a list of the key business areas and the appropriate alert status for each area. There is also a summary of the current business financial status (shown in the inset). With

this type of information, management now has the capability to not just monitor, but also to impact the measurements.

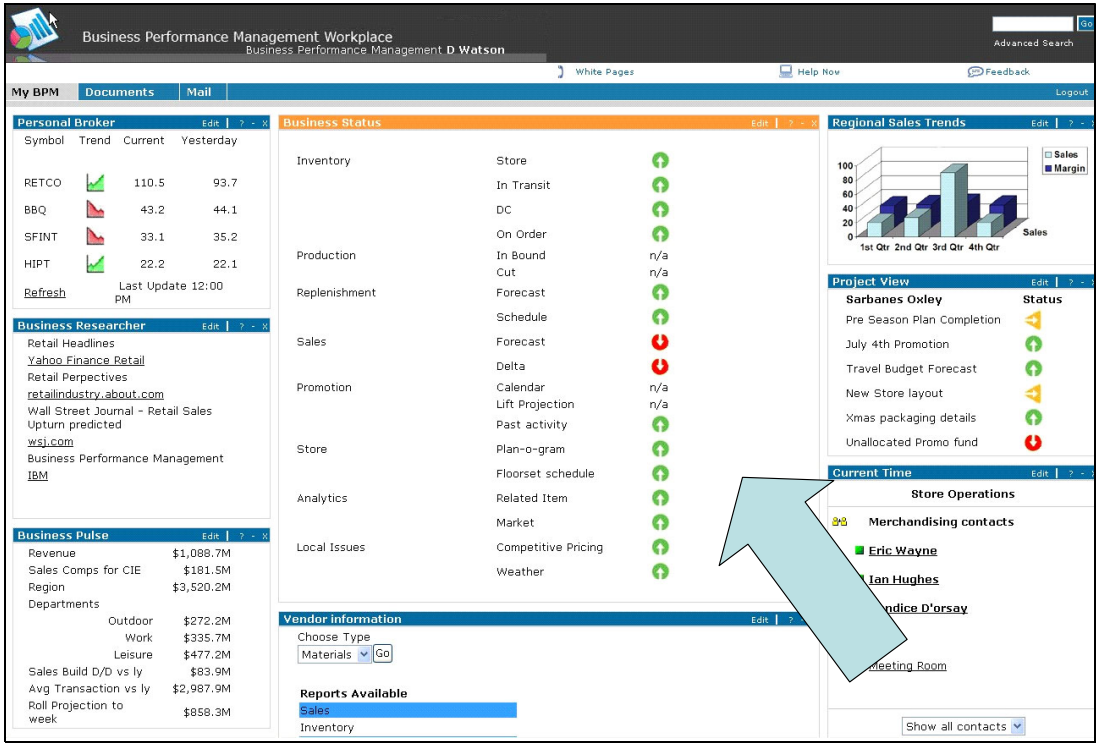


Figure 4-17 Retail Dashboard

Note: The usage of dashboards to help in business performance management is discussed in more detail in the IBM Redbook, *Business Performance Management...Meets Business Intelligence*, SG24-6340.

4.6.5 Data mining applications

Data mining is a relatively new data analysis technique. It is very different from query and reporting and multidimensional analysis in that it uses what is called a *discovery technique*. That is, you do not ask a particular question of the data but rather use specific algorithms that analyze the data and report what they have discovered. Unlike query and reporting and multidimensional analysis where the user has to create and execute queries based on hypotheses, data mining searches for answers to questions that may have not been previously asked. This discovery could take the form of finding significance in relationships between certain data elements, a clustering together of specific data elements,

or other patterns in the usage of specific sets of data elements. After finding these patterns, the algorithms can infer rules. These rules can then be used to generate a model that can predict a desired behavior, identify relationships among the data, discover patterns, and group clusters of records with similar attributes.

Data mining is most typically used for statistical data analysis and knowledge discovery. Statistical data analysis detects unusual patterns in data and applies statistical and mathematical modeling techniques to explain the patterns. The models are then used to forecast and predict. Types of statistical data analysis techniques include:

- Linear and nonlinear analysis
- Regression analysis
- Multi-variant analysis
- Time series analysis

Knowledge discovery extracts implicit, previously unknown information from the data. This often results in uncovering unknown business facts.

Data mining is data driven (see Figure 4-18). There is a high level of complexity in stored data and data interrelationships in the data warehouse that are difficult to discover without data mining. Data mining provides new insights into the business that may not be discovered with query and reporting or multidimensional analysis alone. Data mining can help discover new insights about the business by giving us answers to questions we might never have thought to ask.

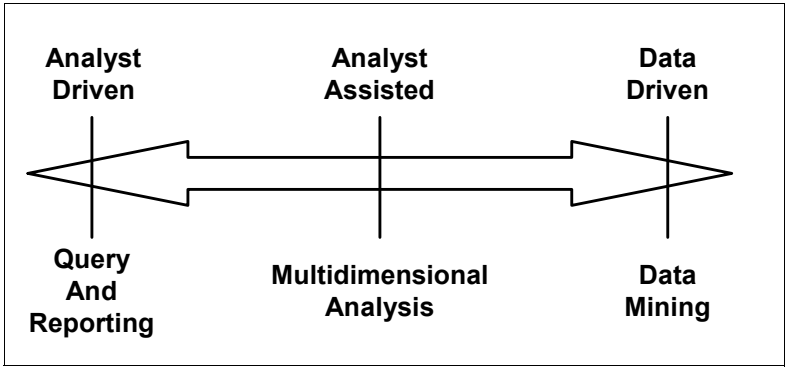


Figure 4-18 Data Mining focuses on analyzing the data content rather than responding to questions

Some data mining tools available in the market are:

- The IBM product for data mining is DB2 Intelligent Miner™, which includes IM Modeling, IM Scoring, and IM Visualization.

- SAS Institute's Enterprise Miner.
- Microsoft Analysis Services.
- Oracle Data mining (separately purchased option within Oracle 10g Enterprise Edition).

5

Dimensional Model Design Life Cycle

In this chapter, we discuss the activities involved in building a dimensional model. This can be a very tedious process because requirements are typically difficult to define. Many times it is only after seeing a result that you can decide that it does, or does not, satisfy a requirement. And, the requirements of an organization change over time. What is valid one day may no longer be valid the next. Regardless, the requirements identified at this point in the development cycle are used to build the dimensional model.

But, where do you start? What do you do first? To help in that decision process, we have developed a dimensional model design life cycle (DMDL) which consists of the following phases:

- Identify business process requirements
- Identify the grain
- Identify the dimensions
- Identify the facts
- Verify the model
- Physical design considerations
- Meta data management

The following sections discuss and describe these phases of the DMDL.

5.1 The structure and phases

The DMDL can help identify the phases and activities that you need to consider in the design of a dimensional model. It is depicted in Figure 5-1. We have used this methodology throughout the book as we work with dimensional models. More specifically, we have used it to help in the example case study documented in Chapter 7, “Case Study: Dimensional model development” on page 333.

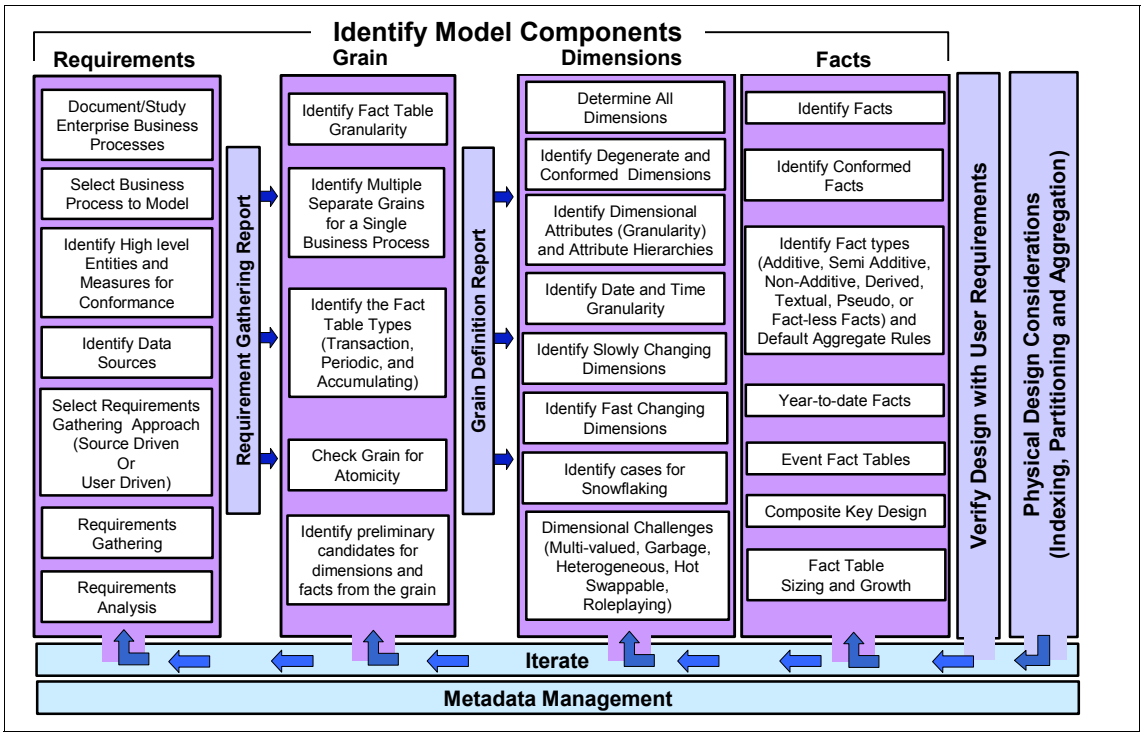


Figure 5-1 Dimensional Model Design Life Cycle

We now describe the phases of the DMDL so you will be familiar with it and better enable its use as you design your next dimensional model:

- **Identify business process requirements:** Involves selection of the business process for which the dimensional model will be designed. Based on the selection, the requirements for the business process are gathered. Selecting a single business process, out of all those that exist in a company, often requires prioritizing the business processes according to criteria, such as business process significance, quality of data in the source systems, and the feasibility and complexity of the business processes.

- **Identify the grain:** Next, we must identify the grain definition for the business process. If more than one grain definition exists for a single business process, then we must design separate fact tables. Avoid forcefully fitting multiple grain definitions into the same fact table. We also need to make sure that we design the grain at the most atomic level of detail so that we can extend the dimensional model to meet future business requirements. In other words, we are able to add new facts and dimensions to the existing model with little or no change to the front-end applications, or any major rework to the existing model.
- **Identify the dimensions:** Here we identify the dimensions that are valid for the grain chosen in the previous step.
- **Identify the facts:** Now we identify the facts that are valid for the grain definition we chose.
- **Verify model:** Before continuing, we must verify that the dimensional model can meet the business requirements. Sometimes it may be required to revisit, and perhaps change, the definition of the grain to assure we can meet the requirements.
- **Physical design considerations:** Now that the model has been designed, we can focus on other considerations, such as performance. It may require tuning by taking actions such as data placement, partitioning, indexing, partitioning, and creating aggregates.

We describe the DMDL by using a retail sales business process. However, there may be concepts of the DMDL that are not applicable to the retail sales business process. For completeness, we cover those concepts in Chapter 6, “Modeling considerations” on page 209.

5.2 Identify business process requirements

During this phase, we identify the business process for which the dimensional model will be designed. However, be aware that a business process may require more than one dimensional model.

In dimensional modeling, the best unit of analysis is the business process in which the organization has the most interest. A business process is basically a set of related activities. Business processes are roughly classified by the topics of interest to the business. To extract a candidate list of high potential business processes necessitates prioritization of requirements. Examples of business processes are customers, profit, sales, organizations, and products.

To help in determining the business processes, use a technique that has been successful for many organizations. Namely, the 5W-1H rule. First determine the

when, where, who, what, why, and how of your business interests. For example, to answer the who question, your business interests may be in customer, employee, manager, supplier, business partner, and/or competitor.

Note: When we refer to a business process, we are not simply referring to a business department. For example, consider a scenario where the sales and marketing department access the orders data. We build a single dimensional model to handle orders data rather than building separate dimensional models for the sales and marketing departments. Creating dimensional models based on departments would no doubt result in duplicate data. This duplication, or data redundancy, can result in many data quality and data consistency issues.

Before beginning, recall the various architectures used for data warehouse design we discussed in 3.3, “Data warehouse architecture choices” on page 57. They were:

- Business-wide enterprise data warehouse
- Independent data warehouse
- Dependent data warehouse

Create a dimensional model for a business process directly from OLTP source systems, as in the case of independent and dependent data warehouse design architectures. Or create a dimensional model for a business process from the enterprise data warehouse, as in the case of the business-wide enterprise data warehouse architecture.

Figure 5-2 on page 107 shows that the source for a dimensional model can be:

- An enterprise wide data warehouse
- OLTP source systems (in the case of independent or dependent data mart architectures)
- Independent data marts (in this situation, you might be interested in consolidating the independent data marts into another data mart or data warehouse)

Note: For more information on data mart consolidation, refer to the IBM Redbook, *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653.

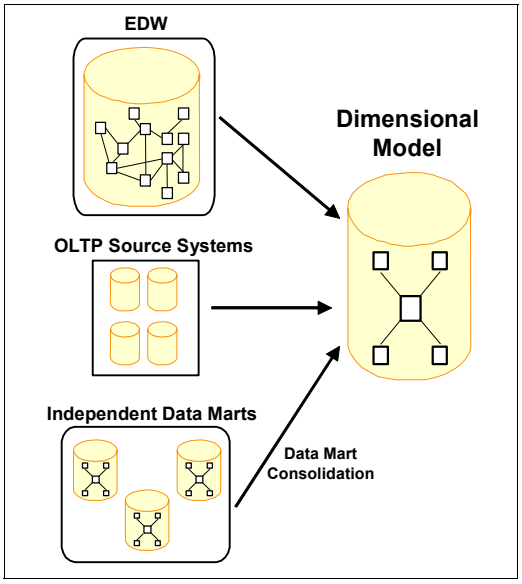


Figure 5-2 Dimensional model sources

Data warehouse and the dimensional model

When you consider the partitioning of the data in a data warehouse, the most common criterion is subject area. As you may remember, a data warehouse is subject-oriented. It is oriented to specific selected subject areas in the organization, such as customer and product. For a practical implementation of a data warehouse, we suggest that the unit of measure is the business process.This is quite different from partitioning in the operational environment.

OLTP systems and the dimensional model

In the operational environment, partitioning is more typically by application or function because the operational environment has been built around transaction-oriented applications that perform a specific set of functions. And, typically, the objective is to perform those functions as quickly as possible. If there are queries performed in the operational environment, they are more tactical in nature and are to answer questions concerned with that instant in time. An example is, “Has the check from Mr. Smith been processed?” Queries in the data warehouse environment are more strategic in nature and intend to ask questions concerned with a larger scope. An example of a query is, “What products are selling well?” or “Where are my weakest sales offices?” To answer those queries, the data warehouse is structured and oriented to subject areas such as product or organization. These subject areas are the most common unit of logical partitioning in the data warehouse.

Figure 5-3 depicts an E/R model (which can be an OLTP or an enterprise data warehouse) that consists of several business processes.

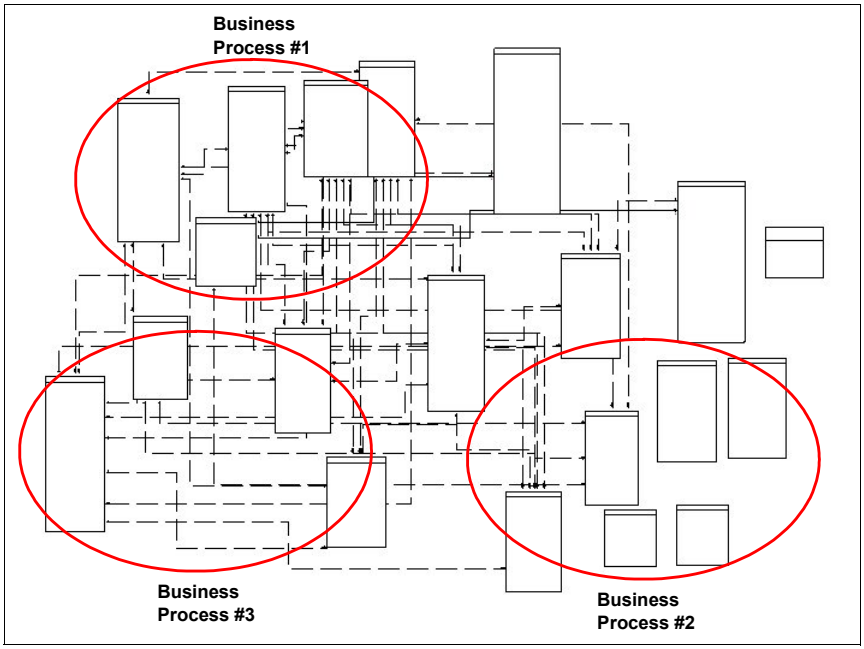


Figure 5-3 E/R model consists of several business processes

5.2.1 Create and Study the enterprise business process list

During this activity, we create a complete enterprise-wide business process list. A sample business process list is shown in Table 5-1 on page 109, along with a number of other assessment factors, such as:

- Complexity of the source systems of each business process
- Data availability of these systems
- Data quality of these systems
- Strategic business significance of each business process

Table 5-1 on page 109 also shows the *value points* along with the assessment factors involved. For example, for the business process *Finance*, the Complexity = High(1). The 1 here is the assigned value point. The value points for each assessment factor are listed in detail in Table 5-2 on page 109.

Table 5-1 Enterprise business process list

Name of business process	Complexity	Data availability	Data quality	Strategic business significance
Retail sales	Low (3)	High (3)	High (3)	High (6)
Finance	High (1)	High (3)	Medium (2)	Medium (4)
Servicing	Low (3)	High (3)	Medium (2)	High (6)
Marketing	Medium (2)	Medium (2)	Medium (2)	Medium (4)
Shipment	Low (3)	Low (1)	High (3)	Low (2)
Supply management	Medium (2)	Low (1)	Medium (2)	Low (2)
Purchase orders	High (1)	Medium (1)	Low (1)	Medium (4)
Labor	Low (3)	Low (1)	Low (1)	High (2)

Table 5-2 shows the value points for each of the assessment factors:

- Complexity
- Data availability
- Data quality
- Strategic business significance

Table 5-2 Value points table for assessment factors

Assessment Factor	Low	Medium	High
Complexity	3	2	1
Data availability	1	2	3
Data quality	1	2	3
Strategic business significance	2	4	6

Note: The value points, or weight, given for the assessment factor, *Strategic business significance*, is more than the other factors. This is simply because the company has decided that strategically significant business processes should be given a higher value than the other assessment factors.

We recommend that you include the assessment factors you think are important for your business processes and assign them with your evaluation of the

appropriate points or weight. For example purposes, we have included only four factors in Table 5-2 on page 109.

This exercise will help you quickly prioritize the business processes for which dimensional models should be built.

5.2.2 Identify business process

In this phase, we prioritize the business processes. The basic idea here is to identify the most and least feasible processes for building a dimensional model.

Table 5-3 shows a business process list in descending order of priority. The priority is based on the number of points in the *Point sum* column. It is a summary of all the assessment factor points. Table 5-3 is created from the enterprise-wide business process list shown in Table 5-1 on page 109.

For example, when we look at the finance business, the point sum column has a value of 10 (1 + 3 + 2 + 4), which is the sum of all assessment factor points.

Table 5-3 Enterprise-wide business process priority listing

Name of business process	Complexity	Data availability	Data quality	Strategic business significance	Point sum
Retail sales	Low (3)	High (3)	High (3)	High (6)	15
Finance	High (1)	High (3)	Medium (2)	Medium (4)	10
Servicing	Low (3)	High (3)	Medium (2)	High (6)	14
Marketing	Medium (2)	Medium (2)	Medium (2)	Medium (4)	10
Shipment	Low (3)	Low (1)	High (3)	Low (2)	9
Supply Mgmt	Medium (2)	Low (1)	Medium (2)	Low (2)	7
Purchase Order	High (1)	Medium (1)	Low (1)	Medium (4)	7
Labor	Low (3)	Low (1)	Low (1)	High (2)	7

From Table 5-3, it is observed that the retail sales process gets the highest priority. Therefore, it is also likely that this will be the business process for which the first dimensional model (and data mart) will be built.

Therefore, it is the business process we have selected for the example case study dimensional model in this chapter.

Note: Table 5-3 serves only as a guideline for identifying high priority and feasible business processes in our example. You will go through a similar methodology in prioritizing the business processes in your company.

5.2.3 Identify high level entities and measures for conformance

The next step is to determine the high level business entities involved in each process. We depict this in Table 5-4. The idea is to determine which entities are common across several business processes. Once identified, we use these entities as common across all dimensional models (data marts) in the enterprise. Each business process will then be tied together through these common (conformed) dimensions.

To create conformed dimensions that are used across the enterprise, the various businesses must agree on the definitions for these common entities. For example, as shown in Table 5-4, the business processes, Retail sales, Finance, Servicing, Marketing, Shipment, Supply management, and Purchase order, should agree on a common definition of the Product entity, because all these business processes have the product entity in common. The product entity then becomes a conformed Product dimension which is shared across all the business processes.

However, getting agreement on definitions of common entities, such as product and customer, can be difficult because they typically vary from one business process to another. And, therefore, changes will likely impact existing applications.

Table 5-4 Business processes and high level entities

High Level Entities → **Business processes**	**Product**	**Selling date**	**Customer**	**Employee**	**Received date**	**Ledger**	**Supplier**	**Warehouse**	**Check**	**Store**	**Shipment company**	**.....**	**And more**
Retail sales	X	X	X	X			X			X			
Finance	X	X											
Servicing	X	X	X	X									
Marketing	X		X	X									
Shipment	X		X	X	X		X	X			X		

High Level Entities → / Business processes	Product	Selling date	Customer	Employee	Received date	Ledger	Supplier	Warehouse	Check	Store	Shipment company	.....	And more
Supply management	X			X	X		X	X					
Purchase order	X			X					X				
Human resources				X									

What are conformed dimensions?

A data warehouse must provide consistent information for queries requesting similar information. One method to maintain consistency is to create dimension tables that are shared (and therefore conformed), and used by all applications and data marts (dimensional models) in the data warehouse. Candidates for shared or conformed dimensions include customers, time, products, and geographical dimensions, such as the store dimension.

What are conformed facts?

Fact conformation means that if two facts exist in two separate locations, then they must have the same name and definition. As examples, revenue and profit are each facts that must be conformed. By conforming a fact, then all business processes agree on one common definition for the revenue and profit measures. Then, revenue and profit, even when taken from separate fact tables, can be mathematically combined.

Establishing conformity

Developing a set of shared, conformed dimensions is a significant challenge. Any dimensions that are common across the business processes must represent the dimension information in the same way. That is, it must be conformed. Each business process will typically have its own schema that contains a fact table, several conforming dimension tables, and dimension tables unique to the specific business function. The same is true for facts.

5.2.4 Identify data sources

In this activity, we identify the data sources involved with the business processes. Table 5-5 on page 113 shows a sample listing of business processes and their respective data sources, along with their owner, location, and platform. Other factors can also be associated to describe and document the data source

in more detail. However, for simplicity of the example, we have chosen the name, owner, location, and platform of the data source.

Table 5-5 Data sources for business process

Business process	Data sources	Owner	Location	Platform
Retail sales	Source 1	Order admin	New Jersey	DB2
Finance	Source 2	Finance admin	New York	DB2
And more				

5.2.5 Select requirements gathering approach

The traditional development cycle focuses on automating the process, making it faster and more efficient. The dimensional model development cycle focuses on facilitating the analysis that will change the process to make it more effective. Efficiency measures how much effort is required to meet a goal. Effectiveness measures how well a goal is being met against a set of expectations.

As previously mentioned, requirements are typically difficult to define. Often, it is only after seeing a result that you can decide that it does, or does not, satisfy a requirement. And, the requirements of an organization change over time. What is valid one day may no longer be valid the next day. Regardless, we use the requirements identified at this point in the development cycle to build the dimensional model.

The question then is how can you build something that cannot be precisely defined? And, how do you know when you have successfully identified the requirements? Although there is no definitive test, we propose that if your requirements address the following questions, then you probably have enough information to begin modeling.

The questions relate to who, what, when, where and how

- **Who** are the people, groups, and organizations of interest?
- **What** functions need to be analyzed?
- **Why** is the data required?
- **When** does the data need to be recorded?
- **Where**, geographically and organizationally, do relevant processes occur?
- **How** do we measure the performance of the functions being analyzed?
- **How** is performance of the business process measured? What factors determine the success or failure?

- **What** is the method of information distribution? Is it a data report, paper, pager, or e-mail (examples)?
- **What** types of information are lacking for analysis and decision making?
- **What** steps are currently taken to fulfill the information gap?
- **What** level of detail would enable data analysis?

There are likely many other methods for deriving business requirements. However, in general, these methods can be placed in one of two categories:

- Source-driven
- User-driven

Figure 5-4 depicts an example of this methodology.

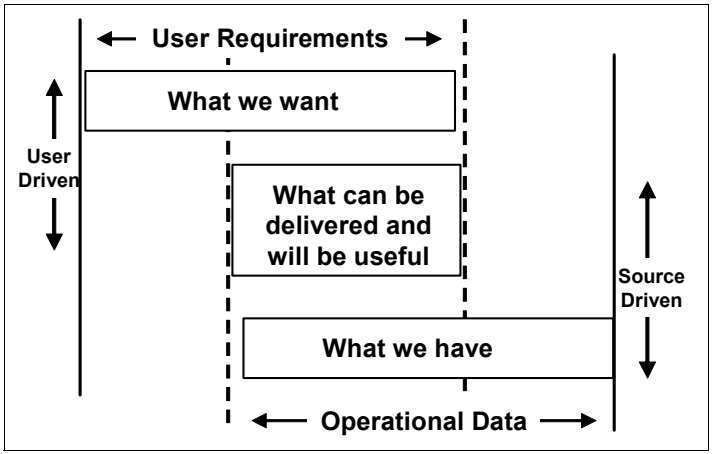


Figure 5-4 Source-driven and User-driven requirements gathering

Source-driven requirements gathering

Source-driven requirements gathering, as the name implies, is a method based on defining the requirements by using the source data in production operational systems. This is done by analyzing an E/R model of source data if one is available or the actual physical record layouts and selecting data elements deemed to be of interest.

The major advantage of this approach is that you know from the beginning that you can supply all the data, because you are already limiting yourself to what is available. A second benefit is that you can minimize the time required by the users in the early stages of the project. However, there is no substitute for the importance and value you get when you get the users involved.

Of course there are also disadvantages to this approach. By minimizing user involvement, you increase the risk of producing an incorrect set of requirements. Depending on the volume of source data you have, and the availability of E/R models for it, this can also be a very time-consuming approach. Perhaps most important, some of the user's key requirements may need data that is currently unavailable. Without the opportunity to identify these requirements, there is no chance to investigate what is involved in obtaining external data. External data is data that exists outside the enterprise. Even so, external data can often be of significant value to the business users.

The result of the source-driven approach is to provide what you have. We think there are at least two cases where this is appropriate. First, relative to dimensional modeling, it can be used to develop a fairly comprehensive list of the major dimensions of interest to the enterprise. If you ultimately plan to have an enterprise-wide data warehouse, this could minimize the proliferation of duplicate dimensions across separately developed data marts. Second, analyzing relationships in the source data can identify areas on which to focus your data warehouse development efforts.

User-driven requirements gathering

User-driven requirements gathering is a method based on defining the requirements by investigating the functions the users perform. This is usually done through a series of meetings and/or interviews with users.

The major advantage to this approach is that the focus is on providing what is really needed, rather than what is available. In general, this approach has a smaller scope than the source-driven approach. Therefore, it generally produces a useful data warehouse or a data mart in a shorter time span.

On the negative side, expectations must be closely managed. The users must clearly understand that it is possible that some of the data they need can simply not be made available for a variety of possible reasons. But, try not to limit the things for which the user asks. Outside-the-box thinking should be promoted when defining requirements for a data warehouse. This prevents you from eliminating requirements simply because you think they might not be possible. If a user is too tightly focused, it is possible to miss useful data that is available in the production systems.

We believe user-driven requirements gathering is the approach of choice, especially when developing dependent data marts or populating data marts from a business-wide enterprise warehouse.

5.2.6 Requirements gathering

During requirements gathering, business users needs are collected and documented. Requirements gathering focuses on the study of business processes and information analysis activities with which users are involved. A user typically needs to evaluate, or analyze, some aspect of the organization's business. It is extremely important that requirements gathering focus on the two key elements of analysis that business users are involved with on a day-to-day basis:

- What is being analyzed?
- The evaluation criteria

Requirements gathering, therefore, is extremely oriented toward understanding the problem domain for which the modeling is done. Typically, requirements at this stage are documented rather informally or, at least, they are not represented in detailed schemas.

Assume that we are designing a dimensional model for a retail sales business process. We identified the retail sales business process from section 5.2.2, "Identify business process" on page 110. Figure 5-5 shows the E/R model for this business process.

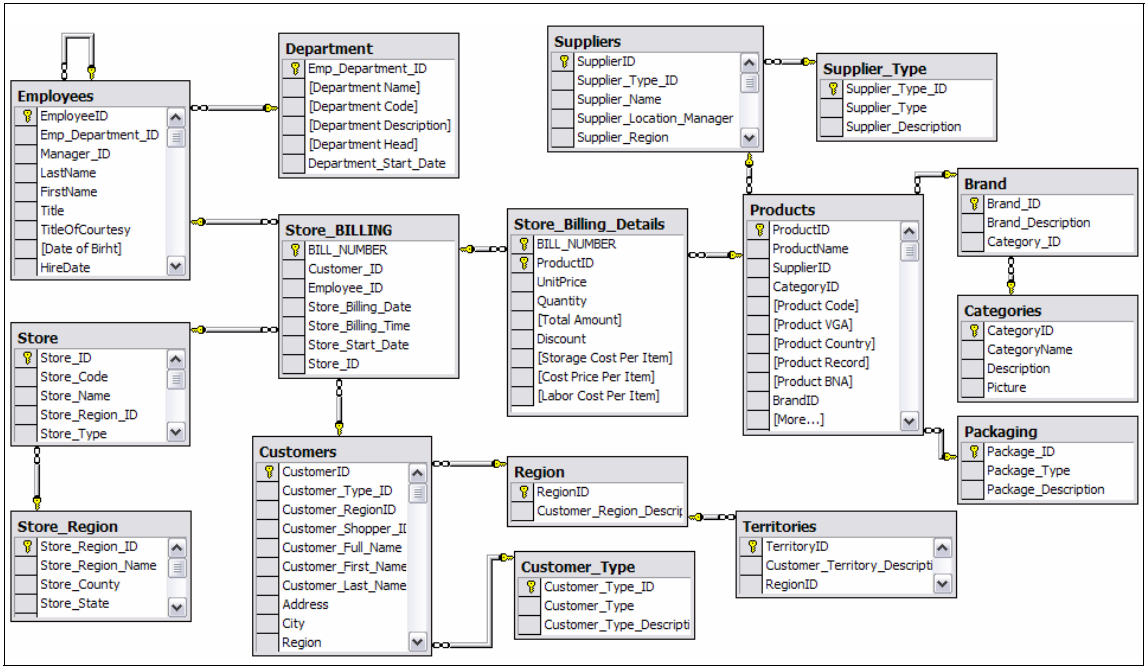


Figure 5-5 E/R Model for the retail sales business process

Now assume that we perform business requirements analysis for the retail sales process. We identify that the business needs the answers to the questions shown in Table 5-6.

Table 5-6 User requirements for the retail sales process

Seq. no.	Business requirement	Importance
Q1	What is the average sales quantity this month for each product in each category?	Medium
Q2	Who are the top 10 sales representatives and who are their managers? What were their sales for the products they sold in the first and last fiscal quarters?	High
Q3	Who are the bottom 20 sales representatives and who are their managers?	High
Q4	How much of each product did U.S. and European customers order, by quarter, in 2005?	High
Q5	What are the top five products sold last month by total revenue? By quantity sold? By total cost? Who was the supplier for each of those products?	High
Q6	Which products and brands have not sold in the last week? The last month?	High
Q7	Which salespersons had no sales recorded last month for each of the products in each of the top five revenue generating countries?	High
Q8	What was the sales quantity of each of the top five selling products on Christmas, Thanksgiving, Easter, Valentines Day, and the Fourth of July?	Medium
Q9	What are the sales comparisons of all products sold on weekdays compared to weekends? Also, what was the sales comparison for all Saturdays and Sundays every month?	High
Q10	What are the 10 top and bottom selling products each day and week? Also at what time of the day do these sell? Assuming there are five broad time periods- Early morning (2AM-6AM), Morning (6AM-12PM), Noon (12PM-4PM), Evening (4PM-10PM), and Late night (10PM-2AM).	Time, Product

The column named *Importance (Low/Medium/High or Critical)* signifies the importance of the questions. Some questions (business needs) may be highly significant for the business where other questions may be less significant in comparison.

In addition to the set of business requirements listed in Table 5-6 on page 117, it is very important to understand how the business wants to preserve the history. In other words, how does the business want to record the changed data. Some of the questions (in regard to maintaining history) that you may ask are listed in Table 5-7.

Table 5-7 Questions relating to maintaining history

Seq. no.	Question	How to maintain history
1	What happens if an employee changes from Region A to Region B?	Overwrite history or maintain history
2	What happens if an employee changes from Manager A to Manager B?	Overwrite history or maintain history
3	What happens if a product changes from existing Category A to Category B?	Overwrite history or maintain history
4	What happens when a product is discontinued?	Overwrite history or maintain history
5	More questions	Overwrite history or maintain history

5.2.7 Requirements analysis

During requirements analysis, informal requirements (as gathered in section 5.2.6, “Requirements gathering” on page 116) are further investigated and high level measures and high level entities (potential future dimensions) are produced.

Table 5-8 shows high level entities and measures identified from the requirements stated in Table 5-6 on page 117 for the retail sales business process. Note that the high level entities are potential future dimensions.

Table 5-8 Requirement Analysis - High level entities and measures

Seq. no.	Business requirement	High level entities	Measures
Q1	What is the average sales quantity this month for each product in each category?	Month, Product	Quantity of units sold
Q2	Who are the top 10 sales representatives and who are their managers? What were their sales in the first and last fiscal quarters for the products they sold?	Sales representative, Manager, Fiscal Quarter, Product	Revenue sales

Seq. no.	Business requirement	High level entities	Measures
Q3	Who are the bottom 20 sales representatives and who are their managers?	Sales representative, Manager	Sales Revenue
Q4	How much of each product did U.S. and European customers order, by quarter, in 2005?	Product, Customers, Quarter, Year	Quantity of units sold
Q5	What are the top five products sold last month by total revenue? By quantity sold? By total cost? Who was the supplier for each of those products?	Product, Supplier	Revenue sales, Quantity of units sold, Total Cost
Q6	Which products and brands have not sold in the last week? The last month?	Product, Brand, Month	Sales Revenue
Q7	Which salespersons had no sales recorded last month for each of the products in each of the top five revenue generating countries?	Salesperson, Product, Customer country, Month	Sales Revenue
Q8	What was the sales quantity of the top five selling products on Christmas, Thanksgiving, Easter, Valentines Day and the Fourth of July?	Holidays, Products	Sales revenue
Q9	What are the sales comparisons of all products sold on weekdays compared to weekends? Also, what was the sales comparison for all Saturdays and Sundays every month?	Products, Weekdays, Weekends	Sales revenue
Q10	What are the 10 top and bottom selling products each day and week? Also, at what time of the day do these sell? Assuming there are five broad time periods- Early morning (2AM-6AM), Morning (6AM-12PM), Noon (12PM-4PM), Evening (4PM-10PM) Late night (10PM-2AM).	Time, Product	Sales revenue

We listed each business requirement in Table 5-8 on page 118 and identified some of the high-level entities and measures. Using this as a starting point, we now filter down the entities and measures as shown in Table 5-9 on page 120.

Also, any hierarchies associated with each of these high-level entities (which could be transformed to dimensions later) are documented.

Table 5-9 Final set of high level entities and measures

Entities (potential future dimensions)	**Hierarchy in entity (potential future dimension)**
Customer (Customer country, such as U.S. or Europe)	Country → Region
Product (Brand, category)	Category → Brand
Selling Date (Holidays, weekends, weekdays, month, quarter, year)	Fiscal Year → Fiscal Quarter → Fiscal Month
Supplier of product	
Time of selling	Hour → Minute
Employee or salesperson or sales representative	
Measures **(Key Performance Indicators)**	Revenue sales, Quantity of units sold, Total cost

Note: The high level dimensions and measures are useful in determining the grain, dimensions, and facts during the different phases of the DMDL.

5.2.8 Business process analysis summary

The final output of the *Identify business process* phase is the requirements gathering report, which contains the following:

- Business process listing
- Business process prioritization
- High level entities and measures, which are common between various business processes
- Business process identified for which the dimensional model will be built
- Data sources listing
- Requirement gathering which contains all business process requirements

- Requirement gathering analysis
- High level entities and measures identified from the requirement analysis

5.3 Identify the grain

In the *Identify the grain* phase, we focus on the second step of the DMDL as shown in Figure 5-6.

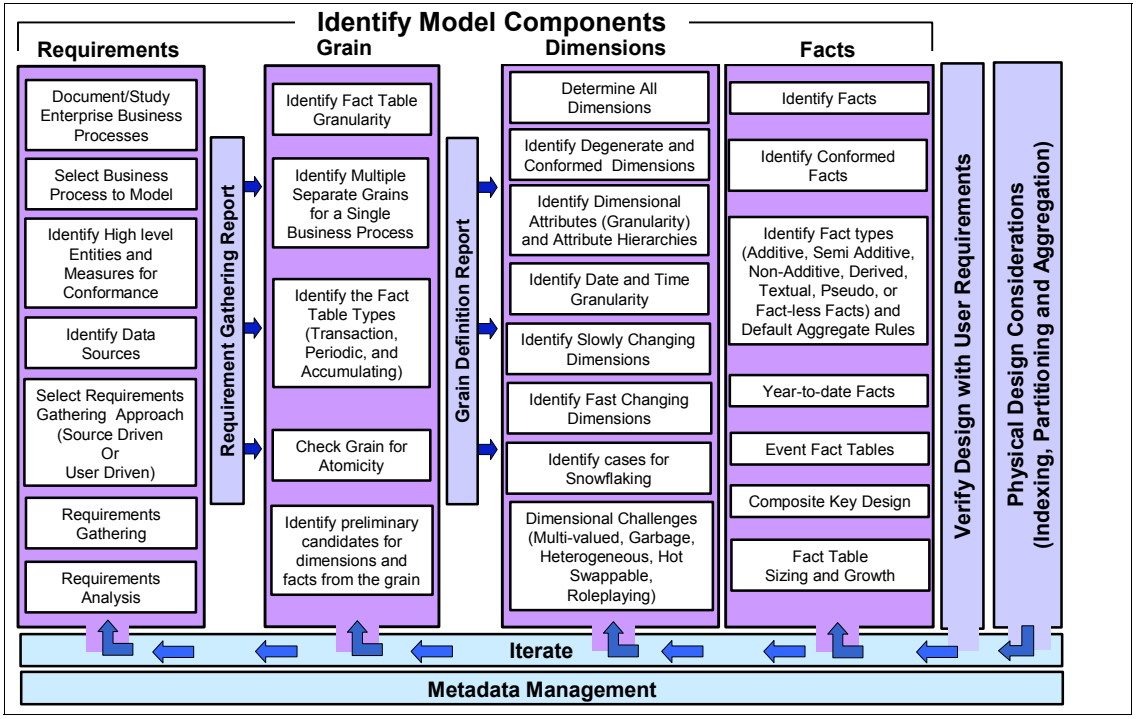


Figure 5-6 Dimensional Model Design Life Cycle

What is the grain?

The following are several characteristics of grain identification:

- When identifying the grain, we must specify exactly what a fact table record means. The *grain* conveys the level of detail associated with the fact table measurements. Identifying the grain also means deciding on the level of detail you want to be made available in the dimensional model. The more detail there is, the lower the level of granularity. The less detail there is, the higher the level of granularity.

- The level of detail available in a star schema (dimensional model) is referred to as the grain. Each fact and dimension table is said to have its own grain or granularity. In other words, each table (either fact or dimension) will have some level of detail associated with it. The grain of the dimensional model is the finest level of detail implied by the joining of the fact and dimension tables. For example, the granularity of a dimensional model consisting of dimensions, date (year, quarter, month, and day), store (region, district, and store) and product (category name, brand, and product) is *product sold in store by day.*
- Both the dimension and fact tables have a grain associated with them. To understand the grain of a dimension table, we need to understand the attributes of the dimension table. Every dimension has one or more attributes. Each attribute associates a parent or child with other attributes. This parent-child relationship provides different levels of summarization. *The lowest level of summarization or the highest level of detail is referred to as the grain*. The granularity of the dimension affects the design, and can impact such things as the retrieval of data and data storage.
- *A grain refers to the level of detail present in each fact table*. Each row should hold the same type of data. For example, each row could contain daily sales by store by product or daily line items by store.

Examples of grain definitions are:

- A line item on a grocery receipt
- A single item on an invoice
- A single item on a restaurant bill
- A line item on a bill received by a hospital
- A monthly snapshot of a bank account statement
- A weekly snapshot of the number of products in the warehouse inventory
- A single airline ticket purchased on a day
- A single bus ticket purchased on a day

What is Granularity?

The fact and dimension tables have a granularity associated with them. In dimensional modeling, the term *granularity* refers to the level of detail stored in a table.

For example, a dimension such as date (year, quarter) has a granularity at the quarter level but does not have information for individual days or months. And, a dimension such as date (year, quarter, month) table has granularity at the month level, but does not contain information at the day level.

Note: Differing data granularities can be handled by using multiple fact tables (daily, monthly, and yearly tables) or by modifying a single table so that a granularity flag (a column to indicate whether the data is a daily, monthly, or yearly amount) can be stored along with the data. However, we do not recommend storing data with different granularities in the same fact table.

5.3.1 Fact table granularity

For our example retail sales business process from section 5.2, “Identify business process requirements” on page 105, we identify the grain definition as *an individual line item on a grocery store bill*. The grain detail is based on the requirements findings that were analyzed and documented in section 5.2.7, “Requirements analysis” on page 118.

The grain definition is represented graphically in Figure 5-7 on page 124. It is important to understand that while gathering business requirements, you should collect documents, such as invoices, receipts, and order memos. These often have information which can be used to define the grain. Also such documents have information which helps identify the dimensions and facts for the dimensional models.

Note: The grain you choose determines the level of detailed information that can be made available to the dimensional model. *Choosing the right grain is the most important step for designing the dimensional model.*

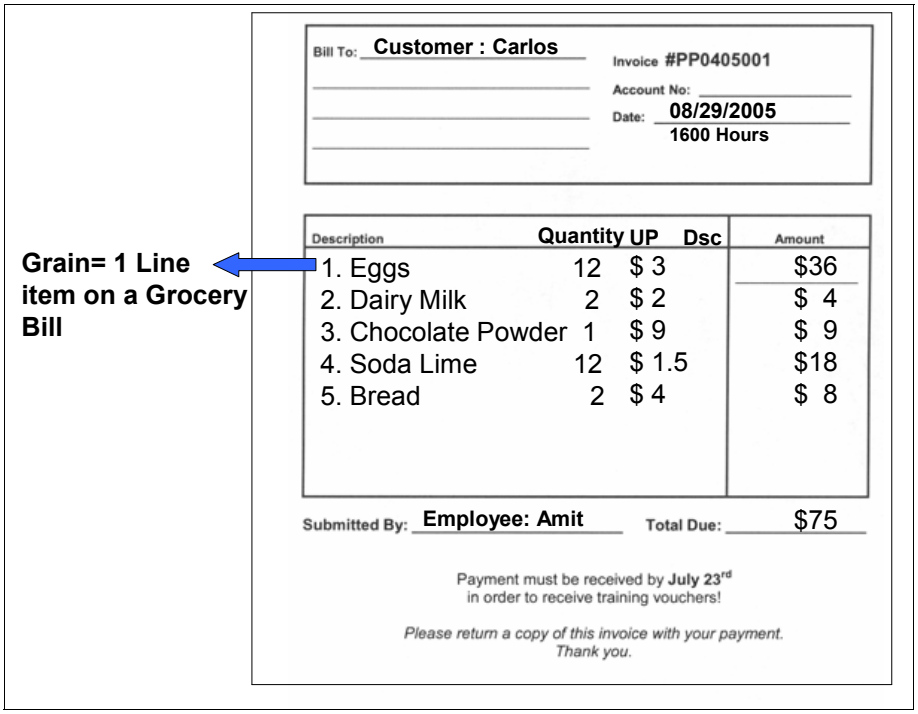


Figure 5-7 Grain example: A single line item on a grocery bill

Guidelines for choosing the fact table granularity

The grain definition is the base of every dimensional model. It determines the level of information that is available. Guidelines for choosing the grain definition are:

- During the business requirements gathering phase, try to collect any documents, such as invoice forms, order forms, and sales receipts. Typically, you will see that these documents have transactional data associated with them, such as order number and invoice number.
- Documents can often point you to the important elements of the business, such as customer and the products. They often contain information at the lowest level that may be required by the business.
- Another important point to consider is the date. Understand to what level of detail a date is associated with a customer, product, or supplier. Is the information in the source systems available at the day, month, or year level? When developing the grain, decide whether you would like information to be stored at a day, month, or year level.

5.3.2 Multiple, separate grains

The primary focus of this activity is to determine if there are multiple grains associated with the business process for which you are designing your dimensional model.

There can be more than one grain definition associated with a single business process. In these cases, we recommend that you design separate fact tables with separate grains and not forcefully try to put all facts in a single fact table.

Differing data granularities can be handled by using multiple fact tables (daily, monthly, and yearly tables, as examples). Also, consider the amount of data, space, and the performance requirements when you decide how to handle multiple granularities.

Criteria for one or multiple fact tables

To determine whether to use one or multiple fact tables, consider the following criteria:

- One of the most important sets of criteria that helps determine the need for one or multiple fact tables are the facts. It is important to understand the dimensionality of the facts to decide whether the facts belong together in one fact table or separate in fact tables with different grains. For example, consider Figure 5-7 on page 124 which shows the grain equivalent to a single line item on a bill. Facts, such as quantity, sales price, and discount per item, are true to the grain. But facts, such as entire order total or entire order quantity, would not be true to the grain definition of a single line item on a bill. We explain the concept of identifying and handling separate grains for the same business process in more detail in “Handling multiple, separate grains for a business process” on page 225.
- Are multiple OLTP source systems involved? Remember, each source system is designed with a particular and very specific purpose. If two source systems were not serving different purposes, it would be better to have one. Often each source system will cater to a particular requirement of the business. Generally, if we are dealing with business processes, such as order management, store inventory, or warehouse inventory, it is likely that separate source systems are involved, and probably the use of separate fact tables is appropriate.
- It is also important to determine if multiple, unrelated business processes are involved. Unrelated business processes involve the creation of multiple separate fact tables. And, it is possible that a single business process may involve creation of separate fact tables to handle facts that have different granularity.

- If you find that a certain dimension is not true to the grain definition, then this is a sign that it may belong to a new fact table with its own grain definition.
- Consider the timing and sequencing of events. It may lead to separate processes handling a single event. For example, a company markets its product. Customers order the products. The accounts receivable department produces an invoice. The customer pays the invoice. After the purchase, the customer may return some of the products or send some back for repairs. If any of the products are out of warranty, this requires new charges, and so on. Here, several processes are involved in the sequence of single purchase event. And, each of these processes are likely working with a particular, and different, point in time. Then, each of these processes would need to be handled using separate fact tables.

For the example of retail sales, assume that we do not have multiple fact tables, because the facts (unit sales price (UP), quantity, cost price (Amount), and revenue earned (Total Due)) defined in Figure 5-7 on page 124 are true to the grain of the fact table. Therefore, we have one single grain definition, which is an individual line item on a bill.

We discuss the concept of identifying multiple separate grain definitions (multiple fact tables) for a single business process in section 6.2.1, “Handling multiple, separate grains for a business process” on page 225.

Multiple granularities in a single fact table

It is possible to have multiple grains in one fact table. This can be accommodated by adding a column called the *granularity flag*. Such a column would indicate whether the data or row in the fact table is at the daily, monthly, quarterly, or yearly level. Even though it is possible to store multiple grains in a single fact table, we do not recommend this approach. We suggest handling multiple grain definitions by designing separate fact tables and star schemas. Then, if desired, the separate star schemas may be related by use of conformed dimensions and conformed facts. For more detail, see section 5.4.3, “Conformed dimensions” on page 144 and 5.5.2, “Conformed facts” on page 174.

5.3.3 Fact table types

In this activity, we identify the type of fact table involved in the design of the dimensional model.

There are three types of fact tables:

- **Transaction:** A transaction-based fact table records one row per transaction. A detailed discussion about the transaction fact table is available in “Transaction fact table” on page 231.

- **Periodic:** A periodic fact table stores one row for a group of transactions that happen over a period of time. A detailed discussion about the periodic fact table is available in “Periodic fact table” on page 232.
- **Accumulating:** An accumulating fact table stores one row for the entire lifetime of an event. As examples, the lifetime of a credit card application from the time it is sent, to the time it is accepted, or the lifetime of a job or college application from the time it is sent, to the time it is accepted or rejected. For a detailed discussion of this topic, see “Accumulating fact table” on page 233.

We summarize the differences among these types of fact tables in Table 5-10. It emphasizes that each has a different type of grain. And, there are differences in ways that inserts and updates occur in each. For example, with transaction and periodic fact tables, only inserts occur. But with the accumulating fact table, the row is first inserted. And as a milestone is achieved and additional facts are made available, it is subsequently updated.

We discuss each of the three fact tables in more detail, with examples, in 6.2.3, “Designing different grains for different fact table types” on page 230.

Table 5-10 Comparison of fact table types

Feature	Transaction	Periodic	Accumulating
Grain	One row per transaction.	One row per time period.	One row for the entire lifetime of an event.
Dimension	Date dimension at the lowest level of granularity.	Date dimension at the end-of-period granularity.	Multiple date dimensions.
Number of dimensions	More than periodic fact type.	Fewer than transaction fact type.	Highest number of dimensions when compared to other fact table types.
Conformed dimensions	Uses shared conformed dimensions.	Uses shared conformed dimensions.	Uses shared conformed dimensions.
Facts	Related to transaction activities.	Related to periodic activities.	Related to activities which have a definite lifetime.
Conformed facts	Uses shared conformed facts.	Uses shared conformed facts.	Uses shared conformed dimensions.

Feature	Transaction	Periodic	Accumulating
Database size	Largest size. At the most detailed grain level, tends to grow very fast.	Smaller than the Transaction fact table because the grain of the date and time dimension is significantly higher.	Smallest in size when compared to the Transaction and Periodic fact tables.
Performance	Performs well and can be improved by choosing a grain above the most detailed.	Performs better than other fact table types because data is stored at a less detailed grain.	Performs well.
Insert	Yes	Yes	Yes
Update	No	No	Yes, when a milestone is reached for a particular activity.
Delete	No	No	No
Fact table growth	Very fast.	Slow in comparison to transaction-based fact table.	Slow in comparison to the transaction and periodic fact table.
Need for aggregate tables	High, primarily because the data is stored at a very detailed level.	No or very Low, primarily because the data is already stored at a high aggregated level.	Medium, because the data is primarily stored at the day level. However, the data in accumulating fact tables is lower than the transaction level.

Note: In the example retail sales business process, the grain is the transaction fact table type.

5.3.4 Check grain atomicity

In this activity, we review the atomicity (level of detail) of the grain to assure it is at the most detailed level. This decision should include consideration for anticipated future needs in order to minimize the potential for a required redesign as business requirements change.

The importance of having detailed atomic grain

Grain of the dimensional model is extremely important for the dimensional design. Even if the business requirements need information at the monthly or quarterly level, it is better to have the information made available at the daily level. This is because the more detailed (atomic) the dimensions are, the more detailed information that canj be provided to the business.

For example, consider the date dimension that has only a year attribute. With this, we cannot get information at the quarter, month, or day level. To maximize available information, it is important to choose a detailed atomic grain. In this example, the choice could perhaps be day.

As an example, assume a grain of *one product sold in a store* in a retail example. Here we would not be able to associate a customer with a particular product purchased because there is only one row for a product. If it were purchased a thousand times by a thousand different customers, we could not know that.

Of course, there are situations where you can always declare higher-level grains for a business process by using aggregations of the most atomic and detailed data. However, the problem is that when a higher-level grain is selected, the number of dimensions are limited and may be less granular. You cannot drill down into these less granular dimensions to get a lower level of detail.

Note: For more detailed discussion on the importance of having a detailed atomic grain, see 6.2.2, “Importance of detailed atomic grain” on page 228.

Trade-offs in considering granularity

Granularity provides the opportunity for a trade-off between important issues in data warehousing. For example, one trade-off can be performance versus the volume of data (and the related cost of storing that data). Another can be a trade-off between the ability to access data at a very detailed level versus performance, and the cost of storing and accessing large volumes of data. Selecting the appropriate level of granularity significantly affects the volume of data in the data warehouse. Along with that, selecting the appropriate level of granularity determines the capability of the data warehouse to satisfy query requirements.

To help make this clear, refer to the example shown in Figure 5-8 on page 130. Here we are looking at transaction data for a bank account. On the side of the high level of detail, we see that 50 is the average number of transactions per account and the size of the record for a transaction is 150 bytes. As a result, it would require about 7.5 KB to store the very detailed transaction records to the end of the month. The side with the low level of detail (with a higher level of granularity) is shown in the form of a summary by account per month. Here, all

the transactions for an account are summarized in only one record. The summary record would require a larger record size, perhaps 200 bytes instead of the 150 bytes of the raw transaction, but the result is a significant savings in storage space.

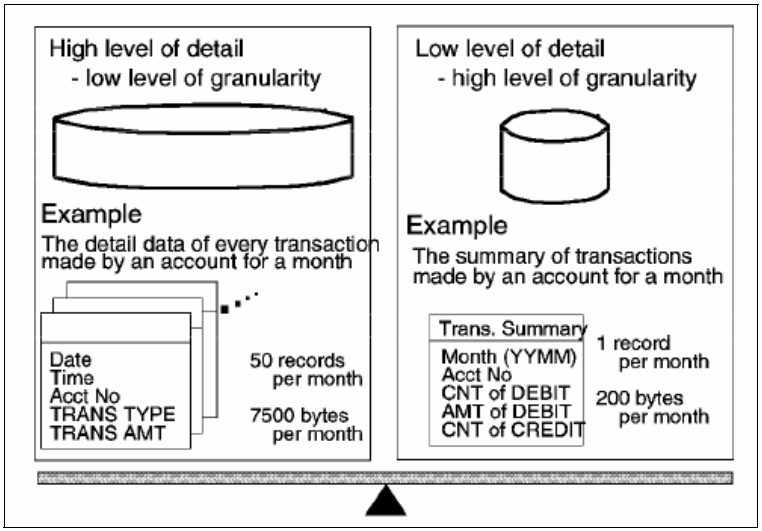


Figure 5-8 Granularity of data: The level of detail trade-off

In terms of disk space and volume of data, a higher granularity provides a more efficient way of storing data than a lower granularity. You also have to consider the disk space for the index of the data as well. This makes the space savings even greater. Perhaps a greater concern is with the manipulation of large volumes of data. This can impact performance, at the cost of more processing power.

There are always trade-offs to be made in data processing, and this is no exception. For example, as the granularity becomes higher, the ability to answer different types of queries (that require data at a more detailed level) diminishes. If you have a very low level of granularity, you can support any queries using that data at the cost of increased storage space and diminished performance.

Look again at Figure 5-8. With a low level of granularity, you can answer the query: *How many credit transactions were there for John's demand deposit account in the San Jose branch last week?* With the higher level of granularity, you cannot answer that question, because the data is summarized by month rather than by week.

If the granularity does not impact the ability to answer a specific query, the amount of system resources required for that same query can still differ

considerably. Suppose that you have two tables with different levels of granularity, such as transaction details and a monthly account summary. To answer a query about the monthly report for channel utilization by accounts, you could use either of those two tables without any dependency on the level of granularity. However, using the detailed transaction table requires a significantly higher volume of disk activity to scan all the data as well as additional processing power for calculation of the results. Using the monthly account summary table requires significantly less resource.

In deciding about the level of granularity, you must always consider the trade-off between the cost of the volume of data and the ability to answer queries.

5.3.5 High level dimensions and facts from grain

In this activity, we identify high level preliminary dimensions and facts from whatever can be understood from the grain definition. No detailed analysis is carried out to identify these preliminary dimensions and facts.

For our retail sales business process, we defined the grain (see 5.3.1, “Fact table granularity” on page 123) as: An individual line item on a grocery bill. This is shown in Figure 5-9.

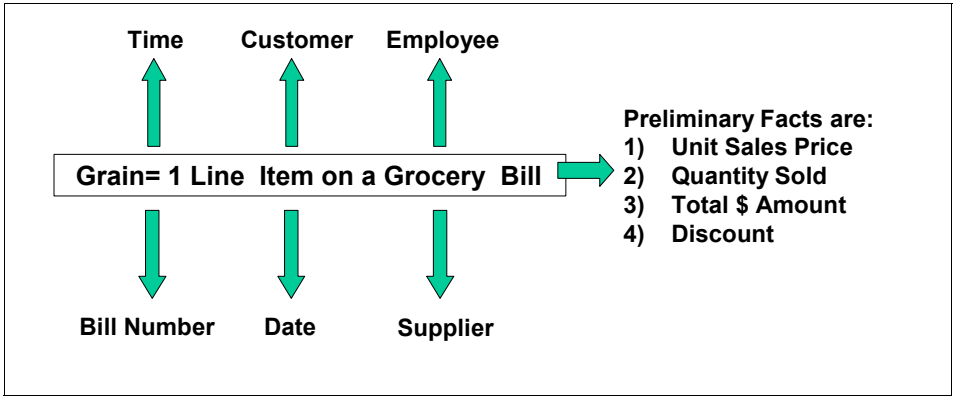


Figure 5-9 Identifying high level dimensions and facts with the grain

Once we define the grain appropriately, we can easily find the preliminary dimensions and facts.

Note: Preliminary facts are facts that can be easily identified by looking at the grain definition. For example, facts such as unit price, quantity, and discount are easily identifiable by looking at the grain. In other words, preliminary facts are easily visible on the grocery store bill we saw in Figure 5-7. However, detailed facts such as cost, manufacturing price per line item, and transportation cost per Item are not preliminary facts that can be identified by looking at the grain definition. Such facts are hidden and typically never visible on the grocery store bill. Preliminary facts are not the final set of facts; the formal detailed fact identification occurs in the *Identify the facts* phase in 5.5, "Identify the facts" on page 169.

Using the one line item grain definition shown in Figure 5-9 on page 131, we can identify all things that are true (logically associated with) to the grain. Table 5-11 shows high level dimensions and facts that are identified from the grain.

Table 5-11 High level dimensions and facts

Dimensions	Facts (KPIs)
Customer country	Sales revenue
Product (Brand, category)	Quantity of units sold
Sale Date	Total cost
Supplier of product	Discount
Time of sale	
Employee, sales person, or sales rep	
Bill	

Note: These preliminary high level dimensions and facts are helpful when we formally identify dimensions (see 5.4.1, "Dimensions" on page 135) and facts (see 5.5.1, "Facts" on page 171) in the later steps of the DMDL. The dimensions and facts get iteratively refined in each of the phases of the DMDL.

5.3.6 Final output of the identify the grain phase

The grain definition report is the final output report for this phase. It consists of one or multiple definitions for the grain of the business process, and the type of fact table (transaction, periodic, or accumulating) being used. It also includes the high level preliminary dimensions and facts.

5.4 Identify the dimensions

In this phase, we focus on the third step of the DMDL, which is *identify the dimensions*. This is depicted in Figure 5-10.

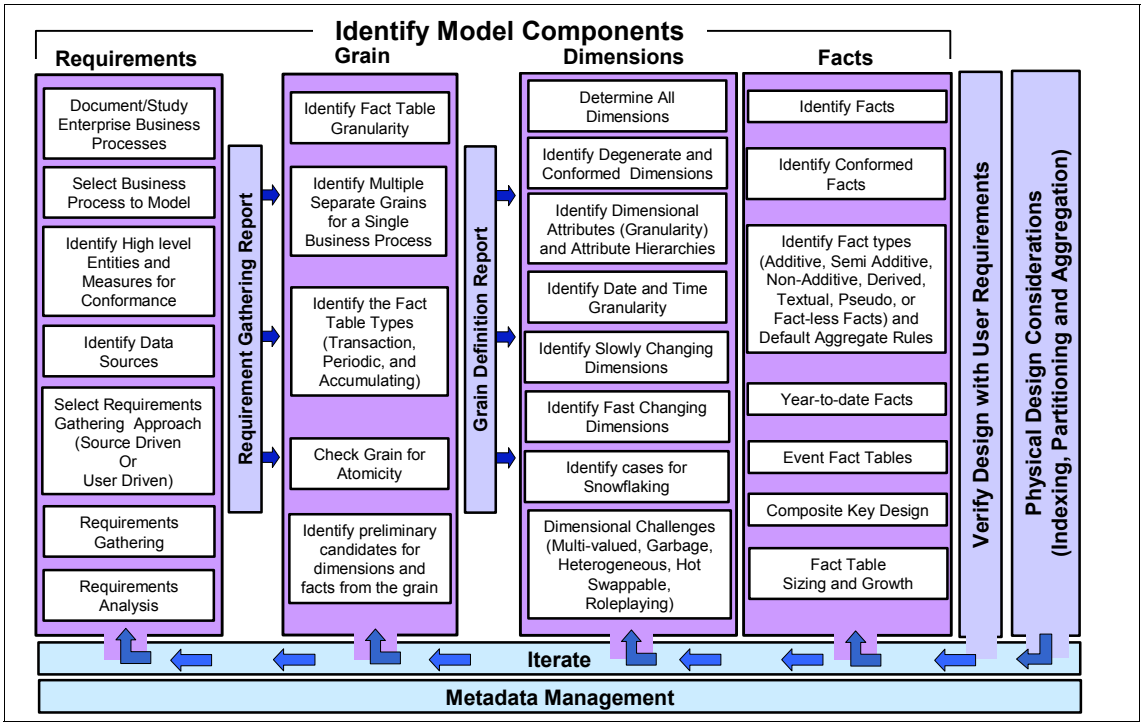


Figure 5-10 Dimensional Model Design Life Cycle

Table 5-12 shows the activities associated with this phase.

Table 5-12 Activities in the identify the dimensions phase

Seq. no.	Activity name	Activity description
1	**Identify dimensions**	Identifies the dimensions that are true to the grain identified in 5.3, “Identify the grain” on page 121.
2	Identify degenerate dimensions	Identifies one or more degenerate dimensions.
3	Identify conformed dimensions	Identifies any existing shared dimensions in the data warehouse or other star schemas used for designing the dimensional model.

Seq. no.	Activity name	Activity description
4	Identify dimensional attributes and dimensional hierarchies	Identifies the dimension attributes. It also identifies hierarchies, such as balanced, unbalanced, or ragged, that may exist in the dimensions. Techniques are suggested to handle these hierarchies in the design.
5	Identify date and time granularity	Identifies the date and time dimensions. Typically these dimensions have a major impact on the grain and size of the dimensional model.
6	Identify slowly changing dimensions	Identifies the slowly changing dimensions in the design. Three techniques (Type-1, Type-2, and Type-3) are described.
7	Identify very fast changing dimensions	Identifies very fast changing dimensions and describes ways of handling them by creating one or more mini-dimensions.
8	Identify cases for snowflaking	Identifies what dimensions need to be snowflaked.
9	Other dimensional challenges to look for are:	Other challenges relating to dimensions:
	→ Identify Multi-valued Dimensions	Looks for multi-valued dimensions and describes ways of handling them, such as by using bridge tables.
	→ Identify Role-Playing Dimensions	Describes ways of looking for dimensions that can be implemented by using role-playing.
	→ Identify Heterogeneous Dimensions	Describes ways of identifying heterogeneous products and implementing them.
	→ Identify Garbage Dimensions	Describes ways to look for low-cardinality fields and using them to make a garbage dimension.
	→ Identify Hot Swappable Dimensions	Describes ways of creating profile-based tables or hot swappable dimensions to improve performance and secure data.

Note: We have listed activities to look into while designing dimensions. The purpose is to make you aware of several design techniques you can use. Knowing they exist can help you make a solid dimensional model.

5.4.1 Dimensions

During this phase, we identify the dimensions that are true to the grain we chose in 5.3, “Identify the grain” on page 121.

Dimension tables

Dimension tables contain attributes that describe fact records in the fact table. Some of these attributes provide descriptive information; others are used to specify how fact table data should be summarized to provide useful information to the business analyst. Dimension tables contain hierarchies of attributes that aid in summarization. Dimension tables are relatively small, denormalized lookup tables which consist of business descriptive columns that can be referenced when defining restriction criteria for ad hoc business intelligence queries.

Important points to consider about dimension tables are:

- Each dimension table has one and only one lowest level element, or lowest level of detail, called the *dimension grain*, also referred to as the *granularity of the dimension*.
- Dimension tables that are referenced, or are likely to be referenced, by multiple fact tables are *conformed dimensions*. If conformed dimensions already exist for any of the dimensions in your model, you are expected to use the conformed versions. If you are developing new dimensions with potential for usage across the entire enterprise, you are expected to develop a design that supports anticipated enterprise needs. If you create a new dimension in your design which you think could potentially be used by other dimensional models or business processes, then you should design the new dimension keeping in mind your business requirements and the future anticipated needs. Conformed dimensions are discussed in more detail in “Conformed dimensions” on page 144.
- Each non-key element (other than the surrogate key) should appear in only one dimension table.
- All dimension table primary keys should be surrogate keys. An OLTP source system primary key should not be used as the primary key of the dimension table. Surrogate keys are discussed in detail in “Primary keys for dimension tables are surrogate keys” on page 139.
- Most dimensional models will have one or more date and time dimensions. Date and time dimensions are handled separately. The concepts relating to date and time dimensions are discussed in “Date and time granularity” on page 155.
- If a dimension table includes a code, then in most cases the code description should be included as well. For example, if region locations are identified by a region code, and each code represents a region name, both the code and the

region name should be included in the dimension table. It is important to understand that the quality of a dimensional model is directly proportional to the quality of the dimensional attributes.

- Typically, dimensional models should not have more than 10-15 dimensions. If you have more dimensions, you might need to find ways to merge dimension tables into one.
- The primary keys (surrogate keys) of the dimension tables should be included in the fact table as foreign keys.
- Typically, each dimension table will have one or more additional *Not Applicable* scenario records. This is primarily because of the fact that the foreign key in a fact table can never be null, since by definition that violates referential integrity. This concept is explained in more detail in “Insert a special customer row for the “Not applicable” scenario” on page 150.
- The rows in a dimension table establish a one-to-many relationship 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 business-oriented textual details, such as product name or customer name. Attributes serve as report labels and query constraints.

Note: Each dimension attribute should take on a single value in the context of each measurement inside the fact table. However, there are situations where we need to attach a multi-valued dimension table to the fact table. In other words, there are situations where there may be more than one value of a dimension for each measurement. Such cases are handled using multi-valued dimensions, which are explained in detail in 6.3.10, “Multi-valued dimensions” on page 288.

To summarize, in this activity we formally identify all dimensions that are true to the grain. By looking at the grain definition, it is relatively easy to define the dimensions. The dimensions for our retail sales example (for a grocery store) are depicted graphically (shown in Figure 5-11 on page 137) using the grocery bill.

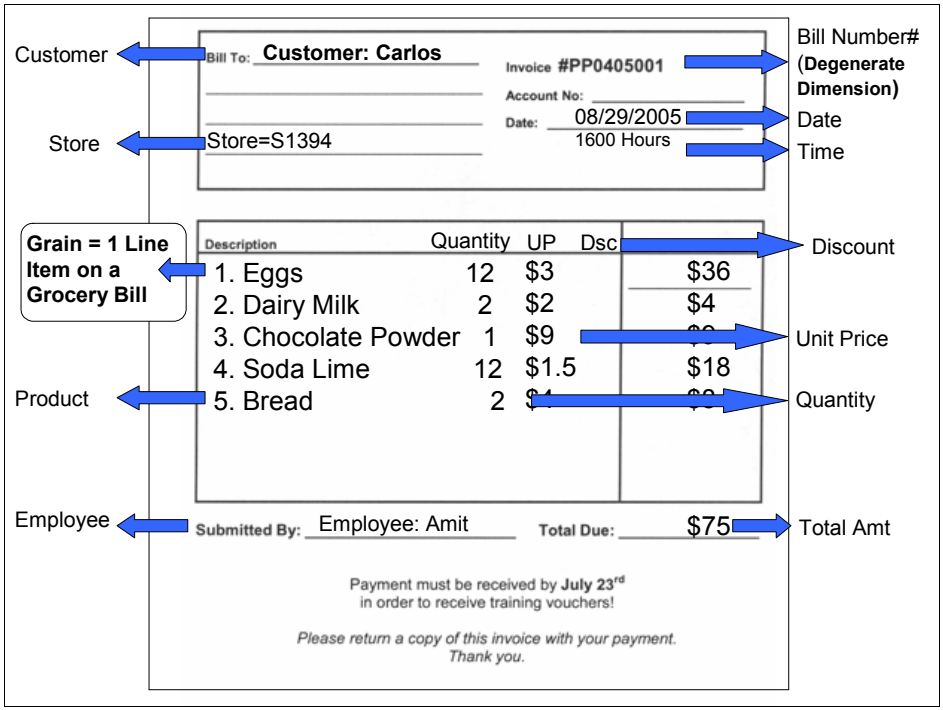


Figure 5-11 Graphical identification of dimensions and facts

List all the dimensions that are thought to be associated with this grain. The dimensions identified are shown in Table 5-13. For the dimensions, also define the level of detail (granularity) that to include in the dimension. However, the descriptive and detailed attributes are defined later in 5.4.4, “Dimensional attributes and hierarchies” on page 145.

Table 5-13 Dimensions and facts from grain

Seq. no.	Dimensions	Dimension granularity
1	Time	It is true to the grain, and has a description of time to the hour and minute level.
2	Customer	It is true to the grain, and this dimension describes the customers.
3	Employee	It is true to the grain, and this dimension describes the employees working in the stores and associated with the retail sale.

Seq. no.	Dimensions	Dimension granularity
4	Supplier	It is true to the grain, and this dimension describes the suppliers of the products.
5	Product	It is true to the grain, and this dimension describes the products, including their brand and category.
6	Date	It is true to the grain, and this dimension describes the different dates on which the products were sold. The date includes the day, month, quarter, and year description.
7	Store	It is true to the grain, and this dimension describes the stores which sold the products.
8	Bill Number (Pos_Bill_Number)	It is true to the grain, and this dimension describes the Bill or receipt of the store. This is handled as a degenerate dimension, described in section 5.4.2, “Degenerate dimensions” on page 142.

The preliminary dimensional schema that we get at this point is shown in Figure 5-12 on page 139. The dimension attributes are mentioned in the dimensional as TBD (To be Determined).

Note: The facts unit price, discount, quantity, and revenue are the preliminary facts identified from 5.3.5, “High level dimensions and facts from grain” on page 131. The facts and several other derived measures are identified in 5.5, “Identify the facts” on page 169.

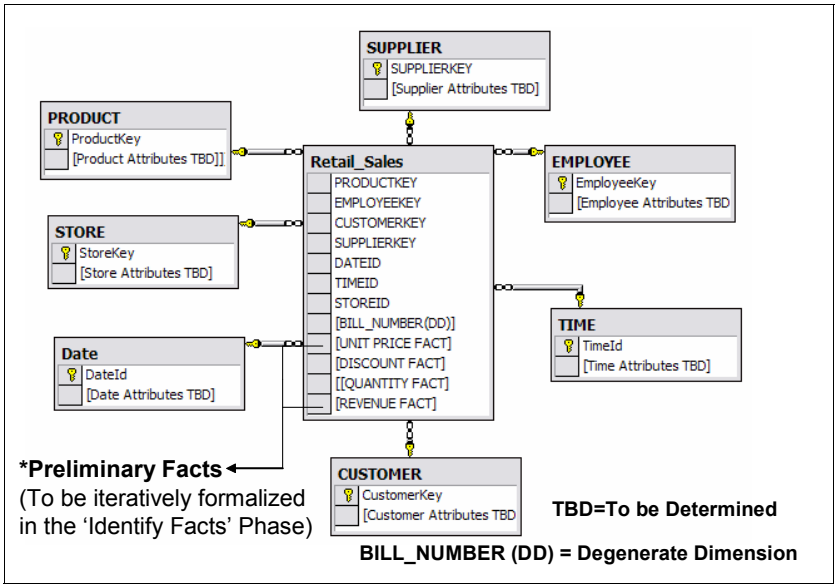


Figure 5-12 Preliminary retail sales grocery store dimensional model

Note: BILL_NUMBER is a degenerate dimension. It has no attributes associated with it and goes inside the fact table. We discuss degenerate dimensions in 5.4.2, "Degenerate dimensions" on page 142.

Primary keys for dimension tables are surrogate keys

It is important that primary keys of dimension tables remain stable. We strongly recommend that surrogate keys are created and used for primary keys for all dimension tables. In this section, we discuss what surrogate keys are, and why it is important to use the surrogate keys as the dimension table primary keys.

What are surrogate keys?

Surrogate keys are keys that are maintained within the data warehouse instead of the natural keys taken from source data systems. Surrogate keys are known by many other aliases, such as dummy keys, non-natural keys, artificial keys, meaningless keys, non-intelligent keys, integer keys, number keys, and technical integer keys. The surrogate keys join the dimension tables to the fact table. Surrogate keys serve as an important means of identifying each instance or entity inside a dimension table.

Reasons for using surrogate keys are:

- Data tables in various OLTP source systems may use different keys for the same entity. It may also be possible that a single key is being used by different

instances of the same entity. This means that different customers might be represented using the same key across different OLTP systems.

This can be a major problem when trying to consolidate information from various source systems. Or for companies trying to create/modify data warehouses after mergers and acquisitions. Existing systems that provide historical data might have used a different numbering system than a current OLTP system. Moreover, systems developed independently may not use the same keys, or they may use keys that conflict with data in the systems of other divisions. This situation may not cause problems when each department independently reports summary data, but can when trying to achieve an enterprise-wide view of the data.

This means that we cannot rely on using the natural primary keys of the source system as dimension primary keys because there is no guarantee that the natural primary keys will be unique for each instance. A surrogate key uniquely identifies each entity in the dimension table, regardless of its natural source key. This is primarily because a surrogate key generates a simple integer value for every new entity.

- Surrogate keys provide the means to maintain data warehouse information when dimensions change. To state it more precisely, surrogate keys are necessary to handle changes in dimension table attributes. We discuss in more detail in “Slowly changing dimensions” on page 159.
- Natural OLTP system keys may change or be reused in the source data systems. This situation is less likely than others, but some systems have reuse keys belonging to obsolete data or for data that has been purged. However, the key may still be in use in historical data in the data warehouse, and the same key cannot be used to identify different entities.

  The design, implementation, and administration of surrogate keys is the responsibility of the data warehouse team. Surrogate keys are maintained in the data preparation area during the data transformation process.
- One simple way improve performance of queries is to use surrogate keys. The narrow integer surrogate keys mean a thinner fact table. The thinner the fact table, the better the performance.
- Surrogate keys also help handle exception cases such as the *To Be Determined* or *Not Applicable* scenarios. This is discussed in “Insert a special customer row for the “Not applicable” scenario” on page 150.
- Changes or realignment of the employee identification number should be carried in a separate column in the table, so information about the employee can be reviewed or summarized, regardless of the number of times the employee record appears in the dimension table. For example, changes in the organization sales force structures may change or alter the keys in the hierarchy.

This is a common situation. For example, if a sales representative is transferred from one region to another, the organization may want to track all sales for the sales representative with the original region for data prior to the transfer date, and sales data for the sales representative in the new region after the transfer date.

To represent this organization of data, the salesperson's record must exist in two places in the Sales_Team dimension table. This is certainly not possible if the sales representative's company employee identification number is used as the primary key for the dimension table, because a primary key must be unique. A surrogate key allows the same sales representative to participate in different regions in the dimension hierarchy.

In this case, the sales representative is represented twice in the dimension table with two different surrogate keys. These surrogate keys are used to join the sales representative's records to the sets of facts appropriate to the various regions in the hierarchy occupied by the sales representative.

The sales representative's employee number should be carried in a separate column in the table so information about the sales representative can be reviewed or summarized regardless of the number of times the sales representative's record appears in the dimension table. The employee number also helps us go back to the OLTP system from where the record was loaded.

Note: Use a separate field in the dimension table to preserve the natural source system key of the entity being used in the source system. This helps us to go back to the original source if we need to track from where (which OLTP Source) the data came into the dimensional model. Also, we are able to summarize fact table data for a single entity in the fact table regardless of the number of times the entity's record appears in the dimension table.

GUID (Globally Unique Identifiers) as surrogate keys

We should strictly avoid the use of GUIDs as primary keys for the dimension tables. GUIDs are known to work well in the source OLTP systems, but they are difficult to use when it comes to data warehouses. This is primarily because of two reasons:

1. The first reason is storage. GUIDs use a significant amount of space compared to their integer counterparts. GUIDs take about 16 bytes each, where an integer takes about 4 bytes.
2. The second reason is that indexes on GUID columns are relatively slower than indexes on integer keys because GUIDs are four times larger.

Note: Every join between dimension and fact tables in the dimensional model should be based only on artificial integer surrogate keys. Use of natural OLTP system primary keys for dimensions should be avoided.

How to identify dimensions from an E/R model

The source of a dimensional model is either the enterprise data warehouse or the OLTP source systems. It would be correct to say both the data warehouse and the OLTP source systems are typically based on E/R models; they are in 3NF. We think if you can create a dimensional model from an E/R model, then you should be able to create dimensional models either from the data warehouse or directly from the OLTP source systems.

The following are the steps involved in converting an E/R model to a dimensional model:

1. Identify the business process from the E/R model.
2. Identify many-to-many tables in the E/R model to convert to fact tables.
3. Denormalize remaining tables into flat dimension tables.
4. Identify date and time from the E/R model.

This process of converting an existing E/R model (which could be a data warehouse or an OLTP source system) is explained in detail in 6.1, “Converting an E/R model to a dimensional model” on page 210.

5.4.2 Degenerate dimensions

In this section, we focus on identifying *degenerate* dimensions, which are dimensions without any attributes. They are not typical dimensions, but often simply a transaction number that is placed inside the fact table. In order to understand the concept of degenerate dimensions, we have to understand the source of the degenerate dimension, which originates in the form of some transaction numbers inside the OLTP system.

All OLTP source systems generally consist of transaction numbers, such as bill numbers, courier tracking number, order number, invoice number, application received acknowledgement number, ticket number, and reference numbers. These transaction numbers in the OLTP system generally tell about the transaction as a whole. Consider the retail sales example (for grocery store) and reanalyze the graphical bill as shown in Figure 5-13 on page 143.

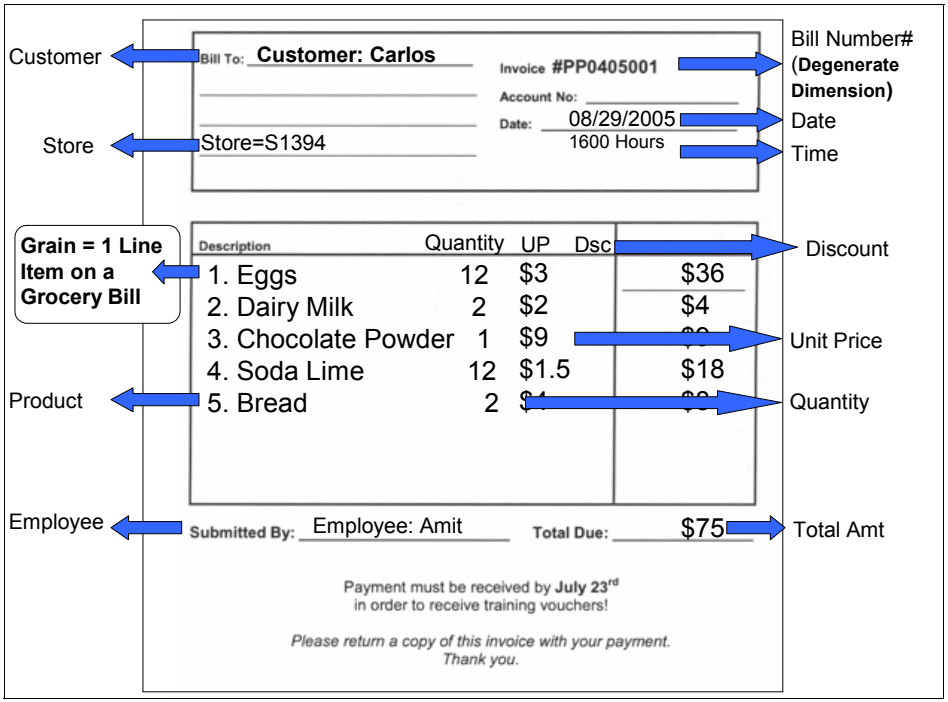


Figure 5-13 Retail sales grocery store bill

Try to analyze the Bill Number# information for the retail sales grocery store example. The Bill Number# is a transaction number that tells us about the transaction (purchase) with the store. If we take an old Bill Number# 973276 to the store and ask its manager to find out information relating to it, we may get all information related to the bill. For example, assume that the manager replies that the Bill Number# 973276 was generated on August 11, 2005. The items purchased were Apples, Oranges and Chocolates. The manager also tells us the quantity, unit price, and discount for each of the items purchased. He also tells us the total price. In short, the Bill Number# tells us the following information:

- Transaction date
- Transaction time
- Products purchased
- Quantity, unit price, and amount for each purchased product

If we now consider the Bill for our retail sales example, the important point to note is that we have already extracted all information related to the Bill Number# into other dimensions such as date, time, and product. Information relating to quantity, unit price, and amount charged is inside the fact table.

A degenerate dimension, such as Bill Number#, is there because we chose the grain to be *an individual line item on a bill*. So, the Bill Number# degenerate dimension is there because the grain we chose represents a single transaction or transaction line item.

The Bill Number# is still useful because it serves as the grouping key for grouping together of all the products purchased in a single transaction or for a single Bill Number#. Although to some, the Bill Number# looks like a dimension key in the fact table, it is not, because all information relating to the Bill Number# has been allocated to different dimensions. Hence, the so-called Bill Number# dimension has no attributes, and we refer it to as a degenerate dimension.

Other important things to consider about degenerate dimensions are:

- How to identify a degenerate dimension for a dimensional design.
- Should we make a separate dimension for the Bill Number#?
- Should we place the Bill Number# inside the fact table? If yes, why?
- How to realize that a degenerate dimension is missing.
- Under what situation will the Bill Number# no longer be a degenerate dimension? We discuss one example in “Identify degenerate dimensions” on page 384. We see that Invoice Number (a type of transaction number) is handled as a separate dimension and not as a degenerate dimension inside the fact table.

We discuss interesting topics about degenerate dimensions in more detail in 6.3.1, “Degenerate dimensions” on page 240.

5.4.3 Conformed dimensions

In this activity, we identify any conformed shared dimensions that are available to use instead of redesigning the dimensions. In this activity, we identify whether a dimension being used already exists inside the enterprise data warehouse or dimensional model.

What are conformed dimensions?

A conformed dimension means the same thing to each fact table to which it can be joined. A more precise definition is that two dimensions are conformed if they share one, more than one, or all attributes that are drawn from the same domain. A dimension may be conformed even if it contains only a subset of attributes from the primary dimension.

Typically, dimension tables that are referenced or are likely to be referenced by multiple fact tables (multiple dimensional models) are called *conformed dimensions*. If conformed dimensions already exist for any of the dimensions in

the data warehouse or dimensional model, you are expected to use the conformed dimension versions. If you are developing new dimensions with potential for usage across the entire enterprise warehouse, you are expected to develop a design that supports anticipated enterprise warehouse needs. In order to find out the anticipated warehouse needs, you might need to interact with several business processes to find out how they would define the dimensions.

To quickly summarize, the *identify conformed dimensions* activity involves the following two steps:

1. Identifying whether a dimension being used already exists. If the dimension being used already exists in the enterprise data warehouse, then that dimension has to be used. If the dimension would exist, then we would not identify dimensional attributes (see “Dimensional attributes and hierarchies” on page 145) for that particular dimension. A dimension may be conformed to the existing dimension even if it contains only a subset of attributes from the primary dimension.
2. For a nonexistent new dimension, create a new dimension with planned long-term cross-enterprise usage. When a dimensional model requires a dimension which does not exist in the enterprise warehouse or any other dimensional models, then a new dimension must be created. While creating this new dimension, you must be certain to interact with enterprise business functions to find out about their future anticipated need of the new dimension.

5.4.4 Dimensional attributes and hierarchies

This activity involves the following:

- Identify dimensional attributes for the dimensions we identified in section “Dimensions” on page 135.
- Identify the various hierarchies (such as balanced, unbalanced, and ragged) associated with each of the dimensions.

Why are good quality dimensional attributes important?

After having identified the dimensions, the next step is to fill the dimensions with good quality attributes. The dimension tables contain business descriptive columns that users reference to define their restriction criteria for ad hoc business queries.

The quality of a good dimensional model is directly proportional to the quality of attributes present inside these dimension tables. The dimension table attributes show up as report labels inside the reports for senior management.

So, which attribute should be included in the dimensions? To help answer that question, here are points to consider:

- Non-key columns are generally referred to as *attributes*. Every dimension table primary key should be a surrogate key, which is usually not considered an attribute because it is simply an integer and is not used for analysis.
- Use a separate field in the dimension table to preserve the natural source system key of the entity being used in the source system.
- A schema design that contains complete, consistent, and accurate attribute fields helps enable queries that are intuitive, and reduces the support burden on the organization responsible for database management and reports.
- A well-designed schema includes attributes that reflect the potential areas of interest and attributes that can be used for aggregations as well as for selective constraints and report breaks.
- If a dimension table includes a code, in most cases, include the code description as well. As an example, if branch locations are identified by a branch code, and each code represents a branch name, then include both the code and the name. Avoid storing cryptic decodes inside a dimensional table attribute to save space.
- Be sure attribute names are unique in the model. If you have duplicate names for different attributes, use the prime term (entity name) to create a distinction. For example, if you have multiple attributes called *Address Type Code*, rename one *Beneficiary Address Type Code*, and another might be *Premium Address Type Code*.
- The dimensional attributes serve as headings of the columns of the report and should be descriptive and easy to understand. For example, for a given situation where you want to store a flag such as 0/1 or Y/N, it is better to store something descriptive, such as Yes/No.
- An attribute can be defined to permit missing values in cases where an attribute does not apply to a specific item or its value is unknown.
- An attribute may belong to more than one hierarchy.
- Use only the alphabetic characters A-Z and the space character. Do not use special characters.
- While naming attributes, do not use possessive nouns. For example, use terms such as Recipient Birth Date rather than Recipient's Birth Date.
- Do not reflect permitted values in the attribute name. For example, the attribute name *Employee Day/Night Code* refers to code values designating day shift or night shift employees working in the grocery store. The attribute must be named to reflect the logical purpose and the entire range of values. For example, *Employee Shift Type Code* which allows for an expandable set of valid values.
- Do not include very large names for building attributes.

- Properly document all dimensional attributes.

Note: It is important to remember that the fact table occupies 85-90% of space in large dimensional models. Hence, saving space by using decodes in dimensional attributes does not save much overall space, but these decodes affect the overall quality and understandability of the dimensional model.

In 5.4.1, “Dimensions” on page 135, we designed the preliminary star schema shown in Figure 5-14.

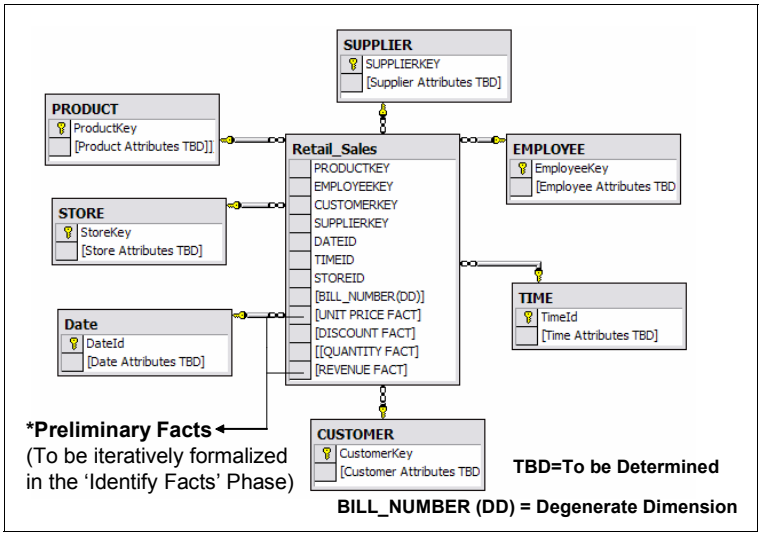


Figure 5-14 Preliminary retail sales grocery store dimensional model

Now fill this preliminary star schema with detailed dimension attributes for each dimension. Before filling in the attributes of various dimension tables, define the granularity of each dimension table as shown in Table 5-14.

Note: While deciding on attributes for each of the dimensions, keep in mind the requirements analysis performed in 5.2.7, “Requirements analysis” on page 118.

Table 5-14 Granularities for dimension tables

Seq. no.	Dimension table	Granularity of the dimension table
1	Employee	This dimension stores information at the single employee level. Also includes manager information.

Seq. no.	Dimension table	Granularity of the dimension table
2	Supplier	Contains information about supplier's company name and address.
3	Date	This dimension stores information down to the day level including month, quarter, and year.
4	Time	This dimension stores information about hour, and a description of the time, such as lunch time, early morning shift, morning, afternoon shift, and night shift.
5	Customer	This dimension stores information about the customer.
6	Product	This dimension stores information about the product, brand, and category name.
7	Store	This dimension stores information about the store dimension.
8	Bill Number	This is a degenerate dimension. It has no attributes and is present as a number (BILL_NUMBER) inside the fact table.

We explain the dimensions, along with their detailed attributes, below:

- **Employee dimension:** Shown in Figure 5-15 on page 149, the employee dimension stores information about the employees working at the store. The grain of the employee dimension table is information about a single employee.

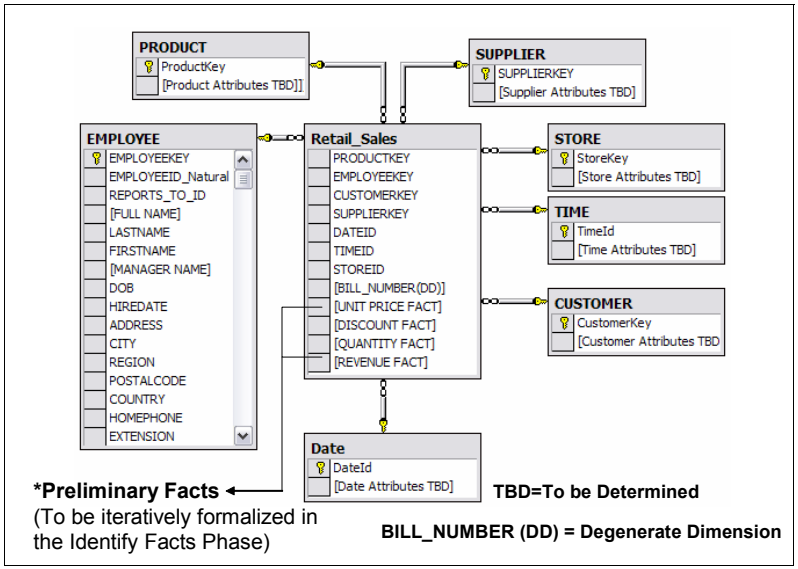


Figure 5-15 Employee dimension

The business requirements analysis in Table 5-8 on page 118 shows that questions Q2 and Q3 require the business to see sales figures for good and bad performing employees, along with the managers of these employees. The manager name column in the employee dimension table helps in that analysis.

- **Supplier dimension:** As shown in Figure 5-16 on page 150, the supplier table consists of attributes that describe the suppliers of the products in the store.

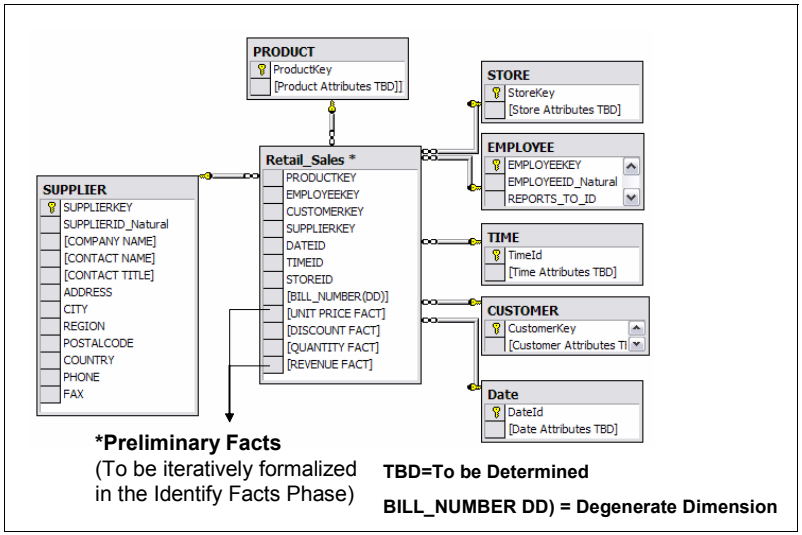


Figure 5-16 Supplier dimension

- **Customer dimension:** The customer dimension is shown in Figure 5-17 on page 151. This dimension consists of customer information.

Insert a special customer row for the "Not applicable" scenario

Typically, in a store environment, many customers are unknown to the store. We need to include a row in the customer dimension, with its own unique surrogate key, to identify a *Customer unknown to store* or *Not applicable*, and to avoid a null customer key in the fact table. The concepts of referential integrity are violated if we put a null in a fact table column declared as a foreign key to a dimension table. In addition, null keys are the source of great confusion to many because they cannot join on null keys.

In the customer dimension table, we insert a *Customer unknown to store* or *Not applicable* row inside the customer table and link the fact table row (measurements) to this row when the customer is unknown.

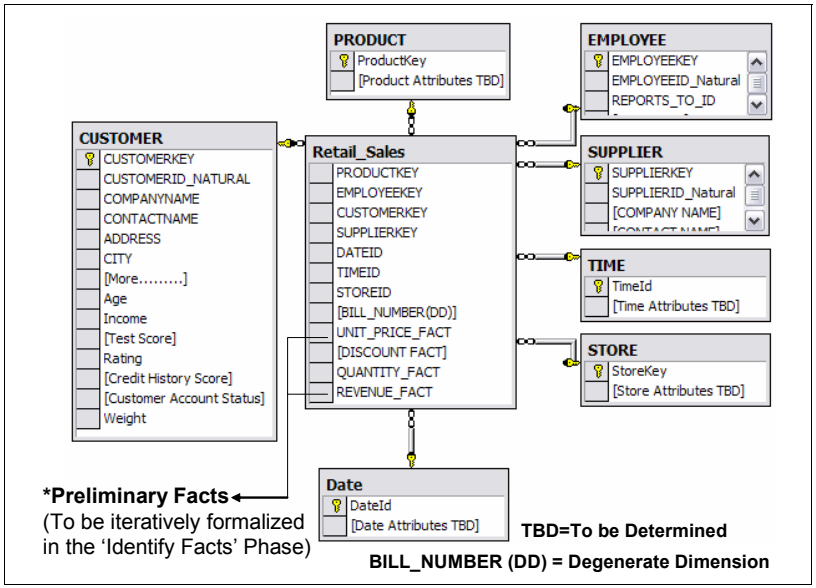


Figure 5-17 Customer dimension

Note: You must avoid null keys in the fact table. A good design includes a row (*Not applicable*) in the corresponding dimension table to identify that the dimension is not applicable to the measurement.

- **Product dimension:** The product dimension, shown in Figure 5-18 on page 152, holds information related to products selling in the stores. It consists of the following hierarchy: Category → Brand → Product.

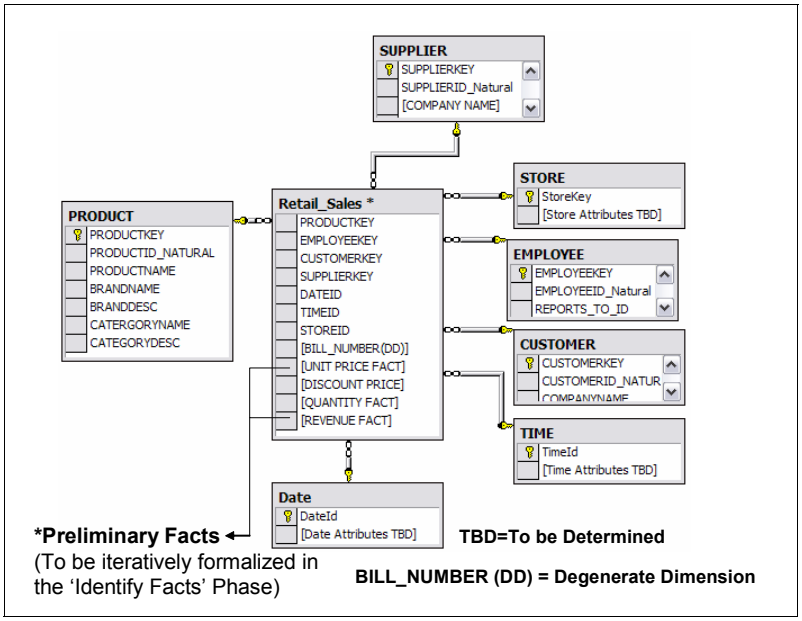


Figure 5-18 Product dimension

- **Store Dimension:** The store dimension holds information related to stores. This table is depicted in Figure 5-19 on page 153.

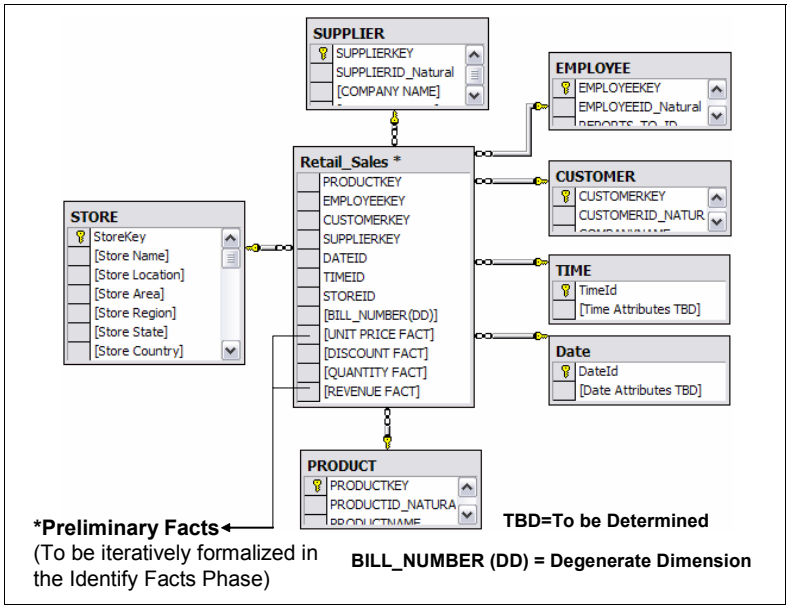


Figure 5-19 Store dimension

- **Date and time dimension:** The date and time dimensions are explained in detail in section “Date and time granularity” on page 155.

Identifying dimension hierarchies

A hierarchy is a cascaded series of many-to-one relationships. A hierarchy basically consists of different levels, each corresponding to a dimension attribute.

In other words, a hierarchy is a specification of levels that represents relationships between different attributes within a hierarchy. For example, one possible hierarchy in the date dimension is Year → Quarter → Month → Day.

There are three major types of hierarchies that you should look for in each of the dimensions. They are explained in Table 5-15 on page 154.

Table 5-15 Different types of hierarchies

Seq. no.	Name	Hierarchy description	How is it implemented?
1	Balanced	In a balanced hierarchy, all the dimension branches have the same number of levels. For more details, see “Balanced hierarchy” on page 249.	See “How to implement a balanced hierarchy” on page 250.
2	Unbalanced	A hierarchy is unbalanced if it has dimension branches containing varying numbers of levels. Parent-child dimensions support unbalanced hierarchies. For more details, see “Unbalanced hierarchy” on page 251.	See “How to implement an unbalanced hierarchy” on page 252.
3	Ragged	A ragged dimension contains at least one member whose parent belongs to a hierarchy that is more than one level above the child. Ragged dimensions, therefore, contain branches with varying depths. For more details, see “Ragged hierarchy” on page 260.	See “How to implement a ragged hierarchy in dimensions” on page 261.

Note: A dimension table may consist of multiple hierarchies. And, a dimension table may consist of attributes or columns which belong to one, more, or no hierarchies.

For the retail sales store example, Table 5-16 shows the hierarchies present inside various dimensions.

Table 5-16 Dimensions and hierarchies

No	Name	Hierarchy description	Type
1	Date	*Calendar Hierarchy*: Calendar Year → Calendar Month → Calendar Week *Fiscal Hierarchy*: Fiscal Year → Fiscal Quarter → Fiscal Month → Fiscal Day	Balanced
2	Time	None	

No	Name	Hierarchy description	Type
3	Product	Category Name → Brand Name → Product Name	Balanced
4	Employee	None	
5	Supplier	Supplier Country → Supplier Region → Supplier City	Balanced
6	Customer	Customer Country → Customer Region → Customer City	Balanced
7	Store	Store Country → Store State → Store Region → Store Area	Balanced

We have a detailed discussion about handling hierarchies in 6.3.4, “Handling dimension hierarchies” on page 248.

5.4.5 Date and time granularity

It is very important to identify the granularity of the date and time dimensions. This is primarily because the date and time dimensions help determine the granularity of the overall dimensional model and the level of information that is stored in it. Choosing a wrong grain in either the date or time dimension may result in important dimensions being omitted from the dimensional model. For example, if we are designing a dimensional model for an order management system and we choose the grain of the date dimension as quarter, we may miss many other dimensions (such as time and employee) that could be included in the model had the date dimension been at the day grain. We explain the date and time dimensions in detail below:

Date dimension

Every data mart has a date dimension because all dimensional models are based on a time series of operations. For example, you typically want to measure performance of the business over a time period. It is also possible that a dimensional model consists of several date dimensions. Such dimensions are usually deployed using a concept called role-playing, which is implemented by views. Role-playing is discussed in detail in section 6.3.9, “Role-playing dimensions” on page 285.

For the retail sales grocery store example, the date dimension is shown Figure 5-20 on page 156. The grain of the date dimension is a single day.

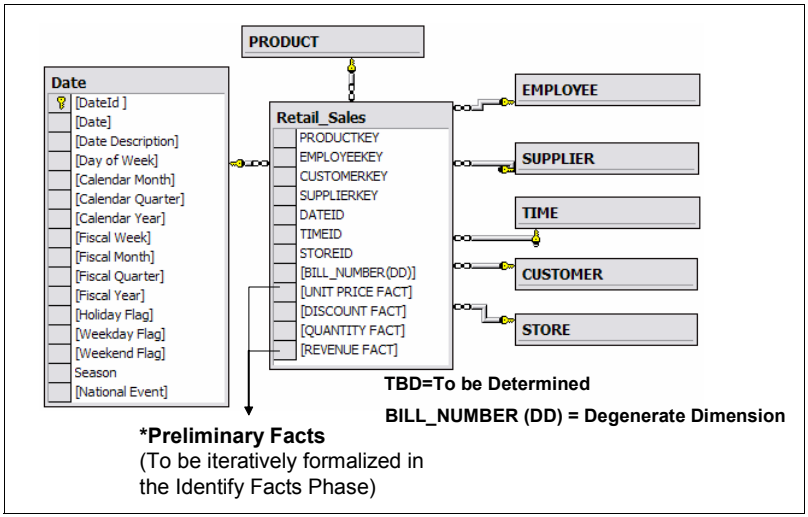


Figure 5-20 Date dimension

Note: The time dimension is handled separately from the date dimension. It is not advised to merge the date and time dimension into one table because a simple date dimension table which has 365 rows (for one year) would explode into 365 (Days) x 24 (Hours) x 60 (Minutes) x 60 (Seconds) = 31536000 rows if we tried storing hours, minutes, and seconds. This is for just one year. Consider the size if it were merged with time.

Typically, the date dimension does not have an OLTP source system connected to it. The date dimension can be built independently even before the actual dimensional design has started. The best way to built a date dimension is to identify all columns needed and then use SQL language to populate the date dimension table for columns such as date, day, month, and year. For other columns, such as holidays, with *Christmas* or *Boxing Day*, manual entries may be required. The same is true for columns such as fiscal date, fiscal month, fiscal quarter, and fiscal years. You may also use the spreadsheet to create the date dimension.

The date dimension in Figure 5-20 consists of the normal calendar and fiscal hierarchies. These hierarchies are shown in Figure 5-21 on page 157.

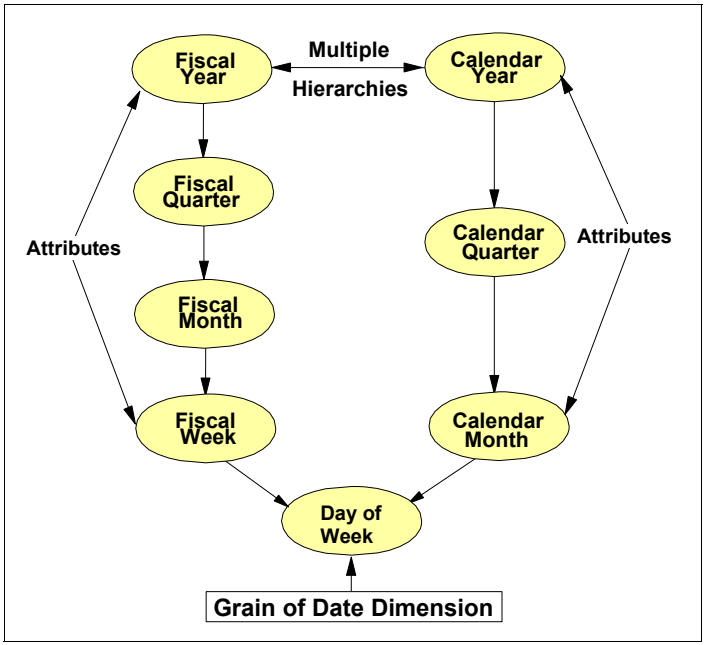


Figure 5-21 Calendar and fiscal hierarchy in the date dimension

Handle date as a dimension or a fact?

Instead of using a separate date table, as shown in Figure 5-20 on page 156, we could have used a date/time column in the fact table. In this way, we could have removed the date table from our dimensional model. We could have used the SQL date functions present inside each database to filter out the day, month, and year as examples. But this design where we replace the date dimension with a column inside the fact table has issues, for the following reasons:

- As shown in Figure 5-20 on page 156, there are several date attributes not supported by the SQL date functions, including fiscal periods, holidays, seasons, weekdays, weekends, and national events. This is one of the primary reasons why we want to have a separate date dimension table. This enables the business to look at the key performance indicators of their business across various fiscal and other date-related attributes which are not possible if we used the SQL date/time column inside the fact table.
- From an ease of use point of view, it is much easier to drag the columns from a date table instead of using complex SQL functions to create the logic for the reports.

Therefore, we think it is important to have a separate date dimension table instead of using a simple date/time column field in the fact table.

Insert a special date row for the "Not applicable" scenario

We discussed the *Not applicable* scenario in "Insert a special customer row for the "Not applicable" scenario" on page 150 for the customer table. Similarly for the date dimension, we insert a *Not applicable* row inside our date dimension table. In the date dimension table, we include a *Date unknown to store* or *Not applicable* or *Corrupt* row inside the date table and link the fact table row (measurements) to this row when the date is unknown, or when the recorded date is inapplicable, corrupted, or has not yet happened.

Recalling the grain for the retail sales grocery example, which is *an individual line item on a bill*, the date dimension is surely going to have a date with each product sold, and hence each fact table row is pointing to one valid row inside the date dimension table.

How to handle several dates across International time zones

The topic of handling date and time across International times zones is discussed in 6.3.3, "Handling date and time across international time zones" on page 248.

Time dimension

The time dimension for our retail sales store example is shown in Figure 5-22. Based on the requirements analysis performed in 5.2.7, "Requirements analysis" on page 118, we designed the time dimension.

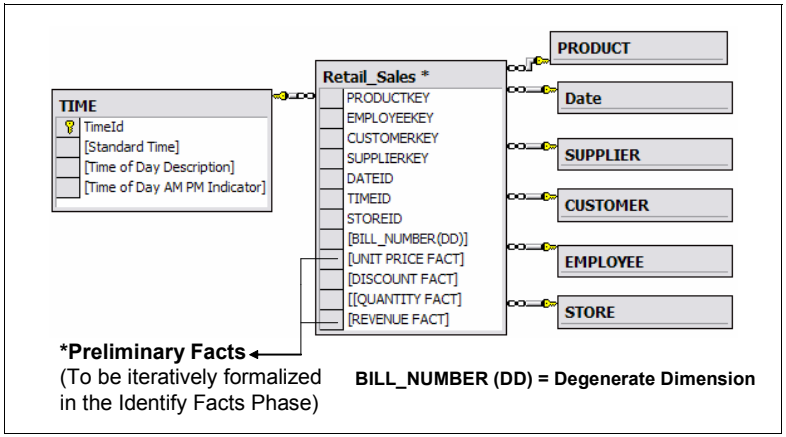


Figure 5-22 Time dimension

Sample rows for the time dimension are shown in Table 5-17 on page 159.

Table 5-17 Sample rows from the time dimension

Time ID	Standard time	Description	Time indicator
1	0600 hours	Early morning	AM
2	1650 hours	Evening	PM
3	2400 hours	Late night	AM

We recommend that time should be handled separately as a dimension and not as a fact inside the fact table.

You can handle time in dimensional modeling in two ways:

- Time of day as a separate dimension
- Time of day as a fact

We discuss more about time handling in 6.3.2, “Handling time as a dimension or a fact” on page 245.

5.4.6 Slowly changing dimensions

In this phase, we identify the slowly changing dimensions and also specify what strategy (Type-1, Type-2, or Type-3) we use to handle the change.

What are slowly changing dimensions?

A slowly changing dimension is a dimension whose attributes for a record (row) change slowly over time.

Assume that David is a customer of an insurance company called INS993, Inc., and first lived in Albany, New York. So, the original entry in the customer dimension table had the record as shown in Table 5-18:

Table 5-18 Insurance customer dimension table

Customer key	Social security number	Name	State
953276	989898988	David	New York

David moved to San Jose, California, in August, 2005. How should INS993, Inc. now modify the customer dimension table to reflect this change? This is the Slowly Changing Dimension problem.

There are typically three ways to solve this type of problem, and they are categorized as follows:

- **Type 1:** The new row replaces the original record. No trace of the old record exists. This is shown in Table 5-19. There is no history maintained for the fact that David lived in New York.

Table 5-19 Type 1 change in customer dimension table

Customer key (Surrogate key)	Social security number	Name	State
953276	989898988	David	California

- **Type 2:** A new row is added into the customer dimension table. Therefore, the customer is treated essentially as two people and both the original and the new row will be present. The new row gets its own primary key (surrogate key). After David moves from New York to California, we add a new row as shown in Table 5-20.

Table 5-20 Type 2 change in customer dimension table

Customer key (Surrogate key)	Social security number	Name	State
953276	989898988	David	New York
953277	989898988	David	California

- **Type 3:** The original record is modified to reflect the change. Also a new column is added to record the previous value prior to the change.

  To accommodate Type 3 slowly changing dimension, we now have the following columns:

  - Customer key
  - Customer name
  - Original State
  - Current State
  - Effective Date

  After David moved from New York to California, the original information gets updated, and we have the following table (assuming the effective date of change is August 15, 2005):

Table 5-21 Type 3 change in customer dimension table

Customer key	Social security number	Customer name	Original state	Current state	Effective date
953276	989898988	David	New York	California	August 15, 2005

Why do we need to handle changes in data?

We need to specify how to handle changes in data. For example, what happens if a customer address or name changes? Do we overwrite this change or does the business need to see each change. Such decisions cannot be made by the dimensional modeler. It is critical to involve the business users to identify how the they would like to see the changes that happen in various business entities.

Activities for slowly changing dimensions

In this phase, we need to identify slowly changing dimensions for the dimensional model designed in Figure 5-23. In addition, we also need to identify how we would handle the slowly changing dimensions and the strategy we would use.

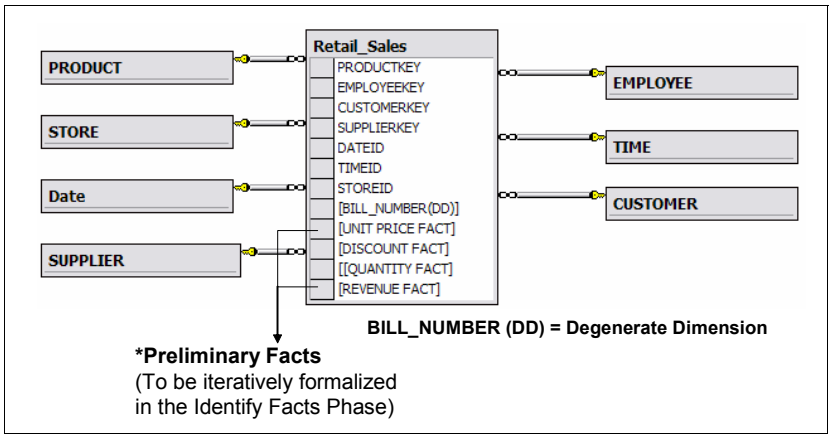


Figure 5-23 Identifying slowly changing dimensions

In Table 5-22, we identify the slowly changing dimensions for our retail sales store star schema, as shown in Figure 5-23.

Table 5-22 Slowly changing dimensions identified for retail sales example

Name	Strategy used	Structure changed?
Product	Type-2	No, only new rows added for each change.
Employee	Type-3	Yes, to show current and previous manager. Current Manager and Previous Manager columns added.
Supplier	Type-1	No, only current rows are updated for each change. No history maintained.
Time	Not applicable	

Name	Strategy used	Structure changed?
Date	Not applicable	
Store	Type-1	No, only current rows are updated for each change. No history maintained.
Customer		Yes, using Fast Changing Dimension strategy discussed in 5.4.7, “Fast changing dimensions” on page 162.

5.4.7 Fast changing dimensions

In this phase, we identify the fast changing dimensions that cannot be handled using the Type-1, Type-2, or Type-3 approaches used for slowly changing dimensions. For our retail sales grocery store example, we show how to handle this fast changing customer dimension.

All other dimensions are handled using the Type-1, Type-2, and Type-3 approach as shown in Table 5-22 on page 161.

What are fast changing dimensions?

Fast changing dimensions are also called rapidly changing dimensions. In 5.4.6, “Slowly changing dimensions” on page 159, we focused on the typically rather slow changes to our dimension tables. The next question is what happens when the rate of change in these slowly changing dimensions speeds up? If a dimension attribute changes very quickly on a daily, weekly, or monthly basis, then we are no longer dealing with a slowly changing dimension that can be handled by using the Type-1, Type-2, or Type-3 approach.

The best approach for handling very fast changing dimensions is to separate the fast changing attributes into one or more separate dimensions which are called *mini-dimensions*. The fact table then has two or more foreign keys—one for the primary dimension table and another for the one or more mini-dimensions (consisting of fast changing attributes). The primary dimension table and all its mini-dimensions are associated with one another every time we insert a row in the fact table.

Activities for the “Identify fast changing dimensions” phase

To handle fast changing dimensions, here are the activities:

- All dimensions are analyzed to find which dimensions change very fast.
- Assume that the Customer Dimension is a fast changing dimension.
- The Customer dimension is analyzed further to understand the impact on the size of the dimension table if the change is handled using the Type-2

approach. If the impact on the size of the dimension table would be huge, then the Type-2 approach is avoided.

Note: The Type-1 approach does not store history, so it is certainly not used to handle fast changing dimensions. The Type-3 approach is also not used because it allows us to see new and historical fact data by either the new or prior attribute values. However, it is inappropriate if you want to track the impact of numerous intermediate attribute values, which would be the case for a fast changing dimension.

- The next step is to analyze the fast changing Customer dimension in detail to identify which attributes of this dimension are subject to change fast. Assume that we identify seven fast changing attributes in the Customer dimension, as shown in Figure 5-24 on page 164. They are age, income, test score, rating, credit history score, customer account status, and weight. Such fast changing attributes are then separated into one new dimension (Customer_Mini_Dimension) table, whose primary key is attached to the fact table as a foreign key. This is also shown in Figure 5-24 on page 164.
- After having identified the constantly changing attributes and putting them in the Customer_Mini_Dimension mini-dimension table, the next step is to convert these identified attributes individually into band ranges. The concept behind this exercise is to force these attributes to take limited, discreet values. For example, assume that each of the above seven attributes takes on 10 different values. Then, the Customer_Mini_Dimension will have 1 million values. Thus, by creating a mini-dimension table consisting of band-range values, we have avoided the problem where the attributes such as age, income, test score, rating, credit history score, customer account status, and weight can no longer change. These attributes cannot change because they have a fixed set of band-range values (see Table 6-18 on page 272) instead of having a large number of values.

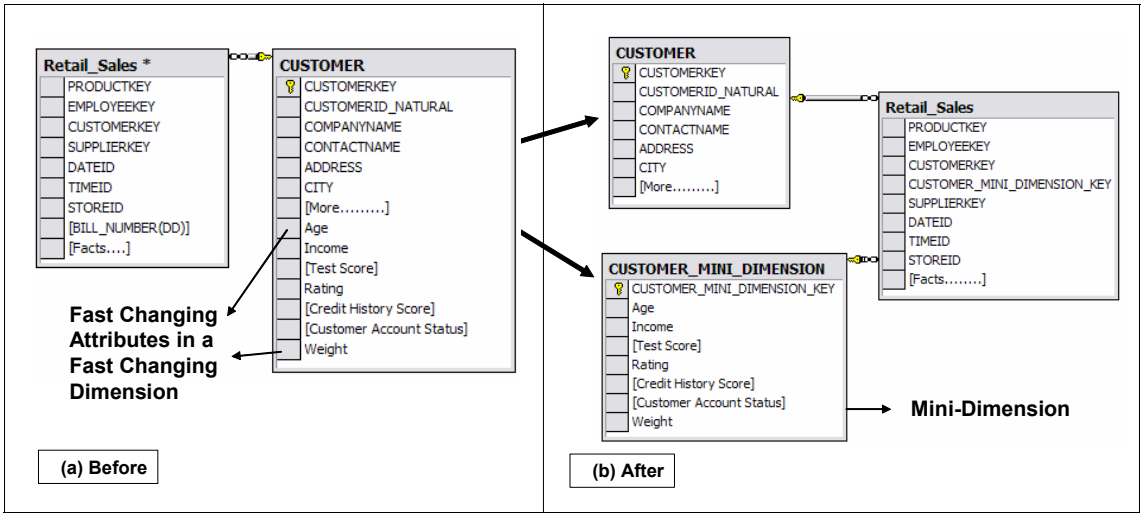


Figure 5-24 Handling a fast changing dimension

Therefore, we have two dimension tables:

a. One table is the primary original fast changing dimension minus the fast changing attributes. This is the Customer table as shown in Figure 5-24.

b. The second table consists of the fast changing attributes which are typically in the mini-dimension. This table is the Customer_Mini_Dimension as shown in Figure 5-24.

Note: It is possible that a very fast changing dimension table may be split into one or more mini-dimensions.

The following topics about fast changing dimensions are discussed in more detail in 6.3.6, “Handling fast changing dimensions” on page 269:

- Fast changing dimensions with a real-time example.
- Mini-dimensions and band range values.
- Fast changing dimensions, resolving issues with band ranges, and Mini-dimensions.
- Snowflaking does not resolve the fast changing dimension problem.

After completing this activity (*Identify Fast Changing Dimensions*), we get the schema as shown in Figure 5-25 on page 165.

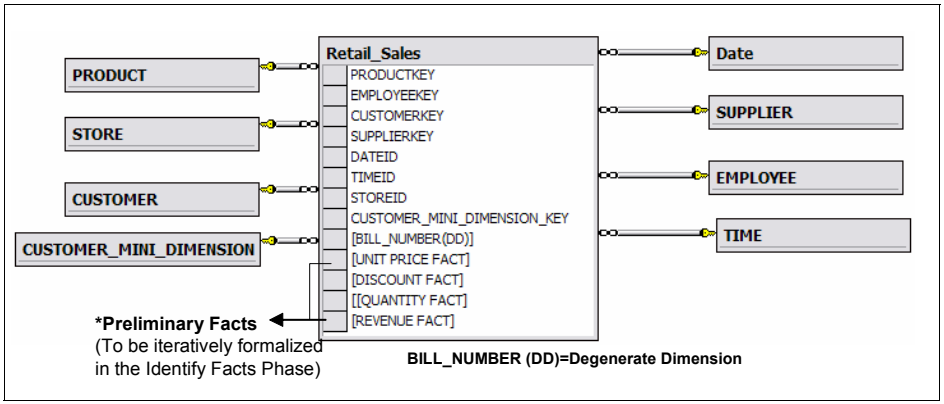


Figure 5-25 Retail sales grocery store star schema

5.4.8 Cases for snowflaking

Further normalization and expansion of the dimension tables in a star schema result in the implementation of a snowflake design. A dimension table is said to be *snowflaked* when the low-cardinality attributes in the dimension have been removed to separate normalized tables and these normalized tables are then joined back into the original dimension table.

Typically, we do not recommend snowflaking in the dimensional model environment because it can impact understandability of the dimensional model and can result in decreased performance because a higher number of tables need to be joined.

For each dimension selected for the dimensional model, we need to identify which should be snowflaked. There are no candidates identified for snowflakes in the grocery store example.

The following topics on snowflaking are discussed in more detail in 6.3.7, “Identifying dimensions that need to be snowflaked” on page 277:

- What is Snowflaking?
- When do you do snowflaking?
- When to avoid snowflaking?
- Under what conditions will snowflaking improve performance?
- Disadvantages of snowflaking.

5.4.9 Other dimensional challenges

In this phase, we identify other special types of dimensions, as shown in Table 5-23.

Table 5-23 Other special dimensions

Seq no.	Type	How it is implemented
1	Multi-valued dimension	**Description** Typically while designing a dimensional model, each dimension attribute should take on a single value in the context of each measurement inside the fact table. However, there are situations where we need to attach a multi-valued dimension table to the fact table, because there may be more than one value of a dimension for each measurement. Such cases are handled using Multi-valued dimensions. **Implementation** Multi-valued dimensions are implemented using Bridge tables. For a detailed discussion, refer to 6.3.10, “Multi-valued dimensions” on page 288.
2	Role-playing dimension	**Description** A single dimension that is expressed differently in a fact table using views is called a role-playing dimension. A date dimension is typically implemented using the role-playing concept when designing a dimensional model using an Accumulating snapshot fact table. This is discussed in more detail in Chapter 6, “Modeling considerations” on page 209 and “Accumulating fact table” on page 233. **Implementation** The role-playing dimensions are implemented using views. This procedure is explained in detail in 6.3.9, “Role-playing dimensions” on page 285.

Seq no.	Type	How it is implemented
3	Heterogeneous dimension	**Description** The concept of heterogeneous products comes to life when you design a dimensional model for a company that sells heterogeneous products with different attributes, to the same customers. That is, the heterogeneous products have separate unique attributes and it is therefore not possible to make a single product table to handle these heterogeneous products. **Implementation** Heterogeneous dimensions can be implemented in following ways: ► Merge all the attributes into a single product table and all facts relating to the different heterogeneous attributes in one fact table. ► Create separate dimensions and fact tables for the different heterogeneous products. ► Create a generic design to include a single fact and single product dimension table with common attributes from two or more heterogeneous products. Implementing heterogeneous dimensions is discussed in more detail in 6.3.12, “Heterogeneous products” on page 292.
4	Garbage dimension	**Description** A garbage dimension is a dimension that consists of low-cardinality columns such as codes, indicators, status, and flags. The garbage dimension is also referred to as a junk dimension. Attributes in a garbage dimension are not related to any hierarchy. **Implementation** The implementation of the garbage dimension involves separating the low-cardinality attributes and creating a dimension for such attributes. This implementation procedure is discussed in detail in 6.3.8, “Identifying garbage dimensions” on page 282.

Seq no.	Type	How it is implemented
5	Hot swappable dimension	**Description** A dimension that has multiple alternate versions that can be swapped at query time is called a Hot Swappable dimension or Profile table. Each of the versions of the hot swappable dimension can be of a different structure. The alternate versions of the hot swappable dimensions access the same fact table but get different output. The different versions of the primary dimension may be completely different, including incompatible attribute names and different hierarchies. **Implementation** The procedure to implement Hot swappable dimensions is discussed in detail in 6.3.13, “Hot swappable dimensions or profile tables” on page 294.

For the retail sales grocery store example, we do not have any of the special dimensions shown in Table 5-23 on page 166. However, we recommend that you refer to Chapter 6, “Modeling considerations” on page 209 to understand how these special dimensions are designed and implemented.

The dimensional model designed for the retail sales grocery business process to the end of the *Identify the dimensions* phase is shown in Figure 5-26.

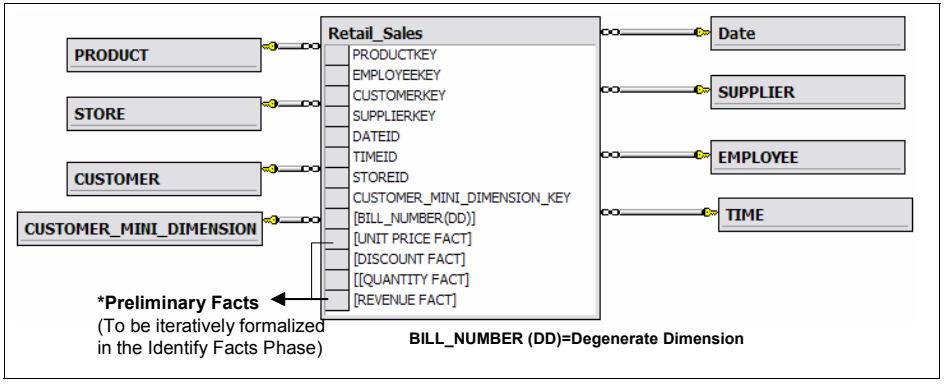


Figure 5-26 Retail sales grocery business process (dimensional model)

The model designed to this point consists of the dimensions and preliminary facts. These preliminary facts are high level facts identified in section 5.3.5, “High level dimensions and facts from grain” on page 131. We iteratively identify additional facts in the next phase, 5.5, “Identify the facts” on page 169.

5.5 Identify the facts

In this phase, we focus on the fourth step of the DMDL, as shown in Figure 5-27.

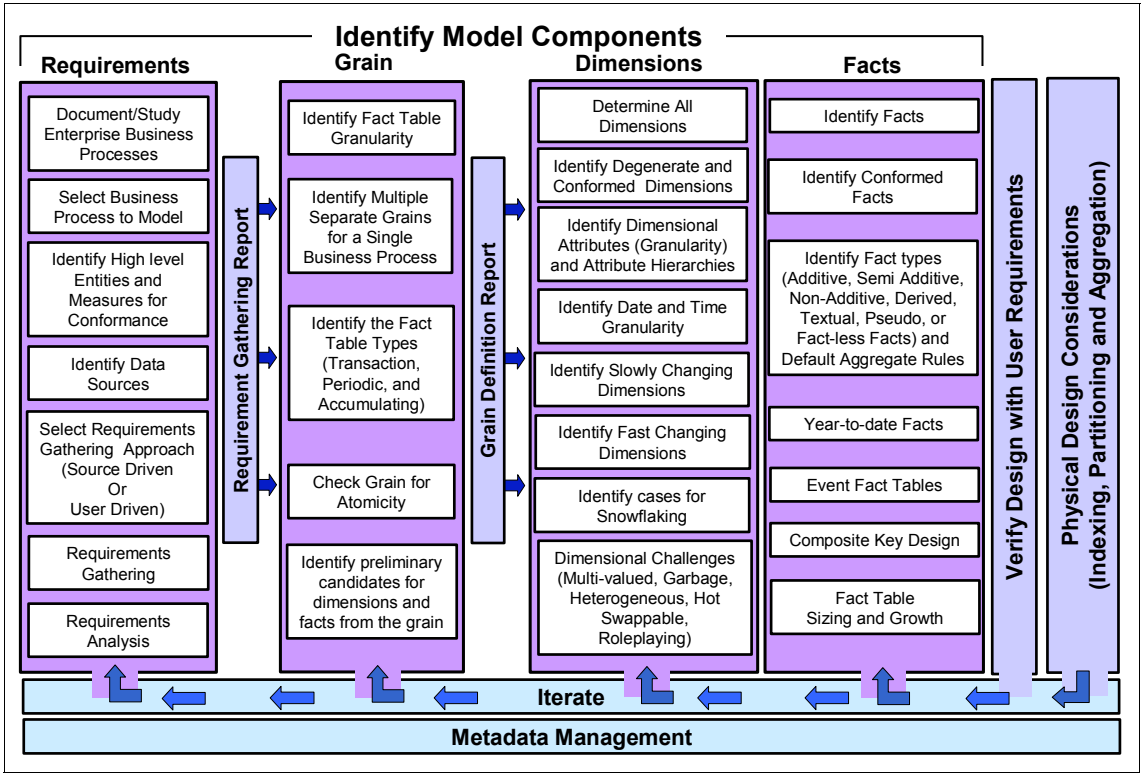


Figure 5-27 Dimensional Model Design Life Cycle

To recall, we identified preliminary dimensions and preliminary facts (for the retail sales grocery business process) in section 5.3.5, “High level dimensions and facts from grain” on page 131. We used the grain definition, as shown in Figure 5-28 on page 171, to quickly arrive at high level preliminary dimensions and facts. In this phase, we iteratively, and in more detail, identify facts that are true to this grain.

Table 5-24 on page 170 shows the activities that are associated with the *Identify the facts* phase.

Table 5-24 Activities in the Identify the facts phase

Activity name	Activity description
Identify facts	This activity identifies the facts that are true to the grain identified in 5.3, “Identify the grain” on page 121.
Identify conformed facts	After we identify the facts, we determine whether any of these facts are conformed. If they are, we use those conformed facts.
Identify fact types	In this activity we identify the fact types, such as: - Additive facts - Semi-Additive facts - Non-Additive facts - Derived facts - Textual facts - Pseudo facts - Factless facts
Year-to-date facts	Year-to-date facts are numeric values that consist of an aggregated total from the start of year to the current date. Here we verify that such facts are not included in a fact table with the atomic level line items.
Event fact tables	Here we describe how to handle events in the event-based fact table. We also highlight the pseudo and factless facts that may be associated with such tables.
Composite key design	We discuss general guidelines for designing the primary composite key of the fact table. We also describe situations when a degenerate dimension may be included inside the fact table composite primary key.
Fact table sizing and growth	Guidelines are described for use by the DBA to predict fact table growth.

Figure 5-28 on page 171 shows the high level dimensions and preliminary facts identified from the grain definition.

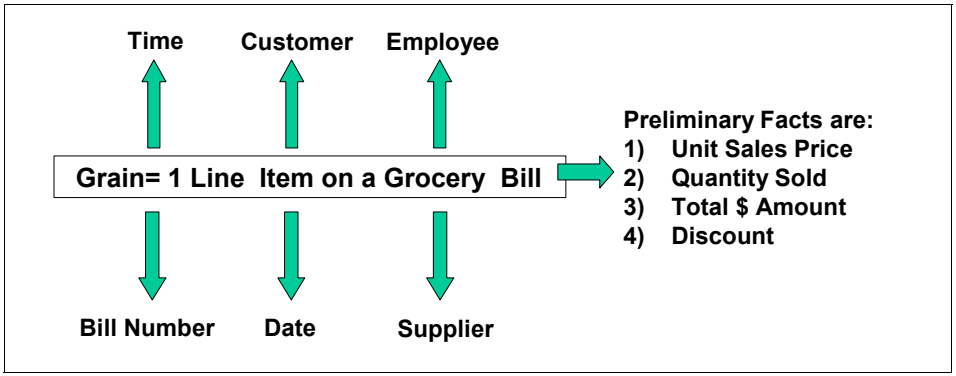


Figure 5-28 Identifying high level dimensions and facts from the grain

5.5.1 Facts

In this activity, we identify all the facts that are true to the grain. These facts include the following:

- We identified the preliminary facts in section “High level dimensions and facts from grain” on page 131. Preliminary facts are easily identified by looking at the grain definition or the grocery bill.
- Other detailed facts which are easily identified by looking at the grain definition as shown in Figure 5-28. For example, detailed facts such as cost per individual product, manufacturing labor cost per product, and transportation cost per individual product, are not preliminary facts. These facts can only be identified by a detailed analysis of the source E/R model to identify all the line item level facts (facts that are true at the line item grain).

For the retail sales grocery store example, we identify the facts that are true to the grain, as shown in Table 5-25. We found other detailed facts, such as cost per item, storage cost per item, and labor cost per item, that are available in the source E/R model.

Table 5-25 Facts identified for the retail sales business process

Facts	Fact description
Unit sales price	Price of a single product.
Quantity sold	Quantity of each individual product that is sold.
Amount	Defined as (Unit sales price) X (Quantity sold).
Discount	Discount given on a single product.
Cost per item	Cost of a single product.

Facts	Fact description
Total cost	Defined as (Cost per item) X (Quantity sold).
Storage cost	Storage cost per product.
Labor cost	Labor cost per product.

Note: It is important that all facts are true to the grain. To improve performance, dimensional modelers may also include year-to-date facts inside the fact table. However, year-to-date facts are non-additive and may result in facts being counted (incorrectly) several times when more than a single date is involved. We discuss this in more detail in 5.5.4, “Year-to-date facts” on page 176.

How to identify facts or fact tables from an E/R model

We identified the facts as shown in Table 5-25 on page 171. The process of identifying these facts involved use of the E/R model for the Retail Sales Business process, as shown in Figure 5-29 on page 173.

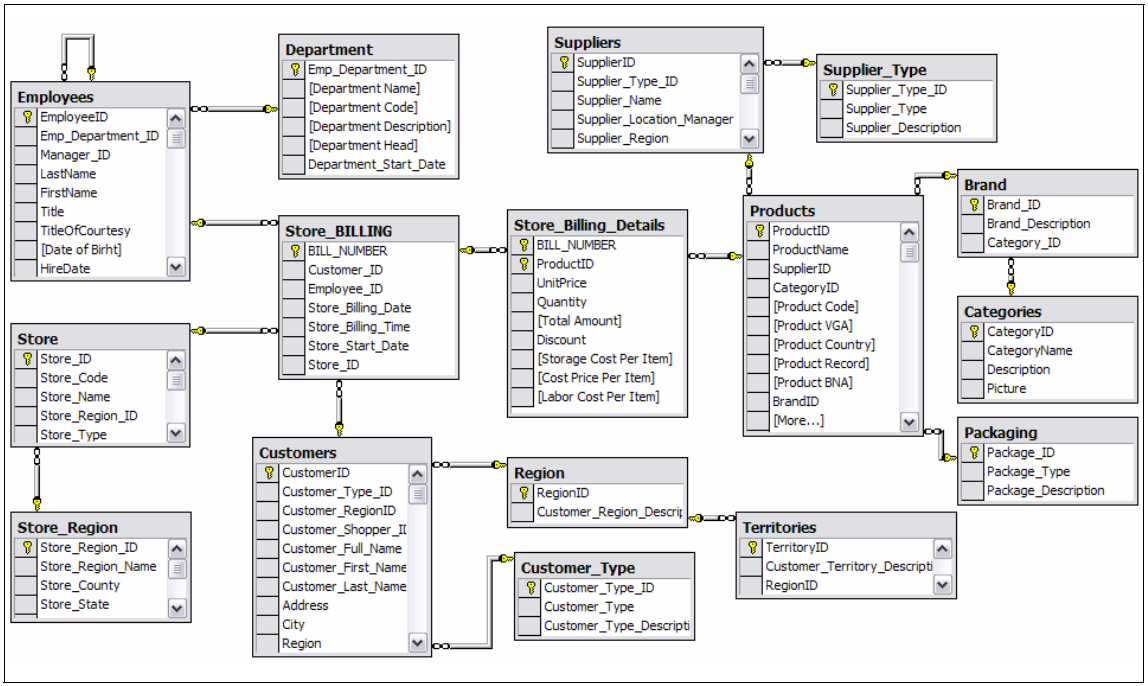


Figure 5-29 E/R Model for the retail sales business process

The following are the steps to convert an E/R model to a dimensional model:

1. Identify the business process from the E/R Model.
2. Identify many-to-many tables in E/R model to convert to fact tables.
3. Denormalize remaining tables into flat dimension tables.
4. Identify date and time from the E/R Model.

This process of converting an existing E/R model (which can be a Data Warehouse or an OLTP source system) is explained in detail in 6.1, “Converting an E/R model to a dimensional model” on page 210.

The final dimensional model, after having identified the facts (shown in Table 5-25 on page 171), is depicted in Figure 5-30 on page 174.

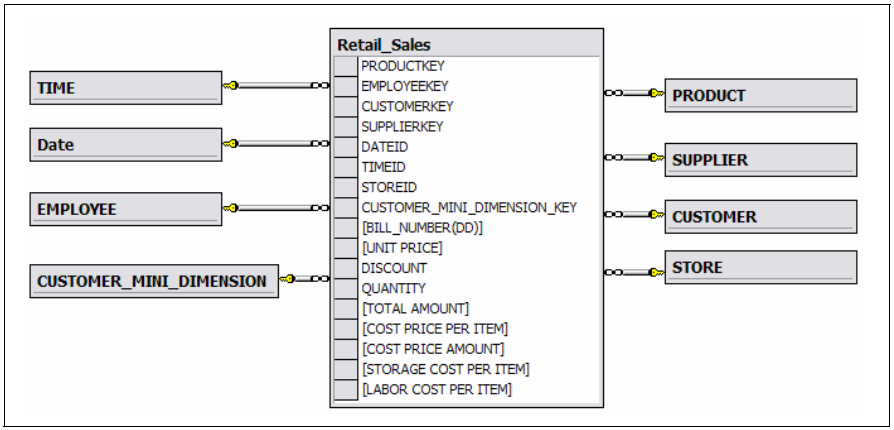


Figure 5-30 Retail sales dimensional model

5.5.2 Conformed facts

A *conformed fact* is a shared fact that is designed to be used in the same way across multiple data marts. So, the shared conformed facts mean the same thing to different star schemas.

Once the facts have been identified, we must determine whether some of these facts already exist inside the data warehouse or in some other data marts. If any do, then ideally we should use these predefined (and hopefully well-tested) facts.

We can assume that for our retail sales business process, there are no conformed facts.

5.5.3 Fact types

The facts inside the fact table could be of several different types, some of which are described in Table 5-26 on page 175.

Table 5-26 Fact types

Fact type	Description
Additive facts	Additive facts are facts that can be added across all of the dimensions in the fact table, and are the most common type of fact. Additive facts may also be called fully additive facts. They are identified here because these facts would be used across several dimensions for summation purposes. It is important to understand that since dimensional modeling involves hierarchies in dimensions, aggregation of information over different members in the hierarchy is a key element in the usefulness of the model. Since aggregation is an additive process, it is good if we have additive facts.
Semi-additive facts	These are facts that can be added across some dimensions but not all. They are also sometimes referred to as partially-additive facts. For example, facts such as head counts and quantity-on-hand (inventory) are considered semi-additive.
Non-additive facts	Facts that cannot be added for any of the dimensions. That is, they cannot be logically added between records or fact rows. Non-additive facts are usually the result of ratios or other mathematical calculations. The only calculation that can be made for such a fact is to get a count of the number of rows of such facts. Table 5-27 shows examples of non-additive facts, and we discuss the process of handling non-additive facts in 6.4.1, “Non-additive facts” on page 297.
Derived facts	Derived facts are created by performing a mathematical calculation on a number of other facts, and are sometimes referred to as calculated facts. Derived facts may or may not be stored inside the fact table.
Textual facts	A textual fact consists of one or more characters (codes). They should be strictly avoided in the fact table. Textual codes such as flags and indicators should be stored in dimension tables so they can be included in queries. Textual facts are non-additive, but could be used for counting.
Pseudo fact	When summed, a pseudo fact gives no valid result. They typically result when you design event-based fact tables. For more detail, see 6.4.4, “Handling event-based fact tables” on page 311.
Factless fact	A fact table with only foreign keys and no facts is called a factless fact table. For more detail, see 6.4.4, “Handling event-based fact tables” on page 311.

Note: Each fact in a fact table should have a default aggregation (or derivation) rule. Each fact in the fact table should be able to be subjected to any of the following: Sum (Additive), Min, Max, Non-additive, Semi-additive, Textual, and Pseudo.

Table 5-27 shows the different facts and their types for the retail sales business process. They are all true to the grain.

Table 5-27 Facts identified for the retail sales business process

Facts	Fact description	Type of fact	Formula
Unit sales price	Sales price of a single product	Non-additive	
Quantity sold	Quantity of each item sold	Additive	
Amount	Total sales	Additive	(unit sales price) X (quantity sold)
Discount	Discount on each item	Non-additive	
Item cost	Cost of a single item	Non-additive	
Total cost	Cost of all items	Additive	(cost per item) X (quantity sold)
Storage cost	Storage cost per item	Non-additive	
Labor cost	Labor cost per item	Non-additive	

Note: A fact is said to be derived if the fact can be calculated from other facts that exist in the table, or that have been also derived. You may decide not to include the derived facts inside the fact table and calculate these derived facts in a reporting application.

5.5.4 Year-to-date facts

The period beginning at the start of the calendar year to the current date is called year-to-date. For a calendar year where the starting day of the year is January 1, the year-to-date definition is a period of time starting from January 1 to the specified date.

Year-to-Date facts are numeric totals that consist of an aggregated total from the start of a year to the current date. For example, assume that a fact table stores sales data for the year 2005. The sales for each month are additive and can be

summed to produce year-to-date totals. If you create a Year-to-Date fact such as Sales_$$_Year_To_Date, then when you query this fact in August 2005, you would get the sum of all sales to August 2005.

Dimensional modelers may include aggregated year-to-date facts inside the fact table to improve performance and also reduce complexities in forming year-to-date queries. However, to avoid confusion it is typically preferred for such facts to be calculated in the report application.

Suggested approaches for handling year-to-date facts are as follows:

- OLAP-based applications
- SQL functions in views or stored procedures

For our retail sales business process dimensional, we do not store any year-to-date facts.

5.5.5 Event fact tables

Event fact tables are used to record events, such as Web page clicks and employee or student attendance. Events do not always result in facts. So, if we are interested in handling event-based scenarios where there are no facts, we use event fact tables that consists of either pseudo facts or factless facts. We explain event fact tables in more detail in 6.4.4, “Handling event-based fact tables” on page 311.

We discuss the topic of event-based fact tables in the *Identify the facts* phase to alert the reader to the considerations associated with an event-based fact table. Some of these considerations are:

- Event-based fact tables typically have pseudo facts or no facts at all.
- Pseudo facts can be helpful in counting.
- The factless fact event table has only foreign keys and no facts. The foreign keys can be used for counting purposes.

Based on the current requirements, our retail sales business process dimensional model has no event-based fact table.

5.5.6 Composite key design

A fact table primary key typically is comprised of multiple foreign keys, one from each dimension table. Such a key is called a *composite, or concatenated*, primary key. However, it is not mandatory to have all the foreign keys included in the primary key.

Also, the combination of all foreign keys of the dimensions in the fact table will not always guarantee uniqueness. In such situations, you may need to include a degenerate dimension as a component in the primary key. It is mandatory that a primary key is unique.

To determine whether a degenerate dimension needs to be included in the fact table primary key to guarantee uniqueness, consider the dimensional model depicted in Figure 5-31.

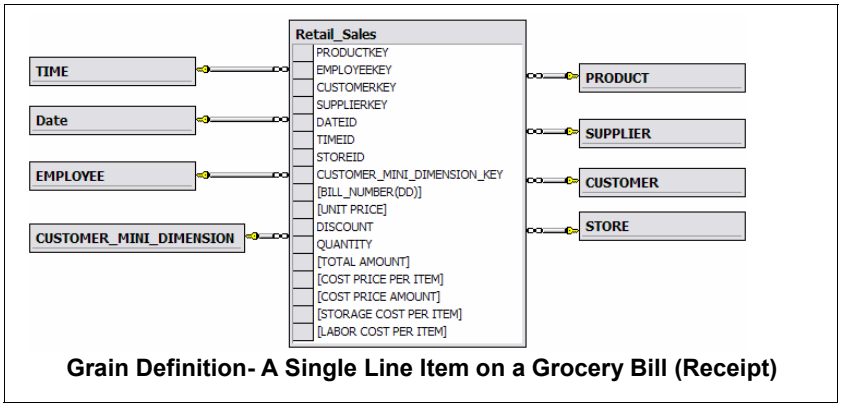


Figure 5-31 Retail sales grocery store business process

The grain definition of this schema is a single line item on a grocery bill. The date dimension stores data at the day level and the time dimension stores the data at the hourly level. In order to build the primary key for this fact table, assume that we create the composite primary key of the fact table consisting of only the foreign keys of all the dimension. Assume that the following two sales occur:

1. On October 22, 2005, Time: 8:00AM, customer C1 buys product P1 from store S1, product P1 was supplied by supplier S1, the employee who sold the product was E1. Customer C1 gets a bill with bill number equal to 787878.
2. On October 22, 2005, Time: 8:30AM, customer C1 buys product P1 from store S1, product P1 was supplied by supplier S1, the employee who sold the product was E1. Customer C1 gets a bill with bill number equal to 790051.

The above sales scenarios are represented in the fact table shown in Figure 5-32 on page 179. We observed that the UNIQUENESS of the fact table row is not guaranteed if we choose the primary key as equal to all foreign keys of all the dimensions. The uniqueness can only be guaranteed only if we include the degenerate dimension (BILL_NUMBER). Of course, we correctly assume that for every new purchase the same customer gets a new bill number.

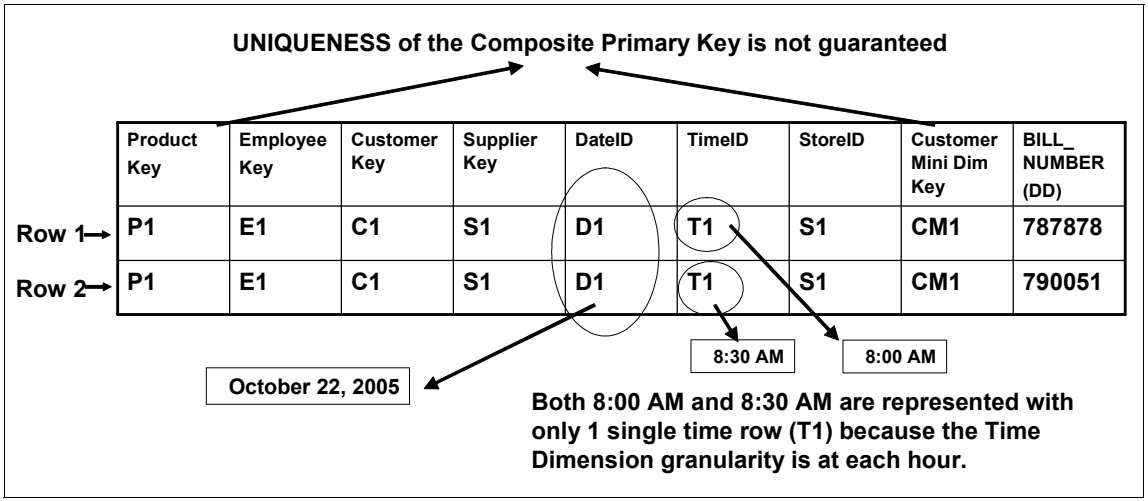


Figure 5-32 Composite fact table design

The composite key for our fact table consists of all the foreign dimension keys and degenerate dimension (Bill_Number).

Note: Typically the fact table primary key consists of all foreign keys of the dimensions. However, the uniqueness of the fact table primary key is not always guaranteed this way. In some scenarios, you may need to include one or more degenerate dimensions in the fact primary key to guarantee uniqueness. On the contrary, in some situations, you may observe that the primary key uniqueness could be guaranteed by including only some of the many foreign keys (of the dimensions) present inside the fact table.

The guarantee of uniqueness of the primary key of the fact table is determined by the grain definition for the fact table. However, a composite key that consists of all dimension foreign keys is not guaranteed to be unique. We discussed one such case as depicted in Figure 5-32. We discuss more about composite primary key design in6.4.3, “Composite key design for fact table” on page 308.

5.5.7 Fact table sizing and growth

In this activity we estimate the fact table size and also predict its growth. There are basically two ways to calculate the growth in fact table data. They are:

- **Understanding the business:** Assume that the retail sales business generates a gross revenue of $100 million. Also assume that the average price of a line item is $2. Then there would be ($100 million)/ ($2) = 50 million line items generated per year, and 50 million rows inserted into the star

schema for the retail sales business process. Recall that the grain for the retail sales grocery business process was *a single line item on a grocery bill*.

- **Technical perspective:** The other way to calculate the exact size of fact table growth is to calculate the size of the foreign keys, degenerate dimensions, and facts. After we calculate the total size of the fact table columns, we multiply it by the number of rows that could be possibly be inserted assuming all permutations of all products that sell in all stores on all days. We may also similarly calculate the growth of the fact table for a year. For example, consider the star schema designed for the retail sales business process. This is shown in Figure 5-33 on page 181. To calculate the maximum possible size of the fact table, perform the following steps:

  a. Calculate the approximate number of rows inside each of the dimensions, assuming that the dimensions have following rows:

     - Time dimension: 4 rows
     - Date dimension: 365 rows for 1 year
     - Product dimension: 100 rows (100 products in all)
     - Store dimension: 2 rows (for 2 stores)
     - Customer dimension: 1 million customers
     - Customer_Mini_Dimension: 5 rows
     - Supplier dimension: 50 suppliers
     - Employee dimension: 10 employees

  b. Calculate the base level of fact records by multiplying together the number of rows for each dimension, calculated in the step above. For example:

     - 4 x 365 x 100 x 2 x 1 million x 5 x 50 x 10 = 730000000 rows or 730 million rows.

     Of course this is a huge number of rows if every product was sold in every store by every employee to every customer.

  c. We calculate the maximum fact table size growth as follows:

     - Number of foreign keys = 8
     - Number of degenerate dimensions = 1
     - Number of facts = 8

  Assuming that the fact table takes 4 bytes for an integer column,

  Total size of 1 row =(8+1+8) x 4 bytes= 68 bytes.

  So, the maximum data growth for this dimensional model is:

  730 million rows x 68 bytes (size of 1 row) = 45 GB

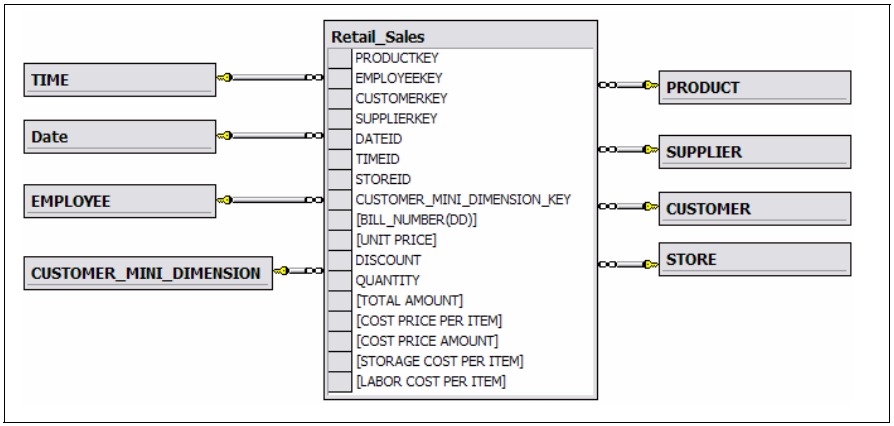


Figure 5-33 Retail sales business process dimensional model

This mathematical fact table growth calculation helps the DBA to calculate the approximate and maximum growth of the fact table. This way the DBA can be made aware of the possible growth of the dimensional model and consider potential performance tuning measures.

5.6 Verify the model

The primary focus of this phase is to test the dimensional model to see if it meets the business requirements. The dimensional model at this point would contain no data. The testing occurs to see if the existing model can answer all questions posed during the requirement gathering phase.

5.6.1 User verification against business requirements

Before we complete the dimensional design and pass it on to the ETL team to begin the ETL design, we must verify the model against the business requirements analyzed in 5.2.7, “Requirements analysis” on page 118. Table 5-28 shows the results of the verification.

Table 5-28 Validate model against requirements

Q. no.	Business requirement	Meets?
Q1	What is the average sales quantity this month for each product in each category?	Yes

Q. no.	Business requirement	Meets?
Q2	Who are the top 10 sales representatives and who are their managers? What were their sales in the first and last fiscal quarters for the products they sold?	Yes
Q3	Who are the bottom 20 sales representatives and who are their managers?	Yes
Q4	How much of each product did U.S. and European customers order, by quarter, in 2005?	Yes
Q5	What are the top five products sold last month by total revenue? By quantity sold? By total cost? Who was the supplier for each of these products?	Yes
Q6	Which products and brands have not sold in the last week? The last month?	Yes
Q7	Which salespersons had no sales recorded last month for each of the products in each of the top five revenue generating countries?	Yes
Q8	What was the sales quantity of each of the top five selling products on Christmas, Thanksgiving, Easter, Valentine's Day, and Fourth of July?	Yes
Q9	What are the sales comparisons of all products sold on weekdays compared to weekends? Also, what was the sales comparison for all Saturdays and Sundays every month?	Yes
Q10	What are the top 10 and bottom 10 selling products each day and week? Also at what time of the day do these sell? Assuming there are 5 broad time periods - Early morning (2AM - 6AM), Morning (6AM - 12PM), Noon (12PM - 4PM), Evening (4PM - 10PM) Late night Shift (10PM - 2AM)	Yes

At this point the dimensional modelers should have the report writers build pseudo logic to answer questions Q1 to Q10 in Table 5-28 on page 181. This process helps in the verification process from a report feasibility standpoint. Then, if there are any missing attributes from the dimension tables, they can be added.

In addition to validating the model against the requirements, also confirm requirements for handling history. Questions that you may need to consider and validate are shown in Table 5-29 on page 183. You also need to validate these history preserving requirements against the dimensional model.

Table 5-29 Questions relating to maintaining history

History requirement	Action
Employee changes from Region A to Region B?	Overwrite or maintain history
Employee changes Manager	Overwrite or maintain history
Product changes from Category A to Category B?	Overwrite or maintain history
More questions	Overwrite or maintain history

5.7 Physical design considerations

In this phase we focus on physical design considerations, as depicted in the DMDL in Figure 5-34.

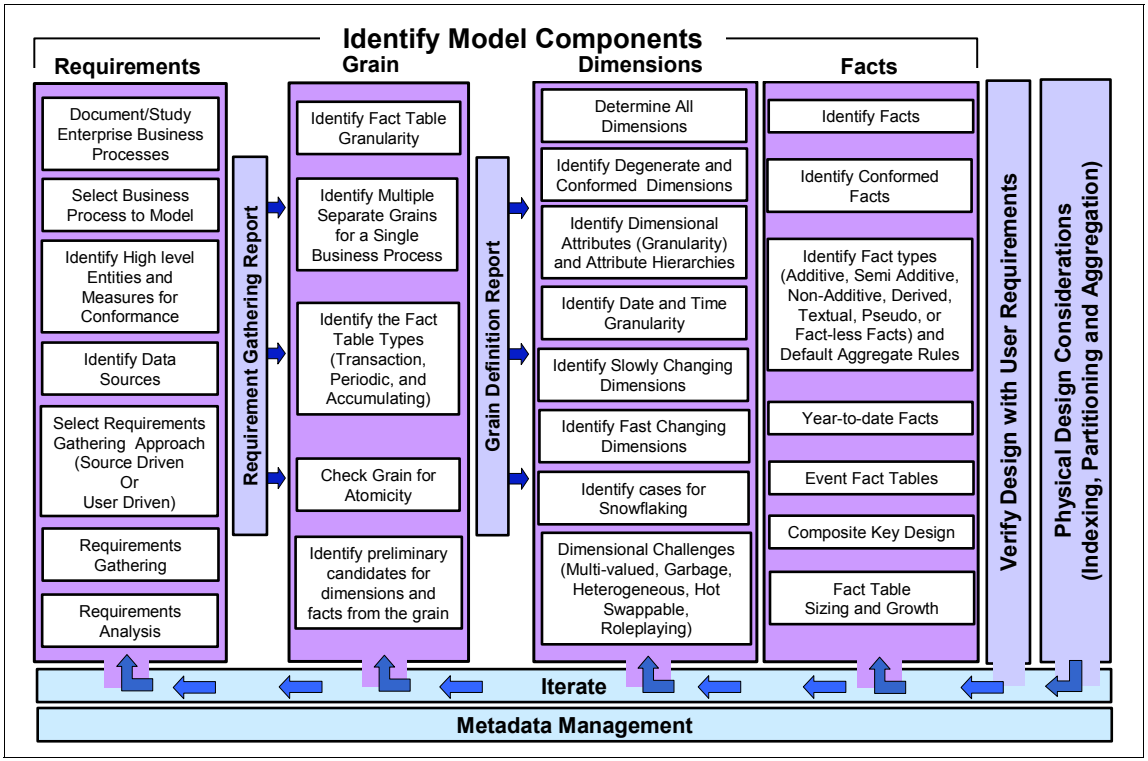


Figure 5-34 Dimensional Model Design Life Cycle

The primary focus of this phase is to design the strategy to handle the following:

- Aggregation

- Aggregate navigation
- Indexing
- Partitioning

5.7.1 Aggregations

Aggregates provide a restricted list of the key columns of a fact table and an aggregation - generally a SUM() - of some or all of the numeric facts. Therefore, a fact table with eight dimensional keys and four numeric facts can be aggregated to a summary table of three dimensional keys, plus two facts.

In simple terms, aggregation is the process of calculating summary data from detail base level fact table records. Aggregates are a powerful tool for increasing query processing speed in dimensional data marts. The aggregation is primarily performed by using attributes of a dimension which are a part of a hierarchy.

A dimension table is made of attributes, and consists of hierarchies. A hierarchy is a cascaded series of many-to-one relationships. Some attributes inside the dimension table typically belong to one or more hierarchy.

Each attribute that belongs to a hierarchy associates as a parent or child with other attributes of the hierarchy. This parent-child relationship provides different levels of summarization. The various levels of summarization provide the business user the ability to drill up or drill down in the report. Highly aggregated data is faster to retrieve than detailed atomic level data. And, the fact table typically occupies a large volume of space when compared to the aggregated data.

The lowest level of aggregation, or the highest level of detail, is referred as the grain of the fact table. The granularity of the dimension affects the design of data storage and retrieval of data. One way to identify candidates for aggregates is to use automated tools that are available in the market, or write your own applications to monitor the SQL generated in response to business queries and identify predictable queries that can be processed using precomputed aggregates rather than ad hoc SQL.

Costs associated with aggregation

Aggregating detailed atomic fact tables improves query performance. However, there are costs associated with aggregation, such as:

- Storage cost
- Cost to build and maintain the ETL process to handle the aggregated tables

Aggregations to avoid

Aggregation should not be considered as a substitute for reducing the size of large detailed fact tables. If data in the fact table is summarized, detailed information in the form of dimensions and facts is often lost. If the business needs detailed data from a summarized fact table, they simply cannot get it. They would need to look for the details back in the source OLTP system that provided the aggregated fact table data. Of course if the business has to go back to the source OLTP systems to get the answers, then the whole purpose of building a dimensional model should be questioned. We discuss the importance of having a detailed atomic grain in more detail in 6.2.2, “Importance of detailed atomic grain” on page 228.

Another important point while creating aggregates is to avoid mixing aggregated data and detailed data by including year-to-date aggregated facts with the detailed facts. This is primarily because year-to-date facts are additive, so it could result in accidental miscalculations. For more on this, see 5.5.4, “Year-to-date facts” on page 176.

Suggested approaches for aggregation

In this section, we provide guidelines for preparing aggregate tables based on detailed (highly atomic) base-level fact tables:

1. Identify all dimensions and their hierarchies from the base level atomic dimensional model. These dimensions and hierarchies are identified from the base-level atomic dimensional model, depicted in Figure 5-35.

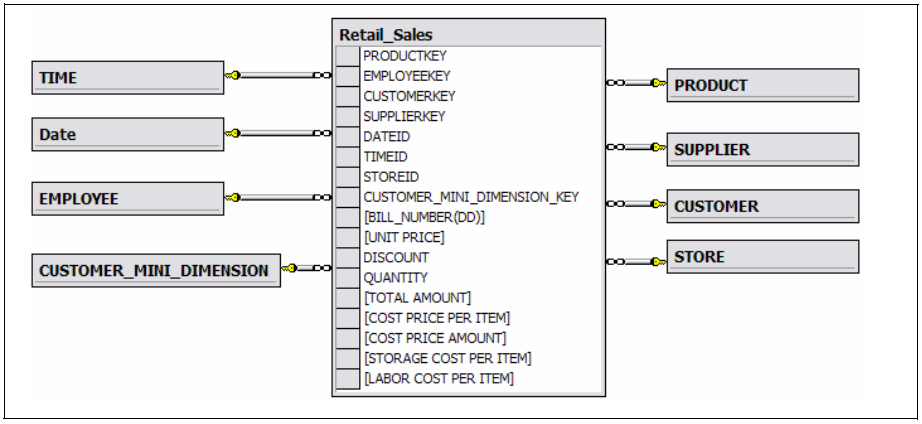


Figure 5-35 Retail sales business process dimensional model

Table 5-30 on page 186 shows all dimensions and all hierarchies associated with each of these dimensions, including their levels.

Table 5-30 Dimensions and their hierarchies

Name	Hierarchy	Type
Date	*Calendar Hierarchy*: Calendar Year → Calendar Month → Calendar Week *Fiscal Hierarchy*: Fiscal Year → Fiscal Quarter → Fiscal Month → Fiscal Day	Balanced
Time	No Hierarchy	
Product	CategoryName → BrandName → ProductName	Balanced
Employee	No Hierarchy	
Supplier	Supplier Country → Supplier Region → Supplier City	Balanced
Customer	Customer Country → Customer Region → Customer City	Balanced
Store	Store Country → Store State → Store Region → Store Area	Balanced
Customer_ Mini_ Dimension	No Hierarchy	

2. Identify all possible combinations of these hierarchy attributes which are used together by business for reporting.

   In this step we identify all attributes from the hierarchies (see Table 5-30) to determine which of these are used frequently together. This is extremely critical, particularly if there are a huge number of dimensions with several hierarchies having several attributes in them.

   Assume from studying the business requirements, we find that the attributes (relating to Date, Product, and Store dimensions) are used together. These attributes are shown in Table 5-31.

Table 5-31 Candidate dimensions and their hierarchies for aggregation

Name	Hierarchy	Type
Date	*Calendar Hierarchy*: Calendar Year → Calendar Month → Calendar Week	Balanced
Product	CategoryName → BrandName	Balanced
Store	Store Country → Store State → Store Region	Balanced

3. Calculate the number of values of each attributes selected for aggregation, in Table 5-31 on page 186.

   It is important to consider the number of values for attributes that are candidates for aggregation. For example, suppose that you have 1 million values for the lowest level product name (assuming that the store sells 1 million different types of products). We do not include the product name as a candidate to be aggregated because we are dealing with a huge number of rows. We use the brand and category attributes of the product hierarchy because the brand has about 10 000 values and category has about 1 500 values. Moreover, the business is more interested in understanding the sales of brands and categories of products, than individual products.

   The number of values each attribute has is indicative of whether the attribute is a candidate to be aggregated. For example, if we find that a low level member in the hierarchy has been included and has a huge number of members (values), then we may drop that particular attribute and choose a higher level attribute which would ideally have fewer values.

   Table 5-32 shows attributes with examples of the number of possible values.

Table 5-32 Dimension attributes with values

Name	Hierarchy	Type
Date	*Calendar Hierarchy*: Calendar Year (12 Years) → Calendar Month (12 Month Names) → Calendar Week (52 Weeks)	Balanced
Product	CategoryName (1,500 Categories) → BrandName (10,000 Brands)	Balanced
Store	Store Country (50 Countries) → Store State (50 States) → Store Region (6000 Regions)	Balanced

4. Validate the final set of attribute candidates and build the aggregated dimensional model.

   In this step we validate the final dimensional attributes identified in Table 5-32. We may also decide to drop one or more attributes if we think that the attribute has a large number of values. The final set of the aggregated dimensional model is shown in Figure 5-36 on page 188. Depending upon the need, one or more such aggregated models can be created.

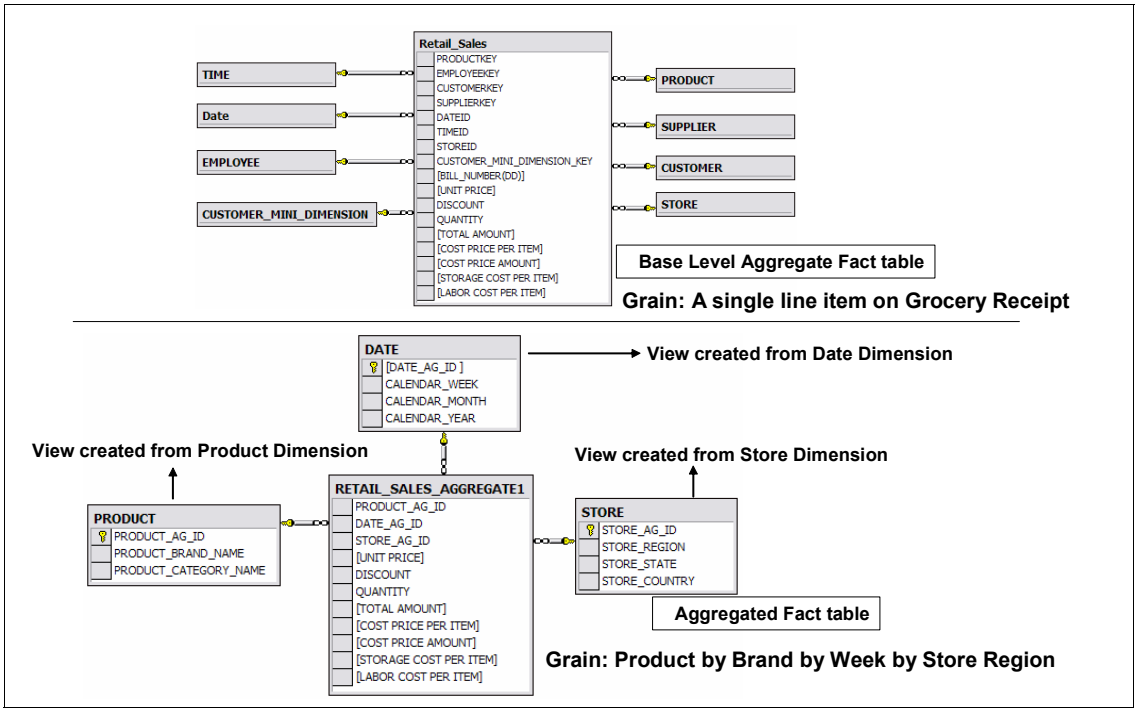


Figure 5-36 Aggregated dimensional design

5.7.2 Aggregate navigation

Aggregate navigation is considered an advanced concept in data warehousing, although the concept is actually simple.

What is aggregate navigation?

Aggregate navigation is software that intercepts SQL requests and transforms them to use the best available aggregates. Aggregate navigation is software that intercepts an SQL request (say SQL1) and transforms it to a new SQL statement (assume SQL2) to be used against a particular aggregate. Aggregate navigation is a technique that involves redirecting SQL queries to appropriate precomputed aggregates. The concept behind this technique is that the SQL queries are intercepted and rewritten to take the best advantage of aggregate tables available inside the warehouse. As shown in Figure 5-37 on page 189, Aggregate navigator
is a software application that sits between the user and data warehouse.

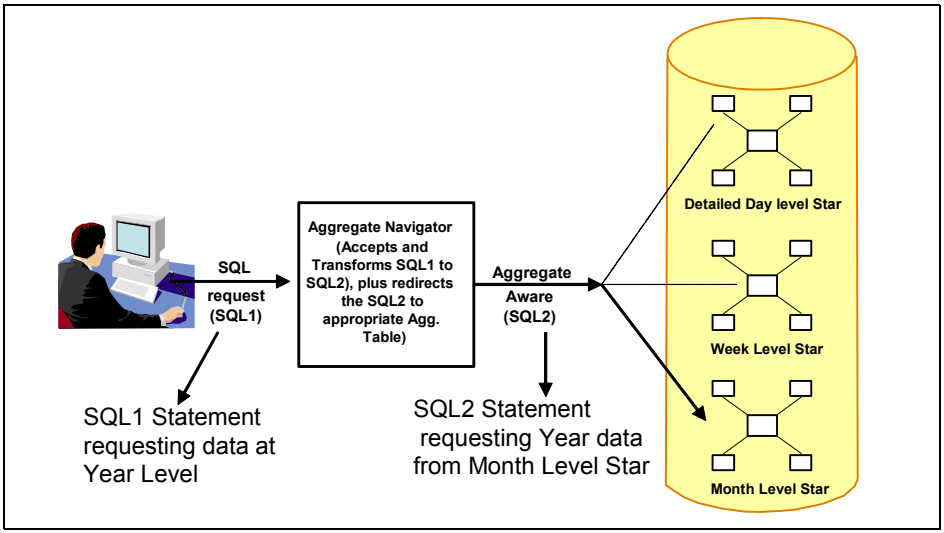


Figure 5-37 Aggregate navigator

The Aggregate navigator accepts the SQL1 statement and analyzes it. For example, as shown in Figure 5-37, the aggregator accepts the SQL1 statement and sees that the user requests data at the year level. Now there are 3 star schemas in Figure 5-37 from which the user can get the data for year level. The aggregate navigator chooses the month level star schema. Choosing the month level star schema instead of the day or week level star schema improves the performance of the query. This is because fewer rows in month need to be summarized to get to the year level data. Had we chosen the day or week, more numbers of rows would have to be summarized to get to the same result.

Aggregate navigation helps to optimize the queries by choosing the most appropriate aggregated schema to get to the desired results. Several database vendors have chosen to implement aggregate navigation software in their databases. In the next section, we discuss optimization with IBM DB2.

DB2 Optimizer and DB2 Cube Views

DB2 Cube Views V8.2 works together with partner BI tools to accelerate OLAP queries. Queries from many data sources, supported by IBM WebSphere Information Integrator, can be accelerated by DB2 Cube Views. DB2 Cube Views works by using cube meta data to design specialized summary tables containing critical dimensions and levels — or slices — of the cube. The DB2 optimizer rewrites incoming queries, and transparently routes eligible queries to the appropriate summary tables for significantly faster query performance. The Cube Views-created summary tables, also called Materialized Query Tables (MQTs), can accelerate all SQL-based queries to the data warehouse, not just those

using a particular tool or interface. The DB2 optimizer rewrites and redirects the incoming queries as shown in Figure 5-38. For a more detailed discussion how the DB2 SQL optimizer navigates between the various base and summary tables, refer to 6.5.1, “DB2 Optimizer and MQTs for aggregate navigation” on page 318.

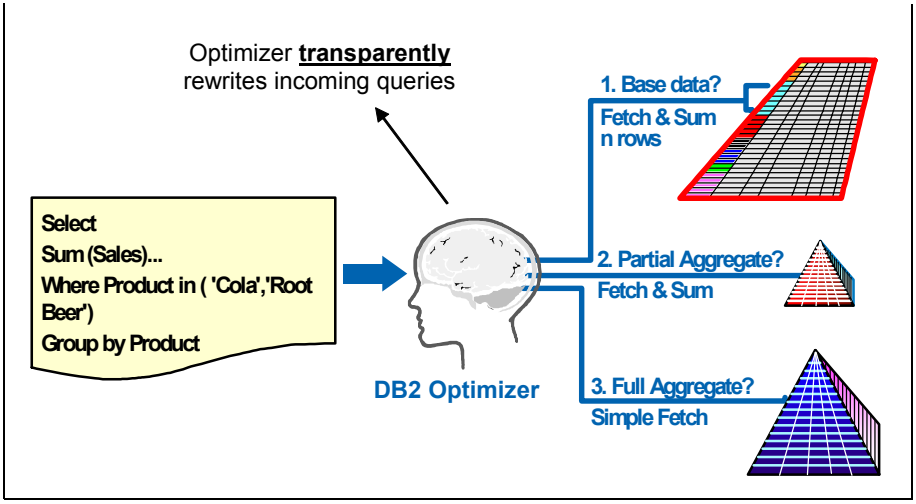


Figure 5-38 DB2 Optimizer

5.7.3 Indexing

Indexing of database tables can improve query performance, for example in situations where a number of tables are joined or a subsets of records are retrieved.

In general, indexes keep information about where the record (row) with a particular key or attribute is physically located in a table. If a table is indexed, the database engine does not need to scan the entire table to look for a row with the specific key or attribute. Instead of scanning the entire table, the database reads the index entries and directly retrieves the rows from the table. It is typically better to scan indexes than the entire table.

Two types of indexes

In this section we provide a brief description of index types. They are:

- **Unique:** Each row is uniquely identified by the value of the index.
- **Non-unique:** The value of the index can identify one or more rows in the table.

In addition, indexes can differ based on their physical organization:

- **B-Tree:** Leaf nodes contain the value of the index and a pointer to the physical row. Use this type of index for attributes with high cardinality, such as PK or FK.
- **Bitmap:** Each unique value has a corresponding bitmap where each bit represents each row in the table, and value 1 in the bitmap means the corresponding row has this value. Use this type of index for attributes with low cardinality. An example is a field storing a status such as active or inactive.

Clustering

B-tree indexes can also be clustered. Here physical rows are sorted in order of the index, which can dramatically decrease the number of I/O operations when tables are read in order of index.

Clustered indexes are most efficient if no modification to tables are performed after the clustered index is created. If new rows are added or old rows are updated and/or deleted, the efficiency of a clustered index decreases because the rows are no longer in the physical order of the index. To correct this, you should recreate clustered indexes from time to time. Although, this can be a rather expensive operation if the tables become very large. In some RDBMS, indexes can be partitioned in a similar way that tables are partitioned. That is, some part of the indexes can be put in separate logical spaces. This can also improve query response time since some parts of index (index partitions) can be completely eliminated from the scan.

Index maintenance

In addition to the advantages that indexes typically bring, there also costs. Indexes require additional space and also increase processing time when records are inserted, deleted, or updated. For example, when a row is updated, the RDBMS must also delete the old index entries corresponding to old values and add new index entries for new values of updated columns in the row.

Note: Activities that may help in maintaining indexes are:

- Use the run statistics utility for all tables and all indexes.
- Reorganize clustered indexes.

Indexes for star schema

To summarize, the star schema model consists of dimension and fact tables. A fact table contains facts and foreign keys (FK) of dimensions. The fact table foreign key (FK) points to corresponding primary keys (PK) in the dimension tables. Dimension tables consist of a primary key and attributes that describe the measures of the fact table.

Fact tables may have one composite primary key to identify a unique set of measures and foreign keys (FK). If the set of foreign keys is not unique, we can introduce one or more degenerate dimensions, which, along with the FKs, will make it a unique key.

Note: The RDBMS optimizer decides when to use and when to not use indexes, using table and index statistics to make the decision.

Indexes for dimensions

Indexing dimension tables is an iterative activity. Dimension tables may have many indexes, and those listed below should be considered as mandatory:

- One unique B-tree index for the primary key (PK) identifying the row in dimension table.
- For each attribute corresponding to a dimension hierarchy level one, a non-unique index should be used - unless it is snowflaked.
- One or more additional non-unique indexes on attributes, to be used as filters in queries.
- Non-unique B-tree index for the foreign key of a snowflaked dimension table.

Indexes for a fact table

First consider if we need to ensure uniqueness within the fact table. If yes, then create a unique B-tree index from foreign keys, and degenerate dimensions if applicable.

Then create a non-unique non-clustering B-tree index for each foreign key and degenerate dimensions.

Do not index numeric facts attributes. If you have need to filter facts by some numeric, you may introduce a new dimension such as a *fact value band*. This table may store band range of values, such as100-9900 and 9901-99992. You can also address this need with an OLAP reporting application. However, we show a technique that can be used to index facts. It is discussed in more detail in “Selective indexes” on page 194

Note: The type of indexes you choose depends on attribute cardinality and on the capabilities of underlying indexes. We suggest using B-tree for high cardinality attributes and Bitmap for low or medium cardinality attributes. For example, if attribute values are highly duplicated, such as gender, status, or color attributes, consider using bitmap indexes. Bitmap indexes, if well designed, can dramatically improve query response times. Multidimensional clustered (MDC) or block indexes (see “MDC indexes” on page 194) are suitable for indexing fact table FKs and degenerate dimensions.

Vendor specific indexing techniques

Several vendors offer special proprietary types of indexes, designed for optimizing queries, for example, the multidimensional clustered (MDC) indexes in IBM DB2 or generalized key (GK) indexes in IBM Informix Extended Parallel Server (XPS).

Generally these indexes are defined either on one, or more than one, table (GK indexes) or on clusters of values from various attributes (MDC indexes). The primary goal of such advanced indexing is to eliminate unnecessary joins and table scans, and to reduce query response time. Generalized key indexes provide the following three types of advanced indexes that can improve OLAP query performance:

- Join indexes
- Virtual indexes
- Selective indexes

Join index

This capability allows you to create an index on a fact table that contains attributes of the dimension table. Using this index can eliminate the need to join the tables.

For example, if we have a SALES fact table, with item and sales amount, and the dimension table PRODUCT, describing items that have attribute BEVERAGE, then you can create an index on the fact table containing BEVERAGE of each item. As an example, you can do that with the following command:

```
CREATE GK INDEX type_of_sale ON SALES
(SELECT PRODUCT.BEVERAGE FROM SALES, PRODUCT
WHERE SALES.PRODUCT_KEY = PRODUCT.PRODUCT_KEY)
```

Then when querying for sales amounts of certain product types, the join of the SALES table and PRODUCT dimension can be eliminated. For example, consider the following query:

```
SELECT SUM(SALES_AMOUNT) FROM SALES, PRODUCT
WHERE SALES.PRODUCT_KEY = PRODUCT.PRODUCT_KEY AND PRODUCT.BEVERAGE =
"SODA"
```

Virtual indexes

This capability allows you to create an index not only on one or more columns, but also on expressions based on those columns. This enables fast queries on computed values, and can save disk space as well.

For example, if you have a table with columns UNITS and UNIT_COST, you can create an index on the *COST_OF_SALE* as follows:

```
CREATE GK INDEX I_cost_of_sale on SALES (SELECT
SALES.UNITS*SALES.UNIT_COST FROM SALES)
```

This index then speeds up queries such as the following:

```
SELECT * FROM SALES WHERE UNITS*UNIT_COST >60
```

Note: Virtual indexes are useful when a fact table contains derived facts.

Selective indexes

This capability enables you to index only part of a large table, saving disk space and speeding up the queries.

For example, you can create an index such as the following:

```
CREATE GK INDEX since1995 ON SALES (SELECT AMOUNT FROM SALES JOIN
DATE_DIM ON SALES.DAY_DIM_KEY=DATE_DIM.DAY_DIM_KEY WHERE DATE_DIM.YEAR
>=1995)
```

Then you can create a fast query such as this:

```
SELECT * FROM SALES WHERE amount >500 AND YEAR >=1995
```

MDC indexes

IBM DB2 V8.x enables you to create advanced clustered indexes, also known as multidimensional clustered (MDC) indexes, which are optimized for OLAP queries. MDC indexes are based on blocks (extents), while traditional indexes are based on rows. Therefore in an MDC index, the leaf entries point to blocks of pages which have rows with the same value of the columns. Traditional index leaf entries point to a physical row on a page. MDC indexes are clustered as rows and grouped by values of columns in the indexes. At least one block is created for each unique value of a dimension.

A *cell* is a unique combination of values across all dimensions. Data in the MDC tables is clustered via cells.

When creating MDC tables, a blocked index is created for each dimension and one covering all dimensions. Blocked indexes optimize query response time because the optimizer can quickly identify blocks of pages with required values in particular columns. Further, the maintenance of MDC tables is simplified as no data reorganization is needed. If there is no cell for new or updated rows or if all blocks for this cell are full, another block is allocated and a new entry for this block is added into block indexes.

Note: When designing MDC tables, the cost of disk space should be considered. Although block indexes demand less disk space, the minimal disk space needed for data in MDC tables is the cartesian product of all cardinalities of all dimensions multiplied by block (extent) size.

You can find a more detailed example of indexing in 6.5.2, “Indexing for dimension and fact tables” on page 324, which includes the following topics:

- Indexes for the star schema (includes indexing for dimensions and facts)
- Differences between indexed and non-indexed star schemas
- Access plans for querying an indexed star schema with SQL
- Access plans for querying a non-indexed star schema with SQL

5.7.4 Partitioning

Partitioning a table divides the table by row, by column, or both. If a table is divided by column, it is said to be vertically partitioned. If by row, it is said to be horizontally partitioned. Partitioning large fact tables improves performance because each partition is more manageable and provides better performance characteristics. Typically, you partition based on the transaction date dimension in a dimensional model. For example, if a huge fact table has billions of rows, it would be ideal for one month of data to be assigned its own partition.

Partitioning the data in the data warehouse enables the accomplishment of several critical goals. For example, it can:

- Provide flexible access to data
- Provide easy and efficient data management services
- Ensure scalability of the data warehouse
- Enable elements of the data warehouse to be portable. That is, certain elements of the data warehouse can be shared with other physical data warehouses, or archived on other storage media.

Guidelines for partitioning star schema data are explained below:

- A large fact table with a large number of rows should be partitioned based on the date dimension.
- Depending upon the growth of the fact table, an individual partition may be created for each unique months of data or each unique quarter of data.
- Large dimension tables, such as customer table of a government agency, having millions of customers may also be partitioned.

We usually partition large volumes of current detail data by dividing it into smaller segments. Doing that helps make the data easier to:

- Restructure
- Index
- Sequentially scan
- Reorganize
- Recover
- Monitor

In addition, other advantages of partitioning are:

- Improved response times because the SQL query only accesses the partition of data needed to answer the query.
- Partitions can be more easily maintained compared to one large table.
- Involves no extra cost, as most partitioning capability is typically included as part of the RDBMS.

Every database management system (DBMS) has its own specific way of implementing physical partitioning, and they all can be quite different. For example, it is an important consideration whether or not the DBMS also supports partition indexing. Instead of DBMS or system level partitioning, you can consider partitioning by application. This would provide flexibility in defining data over time, and portability in moving to the other data warehouses. Notice that the issue of partitioning is closely related to multidimensional modeling, data granularity modeling, and the capabilities of a particular DBMS to support data warehousing.

5.8 Meta data management

Figure 5-39 on page 197 shows that the meta data management block spans the entire DMDL. That is, every phase of dimensional modeling produces some level of meta data.

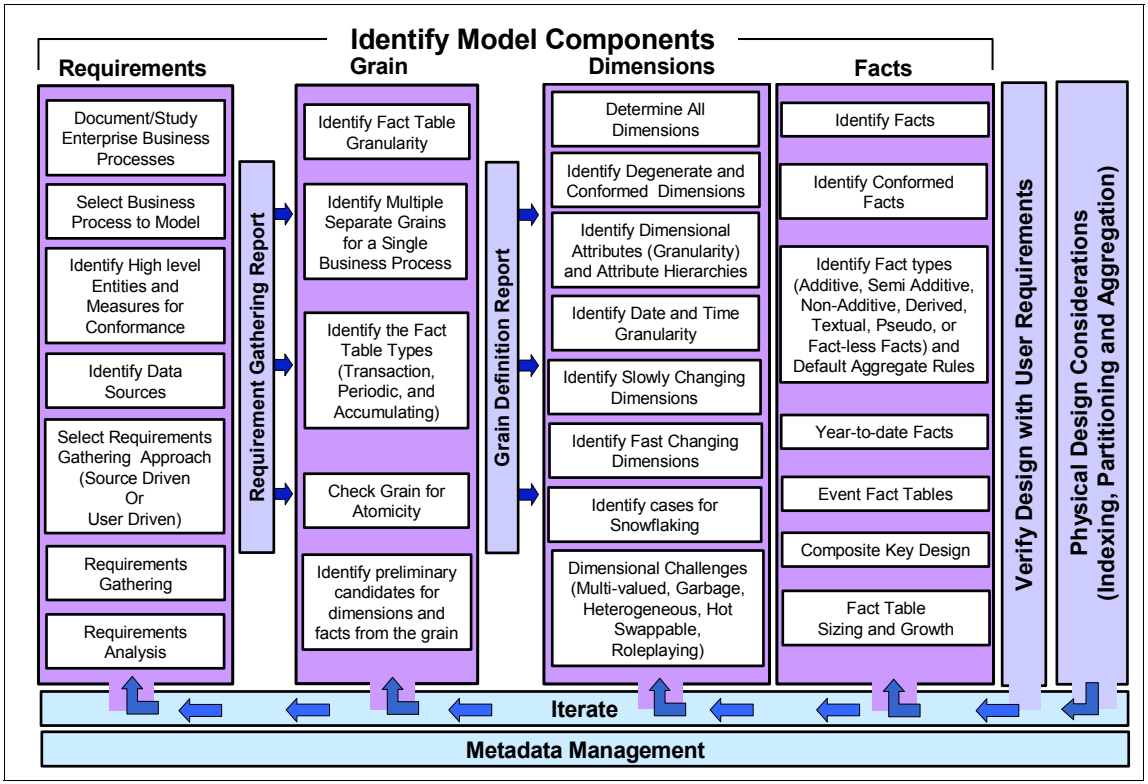


Figure 5-39 Dimensional Model Design Life Cycle

In the traditional development cycle, a model sees only sparse use after completion. This is typically when changes need to be made, or when other projects require the data. In the data warehouse, however, the model is used on a continuous basis. The users of the data warehouse constantly reference the model to determine the data they want to use in their data analysis. The rate of change of the data structure in a data warehouse is much greater than that of operational data structures. Therefore, the technical users of the data warehouse (administrators, modelers, and designers, as examples) will also use the model on a regular basis.

This is where the meta data comes in. Far from just a pretty picture, the model must be a complete representation of the data being stored.

To properly understand the model, and be able to confirm that it meets requirements, you must have access to the meta data that describes the dimensional model in business terms that are easily understood. Therefore, non-technical meta data should also be documented in addition to the technical meta data.

At the dimensional model level, a list should be provided of what is available in the data warehouse. This list should contain the models, dimensions, facts, and measures available as these will all be used as initial entry points for data analysis.

For each model, provide a name, definition, and purpose. The name simply gives something to focus on when searching. Usually, it is the same as the fact. The definition identifies what is modeled, and the purpose describes what the model is used for. The meta data for the model should also contain a list of dimensions, facts, and measures associated with it, as well as the name of a contact person so that users can get additional information when there are questions about the model.

A name, definition, and aliases must be provided for all dimensions, dimension attributes, facts, and measures. Aliases are necessary because it is often difficult to come to agreement on a common name for any widely used object. For dimensions and facts, a contact person should be provided.

Meta data for a dimension should also include hierarchy, change rules, load frequency, and the attributes, facts, and measures associated with the dimension. The hierarchy defines the relationships between attributes of the dimension that identify the different levels that exist within it. For example, in the seller dimension we have the sales region, outlet type (corporate or retail), outlet, and salesperson, as a hierarchy. This documents the roll-up structure of the dimension. Change rules identify how changes to attributes within a dimension are dealt with. In some instances, these rules can be different for individual attributes. Record change rules with the attributes when this is the case. The load frequency allows the user to understand whether or not data will be available when needed.

The attributes of a dimension are used to identify which facts to analyze. For attributes to be used effectively, meta data about them should include the data type, domain, and derivation rules. At this point, a general indication of the data type (such as character, date, and numeric) is sufficient. Exact data type definitions can be developed during design. The domain of an attribute defines the set of valid values. For attributes that contain derived values, the rules for determining the value must be documented.

Meta data about a fact should include the load frequency, the derivation rules and dimensions associated with the fact, and the grain of time or date for the fact. Although it is possible to derive the grain of time for a fact through its relationship to the time dimension, it is worthwhile explicitly stating it here. It is essential for proper analysis that this grain be understood.

5.8.1 Identifying the meta data

The following meta data is collected during the different phases of the DMDL:

- **Identify business process:** The output of this phase results in the *requirements gathering report*. This report primarily consists of the business requirements for the selected business for which you will design the dimensional model. In addition to this, it also consists of various business processes, owners, source systems involved, data quality issues, common terms used across business processes and other business-related meta data.
- **Identify the grain:** The output of this phase results in the *grain definition report,* which consists of one or multiple definitions of the grain for the business process for which the dimensional model is being designed. Also the type of fact table (transaction, periodic, or accumulating) being used is mentioned. The grain definition report also includes high level preliminary dimensions and facts.
- **Identify the dimensions:** The meta data documented for this phase contains the information as shown Table 5-33:

Table 5-33 Identify the dimensions phase meta data

Dimension meta data	Description
Name of dimension	Name of the dimension table.
Business definition	Business definition of the dimension.
Alias	Specifies the other known name by which the business users know the dimension.
Hierarchy	Defines the hierarchies present inside the dimension, such as balanced, unbalanced, or ragged.
Change rules	Specify how to handle slowly changing dimension (type-1, type-2, or type-3) or fast changing dimension.
Load frequency	The frequency of load for this dimension, such as daily, weekly, or monthly.
Load statistics	Consists of meta data such as: → Last load date: N/A → Number of rows loaded: N/A
Usage statistics	Consists of meta data such as: → Average Number of Queries/Day: N/A → Average Rows Returned/Query: N/A → Average Query Runtime: N/A → Maximum Number of Queries/Day: N/A → Maximum Rows Returned/Query: N/A → Maximum Query Runtime: N/A

Dimension meta data	Description
Archive rules	Specifies whether or not data is archived.
Archive statistics	Consists of meta data such as: → Last Archive Date: N/A → Date Archived to: N/A
Purge rules	Specifies any purge rules. For example, customers who have not purchased any goods from the store in the past 48 months will be purged on a monthly basis.
Purge statistics	Consists of meta data such as: → Last Purge Date: N/A → Date Purged to: N/A
Data quality	Specifies data quality checks. For example, when a new customer is added, a search determines if the customer already does business with another location. In rare cases separate branches of a customer are recorded as separate customers because this check fails.
Data accuracy	Specifies the data accuracy. For example, incorrect association of locations of a common customer occur in less than .5% of the customer data.
Key	The key to the dimension table is a surrogate key.
Key generation method	This meta data specifies the process used to generate a new surrogate key for a new dimension row. For example, when a customer is copied from the operational system, the translation table (a staging area persistent table) is checked to determine if the customer already exists in the dimensional model. If not, a new key is generated and the key along with the customer ID and location ID are added to the translation table. If the customer and location already exist, the key from the translation table is used to determine which customer in the dimensional model to update.

Dimension meta data	Description
Source	This includes the following meta data:
	→ Name of the source system table: <Table Name>
	→ Conversion rules: This specifies how the insert/update to the dimension table occurs. For example, rows in each customer table are copied on a daily basis. For existing customers, the name is updated. For new customers, once a location is determined, the key is generated and a row inserted. Before the update/insert takes place, a check is performed for a duplicate customer name. If a duplicate is detected, a sequence number is appended to the name. This check is repeated until the name and sequence number combination is determined to be unique. Once uniqueness has been confirmed, the update/insert takes place.
	→ Selection logic: Only new or changed rows are selected.
Conformed dimension	This specifies if the dimension is a conformed dimension.
Role-playing dimension	This specifies if the dimension is being implemented using the role-playing concept.

Dimension meta data	Description
Attributes (All columns of a dimension)	The meta data for all the dimension attributes includes the following:
	→ Name of the attribute: <Attribute Name>
	→ Definition of the attribute: Attribute definition
	→ Alias of the attribute: <Attribute Name>
	→ Change rules for the attribute: For example, when an attribute changes, then use Type-1, Type-2, or Type-3 strategy to handle the change.
	→ Data Type for the attribute: Data type, such as Integer or Character.
	→ Domain values for the attribute: Domain range such as 1-99.
	→ Derivation rules for the attribute: For example, a system generated key of the highest used customer key +1 is assigned when creating a new customer and location entry.
	→ Source: Specifies the source for this attribute. For example, for a surrogate key, the source could be a system generated value.
Facts	This specifies the facts that can be used with this dimension. Note: Semi-additive facts are additive across only some dimensions.
Subsidiary dimension	This specifies any subsidiary dimension associated with this dimension.
Contact person	This specifies the contact person from the business side responsible for maintaining the dimension.

The meta data information shown in Table 5-33 on page 199 needs to be captured for all dimensions present in the dimensional model.

- **Identify the facts:** The meta data documented for this phase contains the information as shown Table 5-34.

Table 5-34 Identify the facts phase meta data

Fact table meta data	Description
Name of fact table	The name of the fact table.

Fact table meta data	Description
Business Definition	The business definition of the fact table.
Alias	The alias specifies another name by which the fact table is known.
Grain	This specifies the grain of the fact table.
Load frequency	The frequency of load for this fact table such as daily, weekly or monthly.
Load statistics	Consists of meta data such as: → Last load date: N/A → Number of rows loaded: N/A
Usage statistics	Consists of meta data such as: → Average Number of Queries/Day: N/A → Average Rows Returned/Query: N/A → Average Query Runtime: N/A → Maximum Number of Queries/Day: N/A → Maximum Rows Returned/Query: N/A → Maximum Query Runtime: N/A
Archive rules	Specifies whether or not the data is archived. For example, data will be archived after 36 months on a monthly basis.
Archive statistics	Consists of meta data such as: → Last Archive Date: N/A → Date Archived to: N/A
Purge rules	Specifies any purge rules. For example, data will be purged after 48 months on a monthly basis.
Purge statistics	Consists of meta data such as: → Last Purge Date: N/A → Date Purged to: N/A
Data quality	Specifies quality checks for the data. For example, assume that we are designing a fact table for an inventory process. Inventory levels may fluctuate throughout the day as more stock is received into inventory from production and stock is shipped out to retail stores and customers. The measures for this fact are collected once per day and thus reflect the state of inventory at that point in time, which is the end of the working day.

Fact table meta data	Description
Data accuracy	This specifies the accuracy of fact table data. For example, assume that we are designing a fact table for the inventory process. We may conclude that the measures of this fact are 97.5% accurate at the point in time they represent. This may be based on the results of physical inventories matched to recorded inventory levels. No inference can be made from these measures as to values at points in time not recorded.
Grain of the date dimension	Specifies the grain of the date dimension. For example, the date dimension may be at the day level.
Grain of the time dimension	Specifies the grain of the time dimension. For example, the time dimension may be at the hourly level.
Key	The key of the fact table typically consists of concatenation of all foreign keys of all dimensions. In some cases, the degenerate dimension may also be concatenated to guarantee the uniqueness of the primary composite key.
Key generation method	The key generation method specifies how all the foreign keys are concatenated to create a primary key for the fact table. Sometimes a degenerate dimension may be needed to guarantee its uniqueness. This is shown in Figure 5-32 on page 179.
Source	The meta data for the source includes the following:
	→ Name of the Source: <Source Name>
	→ Conversion rules for the source: Rules regarding the conversion. For example, each row in each inventory table is copied into the inventory fact on a daily basis.
	→ Selection Logic: The selection logic behind selecting the rows.
Facts	This specifies the facts involved in the fact table. They could be: → Additive → Non-additive → Semi-additive → Pseudo → Derived → Factless fact → Textual

Fact table meta data	Description
Conformed fact	This specifies whether or not there are any conformed facts in the fact table.
Dimensions	This specifies the dimensions that can validly use these facts. Note: Some facts are semi-additive and can only be used across certain dimensions.
Contact person	This specifies the contact person from the business side responsible for maintaining the fact table.

- **Verify the model:** This phase involves documenting meta data related to the testing (and its result) done on the dimensional model. Any new requests made or changes to the requirements are documented.
- **Physical Design Considerations:** The meta data documented for this phase contains the information as shown in Table 5-35.

Table 5-35 Physical design considerations meta data

Physical design consideration meta data	Description
Aggregation	The aggregation meta data includes the following:
	→ Number of aggregate tables
	→ Dimension tables involved in creating aggregates → Dimension hierarchies involved in creating aggregation
	→ Fact table and facts involved in creating aggregates.
	Other information relating to the aggregate tables includes the following: → Load frequency → Load statistics → Usage statistics → Archive rules → Archive statistics → Purge rules → Purge statistics → Data quality → Data accuracy
Indexing	This specifies the indexing strategy used for dimension and fact tables.

5.9 Summary

In 5.2.2, “Identify business process” on page 110, we created an enterprise business process list for which there are business needs to build a data mart or dimensional model.

After identifying all the business processes, we assessed each of them for a number factor, such as:

- Complexity of the source systems of the business process
- Data availability of these systems
- Data quality of these systems
- Strategic business significance of the business process

After having assessed the various business processes, we developed the final prioritization list shown in Table 5-36.

Table 5-36 Enterprise-wide business process priority listing

Name	Complexity	Availability	Quality	Significance	Points
Retail sales	Low (3)	High (3)	High (3)	High (6)	15
Finance	High (1)	High (3)	Medium (2)	Medium (4)	10
Servicing	Low (3)	High (3)	Medium (2)	High (6)	14
Marketing	Medium (2)	Medium (2)	Medium (2)	Medium (4)	10
Shipment	Low (3)	Low (1)	High (3)	Low (2)	9
Supply management	Medium (2)	Low (1)	Medium (2)	Low (2)	7
Purchase order	High (1)	Medium (1)	Low (1)	Medium (4)	7
Labor	Low (3)	Low (1)	Low (1)	High (2)	7

Table 5-36 helped us to prioritize the business processes for which we can design the dimensional models. In this chapter, we chose the top priority retail sales business process and designed a data mart using the DMDL. After we finish with the design of one data mart, we can then move to the next priority data mart from Table 5-36.

The DMDL helps us segment the larger task of building data marts for the entire organization by choosing a step by step approach of handling each business

process, one at a time. This approach enables you to start small, and complete the project in planned phases.

In addition, you remember that data warehousing itself is a process. That is, typically you are never really finished. You will undoubtedly continue to add people, processes, and products to your enterprise. Or, specifications about those people, processes, and products will change over time. As examples, people get married, have children, and move. This requires modifying, for example, dimensional data that describes those people.

So, as with any data warehousing project, building dimensional models is a continuous and ongoing project. Having a defined DMDL, and using suggestions in this redbook, can help you to build a structured, controlled, planned, and cost-effective approach to building several dimensional models for your data warehousing environment which are integrated by using conformed dimensions and conformed facts.

6

Modeling considerations

In this chapter, we discuss considerations, and challenges that may arise, when designing dimensional models. As examples:

- Converting an E/R model to dimensional model. How do you identify fact and dimension tables from an E/R model? And, how do you convert an E/R model to a dimensional model?
- Identifying the grain.
- Working with degenerate dimensions, dimension hierarchies, time as a fact or dimension, slowly changing dimensions, fast changing dimensions, identifying and handling snowflakes, identifying garbage dimensions, handling multi-valued dimensions, use of bridge tables, handling heterogeneous products, and handling hot swappable dimensions (also referred as profile tables).
- Working with additive and semi-additive facts, composite key design, and event fact tables.
- What about physical design activities, such as indexing?
- Working with changes to data, structure, and requirements.

6.1 Converting an E/R model to a dimensional model

In this section we describe how to convert an E/R model to a dimensional model. A dimensional model can be created from the enterprise data warehouse or directly from OLTP source systems. For additional information, refer to:

- Enterprise data warehouse: “Data warehouse architecture choices” on page 57.
- OLTP Source systems: “Data modeling: The organizing structure” on page 47 in the following sections:
  - “Independent data mart architecture” on page 59
  - “Dependent data mart architecture” on page 61

The following are the steps for converting an E/R model to a dimensional model:

- Identify the business process from the E/R model.
- Identify many-to-many tables in the E/R model to convert to fact tables.
- Denormalize remaining tables into flat dimension tables.
- Identify date and time from the E/R model.

The steps are explained in detail in the following sections.

6.1.1 Identify the business process from the E/R model

It is important to understand that an E/R model can be segmented into multiple dimensional models. An E/R model (which may be an enterprise data warehouse or an OLTP source system) consists of several business processes. This is depicted in Figure 6-1 on page 211. For example, an E/R model for an ERP system includes several business processes, such as retail sales, order management, procurement, inventory, and store and warehouse inventory management.

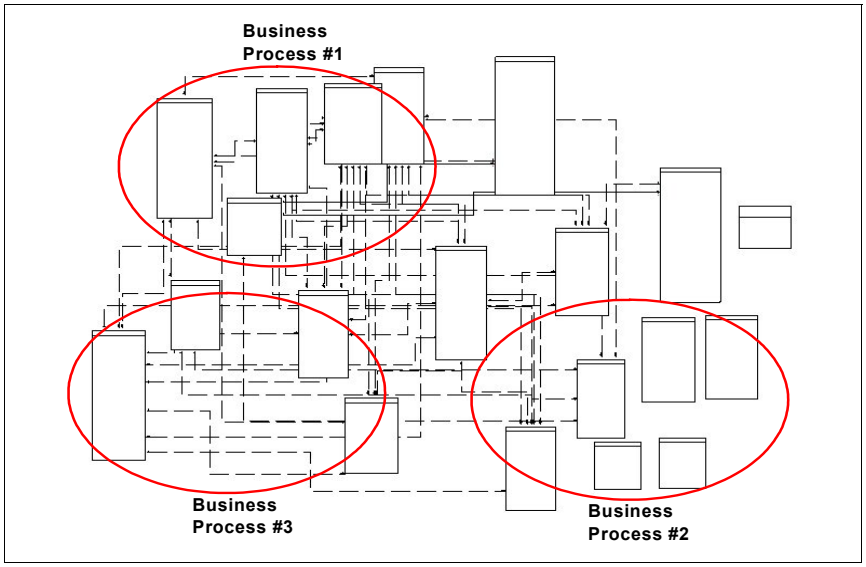


Figure 6-1 E/R model consists of several business processes

6.1.2 Identify many-to-many tables in E/R model

Once the business processes are separated, the next step is to identify the many-to-many tables (many-to-many relationships) in the E/R model and convert them to dimensional model fact tables. These many-to-many relationships contain numeric and additive non-key facts which generally become facts inside the fact table.

The idea behind this step is to identify the transaction-based tables that serve to express many-to-many relationships inside an E/R model.

Every E/R model consists of transaction-based tables which constantly have data inserted, or are updated with data, or have data deleted from them. Some of these tables also express a many-to-many relationship. For example, in an ERP database, there are transaction tables, such as Invoice and Invoice_Details, which are constantly inserted and updated because they are transaction-based tables. However, tables such as Employee and Products in an E/R model may be fairly static.

Description of many-to-many relationships

Many-to-many (m:*n*) relationships add complexity and confusion to the model and to the application development process. The key to resolving m:*n* relationships is to separate the two entities and create two one-to-many (1:*n*)

relationships between them with a third intersect entity. The intersect entity usually contains attributes from both connecting entities.

To resolve an m:*n* relationship, analyze the business rules again. Have you accurately diagrammed the relationship? The telephone directory example has a m:*n* relationship between the name and fax entities, as Figure 6-2 depicts. The business rules say, “One person can have zero, one, or many fax numbers; a fax number can be for several people.” Based on what we selected earlier as our primary key for the voice entity, an m:*n* relationship exists.

A problem exists in the fax entity because the telephone number, which is designated as the primary key, can appear more than one time in the fax entity; this violates the qualification of a primary key. Remember, the primary key must be unique.

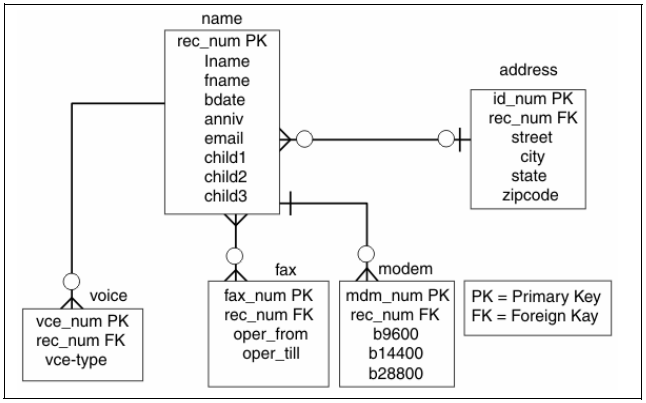


Figure 6-2 Telephone directory diagram to show many to many relationship

To resolve this m:*n* relationship, you can add an intersect entity between the name and fax entities, as depicted in Figure 6-3 on page 213. The new intersect entity, fax name, contains two attributes, fax_num and rec_num.

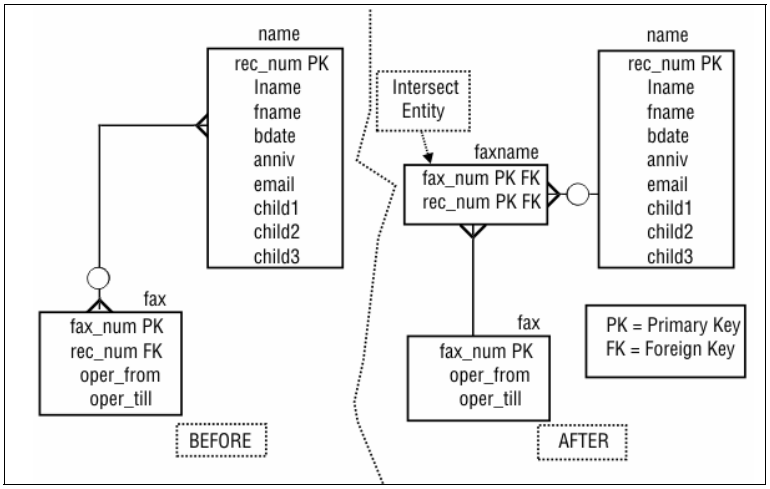


Figure 6-3 Many-to-many relationship

Another example for many-to-many relationship is shown in Figure 6-4.

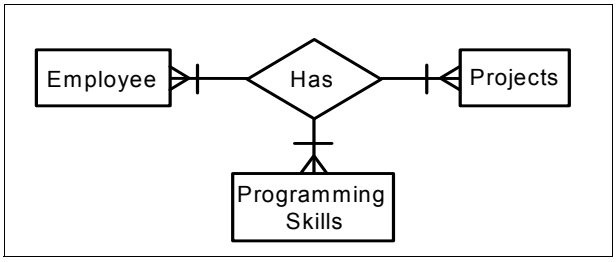


Figure 6-4 Many-to-many relationship

In Figure 6-4, the many-to-many relationship shows that employees use many programming skills on many projects and each project has many employees with varying programming skills.

6.1.3 Denormalize remaining tables into flat dimension tables

The final step involves taking the remaining tables in the E/R model and denormalizing them into dimension tables for the dimensional model. The primary key of each of the dimensions is made a surrogate (non-intelligent, integer) key. This surrogate key connects directly to the fact table.

6.1.4 Identify date and time dimension from E/R model

The last step generally involves identifying the date and time dimension. Dates are generally stored in the form of a date timestamp column inside the E/R model. You will observe that date and time-related columns are generally found in the transaction-based tables.

We explain the process of converting an E/R model to a dimensional model in the section below.

Example: An E/R model conversion

We convert the E/R model shown in Figure 6-5 to a dimensional model using the following steps:

1. **Identify the business process:** We discussed this step in detail in “Identify the business process from the E/R model” on page 210. The business process we identified for our example is retail sales. The E/R model for this retail sales schema is shown in Figure 6-5.

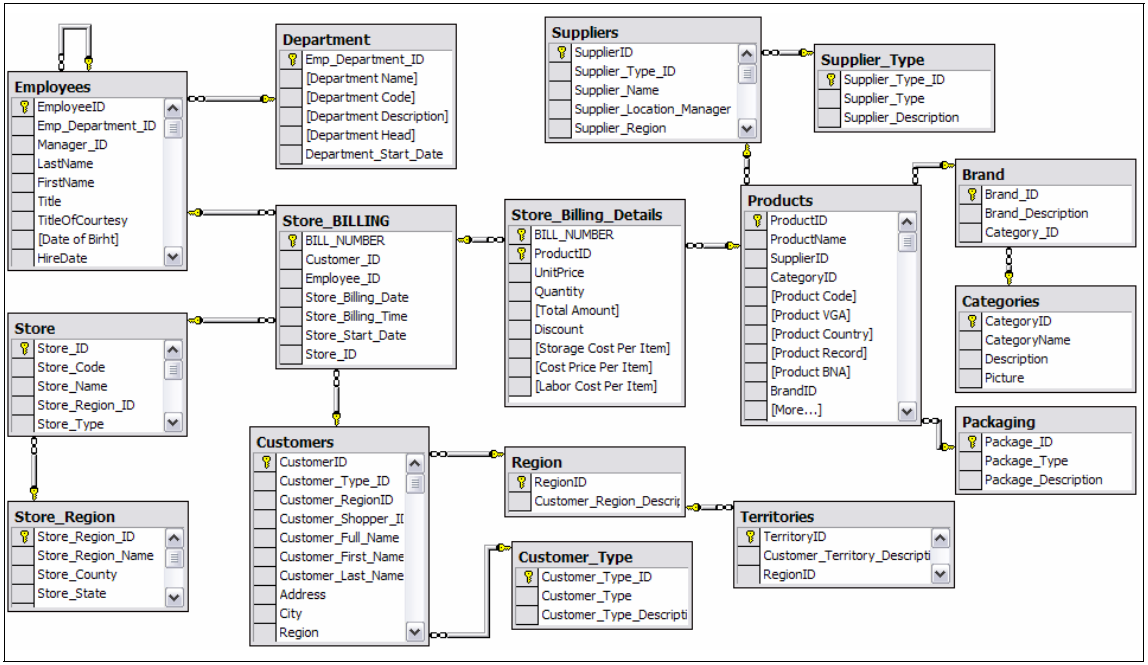


Figure 6-5 E/R model for retail sales business process

2. Identify the many-to-many tables in the E/R model to convert them to fact tables. After identifying the business process as retail sales, and identifying the E/R model as shown in Figure 6-5, the next step is to identify the

many-to-many relationships that exist inside the model. In order to find this, we must segregate the tables inside the E/R model into two types. Transaction-based tables and Non-Transaction based tables, as shown in Table 6-1.

Note: Fact tables in a dimensional model express the many-to-many relationships between dimensions. This means that the foreign keys in the fact tables share a many-to-many relationship.

What is a transaction-based table? In an E/R model, it is one which is generally involved in storing facts and measures about the business. Such tables generally store foreign keys and facts, such as quantity, sales price, profit, unit price, and discount. In transaction tables, records are usually inserted, updated, and deleted as and when the transactions occur. Such tables also in many ways represent many-to-many relationships between non-transaction-based tables. Such tables are larger in volume and grow in size much faster than the non-transaction-based tables.

What is a non-transaction-based table? This is an E/R model which is generally involved in storing descriptions about the business. Such tables describe entities such as products, product category, product brand, customer, employees, regions, locations, services, departments, and territories. In non-transaction tables, records are usually inserted and there are fewer updates and deletes. Such tables are far smaller in volume and grow very slowly in size, compared to the transaction-based tables.

Table 6-1 Transaction and Non-transaction tables in the E/R model

Transaction tables	Non-transaction tables
Store_BILLING [This transaction table stores the BILL_NUMBER, Store Billing date, and time information. The detailed Bill information is contained inside the Store_Billing_Details table.]	Employees: Stores information about employees.
Store_Billing_Details [This transaction table stores the details for each store bill.]	Department: Stores department information for every employee.
	Store: Stores the name of the store and type of store information.
	Store_Region: Stores the region, county, state, and country to which the store belongs.

Transaction tables	Non-transaction tables
	Suppliers: Stores the supplier name and supplier manager-related information.
	Supplier_Type: Stores supplier type information.
	Products: Stores information relating to products.
	Brand: Stores brand information to which different products belong.
	Categories: Stores categories information to which different product brands belong.
	Packaging: Stores packaging-related information for each product.
	Customers: Stores customer-related information.
	Region: Stores different regions to which customers belong.
	Territories: Stores territories to which different customers belong.
	Customer_Type: Stores customer classification information.

The transaction tables are identified in Table 6-1 on page 215, and we depict them in Figure 6-6 on page 217.

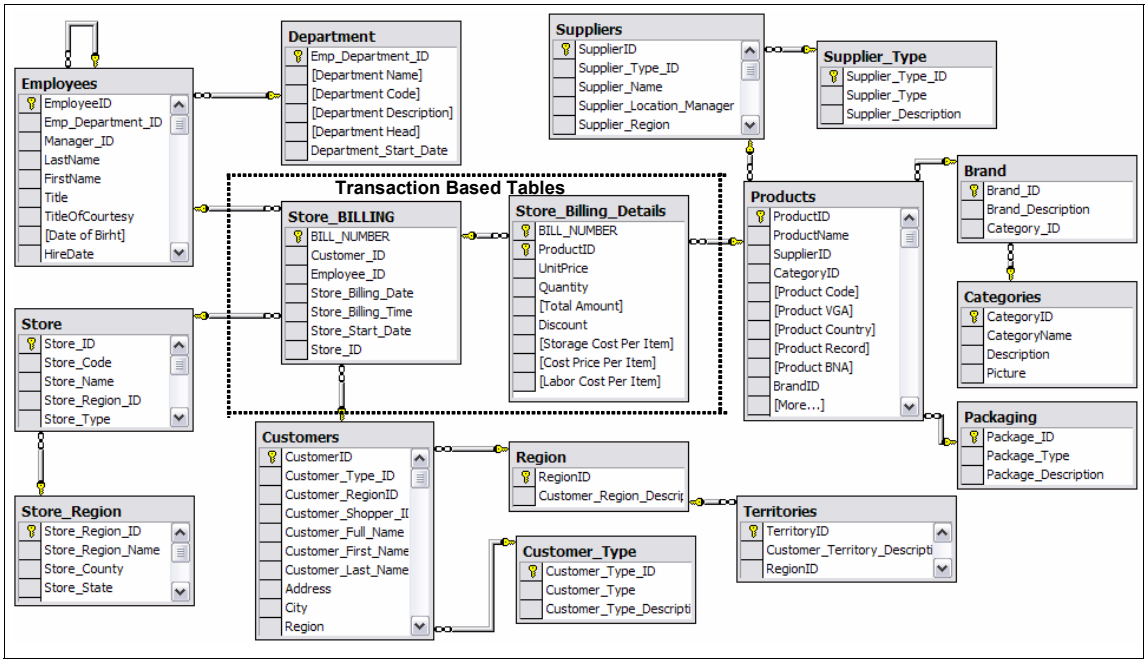


Figure 6-6 Identifying transaction-based tables in the E/R model

The Store_BILLING and Store_Billing_Details tables express a many-to-many relationship that exists between Employee, Store, Customers, and Products. Some of the relationships are explained below:

- Each Employee can sell many products and each Product could be sold by many Employees.
- Each Store could sell many Products and each Product could be sold by many Stores.
- Each Customer could buy many products and each product could be bought by many Customers.
- Each Employee could sell to many Customers and each Customer could purchase from many Employees.

Figure 6-7 on page 218 shows that the Store_BILLING and Store_Billing_Details transaction tables express many-to-many relationships between the different non-transaction tables.

Note: The many-to-many tables in the E/R model are converted to dimensional model fact tables. The many-to-many tables in the E/R model are generally the transaction-oriented tables.

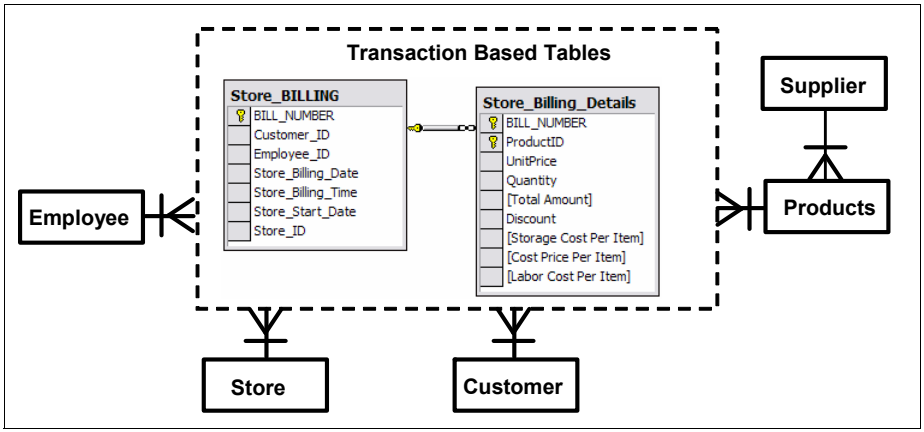


Figure 6-7 Identifying many-to-many relationships in an E/R model

The Store_BILLING and Store_Billing_Details tables are the tables that identify the many-to-many relationship between Employee, Products, Store, Customer, and Supplier tables. The Store_BILLING table stores billing details for an order.

After having identified the many-to-many relationships in the E/R model, we are able to identify the fact table as shown in Figure 6-8.

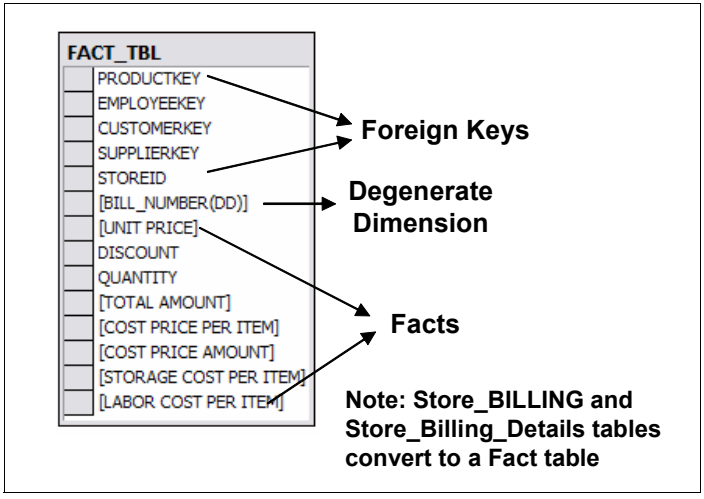


Figure 6-8 Fact table

The fact table design may change depending upon the grain chosen.

3. **Denormalize remaining tables into flat dimension tables:** After we identified the fact table in Step 2, the next step is to take the remaining tables

in the E/R model and denormalize them into dimension tables. The primary key of each of the dimensions is made a surrogate (non-intelligent, integer) key. This surrogate key connects directly to the fact table.

Table 6-2 shows the various E/R tables (see E/R model in Figure 6-6 on page 217) that have been denormalized into dimension tables.

Table 6-2 E/R model to dimension model conversion

Name of tables in E/R model	Corresponding denormalized dimension table	Refer to figure
Customers, Region, Territories, and Customer_Type	Customer	Figure 6-9
Products, Brand, Categories, and Packaging	Product	Figure 6-10
Suppliers and Supplier_Type	Suppliers	Figure 6-11
Employees and Department	Employees	Figure 6-12
Store and Store_Region	Store	Figure 6-13

Figure 6-9 on page 220 shows that the Customers, Region, Territories, and Customer_Type tables in the E/R model are denormalized to form a Customer dimension table. The Customer dimension table has a surrogate key.

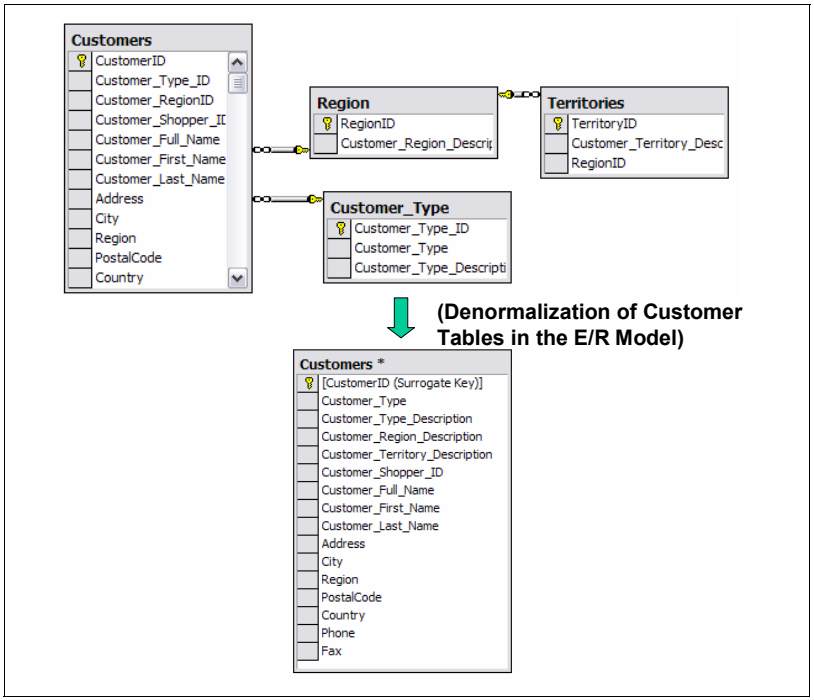


Figure 6-9 Customer dimension table

Figure 6-10 on page 221 shows that the Products, Brand, Categories, and Packaging tables in the E/R model are denormalized to form a product dimension table. The product dimension table has a surrogate key.

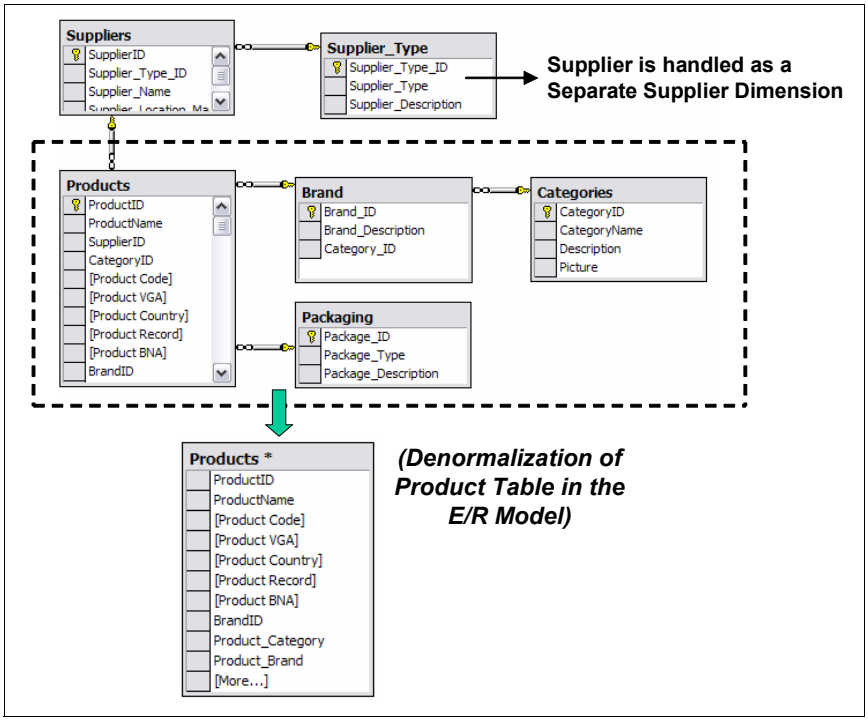


Figure 6-10 Product dimension

Figure 6-11 shows that the Suppliers and Supplier_Type tables in the E/R model are denormalized to form a Supplier dimension table. The Supplier dimension table has a surrogate key.

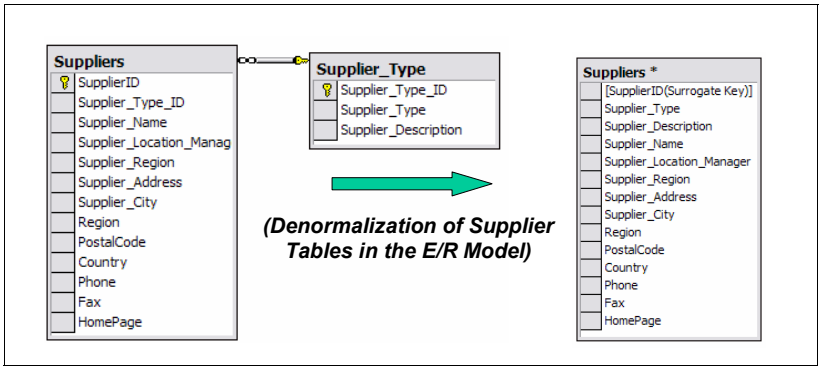


Figure 6-11 Supplier dimension

Figure 6-12 shows that the Employees and Department tables in the E/R model are denormalized to form a Supplier dimension table. The Supplier dimension table has a surrogate key.

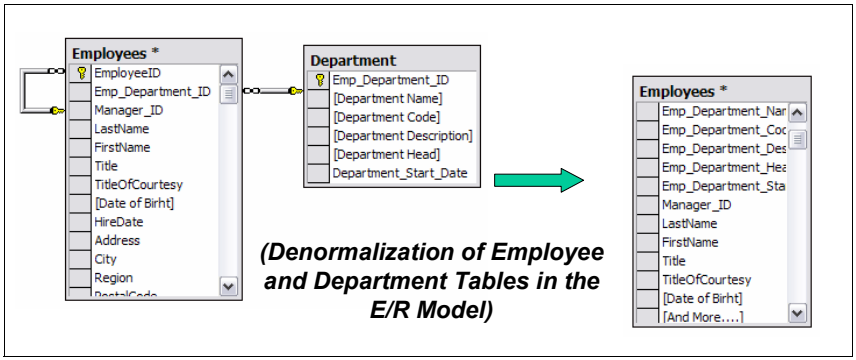


Figure 6-12 Employee dimension

Figure 6-13 shows that the Store and Store_Region tables in the E/R model are denormalized to form a Store dimension table. The Store dimension table has a surrogate key.

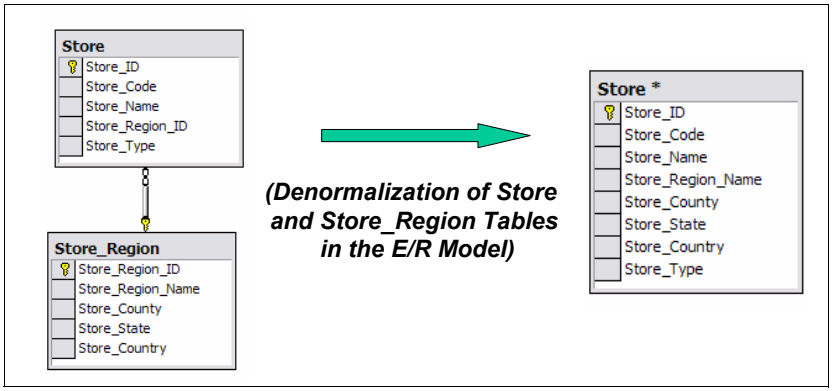


Figure 6-13 Store dimension table

The resulting dimensional model after we normalize the dimension tables is shown in Figure 6-14 on page 223.

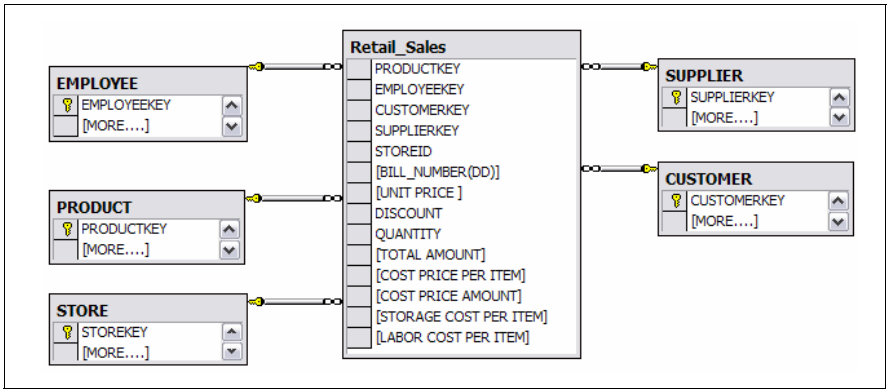


Figure 6-14 Dimensional model after step 3

4. **Add date and time dimensions:** The last step involves identifying the date and time dimension. Dates are generally stored in the form of a date timestamp column inside the E/R model.

   The date and time are stored in the columns called Store_Billing_Date and Store_Billing_Time of the Store_BILLING table of the E/R model as shown in Figure 6-15.

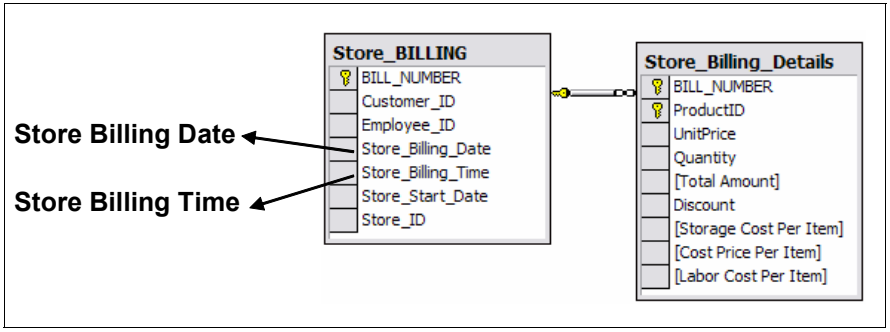


Figure 6-15 Identifying date and time in the E/R model

The E/R models typically have some form of dates associated with them. These dates are generally stored in the transaction-based tables in the form of a date timestamp column. There may also be time associated with the business which is also stored in the form of a date timestamp column present inside the transaction-based tables.

After the date and time columns are identified in the E/R model, they are usually modeled as separate date and time dimensions as shown in Figure 6-16 on page 224.

The final dimensional model

The dimensional model that results from the steps taken is shown in Figure 6-16.

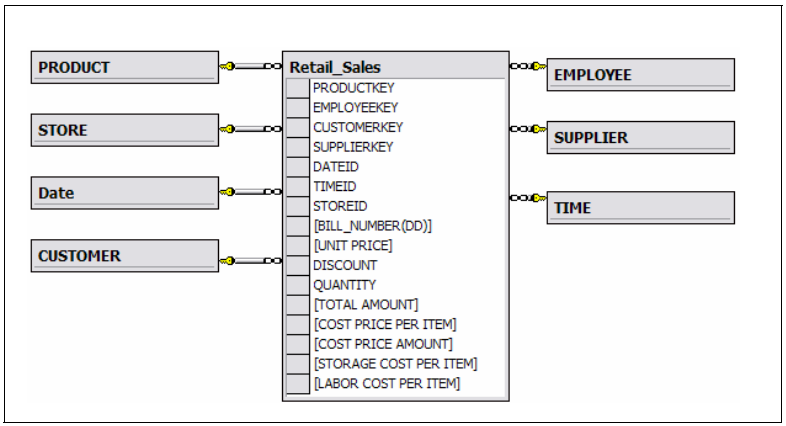


Figure 6-16 The dimensional model

6.2 Identifying the grain for the model

The lowest level of data represented in a fact table is defined as grain. The focus of this section is to discuss the *Identify the grain* component in the DMDL, as depicted in Figure 6-17 on page 225.

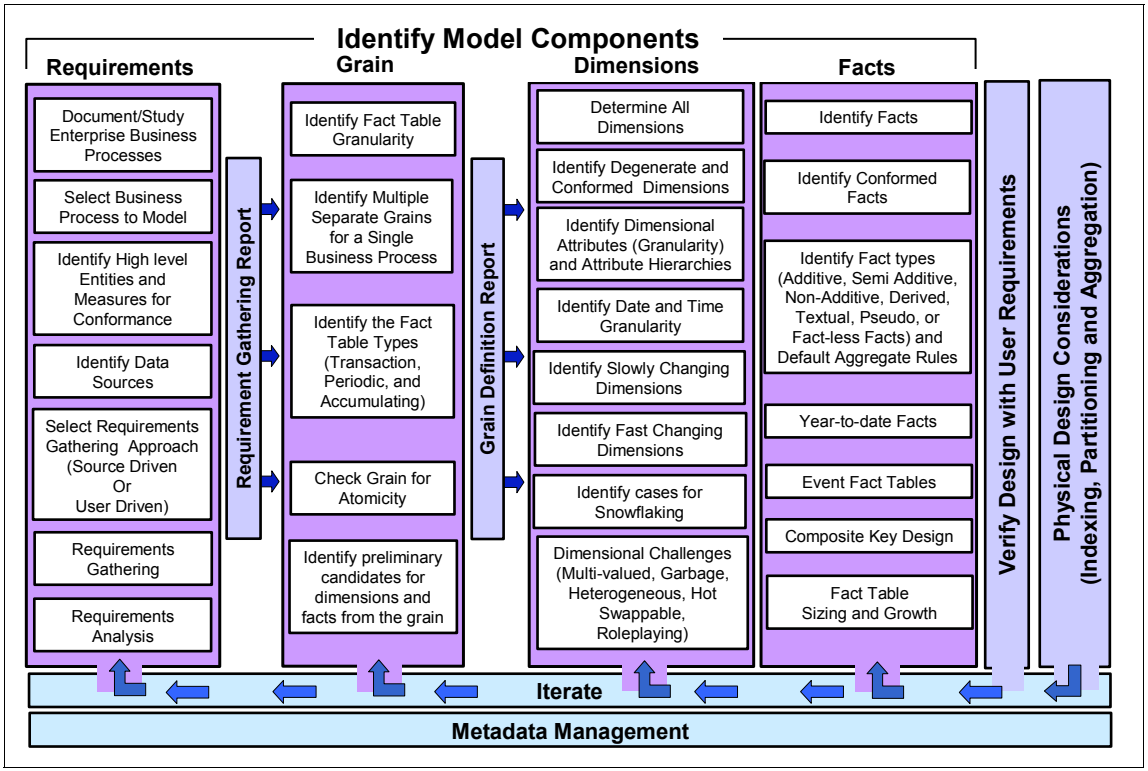


Figure 6-17 Dimensional Model Design Life Cycle

In this section we discuss the importance of having the grain defined at the most detailed, or atomic, level. When data is defined at a very detailed level, the grain is said to be high. When there is less detailed data, the grain is said to be low. For example, for date, a grain of year is a low grain, and a grain of day is a high grain. We also discuss when to consider separate grains for a single business process.

6.2.1 Handling multiple, separate grains for a business process

Typically separate business processes always require separate dimensional models with unique grain definitions. Each business process consists of several facts and dimensions which are different from the other business processes. As discussed in Chapter 5, “Dimensional Model Design Life Cycle” on page 103, we take the following steps to create a dimensional model:

- Identify business process.
- Identify grain.
- Identify dimensions.
- Identify facts.

We explain the concept of multiple fact table grains in the following steps:

1. Business process: Assume that we are creating a dimensional model for the retail sales business of a big clothing merchandise brand which has stores all over the U.S. The business is interested in tracking the sales of the goods from all of its stores. It is also interested in analyzing the reasons for all of its returned clothing products at all stores and in analyzing all suppliers of products based on the percentage of defective returned goods.
2. Identify the grain for the retail sales.

   There are two separate grain definitions for the retail sales business process:

   – For tracking sales in all clothing stores, the grain definition is: One single clothing line item on a bill.

   – For tracking returned clothing goods in all stores, the grain definition is: Every individual clothing item returned by a customer to any store.
3. Identify the dimensions for the different grains.

   – For grain 1 (sales tracking): product, time, customer, date, employee, supplier, and store

   – For grain 2 (returned goods tracking): return date, purchase date, customer, store purchased, products returned, and reasons for return.
4. Identify the facts for the different grains.

   – For grain 1 (sales tracking): unit price, discount, quantity sold, and revenue

   – For grain 2 (returned goods tracking): revenue returned and quantity returned

Note: A single business process may consist of more than one dimensional model. Do not force fit the different facts and dimensions which belong to different dimensional models into a single star schema. We strongly recommend that separate grains are identified for a business process when you are not able to fit facts or dimensions in a single star model.

Figure 6-18 on page 227 shows the dimensional model designed for the sales tracking in the retail sales business process having the grain equivalent to one single line item on a bill.

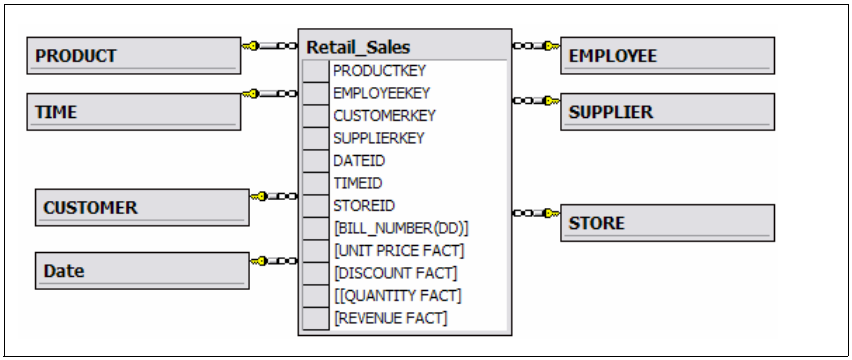


Figure 6-18 Retail sales business star schema for sales tracking

Figure 6-19 shows the dimensional model designed for the retail sales business process having the grain equivalent to every individual item returned by a customer to any store.

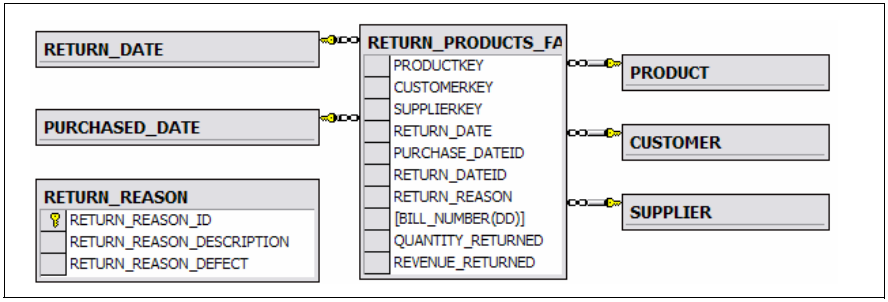


Figure 6-19 Retail Sales Business star schema for tracking returned goods

When to create separate fact tables

When designing the dimensional model for a business process, one or more fact tables can be created. Here are guidelines to consider when deciding to make one or more fact tables for designing the dimensional model for the business process:

- Facts that are not true (valid) to any given grain should not be forced into the dimensional model. Often facts that are not true to a grain definition belong to a separate fact table with its own grain definition.
- Dimensions that are not true (valid) to any given grain should not be forced into the dimensional model. Often such dimensions belong to a separate dimensional model with its own fact table and grain.
- Separate fact tables (dimensional models) should always be created for each unique business process.

6.2.2 Importance of detailed atomic grain

The granularity may be defined as the level of detail made available in the dimensional model. The grain definition is extremely significant from a business, technical, and data mart project standpoint.

It is extremely important that the grain definition is chosen at the most detailed atomic level. The atomic grain is important from three broad perspectives as we discuss below:

- **From a business perspective:**

  From a business perspective, the grain of the fact table dictates whether or not we can easily extend the dimensional model to add new dimensions or facts as, and when, the business requirements change. A dimensional model designed at the lowest level grain (detail) is easy to change. New dimensions or facts can be added to the existing dimensional model without any change to the fact table grain, which means that new business requirements can be delivered from existing dimensional models without the need of much change to an existing one. This is good for businesses whose requirements typically change.

  The dimensional model should be designed at the most detailed atomic level even if the business requires less detailed data. This way the dimensional model has potential future capability and flexibility (regardless of the initial business requirements) to answer questions at a lower level of detail.

  To summarize the importance of grain from a business perspective, we look at an example. Assume your organization wants to analyze customer buying trends by product line and region so that you can develop more effective marketing strategies. Consider the following options:

  – Customer by Product

    The granularity of the fact table always represents the lowest level for each corresponding dimension. When you review the information from the business process, the granularity for customer and product dimensions of the fact table are apparent. Customer and product cannot be reasonably reduced any further. That is, they already express the lowest level of an individual record for the fact table. In some cases, product might be further reduced to the level of product component because a product could be made up of multiple components.

  – Customer by Product by District

    Because the customer buying trends that your organization wants to analyze include a geographical component, you still need to decide the lowest level for region information. The business process indicates that in the past, sales districts were divided by city. But now your organization distinguishes between two regions for the customer base: Region 1 for

California and Region 2 for all other states. Nonetheless, at the lowest level, your organization still includes sales district data, so district represents the lowest level for geographical information and provides a third component to further define the granularity of the fact table.

- Customer by Product by District by Day

  Customer-buying trends always occur over time, so the granularity of the fact table must include a time component. Suppose your organization decides to create reports by week, accounting period, month, quarter, or year. At the lowest level, you probably want to choose a base granularity of day. This granularity allows your business to compare sales on Tuesdays with sales on Fridays, compare sales for the first day of each month, and so forth. The granularity of the fact table is now complete.

  The decision to choose a granularity of day means that each record in the time dimension table represents a day. In terms of the storage requirements, even 10 years of daily data is only about 3,650 records, which is a relatively small dimension table.

- **From a technical perspective:**
  - From a technical perspective, the grain of the fact table has a major impact on the size of the star schema. The more atomic the grain, the bigger the size.
  - The atomic grain results in a huge dimensional model, which can impact the operating cost for performing related tasks such as ETL, as well as the performance.
- **From a project task lists perspective:**
  - The most detailed atomic grain means that the project team needs to represent the data in more detail. This means that the data mart development team will need to understand more E/R or data warehouse-related tables and their corresponding attributes.
  - The more detailed the grain, the more complex related procedures, such as ETL. This means designing more complex procedures and also maintaining more complex meta data.

Note: If the dimensional model is designed with a higher level grain (meanin it is highly aggregated and therefore contains less detailed data), there will be fewer dimensions available for use to perform detailed analyses.

Factors to consider when deciding the grain, are as follows:

- **Current Business Requirements:** The primary factor to consider while deciding the dimensional model grain is the current business requirement. The basic minimum need of the dimensional model grain is to be able to

answer the current business requirements. It is important to remember that the primary purpose for developing a dimensional model is to satisfy a set of business requirements and therefore the grain should be kept at a level of detail which satisfies those requirements.

- **Future Business Requirements:** Another important factor to always consider are the future business requirements. For example, if you design the model to be based on a weekly grain because the business requires only weekly data for its reports, then you may not be able to get reports based on daily data if the business requires daily data in the future. It is important to understand the potential future needs and perhaps design the model at a lower grain than dictated by the present requirements. However, storing the data at a more detailed grain means spending more on maintaining the more atomic grain, but without any present business value. So, it is important that when you design a model with more detailed grain than currently required, you understand the additional overhead costs and prepare satisfactory justification.

How granularity affects the size of the database

The granularity of the fact table also determines how much storage space will be required for the database. For example, consider the following possible granularities for a fact table:

- Product by day by region
- Product by month by region

The size of a database that has a granularity of product by day by region would be much greater than a database with a granularity of product by month by region because the database contains records for every transaction made each day as opposed to a monthly summary of the transactions. You must carefully determine the granularity of your fact table because too fine a granularity could result in a huge database. Conversely, too coarse a granularity could mean the data is not detailed enough for users to perform meaningful queries.

6.2.3 Designing different grains for different fact table types

In the retail sales example, the grain is of the transaction fact table type.

There are three types of fact tables. They are:

- Transaction fact table
- Periodic fact table
- Accumulating fact table

The reason to show different types of fact tables is to emphasize that each typically has different types of grain associated with it. It is important that the

designer be aware of these fact table types so that the designer can use the most appropriate type of the fact tables.

There are also differences in ways the inserts and updates occur inside each of these fact tables. For example, with the transaction and periodic fact tables, only inserts take place and no rows are updated. With the accumulating fact table, the row is first inserted, and then subsequently updated as a milestone is achieved and facts are made available.

We now discuss each of the different fact tables.

Transaction fact table

A transaction-based fact table is a table that records one row per transaction. An example of a transaction-based fact table is shown in Figure 6-20 on page 232. Here the fact table records all transactions that happen for an individual account of a customer of a bank. Assume that a customer named Sherpa makes the following transactions in the month of August 2005 against his bank account:

- Money Withdrawn: $400, Date: August 2, 2005, Time: 4:00AM
- Money Deposited: $300, Date: August 4, 2005, Time: 3:00AM
- Money Withdrawn: $600, Date: August 5, 2005, Time: 2:00PM
- Money Withdrawn: $900, Date: August 6, 2005, Time: 9:00PM
- Money Deposited: $900, Date: August 18, 2005, Time: 7:00AM
- Money Deposited: $800, Date: August 23, 2005, Time: 1:00AM

Figure 6-20 on page 232 shows that in a transaction fact table, a single row is inserted for each bank account transaction (deposit or withdrawal) that Sherpa makes. Important features about the transaction fact table are:

- A single row is inserted for each transaction.
- Typically, the date and time dimensions are represented at the lowest level of detail. For example, the date dimension may be represented at the day level and the time dimension may be represented at the hour or minute level.
- The transaction fact table is known to grow very fast as the number of transactions increases.

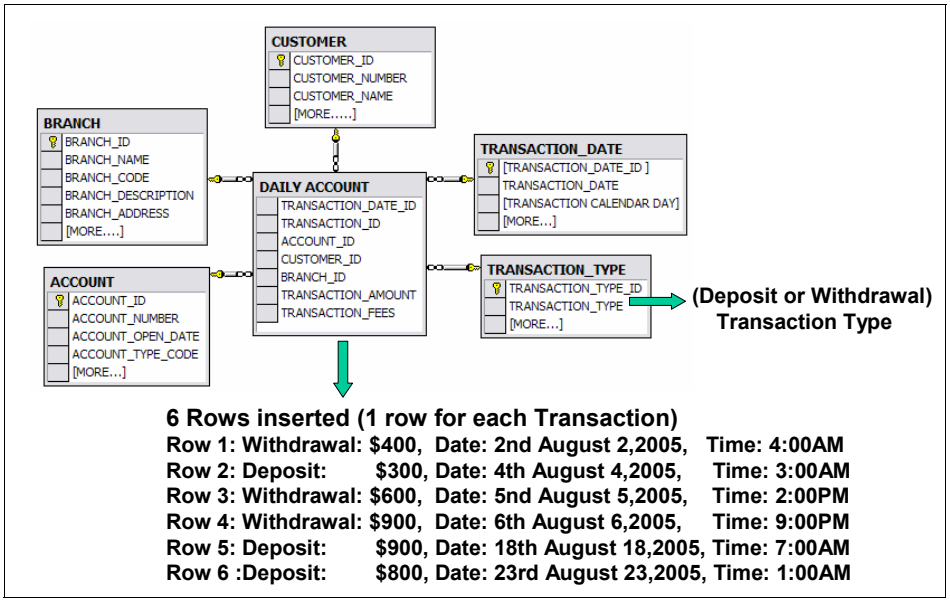


Figure 6-20 Transaction fact table

Periodic fact table

A periodic fact table stores one row for a group of transactions made over a period of time.

An example of a periodic-based fact table is shown in Figure 6-21 on page 233 where the fact table records a single row per month for all transactions against an individual account.

Assume that a customer named Sherpa makes the following transactions in the month of August 2005:

- Money Withdrawn: $400, Date: August 2, 2005, Time: 4:00AM
- Money Deposited: $300, Date: August 4, 2005, Time: 3:00AM
- Money Withdrawn: $600, Date: August 5, 2005, Time: 2:00PM
- Money Withdrawn: $900, Date: August 6, 2005, Time: 9:00PM
- Money Deposited: $900, Date: August 18, 2005, Time: 7:00AM
- Money Deposited: $800, Date: August 23, 2005, Time: 1:00AM

In the transaction-based fact table, we stored six rows for the six transactions shown above. However, for the periodic-based fact table, we consider a time period (end of day, week, month, or quarter) for which we need to store the transactions as a whole.

Figure 6-21 shows a periodic fact table. The grain of the fact table is chosen as the account balance per customer at the end of each month. Observe that we do not record each of the transaction dates separately, but instead use the Month dimension.

Note: It is important to understand that when we design a periodic fact table, certain dimensions are not defined when compared to a transaction fact table type. For example, when comparing the transaction fact table and periodic fact table, we see that certain dimensions such as Transaction_Type, Branch, and Transaction_Date are not applicable at the Periodic (Monthly) grain.

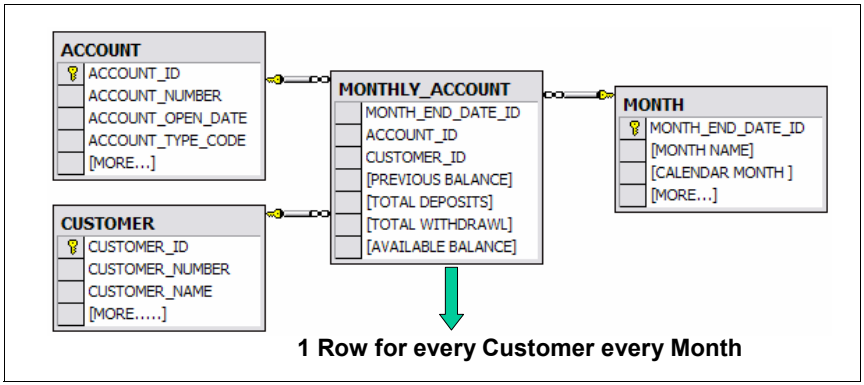


Figure 6-21 Periodic fact table

The periodic fact table design consists of the Month table (Monthly grain). The periodic fact table MONTHLY_ACCOUNT consists of facts, such as Previous_Balance, Total_Deposit, Total_Withdrawal, and Available_Balance, which are true at the periodic month-end grain.

Important features of the periodic fact table are:

- A single row is inserted for each set of activities over a period of time.
- Typically, the date and time dimensions are represented at the higher level of detail. For example, the date dimension may be represented at the month level (instead of day) and the time dimension may be represented at the hour level (instead of seconds or minutes).
- The periodic fact table is known to grow comparatively slowly in comparison to the transaction fact table.

Accumulating fact table

An accumulating fact table stores one row for the entire lifetime of an event. For example, from the lifetime of a credit card application being sent to the time it is

accepted. Another example could be the lifetime of a job or college application being sent to the time it is accepted or rejected by the college or the job posting company.

To understand the concept of an accumulating fact table, consider that a big recruitment company advertises vacancies in many jobs relating to software, hardware, networking, apparel, marketing, sales, food, carpentry, plumbing, housing, house repairs, mechanical, teaching high school, teaching college, senior management, and working in restaurants. About 100 000 vacancies are advertised, in all major newspapers every month. The recruitment company senior management wants to better understand how efficiently their recruitment staff works in matching potential job candidates with the jobs they seek. The senior management wants to understand how long it takes for a prospective candidate to get a job from the time the resume is sent for a particular job vacancy.

The accumulating fact table is shown in Figure 6-22, where the fact table records a single row per job vacancy advertised by the recruitment company.

Note: Accumulating fact tables are typically used for short-lived processes and not constant event-based processes, such as bank transactions.

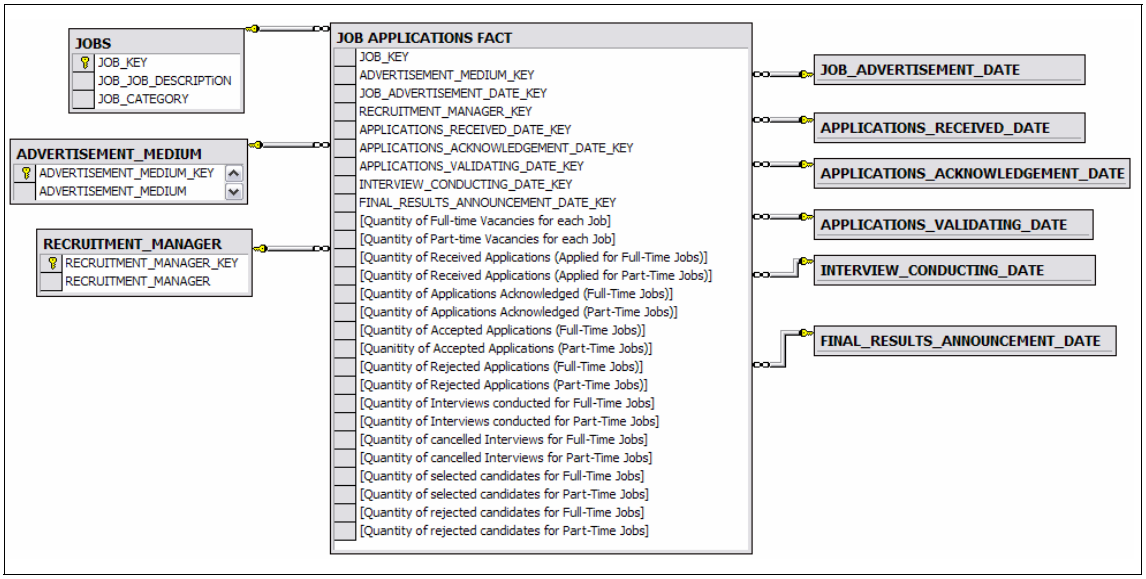


Figure 6-22 Accumulating fact table

It is important to understand that there are several dates involved in the entire job application process. The dates are defined in sequential order in Table 6-3 on page 236. Also after each subsequent date, certain facts become available. This is also shown graphically in Figure 6-23.

Note: There are several dates associated with accumulating fact tables. Each of these dates may be implemented with a single date table using views. The date table, when implemented using views for different dates, is said to have been involved in *Role-playing*. We will discuss Role-playing in more detail in 6.3.9, “Role-playing dimensions” on page 285.

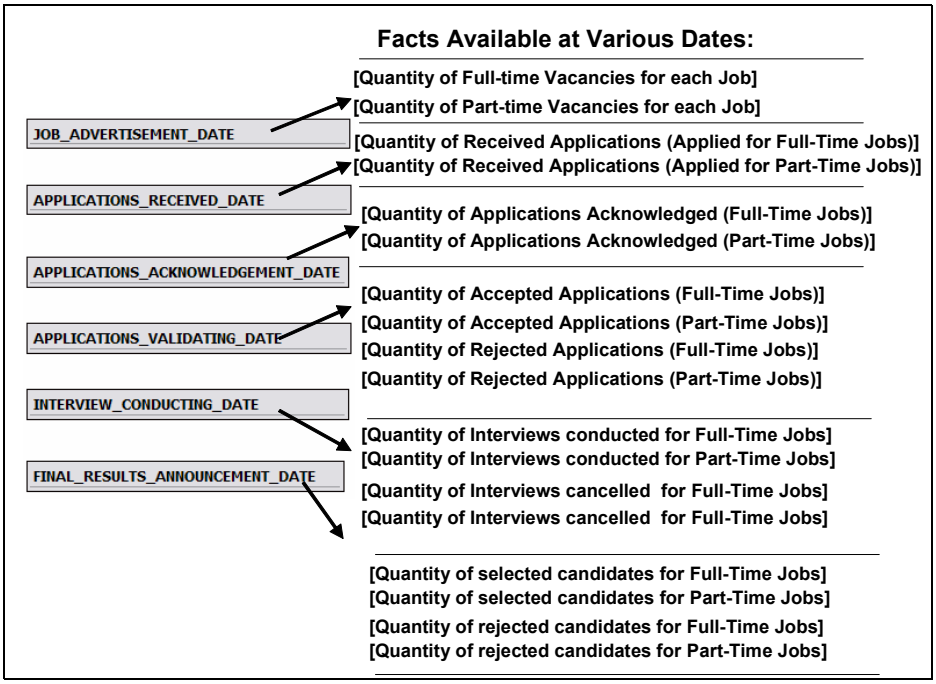


Figure 6-23 Facts associated with each date milestone in accounts fact table

Table 6-3 on page 236 shows the various dates associated with the accumulating fact table and the various facts that are made available on each date.

Table 6-3 Activities defined to understand the concept of accumulating facts

Dates	Available facts at each corresponding date
Job advertisement date	The recruitment company advertises several jobs for several companies. **Important facts available during this date are:** ► Number of full-time vacancies for each job ► Number of part-time vacancies for each job
Applications received date	The application applied date is the date at which different candidates send the applications for the job vacancies. **Important facts available during this date are:** ► Number of received applications (for full-time jobs) ► Number of received applications (for part-time jobs)
Applications acknowledgement date	The application acknowledgement date is the date when each candidate who sent a job application is acknowledged (by e-mail/phone) that their application has been received and is being processed. **Important facts available during this date are:** ► Number of applications acknowledged (for full-time jobs) ► Number of applications acknowledged (for part-time jobs)
Applications validating date	The application validating date is the date on which the applications are validated as matching the required prerequisite requirements for each job. Any application that does not meet prerequisite job requirements is rejected. **Important facts available during this date are:** ► Number of accepted applications (for full-time jobs) ► Number of accepted applications (for part-time jobs) ► Number of rejected applications (for full-time jobs) ► Number of rejected applications (for part-time jobs)
Interview conducting date	The interview date is the date on which candidates are interviewed for the job vacancies. **Important facts available during this date are:** ► Number of interviews conducted for full-time jobs ► Number of interviews conducted for part-time jobs ► Number of cancelled Interviews for full-time jobs ► Number of cancelled interviews for part-time jobs

Dates	Available facts at each corresponding date
Final results announcement date	The final results announcement date is the date on which announcement of the selected candidates is made. **Important facts available during this date are:** ► Number of selected candidates for full-time jobs ► Number of selected candidates for part-time jobs ► Number of rejected candidates for full-time jobs ► Number of rejected candidates for part-time jobs

Comparison between fact tables

Table 6-4 shows a comparison between the various types of fact tables.

Table 6-4 Comparison of fact table types

Feature	Transaction type fact table	Periodic type fact table	Accumulating type fact table
Grain definition of the fact table	One row per transaction. For example one row per line item of a grocery bill.	One row per period. For example, one row per month for a single product sold in a grocery store.	One row for the entire lifetime of an event. For example, the lifetime of a credit card application being sent to the time it is accepted.
Dimensions	Involves date dimension at the lowest granularity.	Involves date dimension at the end-of-period granularity.This could be end of day, end-of week, end-of month, or end-of quarter.	This type of fact table involves multiple date dimensions to show the achievement of different milestones.

Feature	Transaction type fact table	Periodic type fact table	Accumulating type fact table
Total number of dimensions involved	More than periodic fact type.	Less than transaction fact type.	Highest number of dimensions when compared to other fact table types. Generally this type of fact table is associated with several date dimension tables which are based on a single date dimension implemented using a concept of role-playing. This is discussed in 6.3.9, “Role-playing dimensions” on page 285.
Conformed Dimensions	Uses shared conformed dimensions.	Uses shared conformed dimensions.	Uses shared conformed dimensions.
Facts	Facts are related to transaction activities.	Facts are related to periodic activities. For example, inventory amount at end of day or week.	Facts are related to activities which have a definite lifetime. For example, the lifetime of a college application being sent to the time it is accepted by the college.
Conformed Facts	Uses shared conformed facts.	Uses shared conformed facts.	Uses shared conformed dimensions.
Database size	Transaction-based fact tables have the biggest size. If the grain of the transaction is chosen at the most detailed level, these tables tend to grow very fast.	The size of a Periodic fact table is smaller than the Transaction fact table because the grain of the date and time dimension is significantly higher than lower level date and time dimensions present in the transaction fact table.	Accumulating fact tables are the smallest in size when compared to the Transaction and Periodic fact tables.

Feature	Transaction type fact table	Periodic type fact table	Accumulating type fact table
Performance	Performance is typically good. However, the performance improves if you chose a grain above the most detailed because the number of rows decreases.	Performance for Periodic fact table is higher than other fact table types because data is stored at lesser detailed grain and therefore this table has fewer rows.	Performance is typically good. The select statements often require differences between two dates to see the time period in days/weeks/months between any two or more activities.
Insert	Yes	Yes	Yes
Update	No	No	Yes. Only when a milestone is reached for a particular activity.
Delete	No	No	No
Fact table growth	Very fast.	Slow in comparison to transaction-based fact table.	Slow in comparison to the transaction and periodic fact table.
Need for aggregate tables	High need (This is primarily because the data is stored at a very detailed level.)	None or very few (This is primarily because the data is already stored at a highly aggregated level.)	Medium need (This is primarily because the data is stored mostly at the day level. However, the data in accumulating fact tables is less than the transaction level.)

6.3 Identifying the model dimensions

In this section, we discuss issues relating to dimensions. We discussed the Dimensional Model Design Life Cycle in detail in Chapter 5, “Dimensional Model Design Life Cycle” on page 103. Here we focus on the *Identify dimensions* phase of the DMDL, as shown in Figure 6-24 on page 240.

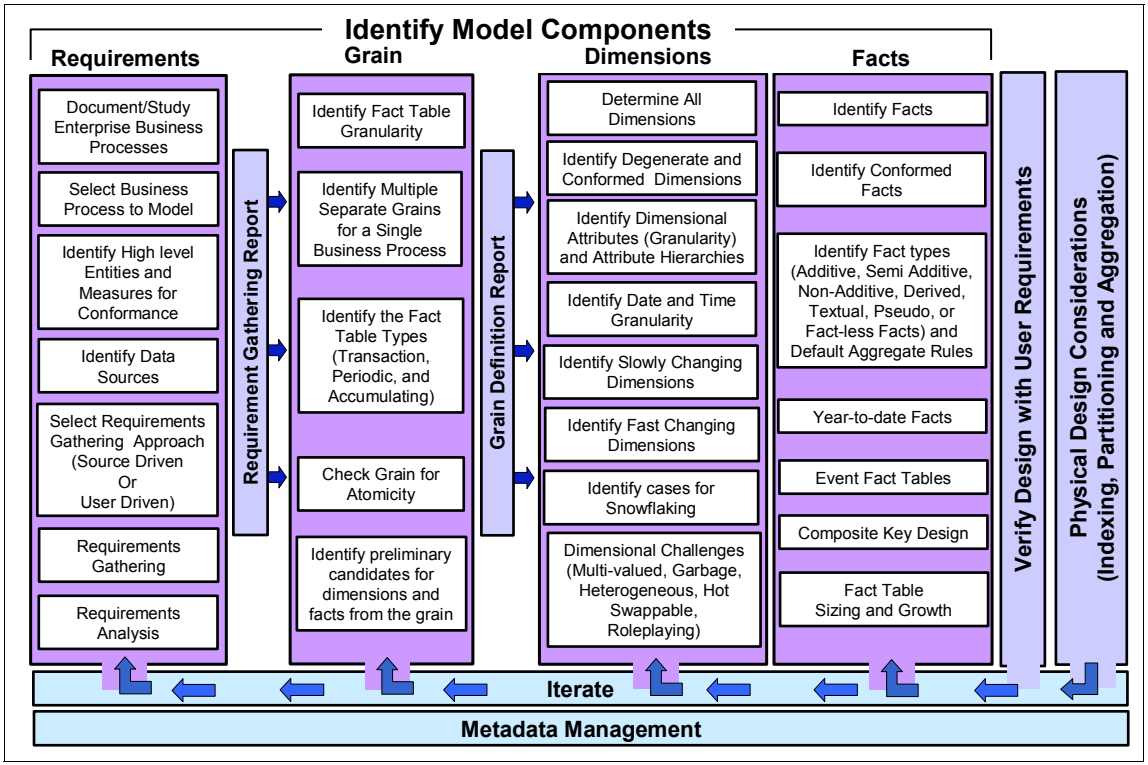


Figure 6-24 Dimensional Model Design Life Cycle

6.3.1 Degenerate dimensions

Before we discuss degenerate dimensions in detail, it is important to understand the following:

A fact table may consist of the following data:

- Foreign keys to dimension tables
- Facts which may be:
  - Additive
  - Semi-additive
  - Non-additive
  - Pseudo facts (such as 1 and 0 in case of attendance tracking)
  - Textual fact (rarely the case)
  - Derived facts
  - year-to-date facts
- Degenerate dimensions (one or more)

What is a degenerate dimension?

A degenerate dimension sounds a bit strange, but it is a dimension without attributes. It is a transaction-based number which resides in the fact table. There may be more than one degenerate dimension inside a fact table.

How to identify a degenerate dimension

All OLTP source systems typically consist of transaction numbers, such as bill numbers, courier tracking numbers, order numbers, invoice numbers, application received acknowledgements, and ticket numbers. These transaction numbers in the OLTP system generally define the transaction.

Consider the grocery store example from Chapter 5, “Dimensional Model Design Life Cycle” on page 103. The dimensional design has a transaction number, such as the Bill Number# shown in Figure 6-25. This Bill Number# represents several line items shown on the graphical bill. Now assume that we choose the grain to be an individual line item on a grocery store bill.

A fact table row is a line item on the bill. And a line item is a fact table row. As shown in Figure 6-25, we have identified the dimensions that describe each line item. The dimensions identified are date, time, product, employee, store, and customer. Other dimensions, such as supplier, are not visible on the graphical bill but are true to the grain.

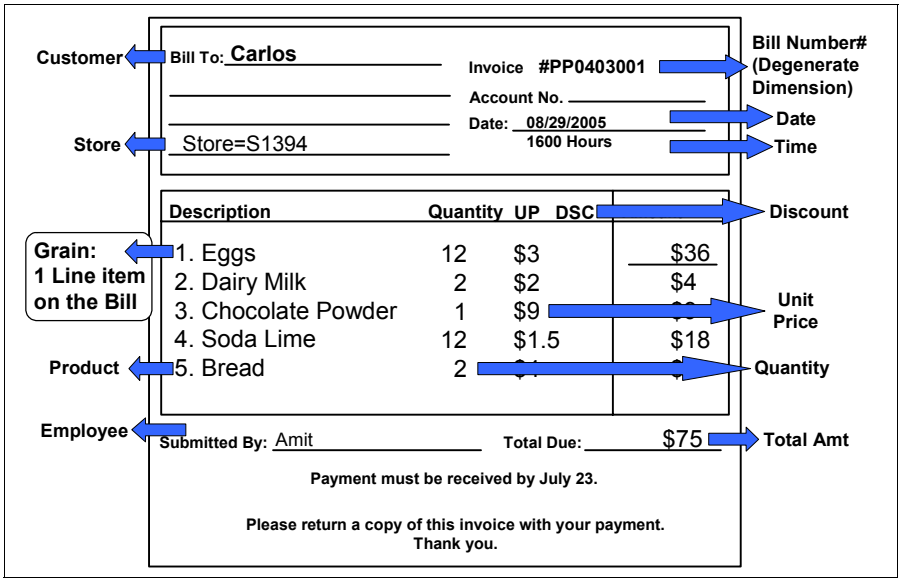


Figure 6-25 Grocery store bill

The next question typically is, "*what do we do with the Bill Number#?"*
We certainly cannot discard it, because the grain definition we chose is a single line item on a bill.

Should we make a separate dimension for the Bill Number#?

To see whether or not we should make a separate dimension, try to analyze the Bill Number# information for the grocery store example. The Bill Number# is a transaction number that tells us about our purchase made at the store. If we take an old Bill Number# 973276 to the store and ask the manager to find out information relating to Bill Number# 973276, we may get all information relating to the bill. For example, assume that the manager replies that Bill Number# 973276 was generated on August 11, 2005. The items purchased were apples, orange, and chocolates. The manager also tells us the quantity, unit price, and discount for each of the items purchased. He also tells us about the total price. In short, the Bill Number # tells us about the following information:

- Transaction date
- Transaction time
- Products purchased
- Store from which bill was generated
- Customer to which the merchandise was sold
- Quantity, unit price, and amount for each purchased product

The important point to note is that we have already extracted all information relating to the Bill Number# into other dimensions, such as date, time, store, customer, and product. Information relating to quantity, unit price, and amount charged is inside the fact table. It would be correct to say that all information that the Bill Number# represents is stored in all other dimensions. Therefore, Bill Number# dimension has no attributes of its own; and therefore it cannot be made a separate dimension.

Should we place the Bill Number# inside the fact table?

The Bill Number# should be placed inside the fact table right after the dimensional foreign keys and right before the numeric facts as shown in Figure 6-26 on page 243.

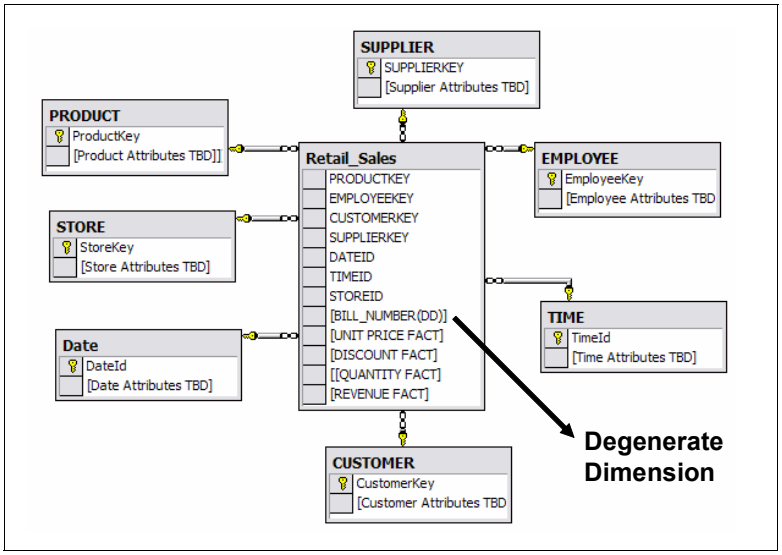


Figure 6-26 Degenerate dimension

The Bill Number#, in SQL language, serves as a GROUP BY clause for grouping together all the products purchased in a single transaction or for a single Bill Number#. Although to some, the Bill Number# looks like a dimension key in the fact table. But, it is not. This is because all information (such as date of purchase and products purchased) relating to the Bill Number# has been allocated to different dimensions.

Note: A degenerate dimension, such as Bill Number#, is there because we chose the grain to be an individual line item on a grocery store bill. In other words, the Bill Number# degenerate dimension is there because the grain we chose represents a single transaction or transaction line item.

How to identify that a degenerate dimension is missing

The best way to identify a missing or a badly designed degenerate dimension is to review your dimensional design and look for any dimension table that has equal or nearly the same number of rows as the fact table. In other words, if, for every row that you insert in the fact table you also have to pre-insert another row in any other dimension table, then you have missed a degenerate dimension.

Let us review the dimensional model shown in Figure 6-27 on page 244. The grain for this dimensional model has been chosen to be an individual line item on a grocery store bill.

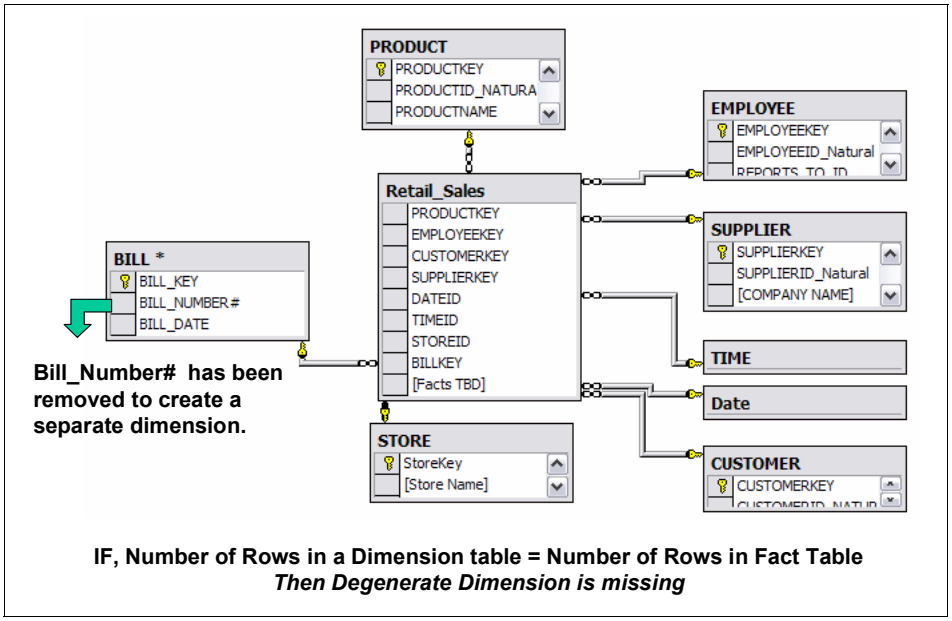


Figure 6-27 Missing degenerate dimension (Bad Design)

Note: Dimension tables are static when compared to fact tables, which grow constantly in size as sales are made on a daily basis and the activity is recorded inside the fact table. In a dimensional design, if the size of a dimension table remains equal or nearly equal to that of a fact table, then a degenerate dimension has been missed. In other words, if for every fact table row to be inserted, correspondingly, an equal or near equal number of rows have to be inserted into a dimension table, then there is a design flaw because a degenerate dimension has been wrongly represented into a new dimension of its own.

In Figure 6-27, we observe that for every purchase a customer makes, an equal number of rows are inserted into the dimension and fact tables. This is because the Bill dimension table has the Bill_Number which is different for each bill. The Bill dimension is not a static dimension like other dimensions.

When will the Bill_Number# no longer be a degenerate dimension?

- When certain data columns belong to the Bill_Number# itself and do not fall into any other dimensions (such as Date, Time, Product, Customer, Supplier, and Employee), then Bill_Number# would no longer be a degenerate dimension. An example of this is explained in a case study in “Identify degenerate dimensions” on page 384. We see that Invoice Number (type of

transaction number) is handled as a separate dimension and not as a degenerate dimension inside the fact table.

This decision however needs to carefully be made by studying the Bill_Number#. If you decide to create a new separate dimension for the Bill_Number#, then you must make sure that it will not contain equal or near equal rows with the fact table. If it does, then there has probably been a mistake.

Note: OLTP transaction numbers, such as bill numbers, courier tracking #, order number, invoice number, application received acknowledgement, and ticket number, usually produce dimensions without any attributes and are represented as degenerate dimensions in the fact table.

6.3.2 Handling time as a dimension or a fact

A date dimension typically represents day, week, month, quarter, or year, where a time dimension represents hours, minutes, and seconds within a day. These are two physically separate dimension structures within the data warehouse, each with their own surrogate keys used to join the fact tables. Because of the unique characteristics of these two dimensions, special considerations can be made when building them. It is not advised to merge the date and time dimension into one table because a simple date dimension table which has 365 rows (for 1 year) would explode into 365 (Days) x 24 (Hours) x 60 (Minutes) x 60 (Seconds) = 31 536 000 rows if we tried storing hours, minutes, and seconds. This is for just 1 year. If we had 10 years of data in our date table, we would have 10 x 365 = 3 650 rows (assuming no leap year). If we now tried to store data at the second, minute, and hour level, we would expand the simple date table from 3 650 rows (for 10 years) to about 3 650 x 24 x 60 x 60 = 315 360 000 rows.

Having now made a point to handle the date and time dimensions separately, we can discuss how to implement the time dimension.

Time in dimensional modeling

Unlike the date dimension, the time-of-day (hour, minute, and second) may be well expressed as a simple numerical fact rather than as a separate time dimension, unless there are textual descriptions of certain periods within the day that are meaningful, such as early morning, late morning, noon, lunch hour, afternoon, evening, night, and late night shift.

- Time of day as a dimension:

  Time is expressed as a dimension when the business needs to understand the sales of its product over a time period which has meaningful textual names such as:

– Early morning (6 a.m. to 8 a.m.)
– Late morning hours (8 a.m. to 11 a.m.)
– Rush hour (11 a.m. to 1 p.m.)
– Lunch hour (1 p.m. to 2 p.m.)

For expressing such scenarios that describe the time, we handle time as a dimension. This is shown in Figure 6-28. The grain of this time dimension is an hour.

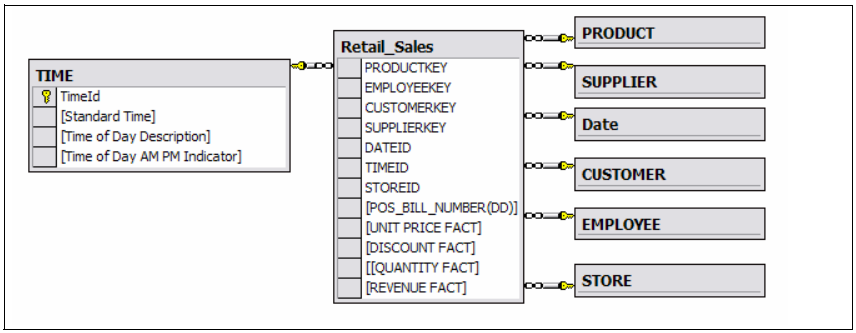


Figure 6-28 Handling time as a dimension

In other words, handling time as a dimension is the correct approach if we need to support the roll-up of time periods into more summarized groupings for reporting and analysis, such as 30-minute intervals, hours, or a.m./p.m. (see Table 6-5 which shows sample time dimension rows) or for business-specific time groupings, such as the week day morning rush period, week day early mornings, late night shifts, and late evenings.

Table 6-5 Time dimension with sample rows

Timeid	Standard time	Time of day description	Time of day a.m. p.m. indicator
1	0100 hours	Late night shift	a.m.
2	0800 hours	Morning	a.m.
3	1100 hours	Rush hour	a.m.
4	2000 hours	Night	p.m.

The other reason you may express time as a dimension is when you want to represent different hierarchies for the time that you are measuring. Some of the time hierarchies the business may want are listed as follows:

– Standard Time hierarchy (Hour → Minute → Second)
– Military Time hierarchy (Hour → Minute → Second)

We have seen in Figure 5-21 on page 157 that date dimension can have multiple hierarchies, such as fiscal and calendar hierarchy inside the same dimension table.

- Time of day as a fact

  We express time as a fact inside the fact table if there is no need to roll up or filter on time-of-day groups, such as morning hour, rush hour, and late night shift. Then in this scenario we have the option of treating time as a simple numeric fact which is stored in the fact table. Here the time of day is expressed as a date-time stamp such as “09-25-2005 10:10:36”.

  Figure 6-29 shows time (TIME_OF_SELLING) expressed as a fact inside the fact table. The TIME_OF_SELLING stores the time as a date-time stamp which includes the precise time in hours, minutes, and seconds at which the product sells inside the store.

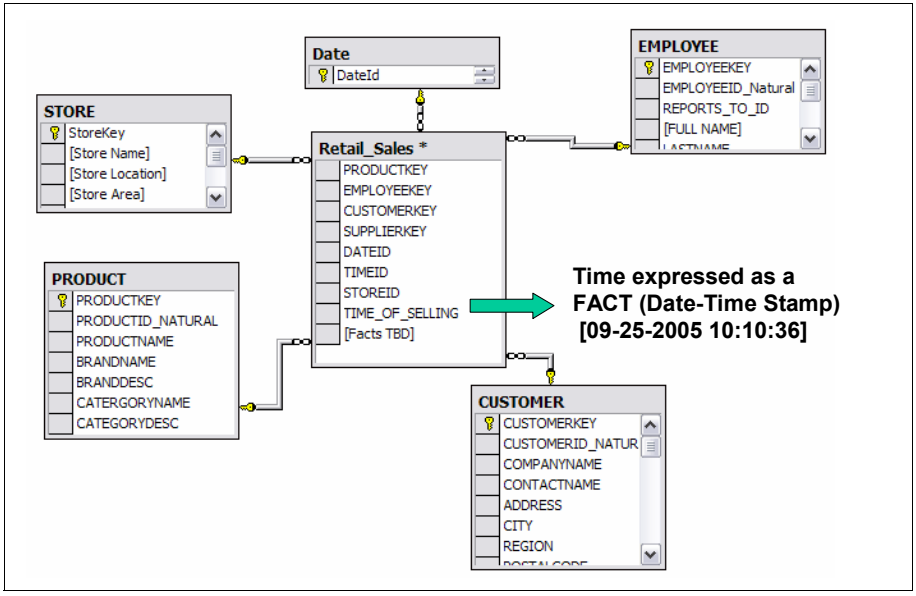


Figure 6-29 Time expressed as a fact

It is important to understand that it is very difficult to generate reports which require roll-ups for business-specific time groupings, such as the week day morning rush period, week day early mornings, late night shifts, and late evenings, using the TIME_OF_SELLING (see Figure 6-29) which is stored as fact.

If we had stored time as a dimension, then we would be able to generate reports to support the roll-up of time periods into more summarized groupings for reporting and analysis, such as 30-minute intervals, hours, or a.m./p.m.

(see Table 6-5 on page 246, which shows sample time dimension rows) or for business-specific time groupings, such as the week day morning rush period, week day early mornings, late night shifts, and late evenings.

6.3.3 Handling date and time across international time zones

In businesses that span multiple time zones, both the date and time of day may need to be expressed both in local time and Greenwich Mean Time (GMT). This must be done with separate date dimensions for local and GMT, and separate time-of-day facts as well. This is shown in Figure 6-30.

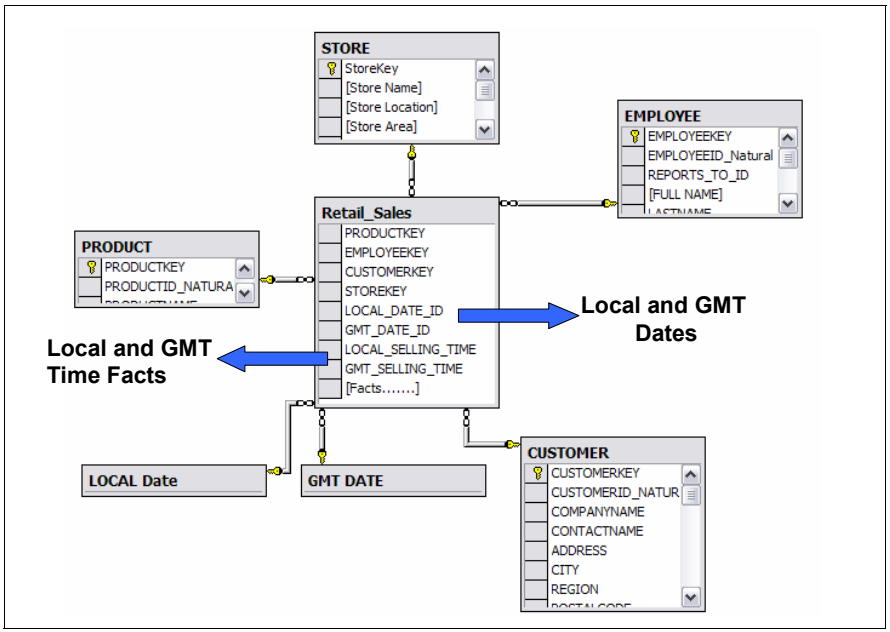


Figure 6-30 Handling International Time Zones

The separate date dimensions are implemented using views, or a concept called *role-playing*. We discuss role-playing in 6.3.9, “Role-playing dimensions” on page 285.

6.3.4 Handling dimension hierarchies

A *hierarchy* is a cascaded series of many-to-one relationships and consists of different levels. Each level in an hierarchy corresponds to a dimension attribute. Hierarchies document the relationships between levels in a dimension. For example, a region hierarchy is defined with the levels Region, State, and City.

In other words, a hierarchy is a specification of levels that represents a relationship between different attributes within a hierarchy. Figure 6-31 shows sample hierarchies.

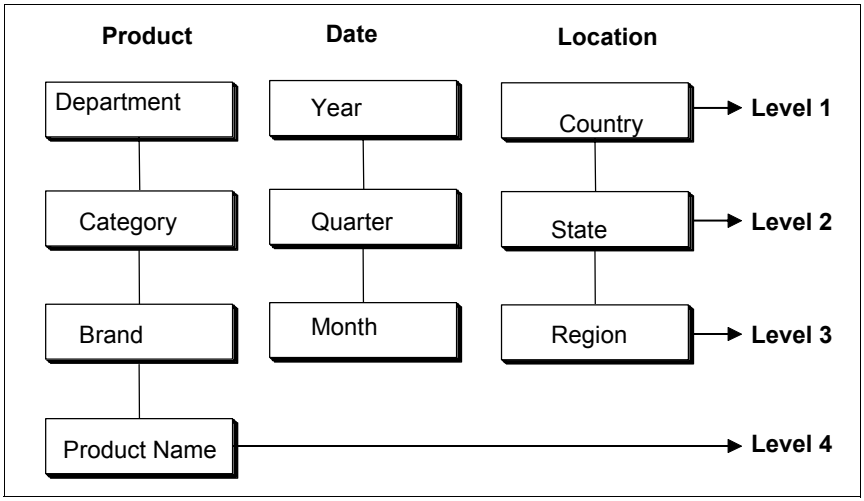


Figure 6-31 Dimension hierarchies

Hierarchies enrich the semantics of data in a dimensional model and correspondingly improve the class of interesting and meaningful queries that can be run against it.

Note: In order to generate data for a report, we can drill down or up on attributes from more than one explicit hierarchy and with attributes that are part of no hierarchy.

We now discuss implementing three types of hierarchies:

- Balanced hierarchy
- Unbalanced hierarchy
- Ragged hierarchy

Balanced hierarchy

A balanced hierarchy is one in which all of the dimension branches have the same number of levels. In other words, the branches have a consistent depth.

The logical parent of a level is directly above it. A balanced hierarchy can represent a date where the meaning and depth of each level, such as Year, Quarter, and Month, are consistent. They are consistent because each level represents the same type of information, and each level is logically equivalent.

Figure 6-32 shows an example of a balanced time hierarchy.

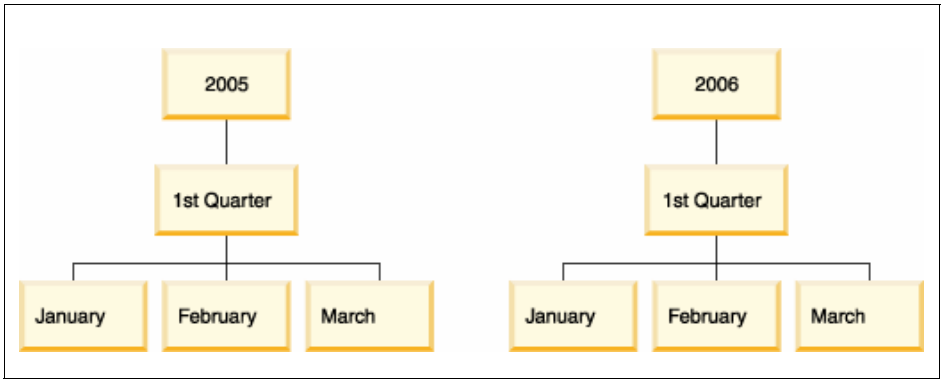


Figure 6-32 Balanced hierarchy

How to implement a balanced hierarchy

A balanced hierarchy consists of a fixed number of levels. Assume that we are designing a date dimension consisting of the hierarchy shown in Figure 6-32. We design the date dimension as shown Figure 6-33. You should have as many fixed attributes inside the dimension table as the number of levels in the hierarchy. This is the number of roll-up levels that you want to track.

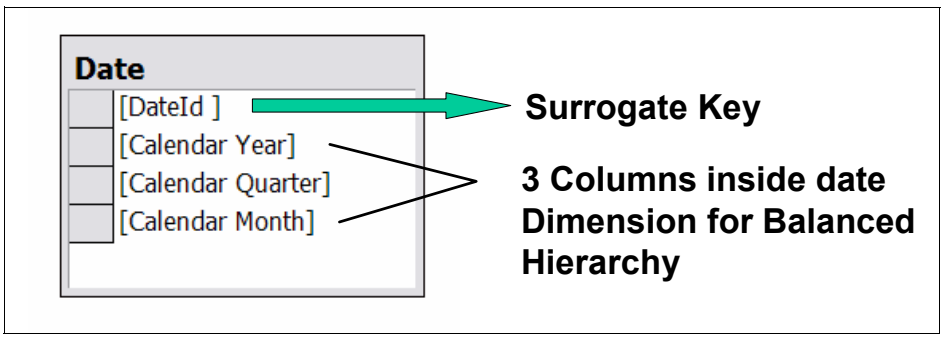


Figure 6-33 Balanced hierarchy for the Date dimension

The date dimension can also have multiple balanced hierarchies in the same dimension table, as shown in Figure 6-34 on page 251.

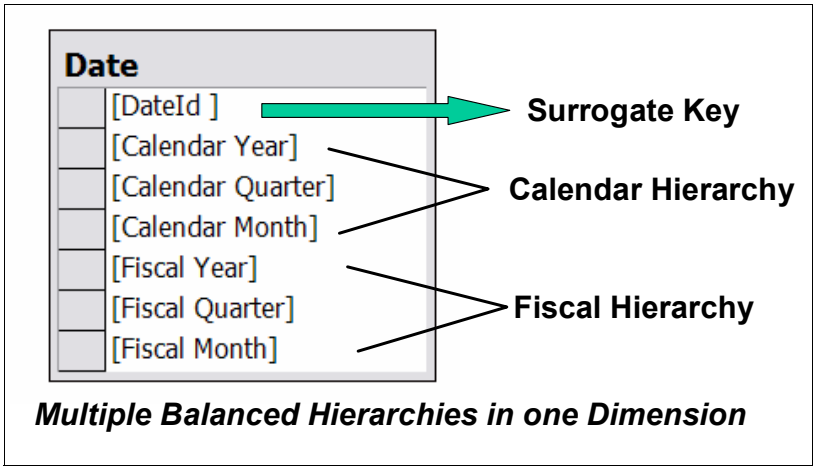


Figure 6-34 Multiple balanced hierarchies in date dimension

Unbalanced hierarchy

A hierarchy is unbalanced if it has dimension branches containing varying numbers of levels. Parent-child dimensions support unbalanced hierarchies.

An unbalanced hierarchy has levels that have a consistent parent-child relationship, but have a logically inconsistent level. The hierarchy branches also can have inconsistent depths. An unbalanced hierarchy can represent an organization chart. For example, Figure 6-35 on page 252 shows a chief executive officer (CEO) on the top level of the hierarchy and two people that branch off below. Those two are the COO and the executive secretary. The COO hierarchy has additional people, but the executive secretary does not. The parent-child relationships on both branches of the hierarchy are consistent. However, the levels of both branches are not logical equivalents. For example, an executive secretary is not the logical equivalent of a chief operating officer.

We can report by any level of the date hierarchy and see the results across year, quarter, or month.

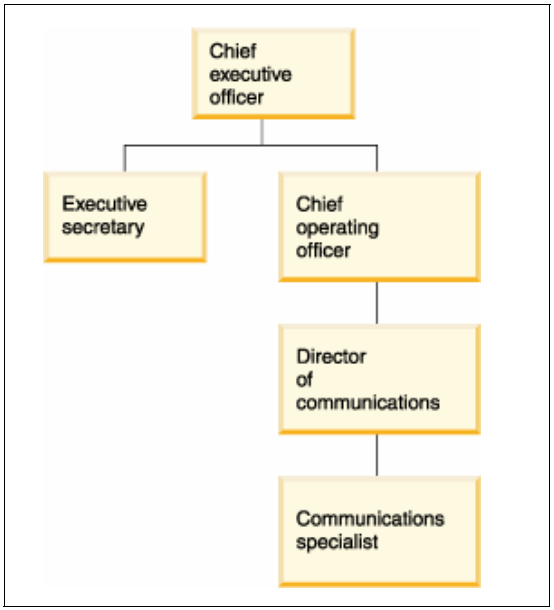


Figure 6-35 Unbalanced hierarchy

How to implement an unbalanced hierarchy

It is important to understand that representing an unbalanced hierarchy is an inherently difficult task in a relational environment. OLAP Cube-based systems are better suited for reporting when the data has unbalanced hierarchies. However, if your hierarchies are symmetrical then arguably either type of technology (OLAP cubes or relational database) is equally capable of providing the answer.

A common example of an unbalanced hierarchy is an organization chart. Employees at a given level may have hundreds of employees, while others at the same level may have none or a few. So, as an example, if you wish to create a report that computes the sales totals for all employees at a given level, you can create this report more efficiently with an OLAP reporting system. For an example of an unbalanced hierarchy, see Figure 6-36 on page 253.

Assume, in this hierarchy, that we want to report the sales for a set of managers. Each box in the figure represents an employee in the organization, and there are are four levels. A large organization with thousands of employee can have many levels. It is important to understand that each employee can play the role of a parent or a child. The report may require summarizing the sales made by one employee or summarizing all sales for one manager (and all people in that management chain). This is equivalent to summarizing the sales revenue to any node in the organizational tree shown in Figure 6-36 on page 253.

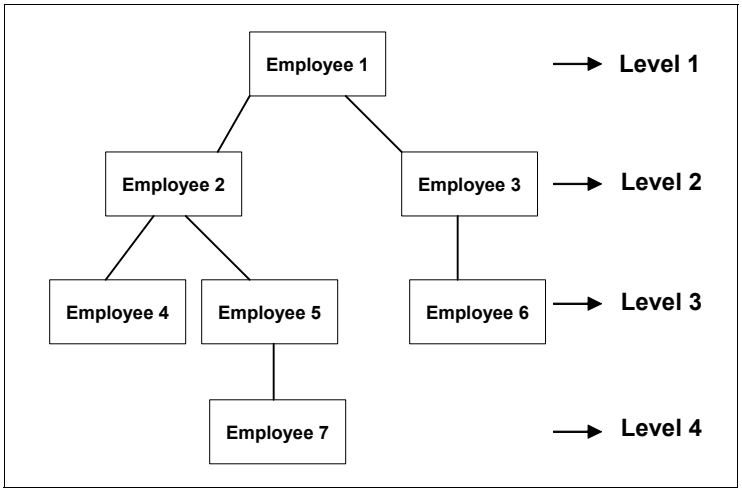


Figure 6-36 Unbalanced hierarchies for an organization chart

The general approach for a parent-child relationship in a relational database is to introduce a child key in the parent table. This is called a *recursive pointer*, and is shown in Figure 6-37.

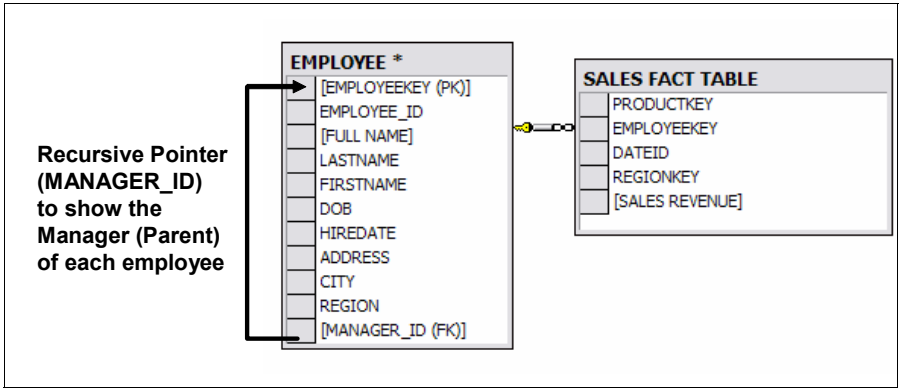


Figure 6-37 Recursive pointer to show parent-child relationship

Issues using a recursive pointer for unbalanced hierarchy

The recursive pointer approach (shown in Figure 6-37) does implement the unbalanced hierarchy (shown in Figure 6-36). However, this approach does not work well at all with SQL language. Why?

Assume that we want a report which shows the total sales revenue at node Employee 2. What this means is that we want the report showing the total sales made by Employee 2, Employee 4, Employee 5, and Employee 7.

We cannot use SQL to answer this question because the GROUP BY function in SQL cannot be used to follow the recursive tree structure downward to summarize an additive fact such as sales revenue in the SALES_FACT table. Because of this, we cannot connect a recursive dimension to any fact table.

The next question is, how do we solve the query using an SQL Group by clause? And, how will we be able to summarize the sales revenue at any Employee node (from Figure 6-36 on page 253).

The answer is to use a bridge table between the EMPLOYEE and SALES_FACT table as shown in Figure 6-38. The bridge table is called a BRIDGE_Traverse_Hierarchy. The aim of the bridge table is to help us traverse through the unbalanced hierarchy depicted in Figure 6-36 on page 253. The bridge table helps us to analyze the sales revenue (SALES_REVENUE in Figure 6-37 on page 253) at any hierarchical level using the SQL language, which was not possible with a
simple recursive pointer as shown in Figure 6-37 on page 253.

Note: The beauty of using a bridge table to solve the problem of traversing an unbalanced hierarchy is that neither the Employee dimension nor the Sales_FACT table changes in any way. The use of a bridge table is optional, and may be used only when you need to traverse through an unbalanced hierarchy as shown in Figure 6-36.

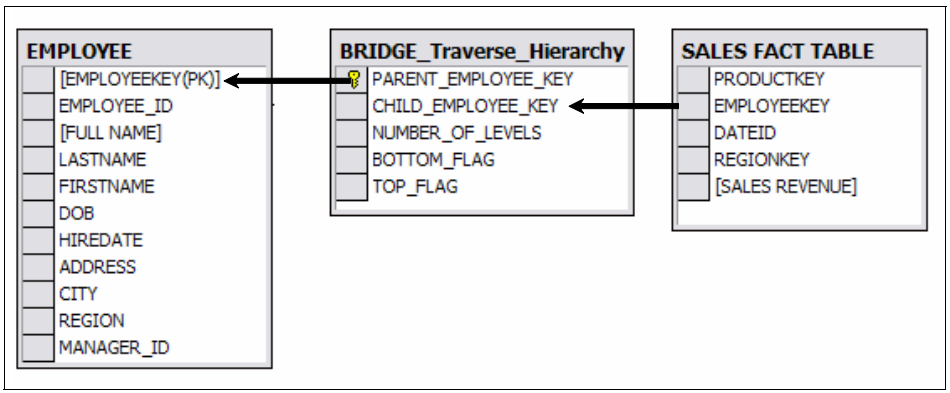


Figure 6-38 Use of a bridge table for descending the unbalanced hierarchy

What does the bridge table contain?

- The Employee dimension table has one row for each employee entity at any level of the hierarchy.
- The bridge table contains a single row for every path as shown in Figure 6-36 on page 253. In other words, the BRIDGE_Traverse_Hierarchy table contains

one row for each pathway from an employee entity to each immediate employee beneath it, as well as a row for the zero-length pathway from an employee to itself.

Note: A zero-length pathway in an unbalanced hierarchy is a pathway to itself. For example, in Figure 6-36, for the Employee 1, the zero-length pathway would signify the distance between Employee 1 and itself.

The BRIDGE_Traverse_Hierarchy table also contains the following other information:

- NUMBER_OF_LEVELS column: This is the number of levels between the parent and child.
- BOTTOM_FLAG: This flag tells us that the Employee node is the bottom most and does not have any employees working under them.
- TOP_FLAG: This flag tells us that the employee is at the top of the organizational hierarchy and does not have any employees working above them.

Table 6-6 shows sample rows for the unbalanced hierarchy in Figure 6-38 on page 254.

Table 6-6 Sample bridge table rows

PARENT_ EMPLOYEE _KEY	CHILD_EMPLOYEE _KEY	NUMBER_OF _LEVELS	BOTTOM_ FLAG	TOP_FLAG
1	1	0	No	Yes
1	2	1	No	No
1	3	1	No	No
1	4	2	Yes	No
1	5	2	No	No
1	6	2	Yes	No
1	7	3	Yes	No
2	2	0	No	No
2	4	1	Yes	No
2	5	1	No	No
2	7	2	Yes	No

PARENT_ EMPLOYEE _KEY	CHILD_EMPLOYEE _KEY	NUMBER_OF _LEVELS	BOTTOM_ FLAG	TOP_FLAG
3	3	0	No	No
3	6	1	Yes	No
4	4	0	Yes	No
5	5	0	No	No
5	7	1	Yes	No
6	6	0	Yes	No
7	7	0	Yes	No

How many rows does the bridge table contain?

To get the answer to this question, we need to analyze the unbalanced hierarchy shown in Figure 6-36 on page 253. The total number of rows in the bridge table are equal to the total number of paths available inside the unbalanced hierarchy. The following are the paths:

1. Employee 1 to Employee 1 (Zero-length pathway to itself)
2. Employee 1 to Employee 2
3. Employee 1 to Employee 3
4. Employee 1 to Employee 4
5. Employee 1 to Employee 5
6. Employee 1 to Employee 6
7. Employee 1 to Employee 7
8. Employee 2 to Employee 2 (Zero-length pathway to itself)
9. Employee 2 to Employee 4
10. Employee 2 to Employee 5
11. Employee 2 to Employee 7
12. Employee 3 to Employee 3 (Zero-length pathway to itself)
13. Employee 3 to Employee 6
14. Employee 4 to Employee 4 (Zero-length pathway to itself)
15. Employee 5 to Employee 5 (Zero-length pathway to itself)
16. Employee 5 to Employee 7
17. Employee 6 to Employee 6 (Zero-length pathway to itself)
18. Employee 7to Employee 7 (Zero-length pathway to itself)

Therefore, the total number of rows in the bridge table named (BRIDGE_Traverse_Hierarchy) is 18. This can be verified from Table 6-6 on page 255.

Now suppose that we want to find the total sales revenue during January 2005 for Employee 2 (Figure 6-36 on page 253) and all its junior employees (Employee 4, Employee 5, and Employee 7).

In order to descend the hierarchy from Employee 2, we make use of Figure 6-38 on page 254. Example 6-1 shows sample SQL necessary to generate the report:

Example 6-1 Sample SQL to generate the report for descending the hierarchy

```
Select E.Full_Name, SUM(F.Sales_Revenue)
From
Employee E, Bridge_Traverse_Hierarchy B, Sales_Fact F, Date D
where
E.EmployeeKey= B.Parent_Employee_Key
and
B.Child_Employee_Key= F.EmployeeKey
and
F.DateID= D.DateID
and
E.[FULL NAME]= 'Employee 2'
// and D.Date= January 2005 Data (Date SQL Code logic depends on Database)
GROUP BY E.Full_Name
```

Now suppose that we want to find out the total sales revenue during January 2005 for Employee 6 (Figure 6-36 on page 253) and all its senior employees (Employee 3 and Employee 1). In order to ascend the hierarchy for Employee 6, we make use of a bridge table, as depicted in Figure 6-39.

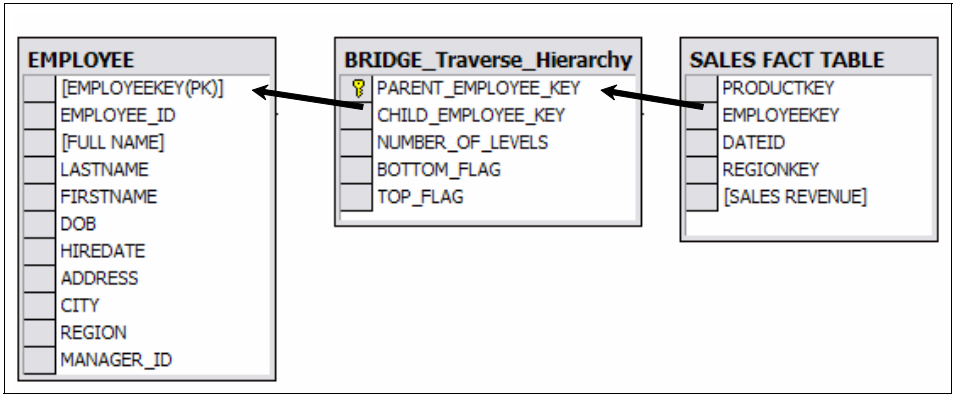


Figure 6-39 Use of the bridge table to ascend the unbalanced hierarchy

Example 6-2 shows sample SQL necessary to generate the report.

Example 6-2 Sample SQL to generate the report for ascending the hierarchy

```
Select E.Full_Name, SUM(F.Sales_Revenue)
From
Employee E, Bridge_Traverse_Hierarchy B, Sales_Fact F, Date D
where
E.EmployeeKey= B.Child_Employee_Key
and
B.Parent_Employee_Key= F.EmployeeKey
and
F.DateID= D.DateID
and
E.[FULL NAME]= ‘Employee 6’
GROUP BY E.Full_Name
```

Joining a bridge table

The bridge table can be joined between the Employee dimension table and fact table in two ways. These are shown in Figure 6-40 on page 259. The two ways are explained as follows:

- For descending the hierarchy

  The fact table joins to the child employee key of the bridge table and the employee dimension table joins to the parent employee key of the bridge table. Descending means selecting an employee and moving down the hierarchy. We can use the NUMBER_OF_LEVELS column to determine how deep to go. If the BOTTOM_FLAG=’Yes’, we know that we have reached the bottom most part of the unbalanced hierarchy.

- For ascending the hierarchy

  The fact table joins to the parent employee key of the bridge table and the employee dimension table joins to the child employee key of the bridge table. The NUMBER_OF_LEVELS column helps us go up the number of levels desired. The TOP_FLAG helps us determine if we have reached the top of the hierarchy at the CEO level.

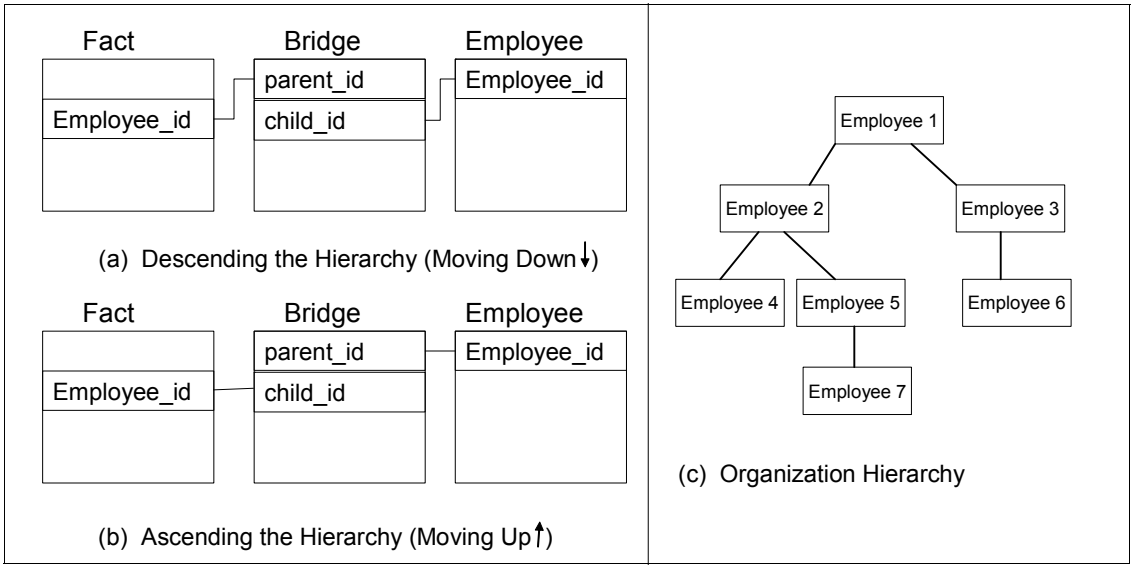


Figure 6-40 Ways of joining a bridge table to fact and dimension tables

Summary: Using a bridge table to navigate an unbalanced hierarchy

There are two join directions that you may use:

- Navigate up the hierarchy
  - Fact joins to subsidiary customer key
  - Dimension joins to parent customer key
- Navigate down the hierarchy
  - Fact joins to parent customer key
  - Dimension joins to subsidiary customer key

Disadvantages of the bridge table approach

- The bridge table data is complex to maintain.
- The bridge table design is too complex to be used directly by the users.

In what scenarios is the bridge table approach used?

A bridge table is used in scenarios where there is a business need to report against an unbalanced organizational hierarchy such as the one discussed in Figure 6-36 on page 253 and Figure 6-40 (c). This approach is typically used in electronic and manufacturing-related industries dealing with a huge number of parts that belong to a higher level assembly.

Ragged hierarchy

A ragged dimension contains at least one member whose parent belongs to a hierarchy that is more than one level above the child (see Figure 6-41). Ragged dimensions, therefore, contain branches with varying depths.

A ragged hierarchy is one in which each level has a consistent meaning, but the branches have inconsistent depths because at least one member attribute in a branch level is unpopulated.

A ragged hierarchy can represent a geographic hierarchy in which the meaning of each level, such as city or country, is used consistently, but the depth of the hierarchy varies. Figure 6-41 shows a geographic hierarchy that has Continent, Country, Province/State, and City levels defined. One branch has North America as the Continent, United States as the Country, California as the Province or State, and San Francisco as the City.

However, the hierarchy becomes ragged when one member does not have an entry at all of the levels. For example, another branch has Europe as the Continent, Greece as the Country, and Athens as the City, but has no entry for the Province or State level because this level is not applicable in this example. In this example, the Greece and United States branches descend to different depths, creating a ragged hierarchy.

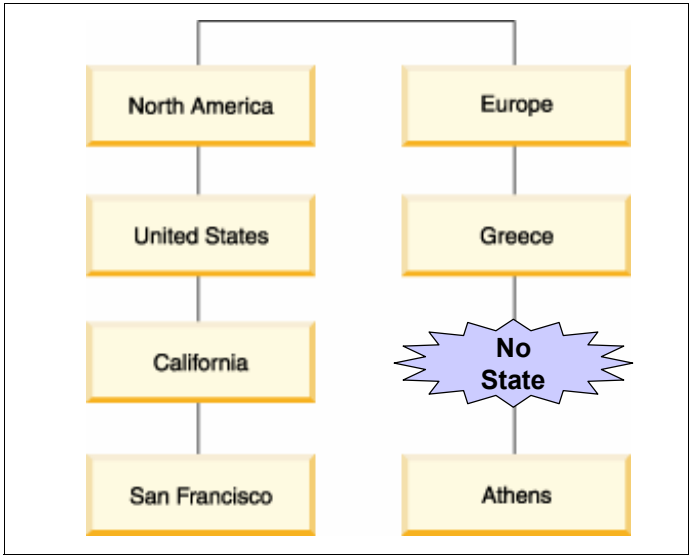


Figure 6-41 Ragged hierarchy

How to implement a ragged hierarchy in dimensions

A ragged hierarchy as shown in Figure 6-41 on page 260 can be implemented in a dimension table as shown in Figure 6-42.

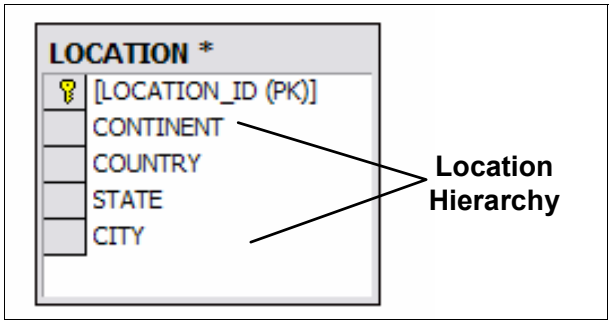


Figure 6-42 Ragged hierarchy implementation in a dimension table

The hierarchy is shown as Continent → Country → State → City. It is possible that attributes such as State will not be populated for Countries, such as Greece, that have no states. In this scenario, we can populate a dummy value of “Not Applicable” or “No States”, rather than null.

6.3.5 Slowly changing dimensions

Handling changes to dimensional data across time can be difficult, and dimensional attributes rarely remain static. A customer address can change, sales representatives come and go, and companies introduce new products to replace older ones.

Changing data in dimension tables can present far-ranging implications when you view the changes over time. For example, assume a company sells car-related accessories.The company decides to reassign a sales territory to a new sales representative. How can you record the change without making it appear that the new sales representative has always held that territory? If a customer name or married status changes, how can you record the change and preserve the old and new version of the name and married status? Designing a dimensional model that accurately and efficiently handles changes is a critical consideration when building a data warehouse. After all, the main reason for building a warehouse is to preserve history.

What are slowly changing dimensions?

A slowly changing dimension is a dimension whose attribute or attributes for a record (row) change slowly over time.

In a dimensional model, the dimension table attributes are not fixed. They typically change slowly over a period of time, but can also change rapidly. The dimensional modeling design team must involve the business users to help them determine a change handling strategy to capture the changed dimensional attributes. This describes what to do when a dimensional attribute changes in the source system. A change handling strategy involves using a surrogate (substitute) key as its primary key for the dimension table.

We now present a few examples of slowly changing dimension attribute change-handling strategies using the star schema shown in Figure 6-43. We also discuss the advantages and disadvantages of each approach.

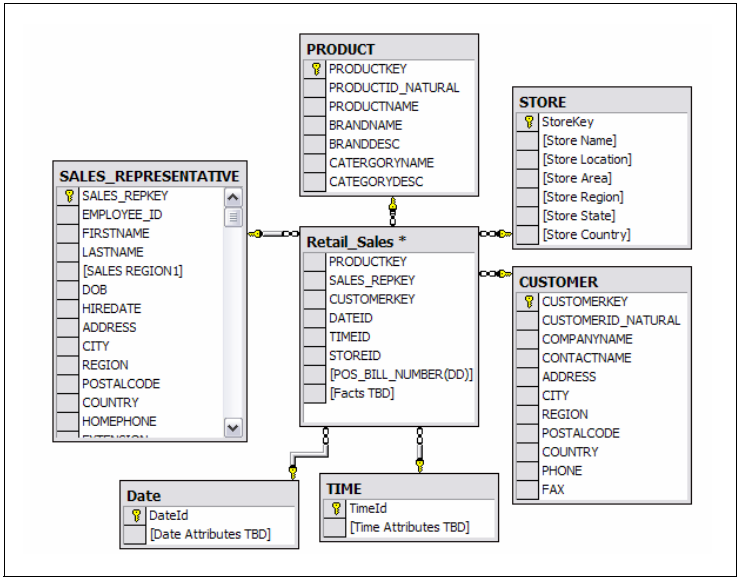


Figure 6-43 Star schema for discussing slowly changing dimensions

The following are approaches for handling slowly changing dimensions:

Type-1 (overwriting the history)

A Type-1 approach overwrites the existing dimensional attribute with new data, and therefore no history is preserved. Now consider the star schema shown in Figure 6-43. Assume that the correct first name of a sales representative is Monika, but is mistakenly written in the table as Monnica, as shown in Table 6-7.

Table 6-7 Sales_representative table

SALES_REPKEY	LASTNAME	FIRSTNAME	SALESREGION
963276	Agarwal	Monnica	New York

When we use a Type-1 approach, we overwrite the old (incorrect) value (Monnica) shown in Table 6-7 on page 262, with the new correct value of Monika, as shown in Table 6-8.

Table 6-8 Sales_representative table (corrected data)

SALES_REPKEY	LASTNAME	FIRSTNAME	SALESREGION
963276	Agarwal	Monika	New York

Updating the FIRSTNAME field once and seeing the change across all fact rows is effective and makes good business sense if the update corrects a misspelling, such as the first name, for example. But suppose the sales representative now correctly named Monika is moved to a new sales region in California. Updating the sales representative dimensional row would update all previous facts so that the sales representative would appear to have always worked in the new sales region, which certainly was not the case. If you want to preserve an accurate history of who was managing which sales region, a Type-1 change might work for the Sales FIRSTNAME field correction, but not for the “SALESREGION” field.

Note: It is very important to involve the business users when implementing the Type-1 approach. Although it is the easiest approach to implement, it may not be appropriate for all slowly changing dimensions since it can disable their tracking business history in certain situations.

The Type-1 approach is summarized in Table 6-9.

Table 6-9 Review of Type-1 approach

Type-1 approach	Description
When to use the Type-1 change handling approach	► This may be the best approach to use if the attribute change is simple, such as a correction in spelling. And, if the old value was wrong, it may not be critical that history is not maintained. ► It is also appropriate if the business does not need to track changes for specific attributes of a particular dimension.
Advantages of the Type-1 change handling approach	► It is the easiest and most simple to implement. ► It is extremely effective in those situations requiring the correction of bad data. ► No change is needed to the structure of the dimension table.

Type-1 approach	Description
Disadvantages of the Type-1 change handling approach	► All history may be lost if this approach is used inappropriately. It is typically not possible to trace history. ► All previously made aggregated tables need to be rebuilt.
Impact on existing dimension table structure	► No impact. The table structure does not change.
Impact on preexisting aggregations	► Any preexisting aggregations based on the old attribute value will need to be rebuilt. For example, as shown in Table 6-8, when correcting the spelling of the FIRSTNAME of Monika (correct name), any preexisting aggregations based on the incorrect value Monnica will need to be rebuilt.
Impact on database size	► No impact on database size.

Type-2 (preserving the history)

A Type-2 approach adds a new dimension row for the changed attribute, and therefore preserves history. This approach accurately partitions history across time more efficiently than other change handling approaches. However, because the Type-2 approach adds new records for every attribute change, it can significantly increase the database's size.

Now consider the star schema shown in Figure 6-43 on page 262. Assume that the sales representative named Monika works for a sales region in California, as shown in Table 6-10. After having worked for fifteen years in California, the region is changed to New Jersey.

Table 6-10 Sales representative table data

SALES_REPKEY	LASTNAME	FIRSTNAME	SALESREGION
963276	Agarwal	Monika	California

With a Type-2 approach, you do not need to make structural changes to the Sales_representative dimension table. But you insert a new row to reflect the region change. After implementing the Type-2 approach, two records appear as depicted in Table 6-11 on page 265. Each record is related to appropriate facts in the fact table shown in Figure 6-43 on page 262, which are related to specific points in time in the time dimension and specific dates of the date dimension.

Table 6-11 Type-2 approach for the sales_representative dimension

SALES_REPKEY	LASTNAME	FIRSTNAME	SALESREGION
963276	Agarwal	Monika	California
963277	Agarwal	Monika	New York

This example illustrates two supplementary but important dimensional design concepts that are the base of a strong dimensional model.

First, the primary key for the Sales_Representative dimension table is a surrogate key. We highly recommend that you avoid designating a primary key that has business value, such as the EMPLOYEE_ID (which is a natural id in the source system). Generally when we design an OLTP database, we use a field that has meaning as a primary key. However, when designing the primary keys for our dimension tables, we want to use non-intelligent keys as primary keys for our dimensions.

This design concept becomes even more important if we are implementing a change handling technique using the Type-2 approach. To prove this, assume that we use the EMPLOYEE_ID (which is the natural id in the source system) as the primary key of the dimension. A Type-2 implementation approach needs to accommodate multiple rows per sales representative depending upon the number of changes that need to be preserved for history purposes.

In the example, the EMPLOYEE_ID never changes. The person is still the same person and has the same employee identification (EMPLOYEE_ID), but his or her information changes over time. Attempting to use the EMPLOYEE_ID field would cause a primary key integrity violation when you try to insert the second row of data for that same person.

The second dimensional model design concept that this example illustrates is the lack of use of effective date attributes (in Sales_Representative dimension) that identify exactly when the change in the Sales_Representative dimension occurred.

A word on effective and expiration date attributes

There is no need to include effective date attributes inside the Sales_Representative dimension because the Date dimension shown in Figure 6-43 on page 262 will effectively partition the data over time when it is related to the fact table. Of course, the effectiveness of this partitioning depends on the grain of the fact table. In our case the grain tracks data at day level using the date dimension and at hour level using the time dimension.

Effective and expiration date attributes are necessary in the staging area because we need to know which surrogate key is valid when loading historical fact records. This is often a point of confusion in the design and use of type-2 slowly changing dimensions.

However, effective and expiration date attributes are very important in the staging area of the data warehouse or dimensional model because we need to know which surrogate key is valid when loading historical fact rows. In the dimension table, we stated that the effective and expiration dates are not needed, though you may still use them as helpful extras that are not required for the basic partitioning of history. It is important to note that if you add the effective and expiration date attributes in the dimension table, then there is no need to constrain on the effective and expiration date in the dimension table to get the correct answer. You can get the same answer by partitioning history using the date or time dimension of your dimensional designs.

A summary of the type-2 change handling strategy is presented in Table 6-12.

Table 6-12 Summary of type-2 approach concepts

Type-2 approach	Description
When to use the Type-2 change handling approach	► When there is need to track an unknown number of historical changes to dimensional attributes.
Advantages of the Type-2 change handling approach	► Enables tracking of all historical information accurately and for an *infinite* number of changes.
Disadvantages of the Type-2 change handling approach	► Causes the size of the dimension table to grow fast. In cases where the number of rows being inserted is very high, then storage and performance of the dimensional model may be affected. ► Complicates the ETL process needed to load the dimensional model. ETL-related activities that are required in the type-2 approach include maintenance of effective and expiration date attributes in the staging area.
Impact on existing dimension table structure	► No change to dimensional structure needed. ► Additional columns for effective and expiration dates are not needed in the dimension table.
Impact on preexisting aggregations	► There is no impact on the preaggregated tables. The aggregated tables are not required to be rebuilt as with the type-1 approach.

Type-2 approach	Description
Impact on database size	► Yes, accelerates the dimensional table growth because with each change in a dimensional attribute, a new row is inserted into the dimension table.
Adding effective and expiration dates to dimension tables	No. This is not necessary in the dimension tables. However, effective and expiration attributes are needed in the staging area because we need to know which surrogate key is valid when we are loading historical fact rows. In the dimension table, we stated that the effective and expiration dates are not needed though you may still use them as helpful extras that are not required for the basic partitioning of history. It is important to note that in case you add the effective and expiration date attributes in the dimension table, then there is no need to constrain on the effective and expiration date in the dimension table in order to get the right answer. You could get the same answer by partitioning history using the date or time dimension of your dimensional designs.

Type-3 (preserving one or more versions of history)

The type-3 approach is typically used only if there is a limited need to preserve and accurately describe history. An example is when someone gets married and there is a need to retain the original surname of the person.

Consider the last name of the person Suzan Holcomb prior to marriage, as shown in Table 6-13. The last name for Suzan is Holcomb.

Table 6-13 Customer table

Customer ID	First name	Old last name	New last name	Married?
963276	Suzan	Holcomb	Holcomb	No

After Suzan gets married to David Williams, she acquires his last name. For such changes where we can predict the number of changes that will happen for a particular dimension attribute (such as last name), we use a type-3 change. This is shown in Table 6-14.

Table 6-14 Type-3 change for customer table

Customer ID	First name	Old last name	New last name	Married?
963276	Suzan	Holcomb	Williams	Yes

In Table 6-14 on page 267, you can preserve one change per attribute. In other words, you can preserve the last name of Suzan only if she marries once. If she marries again, to a second person called Brent Donald, then the type-3 change discards her first last name Holcomb, as shown Table 6-15.

Table 6-15 Type-3 change

Customer ID	First name	Old last name	New last name	Married?
963276	Suzan	Williams	Donald	Yes

Table 6-15 shows that now the new last name is Donald and the old last name column has a value of Williams. The disadvantage with the type-3 approach is that it can preserve only one change per attribute—old and new or first and last. In case you want to preserve two changed attributes for Suzan, then you need to redesign the dimension table as shown in Table 6-16.

Table 6-16 Type-3 two changes

Customer ID	First name	First old last name	Second old last name	New last name	Married
963276	Suzan	Holcomb.	Williams	Donald	Yes

The more historical changes you want to maintain using the type-3 change, the higher the number of additional columns needed in the dimensional table.

Instead of inserting a new dimensional row to hold the attribute change, the type-3 approach places a value for the change in the original dimensional record. You can create multiple fields to hold distinct values for separate points in time.

Important: The type-3 approach enables us to see new and historical fact table rows by either the new or prior attribute values.

There is another disadvantage of using the type-3 approach. This has to do with extracting information from a dimension table which has been implemented using a type-3. Although the dimension structure that was implemented using type-3 has all the data needed, the SQL required to extract the information can be complex.

Extracting a specific value is not that difficult, but if you want to obtain a value for a specific point in time, or multiple attributes with separate old and new values, the SQL statements needed become long and have multiple conditions. Overall, a type-3 approach can store the data of a change, but cannot accommodate multiple changes. Also, summary reporting is very difficult using the type-3 approach.

To summarize the type-3 slowly change handling strategy, we present Table 6-17.

Table 6-17 Summary of type-3 approach concepts

Type-3 approach	Description
When to use the type-3 change handling approach?	► Should only be used when it is necessary for the data warehouse to track historical changes, and when such changes will only occur for a finite number of times. If the number of changes can be predicted, then the dimension table can be modified to place additional columns to track the changes.
Advantages of the type-3 change handling approach	► Does not increase the size of the table as compared to the type-2 approach, since new information is updated. ► Allows us to keep part of history. This is equivalent to the number of changes we can predict. Such prediction helps us modify the dimension table to accommodate new columns.
Disadvantages of the type-3 change handling approach	► Does not maintain all history when an attribute is changed more often than the number in the predicted range, because the dimension table is designed to accommodate a finite number of changes. ► If we designed a dimension table assuming a fixed number of changes, then needed more, then we would have to redesign or risk losing history.
Impact on existing dimension table structure	► The dimension table is modified to add columns. ► The number of columns added depends on the number of changes to be tracked.
Impact on preexisting aggregations	► You may be required to rebuild the preaggregated tables.
Impact on database size	► No impact is there since data is only updated.

6.3.6 Handling fast changing dimensions

In this section we identify the changing dimensions that cannot be handled using the Type-1, Type-2, or Type-3 approaches.

What are fast changing dimensions?

A dimension is considered to be a fast changing dimension if one or more of its attributes changes frequently and in many rows. A fast changing dimension can grow very large if we use the Type-2 approach to track numerous changes. Fast changing dimensions are also called rapidly changing dimensions. Consider a scenario for a customer dimension having 100 000 rows. Assume that we are handling the changes for this customer using the Type-2 approach. Further assume that in a year an average of 10 changes occur for each customer. Therefore in one year the number of rows will increase to 100 000 x 10 = 1 000 000. For some companies, this may still be a small number to manage even using a Type-2 change. That is, for some, the customer dimension may be a slowly changing dimension.

Assume that a customer table has 10 million rows. Imagine the same scenario where, on average, 10 changes occur for a customer each year. This means at the end of the year, the table would grow to about 100 million rows. This is a huge growth. Such a customer dimension may be considered a fast growing dimension. Then, handling such a fast growing dimension using a Type-2 approach is not feasible.

In order to identify the reason for a fast changing dimension, we must look for attributes that have continuously variable values such as age, test score, size, weight, credit history, customer account status, or income.

An appropriate approach for handling very fast changing dimensions is to break off the fast changing attributes into one or more separate dimensions, called *mini-dimensions*. The fact table would then have two foreign keys—one for the primary dimension table and another for the fast changing attributes. These dimension tables would be associated with one another every time we insert a row in the fact table.

Figure 6-44 on page 271 shows an example of a fast changing dimension.

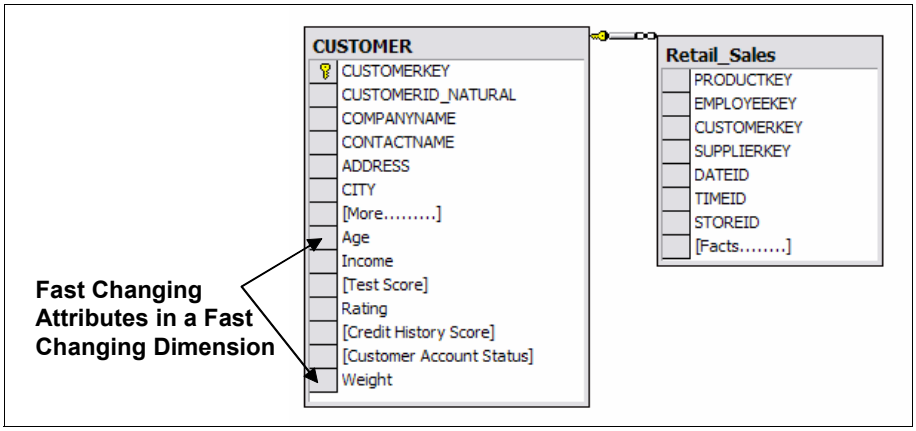


Figure 6-44 Fast changing dimension

The attributes which change rapidly in this fast changing dimension are identified as:

- Age
- Income
- Test score
- Rating
- Credit history score
- Customer account status
- Weight

Solving the fast changing dimensions problem

After having identified the constantly changing attributes, the next step is to convert these identified attributes individually into band ranges. The concept behind this exercise is to force these attributes to take limited discreet values. For example, let us assume that each of the above seven attributes takes on 10 different values, then the Customer_Mini_dimension (Figure 6-45 on page 273) will have 1 million values. Thus, by creating a mini-dimension table consisting of band-range values, we have solved the problem of the situation where the attributes such as age, income, test score, credit history score, customer account status, and weight can no longer change. These attributes cannot change because they have a fixed set of band-range values (see Table 6-18 on page 272) rather than having a large number of values.

Sample rows for this new Mini-dimension, called Customer_Mini_Dimension, are shown in Table 6-18 on page 272.

Table 6-18 Sample mini-dimension rows

Age	Income	Test Score	Rating	Credit_ History_ Score	Customer_ Account_ Status	Weight
18 to 25	$800 to $1000	2-4	8-12	7-9	Good	88
26 to 28	$1000 to $1500	5-9	8-12	10-14	Good	88
29 to 35	$1500 to $3000	10-17	8-12	15-19	Good	88
More	....	....	....	....	....	....

What is a mini-dimension?

A mini-dimension is a dimension that usually contains fast changing attributes of a larger dimension table. This is to improve accessibility to data in the fact table. Rows in mini-dimensions are fewer than rows in large dimension tables because we try to restrict the rows in mini-dimensions by using the band range value concept.

Note: It is important to understand that the mini-dimension created in Figure 6-45 cannot be allowed to grow too large. If it became a fast changing dimension, then it must be split into another mini-dimension, and the same band range technique applied to accommodate the growth.

After identifying the fast changing attributes of the primary customer dimension, and determining the band ranges for these attributes, a new mini-dimension is formed called Customer_Mini_Dimension as shown in Figure 6-45 on page 273. This scenario works reasonably well for a few rapidly changing attributes. The next question is, “What if there are ten or more rapidly changing attributes?”

Should there be a single mini-dimension for all fast changing attributes? Maybe, but the growth of this mini-dimension can easily get out of hand and grow into a huge dimension. The best approach is to use two or more mini-dimensions and force the attributes of each of these dimensions to take on a set of band range values.

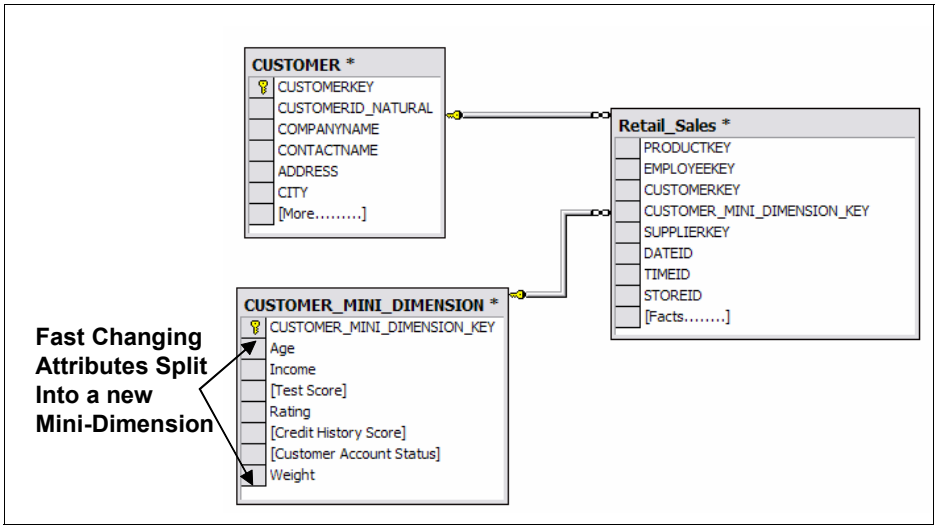


Figure 6-45 Fast changing dimension implementation

As a summary, the approach used with fast changing dimensions involves the following activities:

- We analyze all dimensions identified carefully to find which change very fast.
- The fast changing dimension is analyzed further to see what the impact is on the size of the dimension table if we handle the change using the Type-2 approach. If the impact on the size of the dimension table is huge, then we must avoid the Type-2 approach.
- The next step is to analyze the fast changing dimension in detail to identify which attributes of this dimension are subject to changing fast. Such fast changing attributes are then separated into one or more mini-dimension tables whose primary keys are attached to the fact table as a foreign key. Then there are the following tables:
  - One main dimension table, which was the primary original fast changing dimension, without the fast changing attributes.
  - One or more mini-dimensions where each table consists of fast changing attributes. Each of these mini-dimensions has its own primary surrogate keys.
- Now we study each new mini-dimension and categorize the fast changing attributes into ranges. For example, age can be categorized into a range, such as [1-8], [9-15], [16-25], [26-34], [35-60], and [61 and older]. Although the Type-2 approach should not be needed to track age, age bands are often used for other purposes such as analytical grouping. Assuming that we have the customer date of birth, we can easily calculate age whenever needed.

However, for the purpose of designing a fast changing dimension, we cannot in a vacuum determine the band ranges. Only the business needs should be the deciding factor in determining which continuously variable attributes are suitable for converting to bands.

Snowflaking does not solve the fast changing dimension problem

We now discuss snowflaking and fast changing dimensions. As an example, Figure 6-46 depicts a split Customer dimension and created a Customer mini-dimension. The mini-dimension is snowflaked and attached as a foreign key to the Customer dimension. However, this is not a good design. Why?

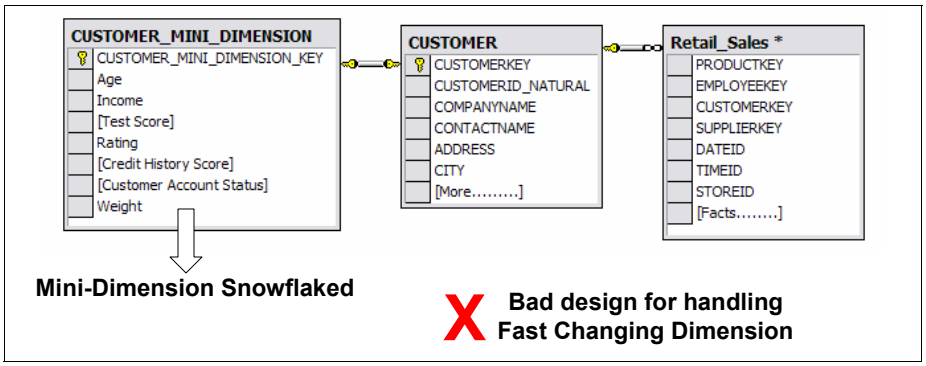


Figure 6-46 Snowflaking a fast changing dimension

Consider the sample rows shown in Table 6-19, for the Customer_Mini_Dimension. Assume that a Customer named Stanislav Vohnik in the Customer table is attached to the 1st row in the Customer_Mini_Dimension table as shown in Table 6-19. At this point, Customer *Stanislav Vohnik*, is only attached to one row.

Now assume that in a single year, customer Stanislav has changes to his profile at least five times, so that he is now attached to all five rows in Table 6-19. That means the average growth for customer Stanislav Vohnik is about five rows every year. Imagine, if we have 10 million customers and each has an average of about five profile changes, the growth will be about 50 million.

Table 6-19 Sample mini-dimension rows

Age	Income	Test Score	Rating	Credit_ History_ Score	Customer_ Account_ Status	Weight
18 to 25	$800 to $1000	2-4	8-12	7-9	Good	88

Age	Income	Test Score	Rating	Credit_ History_ Score	Customer_ Account_ Status	Weight
18 to 25	$1000 to $1500	5-9	8-12	10-14	Good	88
18 to 25	$1500 to $3000	10-17	8-12	15-19	Good	88
18 to 25	$1000 to $1500	5-9	8-12	16-17	Good	88
18 to 25	$1000 to $1500	5-9	8-12	21-25	Very Good	88

We have now introduced the fast changing dimension problem in the Customer dimension table despite creating a mini-dimension with band range values because we snowflaked that dimension.

Had we attached the primary key of the Customer_Mini_Dimension table to the Retail_Sales fact table, as shown in Figure 6-47 on page 276, we could have solved the problem of the fast changing dimension for the Customer table. In this scenario, the changes for the fact table record would be handled by the fact table and the corresponding record of change in the Customer_Mini_Dimension. There would be still only one record for Customer Stanislav Vohnik in the Customer table. The appropriate changes would be handled in the fact table by linking the correct profile of Customer Stanislav Vohnik at the time of purchase of the product. The appropriate profile would be linked to the Customer_Mini_Dimension.

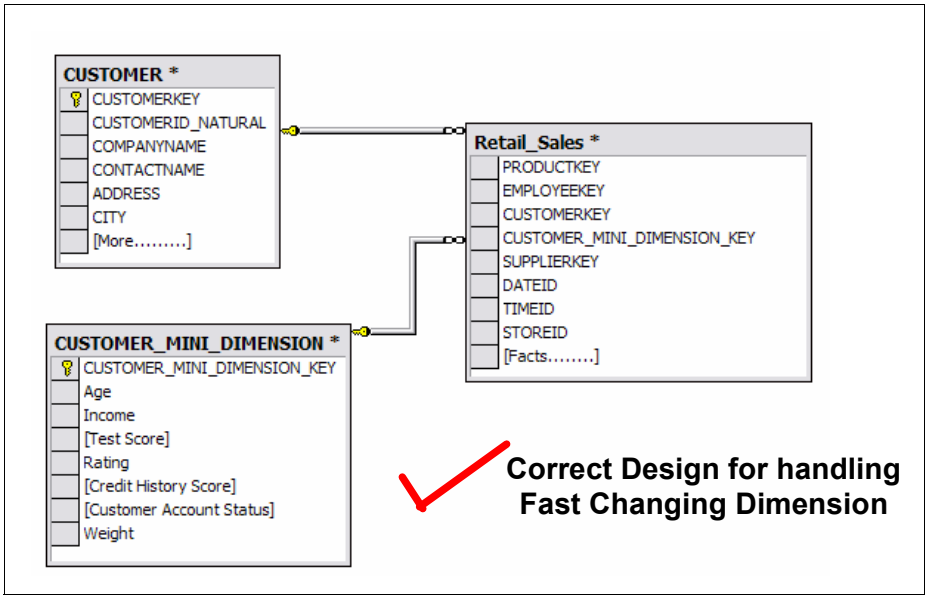


Figure 6-47 Correct approach for handling fast changing dimension

Snowflaking versus mini-dimensions

Figure 6-48 on page 277 shows the difference between snowflaking and creating a mini-dimension out of a fast changing customer dimension table. Fast changing dimensions should be segmented into one or more mini-dimensions and not snowflaked. Snowflaking is appropriate only in cases where low-cardinality attributes in the dimension have been removed to separate normalized tables, and these normalized tables are then joined back into the original dimension table.

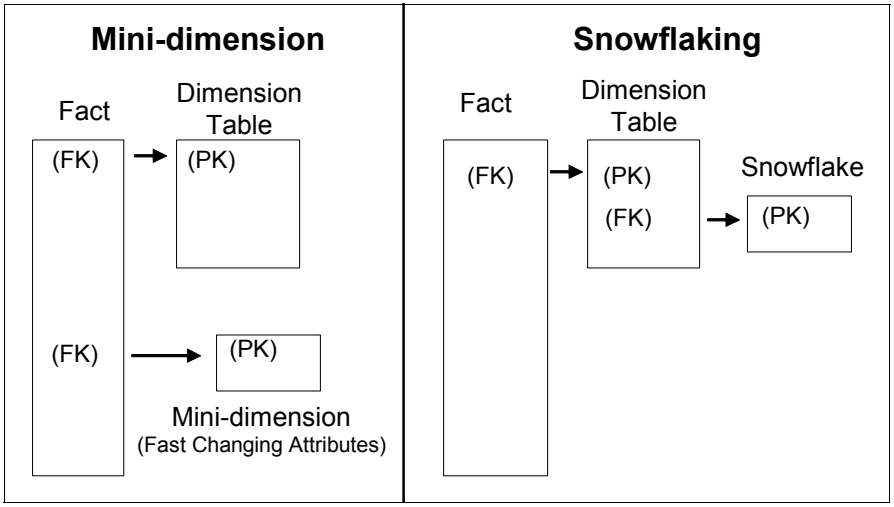


Figure 6-48 Snowflaking versus mini-dimension

How many mini-dimensions for a fast changing dimension?

Assume that we have 10 fast changing attributes for a fast changing dimension. Should there be a separate dimension for each attribute? Perhaps, but the number of dimensions can rapidly get out of control and the fact table will have a large number of foreign keys for these dimensions.

One approach is to combine several of these mini-dimensions into a single physical mini-dimension. This is the same technique used to create a garbage dimension (see6.3.8, “Identifying garbage dimensions” on page 282) that contains unrelated attributes and flags to get them out of the fact table. It is important to understand that the fact table must always be involved to relate customers to their attributes (present in mini-dimension). If the mini-dimension becomes too large, then split the mini-dimension into two or more mini-dimensions.

6.3.7 Identifying dimensions that need to be snowflaked

In this section we look at snowflaking to see when it is practical to implement in dimensional designs. We also look at cases of poor dimensional design where hierarchies have been snowflaked into separate dimension tables, and as a result performance and understandability of the dimensional model have suffered.

What is snowflaking?

Further normalization and expansion of the dimension tables in a star schema result in the implementation of a snowflake design. In other words, a dimension table is said to be snowflaked when the low-cardinality attributes in the dimension have been removed to separate normalized tables and these normalized tables are then joined back into the original dimension table.

Typically, we do not recommend snowflaking in the dimensional model environment, because it can impact understandability of the dimensional model and result in a decrease in performance because more tables will need to be joined to satisfy queries.

When do you snowflake?

Snowflaking a dimension table can typically be performed under the following two conditions:

- The dimension table consists of two or more sets of attributes which define information at different grains.
- The sets of attributes of the same dimension table are being populated by different source systems.

To help understand when and why we snowflake, consider the sample customer table shown in Figure 6-49.

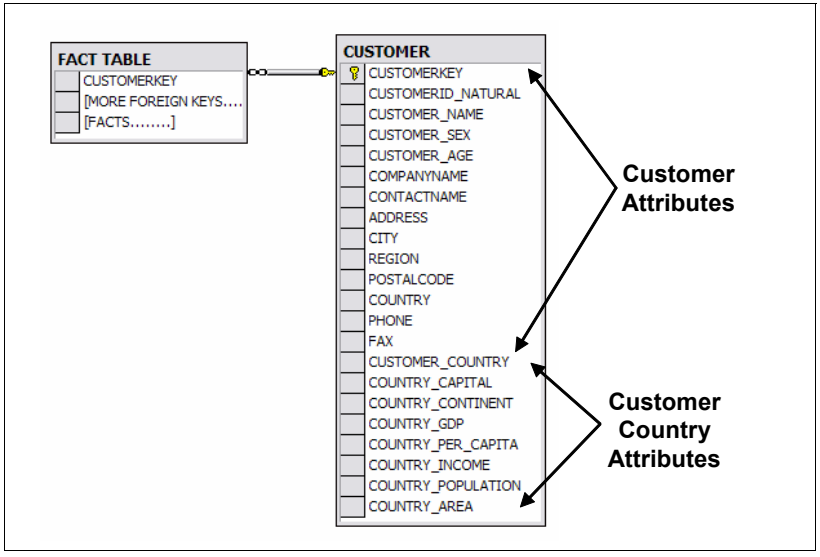


Figure 6-49 Customer dimension

The customer table shows two kinds of attributes. These are attributes relating to the customer and attributes relating to the customer country. Both set of attributes represent a different grain (level of detail) and also both sets of attributes are populated by a different source system.

Such a customer dimension table is a perfect candidate for snowflaking for two primary reasons:

- The customer table represents two different sets of attributes. One set shows customer attributes and the other set shows customer's country attributes. Both of these sets of attributes represent a different level of detail of granularity. One set describes the customer and the other defines more about the country. Also, the data for all customers residing in a country is identical.
- On detailed analysis, we observe that the customer attributes are populated by the company CRM source system, where the country-related attributes are populated from an external demographic firm that has expertise in country demographic attributes. It may also be possible another source system in the company may also be supplying some of these attributes.

The snowflaked customer table is shown in Figure 6-50. The customer dimension table is said to be snowflaked when the low-cardinality attributes (customer's country attributes) in the dimension have been removed to separate a normalized table called country and this normalized table is then joined back into the original customer dimension table.

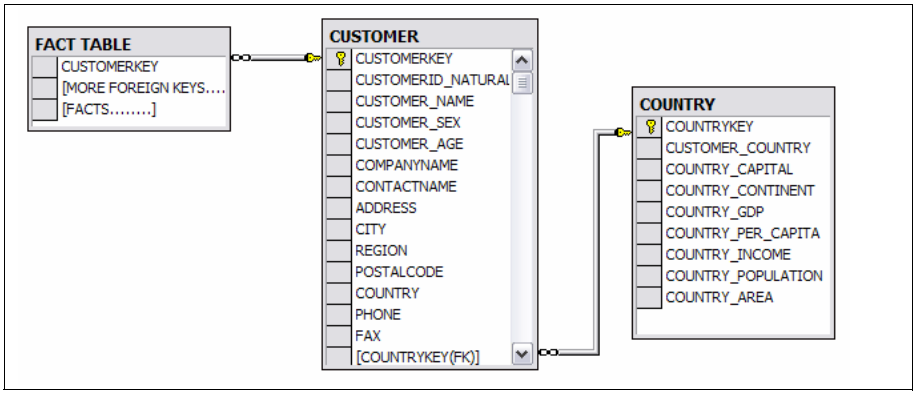


Figure 6-50 Snowflaked customer table

When NOT to snowflake?

Typically, we recommend that you avoid the use of snowflaking, or normalization of dimension tables, unless required and appropriate, as depicted in Figure 6-50.

Dimensional modelers may argue that snowflaking reduces disk space consumed by dimension tables, but the savings are usually insignificant when compared with the entire data warehouse. Moreover the disadvantages in ease of use or query performance far outweigh the space savings achieved by inappropriate snowflakes.

Figure 6-51 shows an inappropriate use of snowflakes where hierarchies that belong to a single product dimension table have been split into separate tables. This has an adverse affect on query performance and understandability.

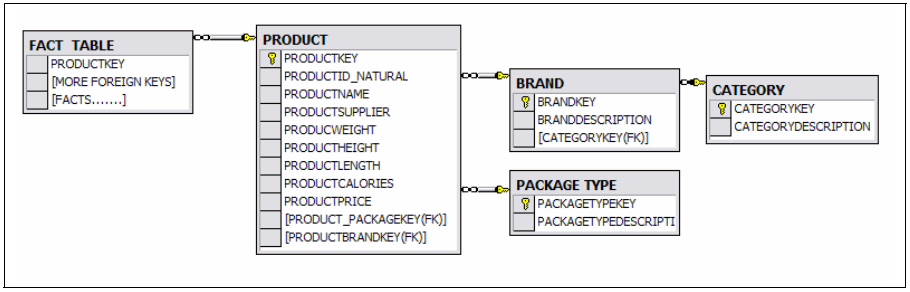


Figure 6-51 Inappropriate use of snowflaking

Note: Do not snowflake hierarchies of one dimension table into separate tables. Hierarchies should belong to the dimension table only and should never be snowflaked. Multiple hierarchies can belong to the same dimension if the dimension has been designed at the lowest possible detail.

Snowflaking for performance improvement

A snowflake schema is a variation on the star schema, in which very large dimension tables are normalized into multiple tables. We do not recommend segmenting hierarchies into snowflake tables, because users typically use different level members of the hierarchy in viewing reports. If hierarchies are split into separate tables, performance may be impacted as a larger number of joins will be required.

In some situations, you may snowflake the hierarchies of a main table. For example, dimensions with hierarchies can be decomposed into a snowflake structure to avoid joins to big dimension tables when you are using an aggregate of the fact table. As an example, if you have brand information that you want to separate from a product dimension table, you can create a brand snowflake that consists of a single row for each brand and that contains significantly fewer rows than the product dimension table.

Figure 6-52 shows a snowflake structure for the brand and product line elements and the brand_agg aggregate table.

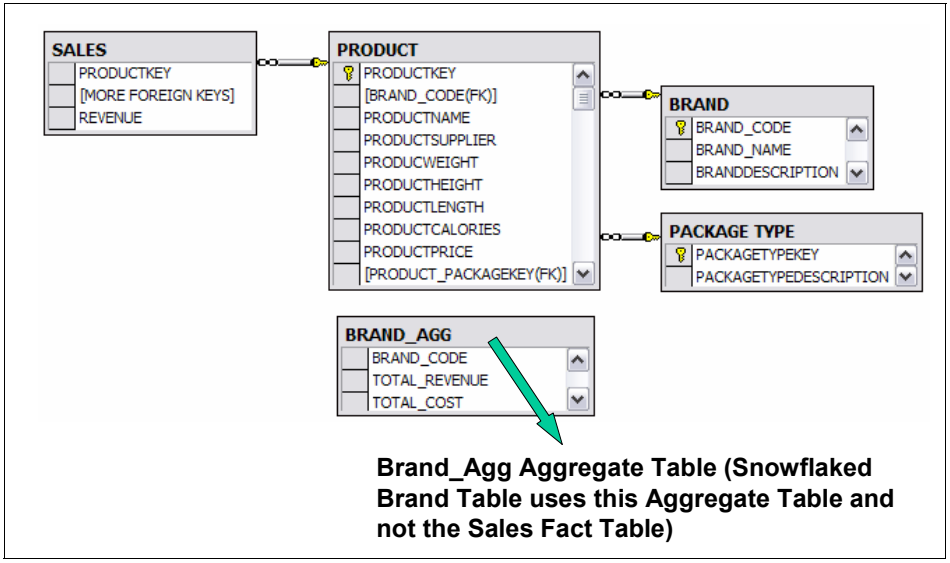


Figure 6-52 Snowflaking to improve performance

By creating an aggregate table that consists of the brand code and the total revenue per brand, a snowflake schema can be used to avoid joining to the much larger sales table. This is shown in the example query on the brand and brand_agg tables, depicted in Example 6-3.

Example 6-3 Query directly on brand table to improve performance

```
SELECT brand.brand_name, brand_agg.total_revenue
FROM brand, brand_agg
   WHERE brand.brand_code = brand_agg.brand_code
   AND brand.brand_name = 'Anza'
```

Without a snowflaked dimension table, a SELECT UNIQUE or SELECT DISTINCT statement on the entire product table (potentially, a very large dimension table that includes all the brand and product-line attributes) would have to be used to eliminate duplicate rows.

While snowflake schemas are unnecessary when the dimension tables are relatively small, a retail or mail-order business that has customer or product dimension tables that contain millions of rows can use snowflake schemas to significantly improve performance.

If an aggregate table is not available, any joins to a dimension element that was normalized with a snowflake schema must now be a three-way join, as shown in Example 6-4. A three-way join reduces some of the performance advantages of a dimensional database.

Example 6-4 Three-way join reduces performance

```
SELECT brand.brand_name, SUM(sales.revenue)
FROM product, brand, sales
   WHERE product.brand_code = brand.brand_code
   AND brand.brand_name = 'Alltemp'
GROUP BY brand_name
```

Disadvantages of snowflaking

The greater the number of tables in a snowflake schema, the more complex the design.

- More tables means more joins, and more joins mean slower queries.
- It is best not to use snowflaking to try to save space. Space saved by snowflaking is extremely small compared to the overall space of the fact table in the dimensional design.

6.3.8 Identifying garbage dimensions

A garbage dimension is a dimension that consists of low-cardinality columns such as codes, indicators, and status flags. The garbage dimension is also referred to as a *junk dimension*. The attributes in a garbage dimension are not related to any hierarchy.

We now review the dimensional model (see Figure 6-53 on page 283) for the grocery store example described in Chapter 5, “Dimensional Model Design Life Cycle” on page 103. The grain for this dimensional model is a single line item on the grocery bill or invoice.

To understand the concept of a garbage dimension, assume that we need to add three additional things to this dimensional model. They are:

- Customer feedback about the store, and customer feedback by employee, categorized as good, bad, and none.
- Method used by the customer to pay for the products, categorized as cash or credit card.
- Bag selected to carry the goods, categorized as plastic, paper, or both.

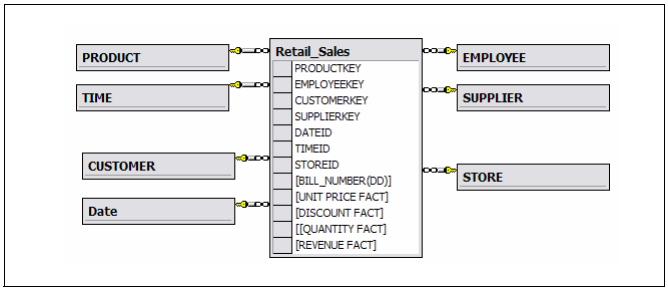


Figure 6-53 Grocery store example

The next question is, how do we accommodate these new attributes in the existing dimensional design. The following are the options:

- Put the new attributes relating to customer feedback, payment type, and bag type into existing dimension tables.
- Put the new attributes relating to customer feedback, payment type, and bag type into the fact table.
- Create new separate dimension tables for customer feedback, payment type, and bag type.

First, we cannot put the new attributes into existing dimension tables because each of the new attributes has a different grain than the product, supplier, employee, time, customer, store, or date table. If we tried to forcibly add these new attributes, we would create a cartesian product for the dimension tables. For example, we merge the payment type (Credit card or Cash) attribute with the product table. Assume that we have 100 product rows in the product table. After adding the payment type attribute to the product table, we have 200 product rows. This is because one product would need to be represented with two rows, for both credit card and cash payment type. Because of this reason, we do not force adjusting these attributes inside any of the existing dimension tables.

Second, we take the fact table. We do not recommend adding low-cardinality fields into the fact table because it would result in a very huge fact table.

Note: Fact tables generally occupy 85-95% of the space in most dimensional designs. To optimize space, recall that fact tables should only contain facts, degenerate dimension numbers, and foreign keys. Fact tables should not be used to store textual facts or low-cardinality text fields.

Our third choice was to create a separate dimension for each of new attributes relating to customer feedback, payment type, and baggage type. The disadvantage of this approach is that we will have three additional foreign keys

inside the fact table. Having the three foreign keys in the table is equivalent of increasing the row size in the fact table by 4 x 3 = 12 bytes. A fact table that has, for example, 100,000 rows inserted into it on a daily basis, would increase about 12 times with an increase of 12 bytes per row. This increased size would impact the performance of the fact table.

Note: Avoid creating separate dimensions for low-cardinality fields such as those which hold flags and indicators.

The best way to solve the problem associated with low-cardinality attributes, such as those relating to customer feedback, payment type, and baggage type, is to create a single dimension called a *garbage dimension*. An example is depicted
in Figure 6-54. A garbage dimension acts as a storage space for the low-cardinality fields.

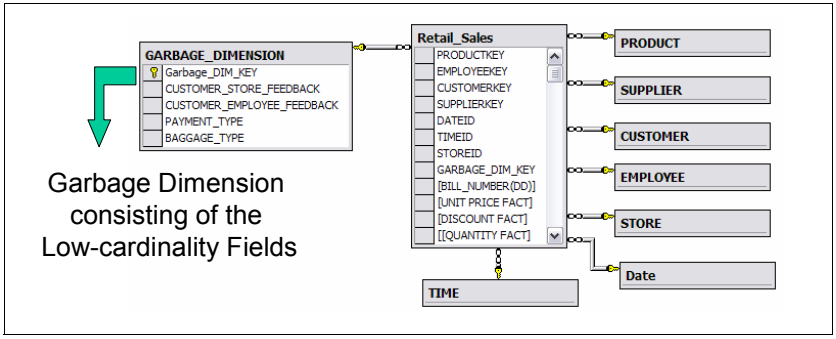


Figure 6-54 Garbage dimension implementation

Table 6-20 on page 285 shows sample rows inside the Garbage_Dimension table. The total number of rows inside a garbage dimension is equal to the number of rows formed after taking the cartesian product of all individual values that reside inside each of the low-cardinality attributes. Therefore, the number of rows inside the Garbage_Dimension table is 3 x 3 x 2 x 3 = 54 rows. A new surrogate key value is also generated for each combination. The following are the possible values for each of the low-cardinality fields:

- Customer feedback: Good, bad, and none
- Customer feedback about employee: Good, bad, and none
- Payment type: Cash or credit card
- Bag type to carry goods: Plastic, paper, or both

Table 6-20 Sample rows inside the garbage dimension

Garbage_ Dim-Key	Customer_Store _Feedback	Customer_ Employee_ Feedback	Payment_ Type	Bag_Type
1	Good	Good	Credit card	Plastic
2	Good	Bad	Credit card	Plastic
3	Good	None	Credit card	Plastic

Note: If the garbage or junk dimension grows too large, then you must split it into two or more dimensions. The total number of rows inside a garbage dimension is equal to the number of rows that are formed after taking the cartesian product of all individual values that reside inside each of the low-cardinality attributes.

How should you load the garbage dimension?

There are two ways to load the garbage dimension. They are:

- Preload the dimension: In this approach, you identify all the possible combinations of values that could result from the cartesian product of all individual values that reside inside each of the low-cardinality attributes. Then you load the dimension table. This approach works well if the number of rows that result from the cartesian product are few.
- Load at run time: This approach works well if the garbage or junk dimension has a huge number of rows. However, in this scenario, you might not pre-populate all combinations.

6.3.9 Role-playing dimensions

A single dimension which is expressed differently in a fact table using views is called a *role-playing dimension*.

Figure 6-55 on page 286 shows a dimensional model designed for an order management business process. There are two date entities (order and received date) at the same day grain involved.

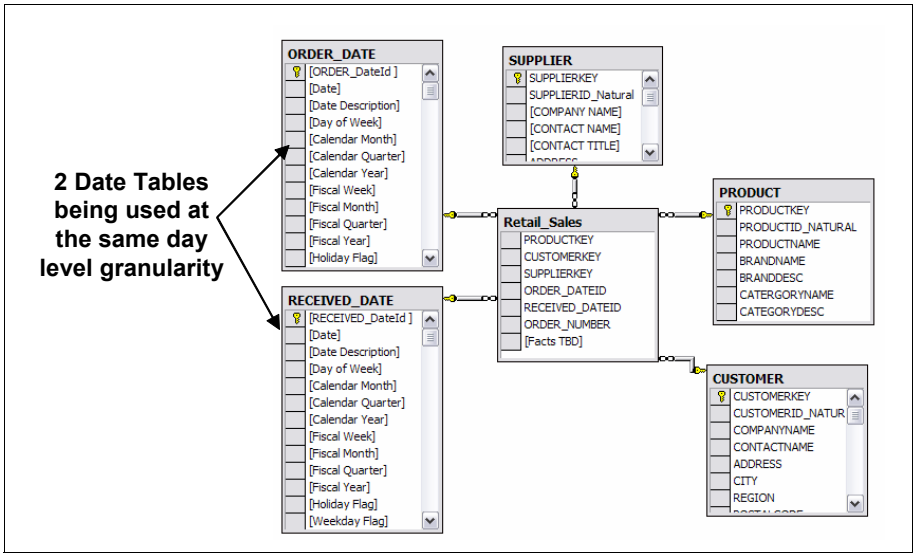


Figure 6-55 Order management dimensional model

Instead of using two separate date tables, Order_Date and Received_Date with the same granularity, we can create two views from a single date table (Date) as shown in Figure 6-56 on page 287.

The Order_Date_View and Received_Date_View serve as the role-playing dimensions as depicted in Figure 6-56 on page 287.

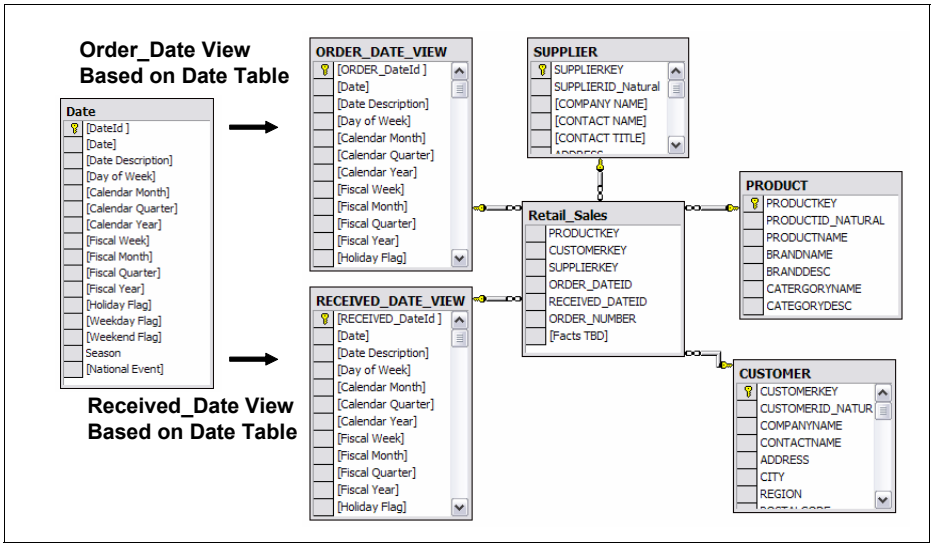


Figure 6-56 Role-playing dimension

The Order_Date_View and Received_Date_View dimension tables are implemented as views, as shown in Example 6-5 and Example 6-6.

Example 6-5 Order_Date_View dimension table implementation

```
Create View ORDER_DATE_VIEW(ORDER_DateId,Order_Date, Order_Date_Description,
Order_Day_of_Week, Order_Calendar_Month, Order_Calendar_Quarter,
Order_Calendar_Year, Order_Fiscal_Week, Order_Fiscal_Month,
Order_Fiscal_Quarter, Order_Fiscal_Year, Order_Holiday_Flag,
Order_Weekday_Flag, Order_Weekend_Flag, Order_Season, Order_National_Event) as
select * from Date
```

There is only one date table, but it is projected differently using views.

Example 6-6 Received_Date_View dimension table implementation

```
Create View Received_DATE_VIEW(Received_DateId,Received_Date,
Received_Date_Description, Received_Day_of_Week, Received_Calendar_Month,
Received_Calendar_Quarter, Received_Calendar_Year, Received_Fiscal_Week,
Received_Fiscal_Month, Received_Fiscal_Quarter, Received_Fiscal_Year,
Received_Holiday_Flag, Received_Weekday_Flag, Received_Weekend_Flag,
Received_Season, Received_National_Event) as
select * from Date
```

6.3.10 Multi-valued dimensions

We discussed in section 5.4.1, “Dimensions” on page 135 that each dimension attribute should take on a single value in the context of each measurement inside the fact table. However, there are situations where we need to attach a multi-valued dimension table to the fact table. There are situations where there may be more than one value of a dimension for each measurement. These cases are handled using multi-valued dimensions.

Consider the design for a dimensional model for a family insurance company. Assume that Mr. John is the family insurance policy account holder and he has the policy account number 963276. Mr. John is married to Lisa and they have a son named Dave. Mr. John later decides to get a joint family policy under the same policy account number 963276. This means is that the three family members hold a joint policy for the same account 963276.

Assume that the family insurance company charges a particular amount for the entire family, which is insured under a common policy account number 963276.

Now assume we are asked to create a data mart for this insurance company and the business users have the following set of requirements:

- Business is interested in analyzing the revenue generated each month after the policy account holders pay differing amounts for their family insurance.
- Business is interested in analyzing the revenue generated for different states and counties.
- Business is also interested in analyzing the data to see individual members associated with each policy. That is, they want to know how many members, and their identity, are associated with a single policy number. For example, the business may be interested to know that for policy account number 963276, Mr. John is the primary policy holder and his wife, Lisa, and their son, Dave, are associated with it.

  To create the dimensional model, we decide the grain of the fact table is one row for each family policy account at the end of each month. We create a dimensional model design as depicted in Figure 6-57 on page 289.

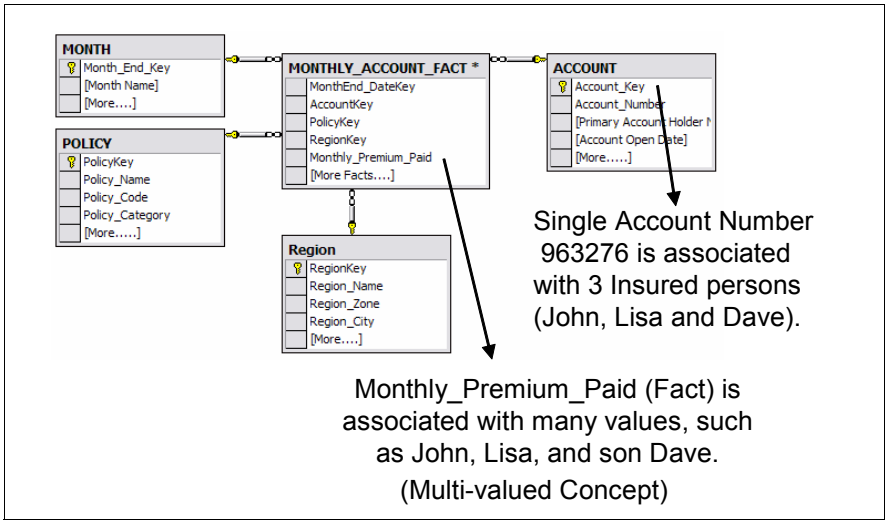


Figure 6-57 Family insurance dimensional model

The multi-valued dimension

As it is shown in Figure 6-57, an individual family insurance policy account can have one, or more, customers associated with it. For example, for the policy account number 963276, there are three insured customers.

To represent that there is more than one customer associated with the fact (Monthly_Premium_Paid), we cannot merely include the customer as an account attribute. By doing so, we would be violating the granularity of the dimension table because more than one individual can be associated with a single family insurance account number.

Similarly, we cannot include an additional customer dimension in the fact table because we have declared the grain to be one row per family policy account. So, if we include the customer dimension we would be violating the grain because there would be one policy account associated to more than one customer. This is a typical example of a multi-valued dimension.

We can solve this multi-valued dimension problem with the use of an Account-Insured_Person bridge table as shown in Figure 6-58 on page 290. The primary key of the Account-Insured_Person bridge table consists of the surrogate Account and Insured_Person foreign keys. Therefore, using the bridge table, we are able to associate three customers, namely Mr. John, his wife, Lisa, and son, Dave, to the same family policy account number without violating the grain definition.

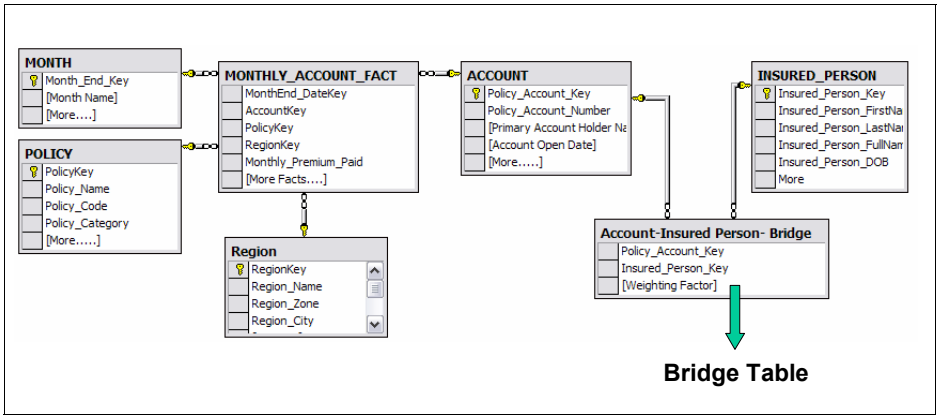


Figure 6-58 Use of a bridge table for multi-valued dimension

What is the weighting factor?

A weighting factor is an important column placed inside the bridge table. We assign a numerical weighting factor to each insured person for each family policy account number such that the sum of all the weighting factors belonging to a single group or account number is exactly 1. The weighting factor is simply a way to allocate the numeric additive facts across the insured persons that are present in the Insured_Person dimension table.

Approaches for handling multi-valued dimensions

There are situations when we need to attach multiple values for a dimension to a single fact table row, or when we need to attach a multi-valued dimension table to the fact table. We described an example of a multi-valued dimension in Figure 6-58. Another example of a multi-valued dimension is where we associate many customers to a single bank account. Or, when multiple diagnoses are associated with single patient. Dimensional modelers usually take one of following approaches for handling multi-valued dimension attributes:

- **Choose one particular value and leave the others:** This is the most frequently used technique, but it should not be forced. If we use this approach, the multi-valued dimension problem is eliminated, but the dimensional model may still not be useful because of missing dimensions.
- **Extend the dimensions list to include a fixed number of multi-valued dimensions:** This is not a good approach because it may increase the number of dimensions for each multi-value. Also, the number of values (dimensions) we may need to create are not fixed and may vary from case to case. If we create a fixed number of multi-valued dimensions and if the number of multi-values increases, then our design will not be flexible enough to handle it well.

- **Use a bridge table:** The bridge table may be inserted in two ways:

  a. Between the fact and dimension table, as shown in Figure 6-59, we chose the grain as one single line item on each order. We observe that for each order completion, there are one or more sales representatives that are involved with the sale and work together to get the order. The weighting factor column shows the relative contribution of each in getting the order. Assume that two persons work to get an order (Order Number# 99). Person A is contributes 90% of the overall time, where Person B only contributes 10%. There would be one row for the entire order (Order Number# 99), but there would be two rows for Person A and Person B in the ORDER-to-SALES REP BRIDGE table. The weighting factor for Person A is .9, since Person A contributed 90%, and for Person B the weighting factor is .1 since Person B's contribution was 10%. There are two rows in the "SALES_REPRESENTATIVE" table.

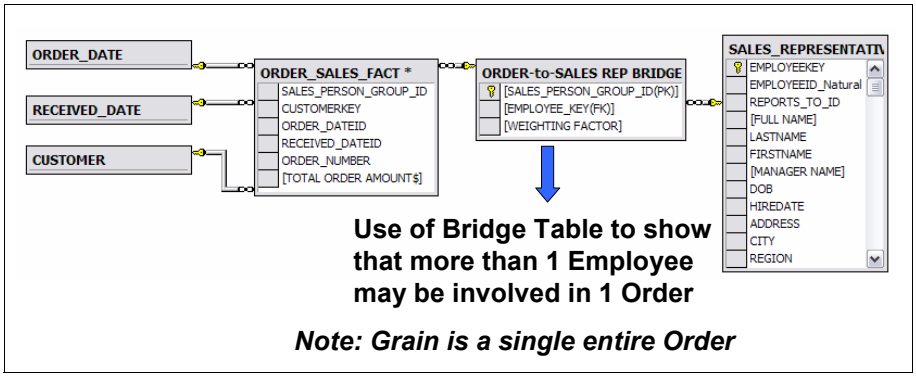


Figure 6-59 Multi-valued dimension example

  b. The other way was depicted in Figure 6-58 on page 290. Here, the bridge table is inserted between the Account dimension table and Insured_Person dimension table to show that multiple persons are associated with the same account.

6.3.11 Use of bridge tables

Bridge tables are used to express a many-to-many relationship between a fact and a dimension table. In dimensional modeling, the bridge tables are primarily used for two reasons:

- To solve the complex unbalanced hierarchies problem as discussed in "Unbalanced hierarchy" on page 251.
- For solving the multi-valued dimension problem as discussed in "Multi-valued dimensions" on page 288.

6.3.12 Heterogeneous products

The term heterogeneous means mixed, assorted, or diverse. This is exactly the state of many businesses. That is, they are selling diverse products to a common customer base. To better understand the concept, assume that we work for an insurance company called ABC 979 Insurance Inc., that sells products in the form of insurance policies. In particular, the company sells home and car insurance. The concept of heterogeneous products comes to light when a company sells different products with different unique attributes to the same customer base.

Moreover, the attributes the business tracks for home insurance are very different from those collected for car insurance. This is a typical example of a company selling heterogeneous products.

There are several approaches of handling heterogeneous products within the company. Several approaches are explained below:

- **Merge attributes:** Merge all the attributes into a single product table and all facts relating to the heterogeneous attributes in one fact table. This approach is depicted in Figure 6-60. The disadvantage of this approach is that all unrelated attributes for home and car insurance are merged into one single dimension. This makes the insurance dimension very large. Also for every row inside such a dimension, many columns belonging to home insurance are NULL for car insurance, because they belong to a different type. For this approach, a single fact table is created in which all facts for all heterogeneous products are merged, so the fact table grows very large.

  You should avoid this approach because the huge size of the tables can result in performance issues.

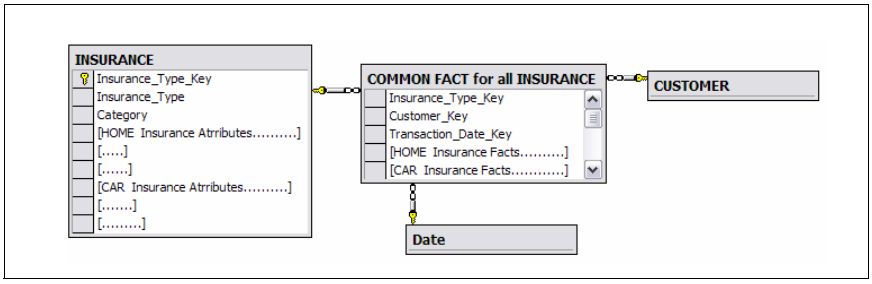


Figure 6-60 Merge all heterogeneous product attributes into one table

- **Separate tables:** Create separate dimension and fact tables for different heterogeneous products, as depicted in Figure 6-61 on page 293. With this approach, we create separate dimensions and facts for the heterogeneous products.

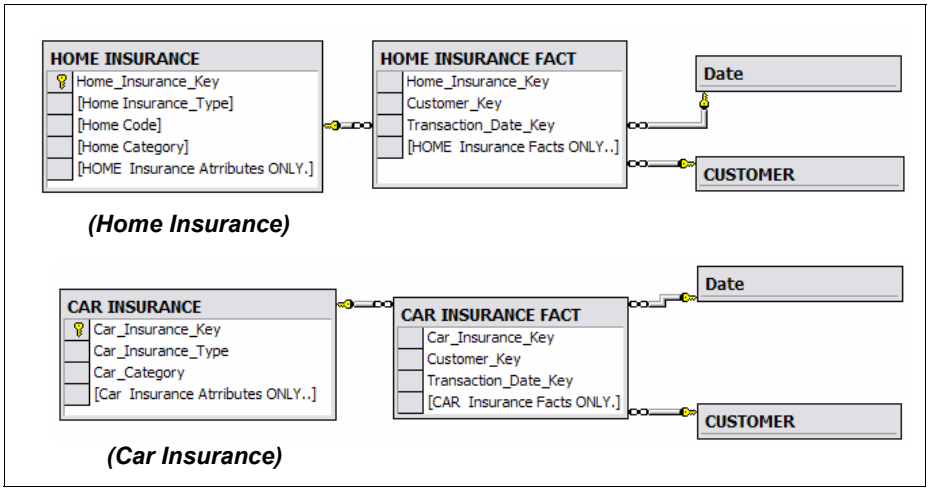


Figure 6-61 Separate dimension and fact tables to handle heterogeneous products

The advantage of having separate dimension and fact tables for such heterogeneous products is that the specifics about these individual businesses (Home and Car Insurance) can be analyzed in more detail.

- **Generic design:** Create a generic design to include a single fact and single product dimension table with common attributes from two or more heterogeneous products, as depicted in Figure 6-62. In this approach, we identify common attributes between heterogeneous dimensions and model them as a single dimension. Also, we identify and create generic facts, such as Transaction Amount or Revenue, earned for the two heterogeneous dimensions. This way the business can analyze the two business processes together for common facts.

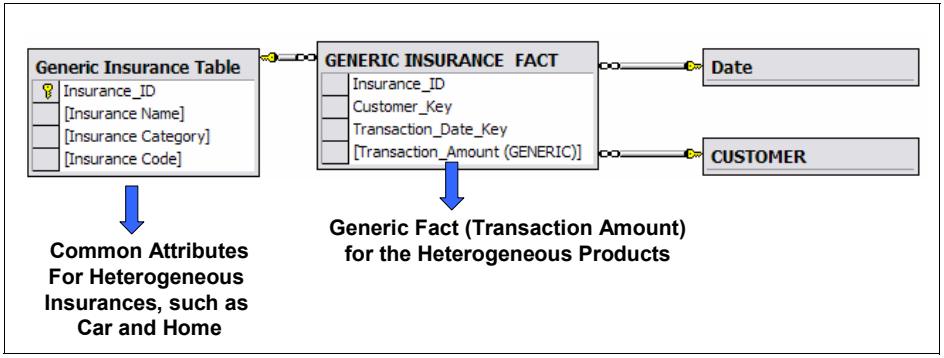


Figure 6-62 Generic design to handle heterogeneous products

The disadvantage of this approach is that only common attributes between the heterogeneous types of insurance (Home and Car) are available for analysis. Also, there is only a generic fact called the Transaction Amount. However, for detailed, specific business facts and attributes about each type of insurance (Car and Home), management can refer to the separate dimensional models as shown in Figure 6-61 on page 293.

Note: The concept of heterogeneous products applies to situations where the customer base is common. That is, if the company is selling heterogeneous products to altogether different customers, then such products would typically belong to separate data marts and perhaps also separate data warehouses.

6.3.13 Hot swappable dimensions or profile tables

A dimension that has multiple alternate versions of itself that can be swapped at query time is called a *hot swappable dimension or profile table*. Each of the versions of the hot swappable dimension can be of different structures. The alternate versions of the hot swappable dimensions access the same fact table, but with different output. The different versions of the primary dimension may be completely different, including incompatible attribute names and different hierarchies.

Unlike in a relational database, building hot swappable dimensions in OLAP is complicated because the joins between the tables cannot be specified at the query time.

Note: Swappable dimensions may be implemented to improve performance of the dimensional model. The swappable dimensions also help create more secure dimensions.

How to implement a swappable dimension

- Create separate physical dimensions from the primary dimension, which has only a subset of data (columns and rows) of interest. The new dimension can then be swapped instead of using the primary dimension at run time by joining the key of the swappable dimension to the foreign key inside the fact table. The fact table remains the same for the primary dimension and all swappable dimensions.
- Create one or more views based on the primary dimension. These views can be swapped at run time to join to the fact table instead of the primary dimension table.

  Figure 6-63 on page 295 shows a dimensional model for an order management system, with dimensions such as Product, Customer, Supplier, and Sales_Channel for which we have created swappable dimensions.

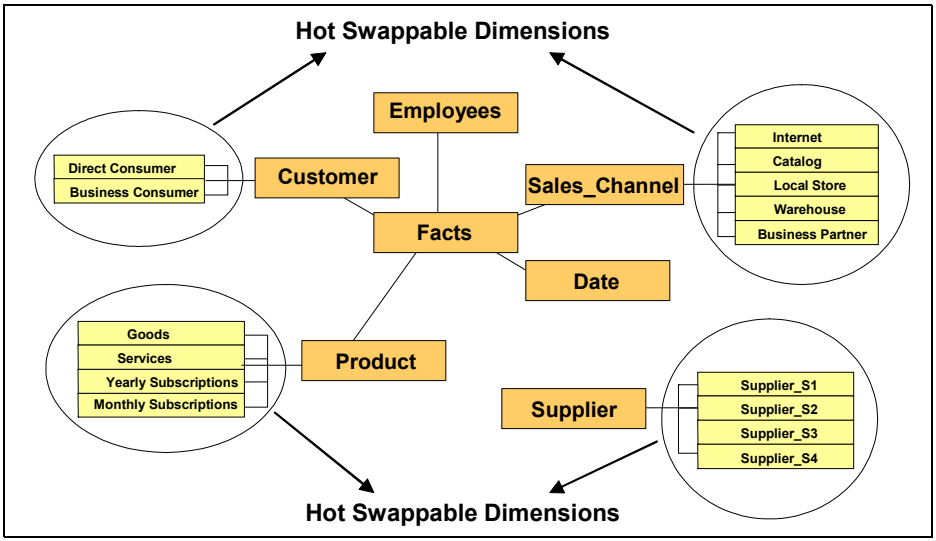


Figure 6-63 Hot swappable dimensions

As shown in Figure 6-63, all business users still use the same fact table “Sales_Fact” which is still stored in a single place. However, each business user may swap the different versions of dimensions at query time.

Note: An important concept behind using swappable dimensions is that each of the swappable dimensions can be used in place of the primary dimension to join to the same fact table. The swappable dimensions have less data when compared to the primary dimension and so this way fewer rows (of swapping dimension) join to the huge fact table, and therefore performance is faster. Also, using swappable dimensions, you can secure data by allowing the user to access only the swapped dimension which contains restricted data. This way the user views only the swapped dimension and not the entire primary dimension.

The description of each primary dimension and its corresponding swappable dimension is explained in Table 6-21 on page 296.

Table 6-21 Primary dimension and corresponding swappable dimensions

Primary dimension	Swappable dimension	Description for swappable dimension
Product	Goods	The Goods swappable dimension is derived from product dimension, and consists of all products which are goods.
	Services	The Services swappable dimension is derived from product dimension, which consists of all products which are offered as services.
	Yearly Subscription	The Yearly Subscription swappable dimension is derived from product dimension and consists of all products which are subscribed to yearly.
	Monthly Subscription	The Monthly Subscription swappable dimension is derived from product dimension and consists of all products which are subscribed to monthly.
Date	None	None
Sales_Channel	Internet	The Internet swappable dimension is derived from Sales_Channel dimension and consists of all products that are sold through the Web.
	Catalog	The Catalog swappable dimension is derived from Sales_Channel dimension and consists of all products which are sold through the catalog.
	Local_Store	The Local_Store swappable dimension is derived from Sales_Channel dimension and consists of all products that are sold through the local stores.
	Warehouse	The Warehouse swappable dimension is derived from Sales_Channel dimension and consists of all products that are sold through the warehouses.
	Business_ Partner	The Business_Partner swappable dimension is derived from the Sales_Channel dimension and consists of all products which are sold through the business partners.

Primary dimension	Swappable dimension	Description for swappable dimension
Supplier	Supplier_S1 (Goods)	The Supplier_S1 swappable dimension is derived from Supplier dimension, and consists of all products (goods) which are supplied by the supplier “S1”.
	Supplier_S2 (Goods)	The Supplier_S2 swappable dimension is derived from Supplier dimension, and consists of all products (goods) which are supplied by the supplier “S2”.
	Supplier_S3 (Services)	The Supplier_S3 swappable dimension is derived from Supplier dimension, and consists of all products (services) which are supplied by the supplier “S3”.
	Supplier_S4 (Subscriptions)	The Supplier_S4 swappable dimension is derived from Supplier dimension, and consists of all products (all subscriptions) which are supplied by the supplier “S4”.
Customer	Direct_ Consumer	The Direct_Consumer swappable dimension is derived from Customer dimension, and consists of all direct customers.
	Business_ Consumer	The Business_Consumer swappable dimension is derived from Customer dimension, and consists of all business customers.

6.4 Facts and fact tables

In this section, we discuss challenges that surface because of the particular attributes being used with facts and fact tables.

6.4.1 Non-additive facts

Non-additive facts are facts which cannot be added meaningfully across any dimensions. In other words, non-additive facts are facts where the SUM operator cannot be used to produce any meaningful results. Examples of non-additive facts include:

- **Textual facts:** Adding textual facts does not result in any number. However, counting textual facts may result in a sensible number.
- **Per-unit prices:** Adding unit prices does not produce any meaningful number. For example, unit sales price or unit cost is strictly non-additive

because adding these prices across any dimension will not yield a meaningful number. If instead we store the unit cost as an extended price (such as per-unit cost x quantity purchased), it is correctly additive across all dimensions.

- **Percentages and ratios:**

  A ratio, such as gross margin, is non-additive. Non-additive facts are usually the result of ratio or other calculations, such as percentages. Whenever possible, you should replace such facts with the underlying calculation facts (numerator and denominator) so you can capture the calculation in the application as a metric. It is also very important to understand that when adding a ratio, it is necessary to take sums of numerator and denominator separately and these totals should be divided.

- **Measures of intensity:** Measures of intensity such as the room temperature are non-additive across all dimensions. Summing the room temperature across different times of the day produces a totally non-meaningful number as shown in Figure 6-64. However, if we do an average of several temperatures during the day, we can produce the average temperature for the day, which is a meaningful number.

Month	Avg. Units Price
January	$10
February	$20
March	$30
Total	$60 X

(a) Average Units Price (Non-Additive)

Day	Temperature
Jan 9th (6 AM)	65 F
Jan 9th (7 AM)	65 F
Jan 9th (9 AM)	65 F
Total	195 F X

(b) Temperature (Non-Additive)

Figure 6-64 Non-additive facts

Note: You may avoid storing non-additive facts, such as unit prices, as facts in the fact table. Instead you may store fully additive facts such store quantity sold and amount charged (unit price x quantity). Then to get the unit price of the product, just divide the amount charged by the quantity.

- **Averages:** Facts based on averages are non-additive. For example, average sales price is non-additive. Adding all the average unit prices produces a meaningless result as shown in Figure 6-64.
- **Degenerate Numbers:** Degenerate dimensions are also non-additive. Number, such as order number, invoice number, tracking number,

confirmation number, and receipt number, are stored inside the fact table as degenerate dimensions. Counts of such numbers produce meaningful results such as total number of orders received or total number of line items in each distinct order.

Note: Non-additive facts, such as degenerate dimensions, may be counted. A count of degenerate dimensions, such as order numbers, produces meaningful results.

6.4.2 Semi-additive facts

Semi-additive facts are facts which can be summarized across some dimensions but not others. Examples of semi-additive facts include the following:

- Account balances
- Quantity-on-hand

We now discuss these types of semi-additive facts in more detail and see how to handle these in dimensional modeling and in reports.

(a) Account balances

Account balances are typically semi-additive facts. Consider a customer named Chuck whose account balance is shown in Figure 6-65. Chuck has an initial deposit of $1000 on January 1, 2005, withdraws $100 on January 2, deposits $600 on January 7, withdraws $500 on January 9, withdraws $300 on January 15, and finally deposits $1000 on January 31. His account balance at the end of January 2005 is $1700.

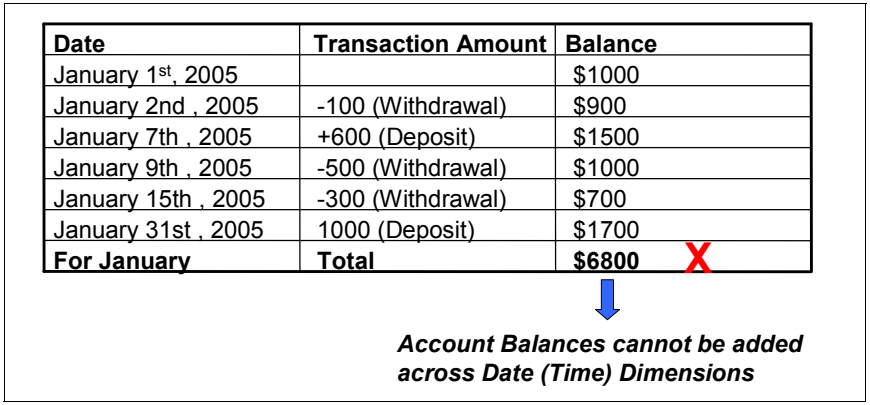


Figure 6-65 Account balances are semi-additive

As shown in Figure 6-65 on page 299, adding the monthly balances across the different days for the month of January results in an incorrect balance figure. However, if we average the account balance to find out daily average balance during each day of the month, it would be valid.

How can account balance be calculated across other dimensions?

We mentioned that a semi-additive fact is a fact that is additive across some dimensions but not others. Consider the star schema shown in Figure 6-66 on page 301. We see how and why the account balance is additive across the Customer, Branch, and Account dimensions, but not across the Date dimension. This is shown in Table 6-22.

Table 6-22 Account balance is semi-additive

Dimension	Account balance additive?	Why?
Date	**NO**	Explained in example in Figure 6-65
Branch	Yes	See SQL query in Example 6-7 and result in Figure 6-67.
Account	Yes	See SQL query in Example 6-8 and result in Figure 6-68.
Customer	Yes	See SQL query in Example 6-9 and result in Figure 6-69.

With the help of SQL queries, we can see that adding account balance along other dimensions, except date, can provide a meaningful measure for the total amount of money the bank is holding at any given point in time.

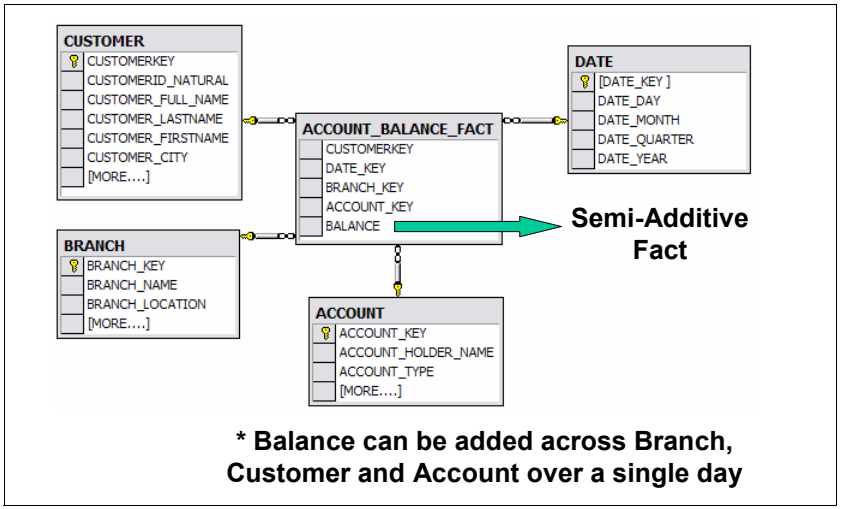


Figure 6-66 Account balance star schema

We use SQL queries to show that account balances for a single day are additive across the following dimension:

- Account balance is additive across Branch. It can be added at a single day across the branch dimension. The SQL code to show this is displayed in Example 6-7.

Example 6-7 Sample SQL showing balance is additive across branch dimension

```
Select B.BRANCH_NAME, SUM(F.BALANCE)
From
Branch B, Account_Balance_Fact F, Date D
where
B.BRANCH_KEY= F.BRANCH_KEY
and
F.DATE_KEY= D.DATE_KEY
and
D.DATE_DAY='1'
and
D.DATE_MONTH='January'
and
D.DATE_YEAR='2005'
GROUP BY B.BRANCH_NAME
```

The result of the query in Example 6-7 is shown in Figure 6-67 on page 302. It shows that the balance is additive at the branch level.

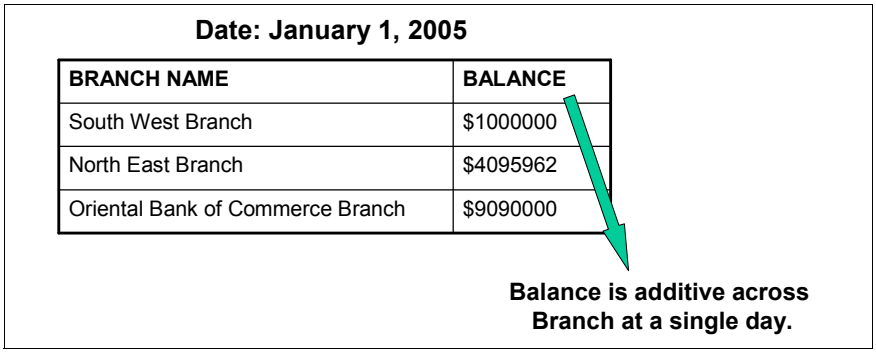


Figure 6-67 Balance is additive across the branch dimension

Note: It is important to understand that balance is semi-additive across the branch dimension only for a particular day. If we do not include the *where clause* for a day, then the SUM we get is an incorrect figure.

- Account balance is additive across Account dimension. Account balance can be added at a single day across the account dimension. The SQL code to do this is shown in Example 6-8.

Example 6-8 Sample SQL showing Balance is additive across Account dimension

```
Select A.ACCOUNT_TYPE, SUM(F.BALANCE)
From
Account A, Account_Balance_Fact F, Date D
where
A.ACCOUNT_KEY= F.ACCOUNT_KEY
and
F.DATE_KEY= D.DATE_KEY
and
D.DATE_DAY='1'
and
D.DATE_MONTH='January'
and
D.DATE_YEAR='2005'
GROUP BY A.ACCOUNT_TYPE
```

The result of the query shown in Example 6-8 is shown in Figure 6-68 on page 303. It shows that balance is additive across the Account dimension (per account type).

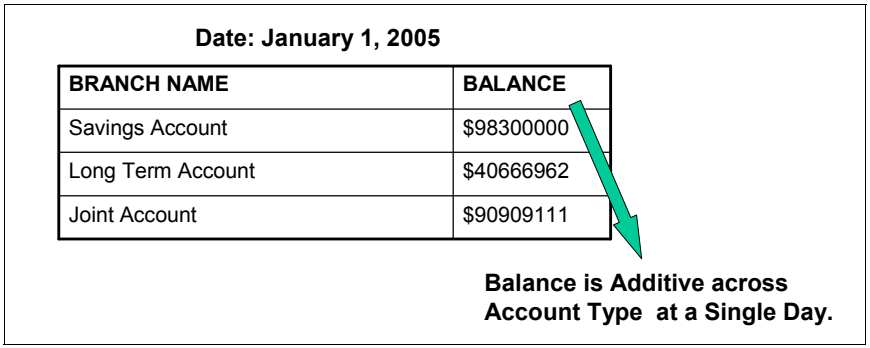


Figure 6-68 Balance is additive across the account dimension

Note: It is important to understand that balance is semi-additive across the branch dimension only for a particular day. If we do not include the *where clause* for a day, then the SUM we get is an incorrect figure.

- Account balance is additive across Customer dimension. It can be added at a single day across the customer dimension. The SQL code to do this is shown in Example 6-9.

Example 6-9 Sample SQL showing Balance is additive across Customer dimension

```
Select C.CUSTOMER_CITY, SUM(F.BALANCE)
From
Customer C, Account_Balance_Fact F, Date D
where
C.CUSTOMER_KEY= F.CUSTOMER_KEY
and
F.DATE_KEY= D.DATE_KEY
and
D.DATE_DAY='1'
and
D.DATE_MONTH='January'
and
D.DATE_YEAR='2005'
GROUP BY C.CUSTOMER_CITY
```

The result of the query shown in Example 6-9 is shown in Figure 6-69 on page 304. It shows that balance is additive per customer city.

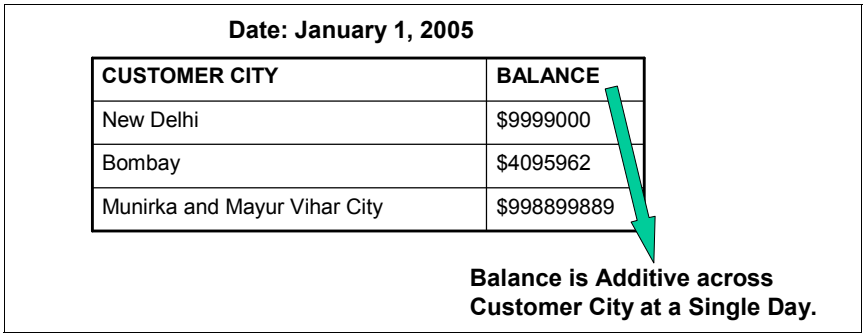


Figure 6-69 Balance is additive across the customer dimension

Note: It is important to understand that balance is semi-additive across the branch dimension only for a particular day. If we do not include the *where clause* for a day, then the SUM we get is an incorrect figure.

Also, if instead of choosing customer city we choose customer name, the sum of the balance could get summed up as a single value for all people having the same names. That is, if there were the customers with the name Suniti, all three balances would have summed up into a single name Suniti. Although balance is a semi-additive fact across the Customer dimension, it is for the user to decide whether or not the case for semi-additivity is good or not for a particular column.

(b) Quantity-on-hand (Inventory)

Quantity-on-hand (inventory or stock remaining) is another typical semi-additive fact. To understand this, consider a store that maintains inventory for each of its products at the end of every month. This is shown in Figure 6-70 on page 305. For a product named P99, the store has an inventory of 9000 at the end of January, 2005. For the same product P99, the store has an inventory of 5000 at the end of February, 8000 at the end of March, 1000 at the end of April, and 2000 at the end of May. The store's final inventory (quantity-on-hand) for product P99 at the end of June, is 1000.

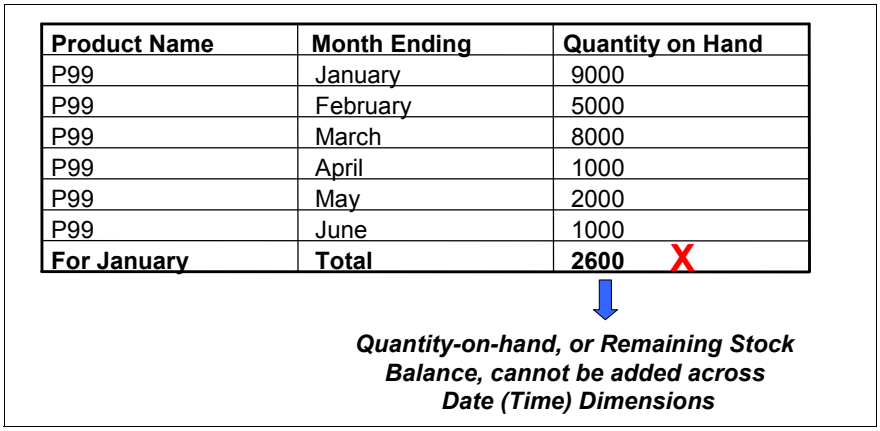


Figure 6-70 Quantity-on-hand is semi-additive

As shown in Figure 6-70, adding the month end quantity-on-hand stocks across the different months results in an incorrect balance figure. However, if we average the quantity on hand to find out the monthly average balance during each month of the year, it is valid.

How can Quantity-on-hand be calculated across other dimensions?

A semi-additive fact is a fact that is additive across some dimensions, but not others. Consider the star schema shown in Figure 6-71 on page 306. We can see that the quantity-on-hand is additive across the product and store dimensions but not across the Date dimension. This is shown in Table 6-23.

Table 6-23 Quantity-on-hand is semi-additive

Dimension	Account balance additive?	Why?
Date	**NO**	Explained in Figure 6-65.
Product	Yes	See SQL query in Example 6-7 and result in Figure 6-67.
Store	Yes	See SQL query in Example 6-8 and result in Figure 6-68.

With the help of SQL queries, we will see that adding quantity-on-hand along other dimensions such as store can provide a meaningful measure for the total quantity of products the stores are holding at any given point in time. We will also see that adding quantity of stock remaining across product (category) or product

(name) gives us an idea of the total stock remaining (across all stores) at a given point in time.

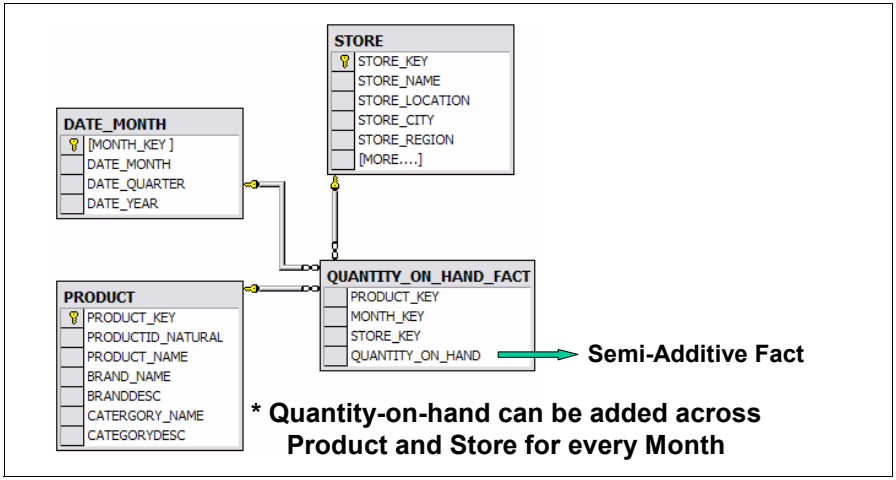


Figure 6-71 Store inventory on hand star schema

We use SQL queries to show that quantity-on-hand for every month is additive across the following dimensions:

- Store dimension. In Example 6-10, we provide the SQL code we used to show that Quantity-on-hand can be added for every month across the store dimension.

Example 6-10 Sample SQL: Quantity-on-hand additive across Store dimension

```
Select S.STORE_CITY, SUM(F.QUANTITY_ON_HAND)
From
Store S, Quantity_On_Hand_Fact F, Date_Month D, Product P
where
S.STORE_KEY= F.STORE_KEY
and
F.MONTH_KEY= D.MONTH_KEY
and
F.PRODUCT_KEY= P.PRODUCT_KEY
and
D.DATE_MONTH='January'
and
D.DATE_YEAR='2005'
and
P.PRODUCT_NAME='P99' // NOTE: Important to include otherwise we get a sum of
//all unrelated products
GROUP BY S.STORE_CITY
```

The result of the query from Example 6-10 on page 306 is shown in Figure 6-72. It shows the quantity-on-hand addition per store city for a particular product. However, it is important to note that you should include the product name in the where clause to specify a particular product. If we did not include the product in the where clause, we would get a report showing the sum of all products (P1 + P2 + P3 + P4 + P5 + ... P*n*) that are present in any store city. We certainly do not want that because if we add the sum of all remaining products (quantity-on-hand) for products such as CDs, pencils, paper, books, and pens, it would result in a meaningless value.

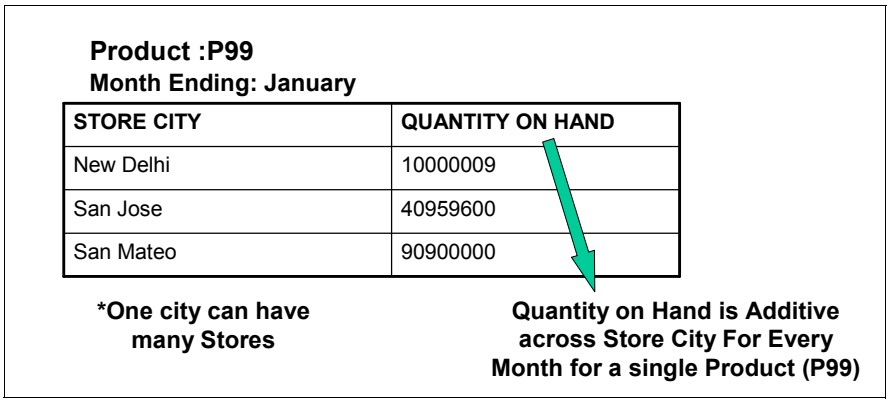


Figure 6-72 Quantity-on-hand is additive across the store dimension

Note: It is important to understand that balance is semi-additive across the branch dimension only for a particular day. If we do not include the *where clause* for a day, then the SUM we get is an incorrect figure.

- Quantity-on-hand is additive across product. It can be added for every month across the product (category) dimension. When we sum up all products present inside a category of a store, we have the quantity-on-hand for all products belonging to a particular category of a particular store. The SQL code to show this is depicted in Example 6-11.

Example 6-11 Sample SQL: Quantity-on-hand additive across product dimension

```
Select P.PRODUCT_CATEGORY, SUM(F.QUANTITY_ON_HAND)
From
Quantity_On_Hand_Fact F, Date_Month D, Product P, Store S,
where
F.MONTH_KEY= D.MONTH_KEY
and
F.PRODUCT_KEY= P.PRODUCT_KEY
and
```

```
F.STORE_KEY= S.STORE_KEY
and
D.DATE_MONTH='January'
and
D.DATE_YEAR='2005'
and
S.STORE_NAME='S1'

GROUP BY P.PRODUCT_CATEGORY
```

The result of the query shown in Example 6-11 on page 307 is shown in Figure 6-73. However, there is an important note here. It is important to include the store name in the where clause to make it a particular store. If we do not include the store in the where clause, we would get a report showing the sum of all products belonging to the same category (P1 + P2 + P3 + P4 + P5 + ...P*n*) that are present in any store. The result shows that quantity-on-hand is additive per product category per store.

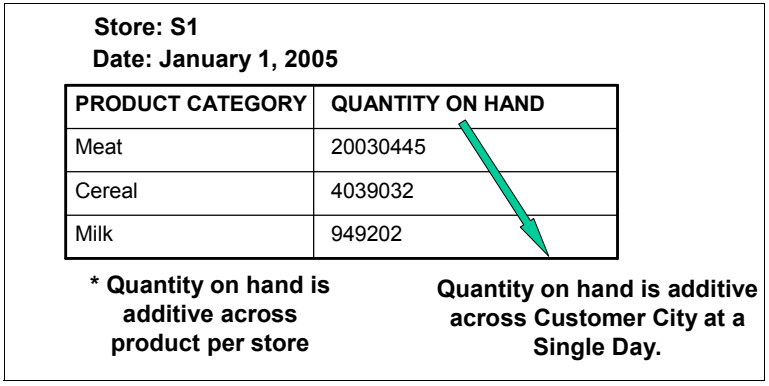


Figure 6-73 Quantity-on-hand is additive across the product dimension

Note: It is important to understand that "quantity on hand" is semi-additive across the branch dimension only for a particular day. If we do not include the *where clause* for a day, then the SUM we get is incorrect.

6.4.3 Composite key design for fact table

A fact table primary key is typically comprised of multiple foreign keys, one from each dimension table. Such a key is called a *composite or concatenated primary key*. It is not a mandatory rule to have all the foreign keys included as the primary key of the fact table. Sometimes, only a few combinations of foreign keys will be used.

Also, it is not always true that the combination of all foreign keys of the dimensions in the fact table will guarantee the uniqueness of the fact table primary key. In such situations, you may need to include the degenerate dimension as a component inside the primary key of the fact table. It is mandatory that such a primary key be unique.

Note: It is important to understand that some or all foreign keys (of dimensions) present inside the fact table may guarantee uniqueness and may be used to create the primary key of the fact table.

Composite primary keys and uniqueness

Does the composite primary key design consisting of all dimension foreign keys guarantee uniqueness? The answer is that uniqueness of the composite primary key of the fact table is guaranteed by the grain definition and it is not a rule that all dimension keys will always be unique if the grain definition is not enforced properly. For example, consider the fact table shown in Figure 6-74 on page 310. The granularity of the fact table is at the day level. Therefore, we need to sum all of the transactions to the day level. That would be the definition of the fact.

Consider product dimension, store dimension, date dimension, and fact table as shown in Figure 6-74 on page 310. Assume that the product dimension has only two bicycle brands - C1 and C2. The date dimension is at the day level and contains one row for each day.

Assume that you sell the following number of bicycles on October 22, 2005:

- 4 Bicycles of Brand C1 in the morning at 8:00 a.m.
- 5 Bicycles of Brand C1 in the evening at 6:00 p.m., before the store closes.

Since the grain of the fact table in Figure 6-74 on page 310 is one row for each product sold in each store, we will have one row inserted for the sales of Bicycle of Brand C1 on October 22, 2005. In this single row, the fact (Sales_Quantity) would be 9 (4 + 5).

Grain guarantees UNIQUENESS of the composite fact primary key

If we design the primary key of the fact table (see Figure 6-74 on page 310) as the combination of the foreign keys of all dimensions, then the (Retail_Sales) primary key would be composite key of Product_ID, Store_ID, and Date_ID. This primary key is guaranteed to be UNIQUE because of the fact that only one row will be inserted for each product in each store on a single day. The composite primary key UNIQUENESS is guaranteed by the GRAIN of the fact table which says that only ONE row is inserted for each product in each store (no matter how many times it is sold during the day).

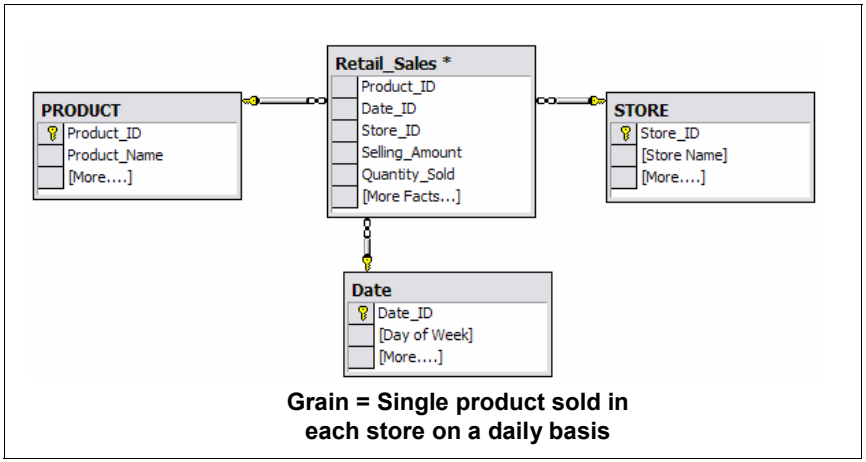


Figure 6-74 Retail store schema

Suppose that we want to report sales on an hourly basis. To do so, we design a star schema as shown in Figure 6-75 on page 311. Assume during a day (October 14, 2005) we have the following sales:

- 5 Bicycles of Brand C1 sell at 8:00 a.m.
- 5 Bicycles of Brand C1 sell at 8:30 a.m.
- 2 Bicycles of Brand C1 sell at 11:00 a.m.
- 3 Bicycles of Brand C1 sell at 4:00 p.m.
- 2 Bicycles of Brand C1 sell at 8:00 p.m.

Since the grain of the fact table in Figure 6-75 on page 311 is one row for each product sold in each store on an hourly basis, we have 4 rows (instead of five) inserted for the sales of Bicycles of Brand C1 for October 14, 2005.

Note: Only one single row (Selling_Amount=10) will be inserted, two different times (8:00 a.m. and 8:30 a.m.) because the grain definition asks us to track sales on an hourly basis. If several sales occur for the same product in the same hour, then only one single row would be inserted for the product for that hour. In other words, the UNIQUENESS of the fact table row (fact composite primary key) is guaranteed by the GRAIN definition of the fact table.

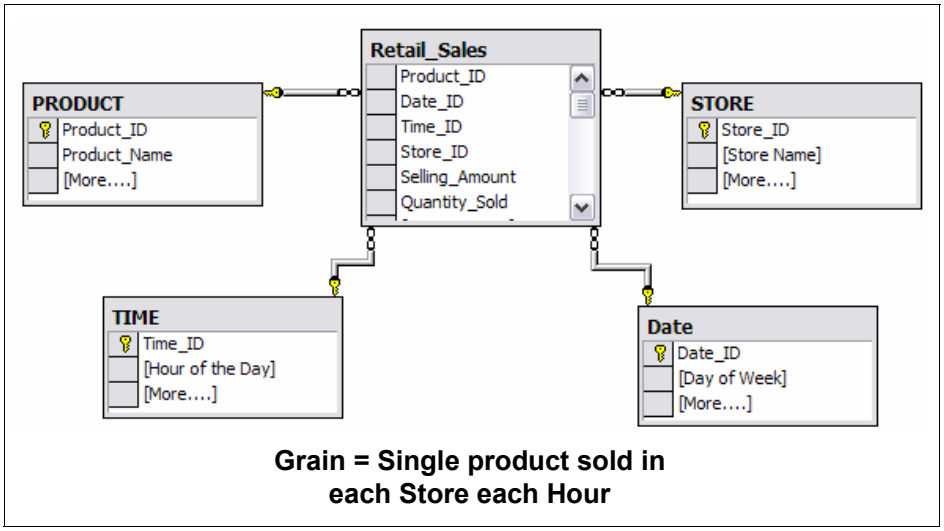


Figure 6-75 Sales schema to report hourly sales

If we design the primary key of the fact table in Figure 6-75 as a composite key consisting of the Product_ID, Date_ID, Time_ID, and Store_ID, the primary key would be guaranteed to be UNIQUE because the GRAIN of the fact table assures that only a single row will be inserted for a single product sold on an hourly basis.

Note: One or more degenerate dimensions may be needed to guarantee the uniqueness of the fact table row. A detailed discussion is available in 5.5.6, “Composite key design” on page 177.

6.4.4 Handling event-based fact tables

Event fact tables are tables that record events. For example, event fact tables are used to record events such as Web page clicks and employee or student attendance. Events, such as a Web user clicking on a Web page of a Web site, do not always result in facts. In other words, millions of such Web page click events do not always result in sales. If we are interested in handling such event-based scenarios where there are no facts, we use event fact tables which consist of either pseudo facts or these tables have no facts (factless) at all.

From a conceptual perspective, the event-based fact tables capture the many-to-many relationships between the dimension tables. To better understand the concept of event tables, consider the following two examples:

- Example one: Hospital and insurance revenue example

- Example two: Employee attendance recording

Note: Event-based fact tables may have pseudo facts or no facts (factless facts) at all. We discuss factless fact tables in Example one: Hospital and insurance revenue example and pseudo facts in “Example two: Employee attendance recording” on page 315.

Assume an Internet Service Provider is tracking Web sites and their Web pages, and all the visitor mouse clicks. We consider each of these examples:

Example one: Hospital and insurance revenue example

Consider an example of a hospital which is very busy with patients for the entire day. Figure 6-76 on page 313 shows a simple star schema that can be used to track the daily revenue of the hospital. The grain of the star schema is a single patient visiting the hospital in a single day. The following are the dimension tables in the star schema:

- Insurance dimension: Contains one row for each insurance the patient has
- Date dimension: The data at the daily level
- Hospital dimension: A single row for each hospital
- Customer dimension: One row for each customer

The insurance dimension consists of a *Not applicable* row or the *Does not have insurance* row. We discussed the concept of *Not applicable* rows that may be present inside a dimension table in “Insert a special customer row for the “Not applicable” scenario” on page 150.

If a patient visits a hospital but does not have insurance, then the row in the insurance dimension will be the *Does not have insurance row*. However, if a patient visits a hospital and has insurance, then the insurance row specifying the type of insurance is attached to the fact table row.

What this means is that there may be several patients that have insurance, but they are visible inside the fact table only when they visit the hospital. Once they visit the hospital, a row appears for these patients inside the fact table HOSPITAL_DAILY_REVENUE.

So if we have one million customers who have purchased insurance in a month and only 1 000 of these visit the hospital, then we would not be able to tell which of the 999 000 customers had insurance, but did not visit the hospital.

From the star schema shown in Figure 6-76 on page 313, we can know about a customer (patient) having insurance only after the customer visits the hospital and appears in the fact table. This is primarily because the HOSPITAL_DAILY_REVENUE fact table records sales-related activity. The

customer appears in this sales table if, and only if, the customer visits the hospital and pays for the treatment received. If a customer has insurance, but does not visit any hospital, we cannot tell whether or not they have insurance just by looking at the schema shown in Figure 6-76. This is also because of the fact that the Hospital and Insurance are independent dimensions only linked by the HOSPITAL_DAILY_REVENUE fact table.

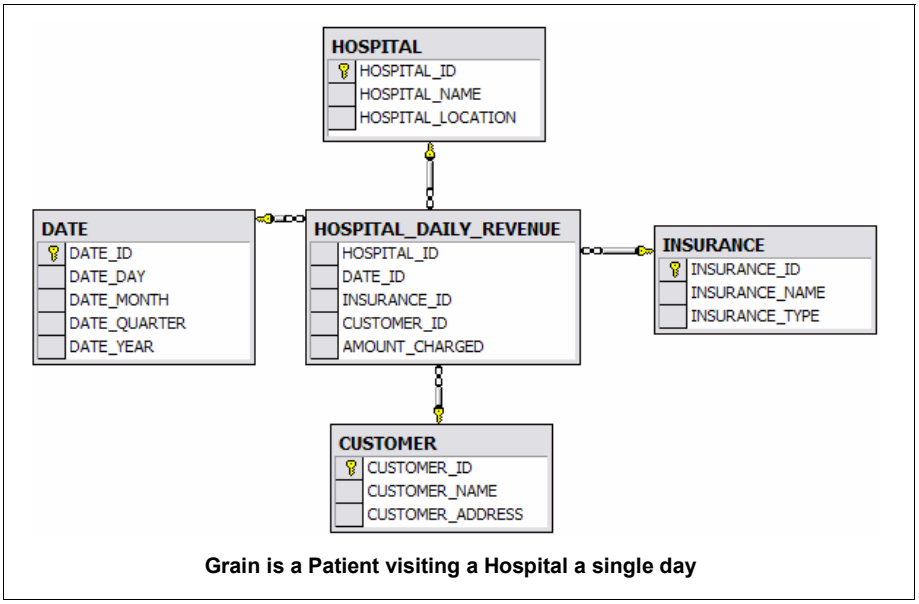


Figure 6-76 Hospital revenue generating schema

The hospital schema shown in Figure 6-76 can only tell if a customer has insurance if the customer visits the hospital.

How to find customers who have insurance but who never came to the hospital.

To find out about customers who have insurance, but who have not visited the hospital, there are two options:

- Insert a single row for all customers in the fact table (see Figure 6-76) on a daily basis, irrespective of whether or not they visit the hospital. If the customer does not visit the hospital, then have the fact AMOUNT_CHARGED show a value of “0”. However, we do not recommend this option of inserting a row for each customer on a daily basis irrespective of whether the customer visited the hospital, because the fact table will grow at a very fast rate and result in performance issues.

- Create the event fact table star schema as shown in Figure 6-77 on page 314, which consists of the following tables:
  - Date dimension: Contains the date on which the person is insured.
  - Insurance dimension: One row for each type of insurance.
  - Customer dimension: A single row for each customer.
  - Insurance_Event _Fact table: An event-based factless fact table. It does not record anything other than the fact that a customer is insured. A customer getting insured is treated as an *Event* and so a row is inserted into the INSURANCE_EVENT_FACT table.

Note: The INSURANCE_EVENT_FACT fact table is a factless fact table. This factless fact table contains only foreign keys and has no facts. It is a factless fact table that represents the many-to-many relationship between the various dimensions such as date, customer, and insurance.

Finding customers who have insurance but never visited a hospital:

To find out about customers who have insurance but never visited any hospital, we do the following:

1. From Figure 6-76 on page 313, find all customers who visited the hospital.
2. From Figure 6-77, find all customers who have insurance.
3. Find the set difference from step 1 and 2. This would tell you which customers had insurance but did not visit the hospital.

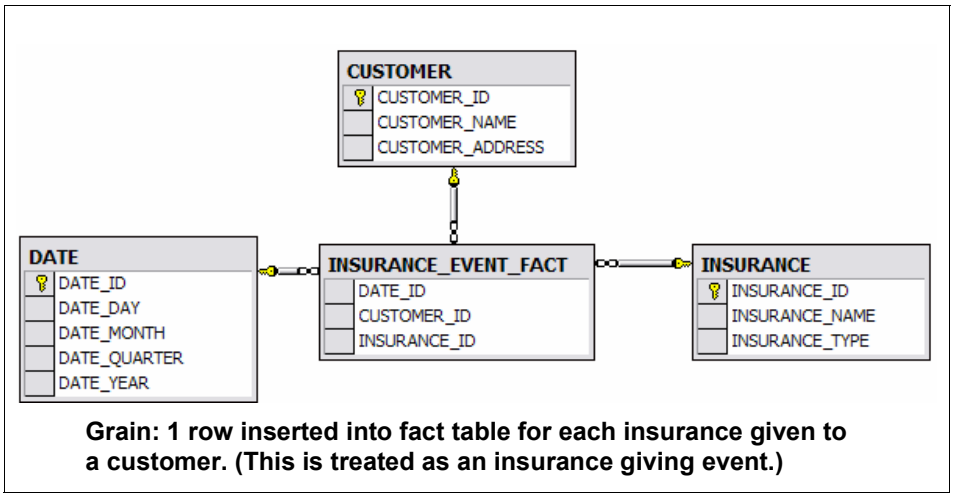


Figure 6-77 Insurance event-based star schema

Note: The factless table shown in Figure 6-77 is used in conjunction with an ordinary sales fact table (see Figure 6-76) to answer the question *which customers had insurance but did not go to the hospital?*

Example two: Employee attendance recording

In this section we discuss an event fact table consisting of pseudo facts. The event-based pseudo fact table is designed for recording the attendance of all employees working for a company. Important points to consider while designing the attendance recording system for the company are:

- Attendance needs to be recorded for employees in the company.
- Each employee belongs to a department.
- A department may or may not have employees.
- Attendance is recorded on a daily basis. The time of the employees attending the office is also mandatory.
- An employee may or may not have a manager. Some employees do not have managers because they are their own managers.
- The reporting manager of the employee also needs to be tracked.

For the requirements stated above, we designed a star schema as shown in Figure 6-78 on page 316. The grain of the attendance tracking schema is *attendance of a single employee every day*. The star schema consists of the following dimensions:

- Date dimension: Consists of date data at the day level.
- Time dimension: Contains the time of day and the description of the event, such as On Time, Late, Early Shift, Afternoon Shift, Night Shift, or Absent.
- Employee dimension: Data about the employee.
- Department: Department number to which an employee belongs. This table also contains a *Belongs to No Department* row for employees who have no department.
- Reporting Manager: Contains all the managers information. It also contains a row called *No Manager Assigned Yet* and another row called *Self-Managed*. These special rows are for people who have yet not been assigned managers or for people who are their own managers.
- Project dimension: All the projects in the company. This table also contains a special row called *No Project Assigned Yet*. This is for employees who have not been assigned any project, or are into administrative work.

- Client dimension: One row for each company client. This table also contains a special row called *Employee does not work for Client,* for employees who have no clients as of yet.

The star schema shown in Figure 6-78 contains the EMPLOYEE_DAILY_ATTENDANCE fact table. The grain of this fact table is one row per employee attendance during each day. One row is inserted into the EMPLOYEE_DAILY_ATTENDANCE fact table each time an employee swipes his badge while entering the company. The fact, called ATTENDANCE_COUNT, gets a value of 1 when the attendance is recorded. For employees who did not attend during a given day, a row is inserted with ATTENDANCE_COUNT equal to 0.

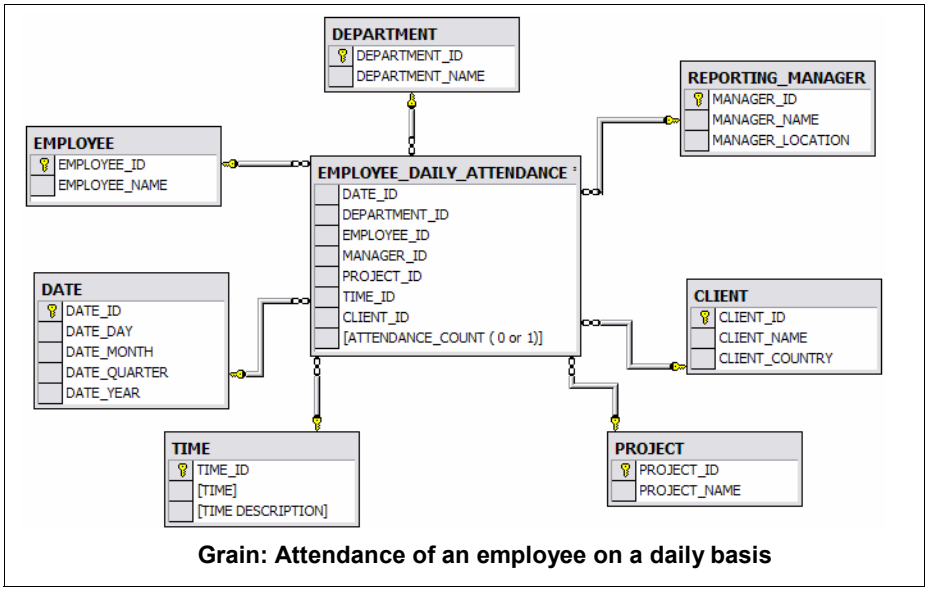


Figure 6-78 Employee attendance recording

To count the total employee attendance for a month, query on the EMPLOYEE_DAILY_ATTENDANCE fact table and restrict data for an employee with ATTENDANCE_COUNT=1. We use the COUNT() function to count the employee attendance. We could also use the SUM() function and get the same result.

Similarly if we want to find the number of times an employee has been absent in a year, we query the EMPLOYEE_DAILY_ATTENDANCE fact table, restrict the data for the employee for a particular year, and set ATTENDANCE_COUNT=0. We use the COUNT() function instead of SUM because the SUM of all ATTENDANCE_COUNT=0 will give us 0 where the COUNT of

ATTENDANCE_COUNT=0 will give the number of times an employee was absent.

Why is the ATTENDANCE_COUNT a pseudo fact?

It is important to understand that ATTENDANCE_COUNT is a pseudo fact. Instead of using a “1” for employee presence, we could have used any other number or even a character like “Y”. Similarly for tracking the absence of employees, we could have set ATTENDANCE_COUNT equal to a character such as “N” or alternatively have used any number other than “0”.

Assume that for tracking employees present on a daily basis, we set ATTENDANCE_COUNT=’10’. In this case we cannot SUM the attendance for a month (30 days x 10) and say that the attendance is 300. However, if we use COUNT, we get a correct number 30, assuming the employee worked for 30 days.

Similarly, assume that for tracking employee absence on a daily basis, we set ATTENDANCE_COUNT=’9’, which signifies that the employee is ABSENT. In this case we cannot SUM the attendance for a month (30 days x 9) and say that the absence is 270 times. However, if we use COUNT, we get a correct number 30, assuming the employee was absent for 30 days.

(c) Internet Web page click event tracking

Figure 6-79 on page 318 shows a factless fact table star schema to track the Web sites and each of the pages that has been visited. The grain is a *single click to a Web page*. It is important to understand that the fact table named WEB_PAGE_CLICK_FACT is an event-based table and does not have any facts. The WEB_PAGE_CLICK_FACT fact table is used to represent a Web page click event. The fact table captures many-to-many relationships between the dimension tables and does not contain any facts. To calculate the total number of Web page clicks per day, we can use the COUNT function on all the combination of foreign keys.

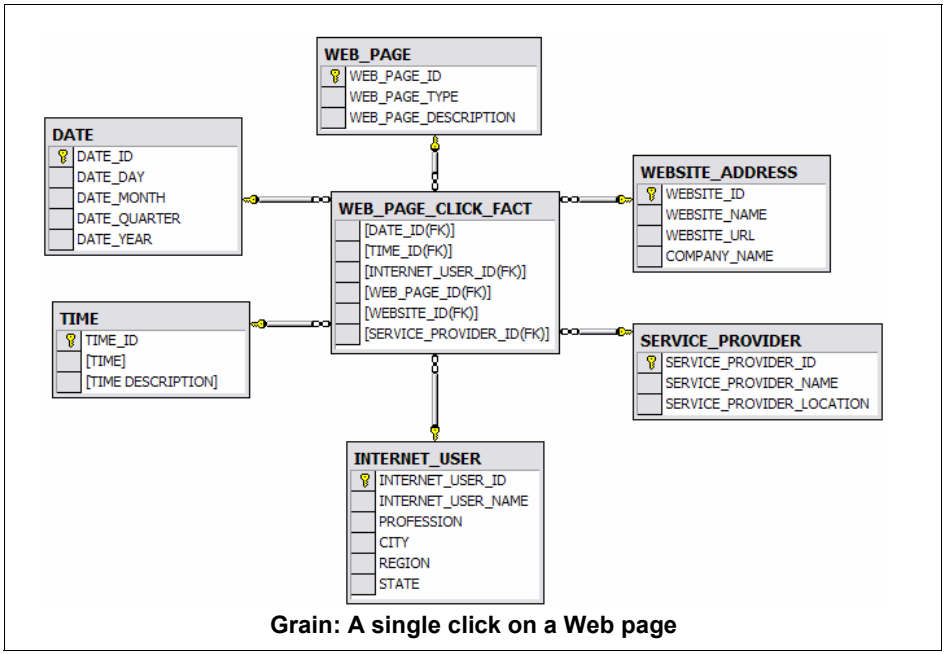


Figure 6-79 Web page clicking event star schema

6.5 Physical design considerations

In this section, we discuss the following topics:

- DB2 Optimizer and MQTs for aggregate navigation
- Indexing for dimension and fact tables

6.5.1 DB2 Optimizer and MQTs for aggregate navigation

DB2 Cube Views™ uses DB2 summary tables to improve the performance of queries issued to cube models and cubes. A summary table is a special type of a materialized query table (MQT) that specifically includes summary data.

You can complete expensive calculations and joins for queries ahead of time and store that data in a summary table. When you run queries that can use the precomputed data, DB2 UDB will reroute the queries to the summary table, even if the query does not exactly match the precomputed calculations. By using simple analytics such as SUM and COUNT, DB2 UDB can dynamically aggregate the results from the precomputed data, and many different queries can be satisfied by one summary table. Using summary tables can dramatically

improve query performance for queries that access commonly used data or that involve aggregated data over one or more dimensions or tables.

Figure 6-80 shows a sample dimensional model based on a snowflake schema with a Sales facts table, and Time, Market, and Product dimensions. The fact table has measures and attributes keys, and each dimension has a set of attributes and is joined to the facts object by a facts-to-dimension join.

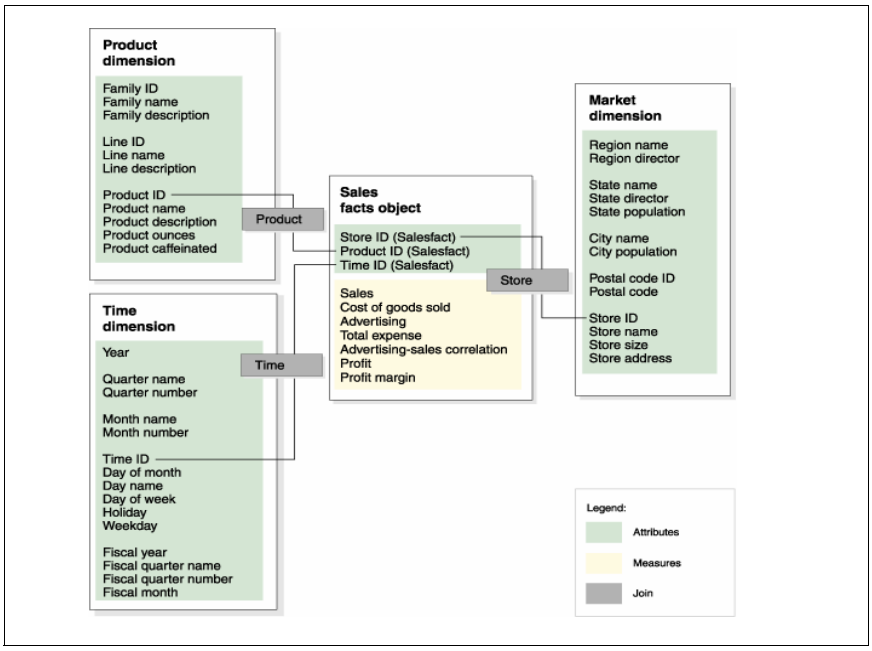


Figure 6-80 Sales dimensional model

The hierarchy for each dimension in the dimensional model is shown in Figure 6-81 on page 320. The boxes connected by the thick dark lines across the hierarchies represent the data that actually exists in the base tables. Sales data is stored at the Day level, Store level, and Product level. Data above the base level in the hierarchy must be aggregated. If you query a base table for sales data from a particular month, DB2 UDB dynamically adds the daily sales data to return the monthly sales figures.

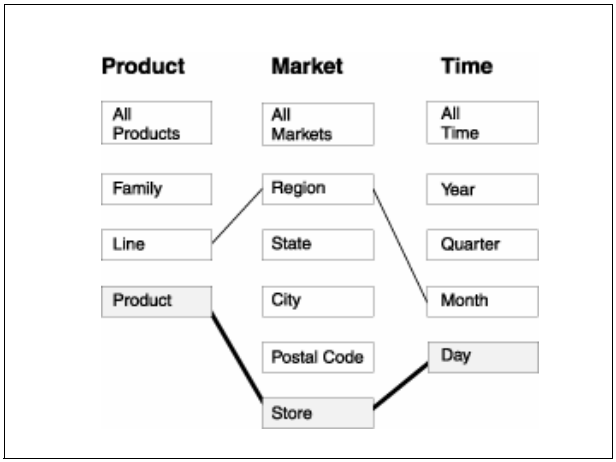


Figure 6-81 Hierarchy showing product, market, and time

The sample query in Example 6-12 shows the sales data for each product line, in each region, by each month in 2004.

Example 6-12 Sample SQL showing sales by product line, region, and month

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM TIME, STORE, LOCATION, PRODUCT, LINE, SALESFACT
WHERE SALESFACT.STOREID = STORE.STOREID
  AND STORE.POSTALCODEID = LOCATION.POSTALCODEID
  AND SALESFACT.PRODUCTID = PRODUCT.PRODUCTID
  AND PRODUCT.LINEID = LINE.LINEID
  AND SALESFACT.TIMEID = TIME.TIMEID
  AND YEAR = '2004'
 GROUP BY LINEID, MONTH_NUMBER;
```

The line connecting Line-Region-Month in Figure 6-81 represents the slice that the query accesses. It is a slice of the dimensional model and includes one level from each hierarchy. You can define summary tables to satisfy queries at or above a particular slice. A summary table can be built for the Line-Region-Month slice that is accessed by the query. Any other queries that access data at or above that slice including All Time, Year, Quarter, All Markets, All Products, and Family can be satisfied by the summary table with some additional aggregating. However, if you query more detailed data below the slice, such as Day or City, the summary table cannot be used for this more granular query.

In Figure 6-82 on page 321, a dotted line defines the Line-State-Month slice. A summary table built for the Line-State-Month slice can satisfy any query that accesses data at or above the slice. All of the data that can be satisfied by a

summary table built for the Line-State-Month slice is included in the set of boxes inside the “dashed line” area.

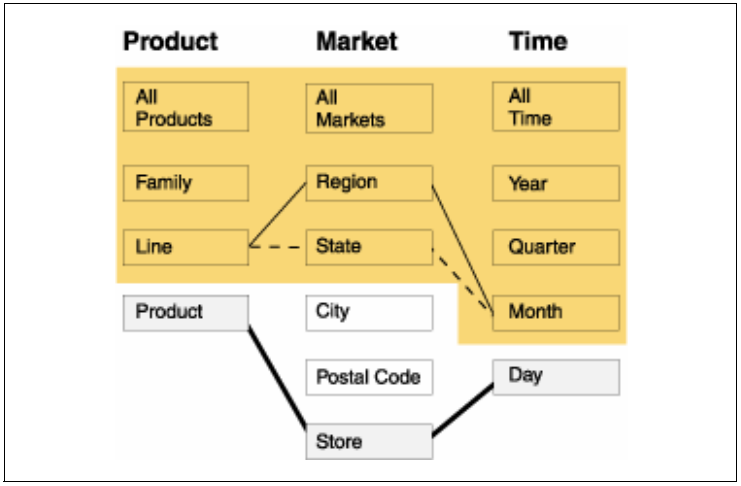


Figure 6-82 Summary table slice

The rewriter in the DB2 SQL compiler knows about existing summary tables, and can automatically rewrite queries to read from the summary table instead of the base tables. Rewritten queries are typically much faster because the summary tables are usually much smaller than the base tables and contain preaggregated data. Queries continue to be written against the base tables. DB2 UDB decides when to use a summary table for a particular query and will rewrite the query to access the summary tables instead, as shown in Figure 6-83 on page 322. The rewritten query accesses a summary table that contains preaggregated data. A summary table is often significantly smaller, and therefore significantly faster, than the base tables and returns the same results as the base tables.

You can use the DB2 EXPLAIN facility to see if the query was rerouted, and if applicable, to which table it was rerouted.

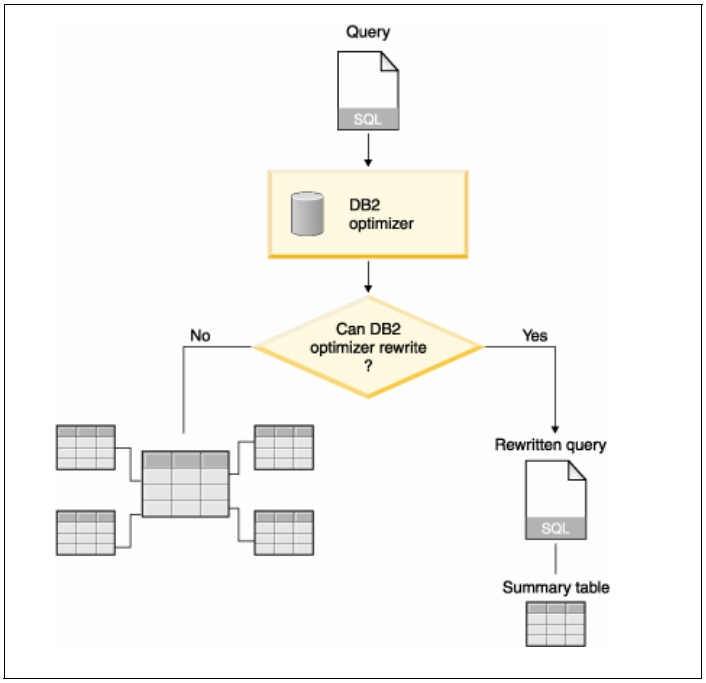


Figure 6-83 Query Rewrite using DB2 Optimizer

The query to see the sales data for each product line, in each region, by each month in 2004, can be rewritten to use the summary table built for the Line-Region-Month slice. The original query is shown in Example 6-13.

Example 6-13 Sample SQL to show original query

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM TIME, STORE, LOCATION, PRODUCT, LINE, SALESFACT
WHERE SALESFACT.STOREID = STORE.STOREID
  AND STORE.POSTALCODEID = LOCATION.POSTALCODEID
  AND SALESFACT.PRODUCTID = PRODUCT.PRODUCTID
  AND PRODUCT.LINEID = LINE.LINEID
  AND SALESFACT.TIMEID = TIME.TIMEID
  AND YEAR = '2004'
 GROUP BY LINEID, MONTH_NUMBER;
```

The rewritten query is shown in Example 6-14.

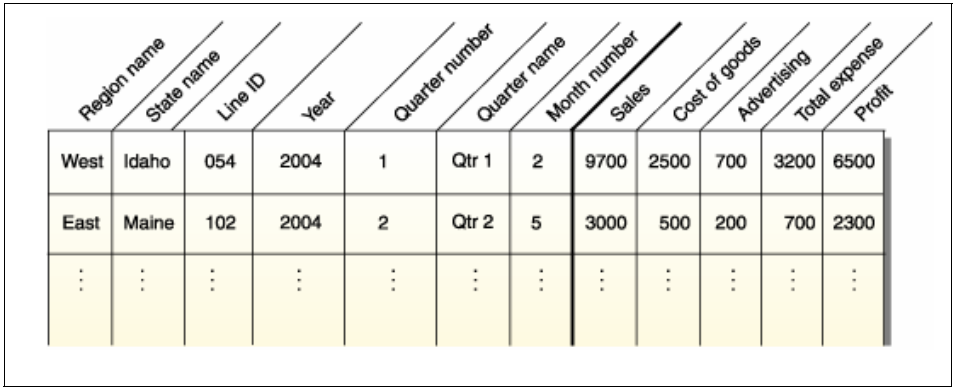
Example 6-14 SQL to show rewritten query

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM SUMMARYTABLE1
WHERE YEAR = '2004'
```

```
GROUP BY LINE_ID, REGION_NAME, MONTH_NUMBER;
```

The rewritten query is much simpler and quicker for DB2 UDB to complete because the data is preaggregated and many of the table joins are precomputed so DB2 UDB accesses one small table instead of six tables, including a large fact table. The savings with summary tables can be tremendous, especially for schemas that have large fact tables. For example, a fact table with 1 billion rows might be preaggregated to a summary table with only 1 million rows, and the calculations involved in this aggregation occur only once instead of each time a query is issued. A summary table that is 1000 times smaller is much faster than accessing the large base tables.

In this example, Figure 6-84 shows the summary table for the Line-State-Month slice. DB2 UDB needs to calculate data for Region from the higher level State instead of from the lower level Store, so the summary table has fewer rows than the base tables because there are fewer states than stores. DB2 UDB does not need to perform any additional calculations to return sales data by Month and Line because the data is already aggregated at these levels. This query is satisfied entirely by the data in the summary table that joins the tables used in the query ahead of time and the joins do not need to be performed at the time the query is issued. For more complex queries, the performance gains can be dramatic.



Region name	State name	Line ID	Year	Quarter number	Quarter name	Month number	Sales	Cost of goods	Advertising	Total expense	Profit
West	Idaho	054	2004	1	Qtr 1	2	9700	2500	700	3200	6500
East	Maine	102	2004	2	Qtr 2	5	3000	500	200	700	2300
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 6-84 Summary table created for line-region-month slice

In some cases, a query might access an attribute that is related to an attribute that is included in the summary table. The DB2 optimizer can use functional dependencies and constraints to dynamically join the summary table with the appropriate dimension table.

When the Optimization Advisor recommends a summary table, all of the measures in the dimensional model are included. In this example, the

SalesFacts object has only five measures including sales, cost of goods, advertising, total expense, and profit, which are all included in the summary table. If you define fifty measures for your dimensional model, all fifty measures are included in the summary table. The Optimization Advisor does not need to include all of the related attributes that are defined for a level in the summary table because DB2 Cube Views defines functional dependencies between the attributes in a level.

6.5.2 Indexing for dimension and fact tables

In this section we show how to design indexes for one particular star schema. The output of our activity is data definition language (DDL).

We also show the different access paths the RDBMS optimizer uses for indexed and non-indexed star schemas.

We have a star schema with three dimensions (date, customer, and product) and a fact table called SALES. The fact table has three foreign keys. The three foreign keys together with the degenerate dimension INVOICE_NO (Invoice Number) forms the primary key of fact table. This is shown in Figure 6-85 on page 325.

The following are the hierarchies involved with the dimensions:

- The hierarchy of the PRODUCT dimension is Supplier → Beverage group → Beverage → Name.
- The hierarchy for the CUSTOMER dimension is Region → Country → State City → Name.
- The hierarchy with the DATE dimension is Year → Quarter → Month_of_year → Day.

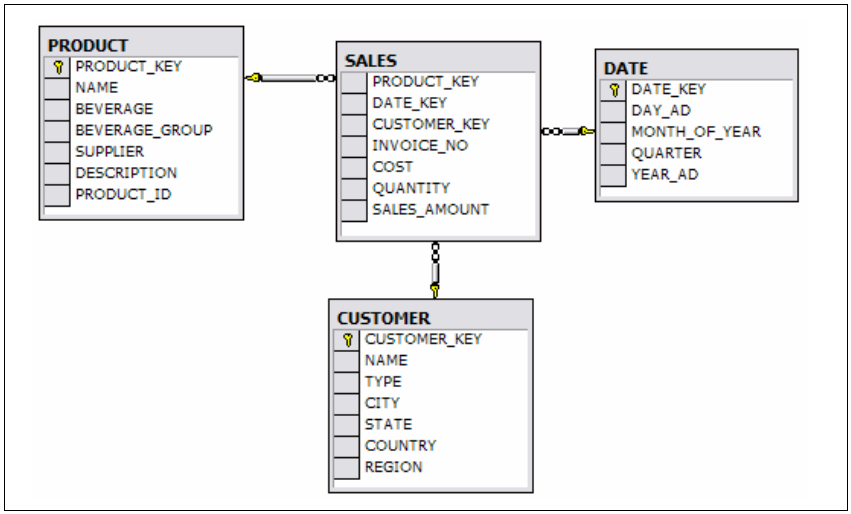


Figure 6-85 Star Schema

First of all we design indexes for dimensions tables

- Primary key (PK), if not generated by RDBMS
- Unique indexes for each dimension hierarchy supporting joins (create them if they are not generated by RDBMS automatically):

```
CREATE INDEX CUSTOMER_PK_IDX ON CUSTOMER (CUSTOMER_KEY)
CREATE INDEX PRODUCT_PK_IDX ON PRODUCT(PRODUCT_KEY)
CREATE INDEX DATE_PK_IDX ON DATE (DATE_KEY)
```

- Non-unique indexes for each dimension's hierarchy level:

```
CREATE INDEX CUSTOMER_NA_IDX ON CUSTOMER (NAME,CUSTOMER_KEY)
CREATE INDEX CUSTOMER_TY_IDX ON CUSTOMER (TYPE,CUSTOMER_KEY)
CREATE INDEX CUSTOMER_CI_IDX ON CUSTOMER (CITY,CUSTOMER_KEY)
CREATE INDEX CUSTOMER_ST_IDX ON CUSTOMER (STATE,CUSTOMER_KEY)
CREATE INDEX CUSTOMER_CO_IDX ON CUSTOMER (COUNTRY,CUSTOMER_KEY)
CREATE INDEX CUSTOMER_RE_IDX ON CUSTOMER (REGION,CUSTOMER_KEY)


CREATE INDEX DATE_D_IDX ON DATE (DAY_AD,DATE_KEY)
CREATE INDEX DATE_M_IDX ON DATE (MONTH_OF_YEAR,DATE_KEY)
CREATE INDEX DATE_Q_IDX ON DATE (QUARTER,DATE_KEY)
CREATE INDEX DATE_Y_IDX ON DATE (YEAR_AD,DATE_KEY)


CREATE INDEX PRODUCT_NA_IDX ON PRODUCT (NAME,PRODUCT_KEY)
```

```
CREATE INDEX PRODUCT_BE_IDX ON PRODUCT (BEVERAGE,PRODUCT_KEY)
CREATE INDEX PRODUCT_BG_IDX ON PRODUCT (BEVERAGE_GROUP,PRODUCT_KEY)
CREATE INDEX PRODUCT_SU_IDX ON PRODUCT (SUPPLIER,PRODUCT_KEY)
```

Note: Including the PK of the dimensional table in the index eliminates additional fetching of the PK values from the dimension table.

- Non-unique Index for the product attribute product_id

```
CREATE INDEX PRODUCT_PID_IDX ON PRODUCT (PRODUCT_ID,PRODUCT_KEY)
```

- Optionally create indexes for selected dimension hierarchy:

```
CREATE INDEX PRODUCT_H1_IDX ON PRODUCT (BEVERAGE,NAME,PRODUCT_KEY)
```

Note: Including the dimension hierarchy in an index eliminates additional fetching of attributes from dimension tables. This approach is not typically applicable, because the index tables can be quite large.

Index for fact table foreign keys

```
CREATE INDEX SALES_FK_PROD_IDX ON SALES (PRODUCT_KEY)
CREATE INDEX SALES_FK_CUST_IDX ON SALES (CUSTOMER_KEY)
CREATE INDEX SALES_FK_DATE_IDX ON SALES (DATE_KEY)
```

Note: After each change in the physical database you should update statistics on tables and indexes. Following is the DB2 command for this action:

```
RUNSTATS ON TABLE TEST.SALES ON ALL COLUMNS WITH DISTRIBUTION ON ALL
COLUMNS AND INDEXES ALL

RUNSTATS ON TABLE TEST.PRODUCT ON ALL COLUMNS WITH DISTRIBUTION ON
ALL COLUMNS AND INDEXES ALL

RUNSTATS ON TABLE TEST.CUSTOMER ON ALL COLUMNS WITH DISTRIBUTION ON
ALL COLUMNS AND INDEXES ALL

RUNSTATS ON TABLE TEST.DATE ON ALL COLUMNS WITH DISTRIBUTION ON ALL
COLUMNS AND INDEXES ALL
```

Differences between indexed and non-indexed star schemas

In the following section we show the differences between the access paths the RDBMS optimizer chooses for indexed and non-indexed star schemas for D*icing*

and Slicing. Generally if an index is not created, the RDBMS must read all tables in one continuous scan.

Access plans for OLAP activity

In this section we show the access plans of the query optimizer, and which indexes are used. We also show the same execution times for OLAP Slice and Dice query.

For DICE

In Figure 6-86 we show a Dice example for beverage group and region.

			Region	AMERICA	ASIA	EU	Total
			Metrics	Sales Amount	Sales Amount	Sales Amount	Sales Amount
Beverage Group	Beverage	Product					
Whiskey	Whiskey Stan Long	Stan Long 0,75l				269,040	**269,040**
		Stan Long 0,35l			293,150		**293,150**
		Stan Long 0,50l				210,080	**210,080**
		Total			**293,150**	**479,120**	**772,270**
	Total				**293,150**	**479,120**	**772,270**
WINE	Wine Oak	Wine Oak 1,0l		26,350	117,040	4,263,230	**4,406,620**
		Total		**26,350**	**117,040**	**4,263,230**	**4,406,620**
	Wine Red	Wine Red 0,75l			460,578	3,384,930	**3,845,508**
		Wine Red 1,0l		327,630	1,047,510	204,820	**1,579,960**
		Total		**327,630**	**1,508,088**	**3,589,750**	**5,425,468**
	Wine White	Wine White 1,0l				342,550	**342,550**
		Total				**342,550**	**342,550**
	Total			**353,980**	**1,625,128**	**8,195,530**	**10,174,638**
Total				**353,980**	**1,918,278**	**8,674,650**	**10,946,908**

Figure 6-86 Beverage group: Beverage: Product x Region

The figures and labels in Figure 6-86 were obtained by the SQL statement shown in Example 6-15.

Example 6-15 Select statement for Dice

```
select P.BEVERAGE_GROUP,P.BEVERAGE,C.REGION,
sum(S.SALES_AMOUNT) as SALES_AMOUNT
fromTEST.SALES S
   joinTEST.PRODUCTP
     on (S.PRODUCT_KEY = P.PRODUCT_KEY)
   joinTEST.CUSTOMER C
     on (S.CUSTOMER_KEY = C.CUSTOMER_KEY)
group byP.BEVERAGE_GROUP,
   P.BEVERAGE,
   C.REGION
```

Figure 6-87 shows the access plan of the optimizer for the SQL Dice example. The optimizer decided to read entire tables in one scan, and then used a hash join rather than fetching rows by index and joining them with the other table rows in a loop.

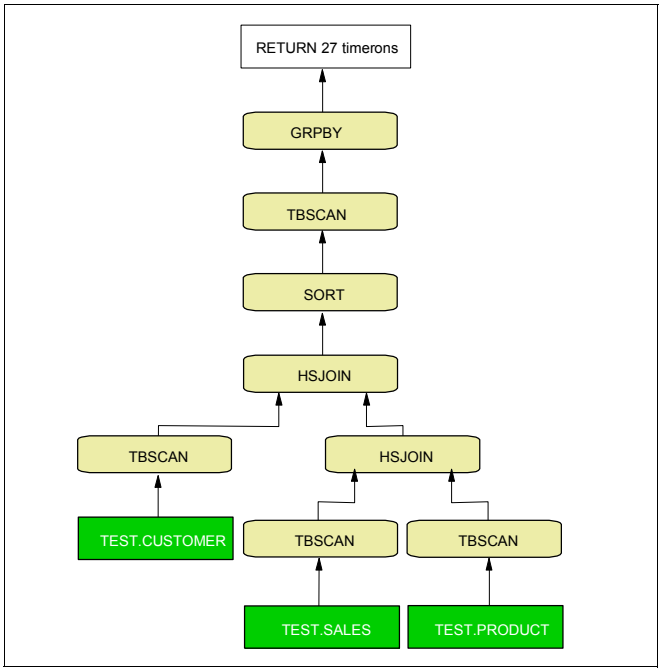


Figure 6-87 Access plan for Dice

For Slice

In Figure 6-88, we show a slice example for beverage *Wine Oak*.

		Metrics	Sales Amount
Beverage	**Product**		
Wine Oak	Wine Oak 1,0l		4,406,620
	Total		**4,406,620**
Total			**4,406,620**

Figure 6-88 Slice for Product dimension

The SQL query for the slice where `BEVERAGE_GROUP='Wine Oak'`, is shown in Example 6-16 on page 329.

Example 6-16 Select statement for Dice

```
select S.BEVERAGE, NAME, sum(P.SALES_AMOUNT) as SALES_AMOUNT
from TEST.SALES P
join  TEST.PRODUCT S
on (P.PRODUCT_KEY = S.PRODUCT_KEY)
whereS.BEVERAGE in ('Wine Oak')
group by  S.BEVERAGE, NAME
```

In Figure 6-89 on page 330 we show the access plan for the star schema without indexes, with indexes, and with an index (Product_H1_IDX) on a the following attributes (BEVERAGE,NAME,PRODUCT_KEY). We get the following results:

- Without indexes, the optimizer reads the data with table scans.
- With PK and FK indexes, the optimizer use the Indexes to fetch and join Product and Sales tables. Therefore, reading the entire fact table was eliminated.
- With Index carrying the Product dimension hierarchy, the access path is simpler because it eliminates a few tasks on the product table. However, this does not significantly decrease execution time.

The estimated processing time for fact table (1000000 rows) and Product dimension in our scenario is listed in Table 6-24. Results show that the indexed star schema for our Slice example performs best.

Table 6-24 Estimated and measured processing times

Index	Estimated [timerons*]
NO	27
PK, FK, and dimension	10
PK, FK dimension, and Product hierarchy	9

* ***Timerons*** are a hybrid value made up of the estimated CPU, elapsed time, I/O, and buffer pool consumption that DB2 expects the SQL will consume when using a particular access path.

Note: The estimated time given by the RDBMS tool is a good guide for designing indexes, but only repeatable measurement of real execution (10 times) gives reliable average values. We used the *DB2 Command Center* access plan utility to get estimates and access plans, and DB2batch to measure execution values.

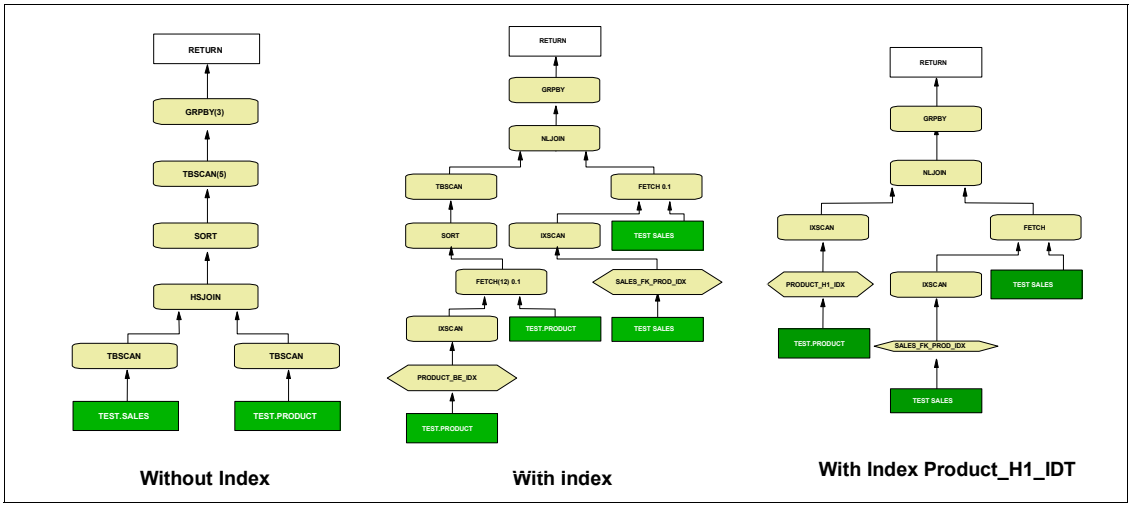


Figure 6-89 Access plan for slice

6.6 Handling changes

In this section we discuss how to deal with changes. The dimensional model should be designed to handle changes to the:

- Data
- Structure
- Business requirements

6.6.1 Changes to data

To maintain history in the dimensional model, we have different change handling strategies. The strategies depend upon whether the dimension changes slowly or changes very fast. The following list describes different approaches for handling changes to various types of dimensions:

- **Slowly changing dimensions:** Slowly changing dimensions change slowly over a period of time. The different approaches to handle slowly changing dimensions are:
  - Type-1: Overwrite the existing value.
  - Type-2: Insert a new row.
  - Type-3: Add one or more columns in the dimension to store changes.

  We discussed slowly changing dimensions in detail in 6.3.5, “Slowly changing dimensions” on page 261.

- **Very fast changing dimensions:** Fast changing dimensions are dimensions whose attributes change very rapidly over a period of time. The approach to handling fast changing dimension is to split the rapidly changing dimension into one or more mini-dimensions. We discussed fast changing dimensions in detail in “Handling fast changing dimensions” on page 269.

6.6.2 Changes to structure

Changes to the structure of a dimensional model may occur because of any of the following reasons:

- **Addition of a new dimension to the star schema:** This may occur when the business needs to view existing data across a new dimension are not present in the model. The new dimension can be easily added to the model if it does not violate the grain of the fact table. This is one of the primary reasons we recommend to design the model at the lowest granularity. Then additional changes to business requirements in the form of new dimensions can be added easily to the existing structure. Assume that you add a new dimension to an existing model on 2/2/2005. Whenever you do so, you must be sure to include an *informational* row containing information, such as *No Data for prior to Date 2/2/2005,* inside this new dimension. All previous rows in the fact table are then linked to this row for all rows inserted in the fact table before the new dimension was made available.
- **Addition of a new attribute to a dimension:** A new attribute can be added to any existing dimension provided the new attribute is true to the grain. If the attribute takes on a single value in the context of the fact measurements, then the new attribute can be added to the existing dimensional model. Assume that you add a new attribute to an existing model on 2/2/2005. When you do so, you must be sure to include an informational value, such as *Not Applicable prior to Date 2/2/2005,* inside this new attribute for all the old dimension rows for which new dimension attribute is not applicable and has no value.
- **Addition of a new fact to the fact table:** A new fact can be added to the existing fact table provided the new fact is true to the grain. If the fact is valid for the existing grain definition, then it can be added to the existing star schema. Assume that you add a new fact to an existing fact table on 2/2/2005. Whenever you do so, you must be sure to include an informational value such as *0* inside this new fact for all the old fact rows for which new fact is not applicable and has no value.
- **Granularity of the dimensional model:** The change in the grain definition is also a cause for the structure of the dimensional model to change. This occurs primarily when the existing dimensional model was designed at a lower grain, thereby missing certain dimensions and facts which were available at a lower grain definition. If the business at a later time requires new

dimensions to be made available at an existing lower detailed grain model, this is generally possible only if the entire model is redesigned by defining a more detailed atomic grain.

For more information on this topic, refer to 6.2.2, “Importance of detailed atomic grain” on page 228.

6.6.3 Changes to business requirements

Dimensional models, if designed at the most atomic detailed grain, can accommodate a change in business requirements very easily. If the grain is defined carefully, then the dimensional model can withstand business requirement changes easily without any change in the front-end applications. The change is typically transparent to applications.

Typically, any change to business requirements may lead to one or more of the following:

- Addition of a new dimension
- Addition of a new dimensional attribute
- Addition of a new fact
- Change in the granularity of the fact table or the entire dimensional model

7

Case Study: Dimensional model development

In this chapter we describe in detail how to methodically apply the Dimensional Model Design Life Cycle (DMDL) when designing a dimensional model.

The goal of this case study is to show the usage of the DMDL technique when applied to specific business requirements. For that, we will follow the DMDL technique step by step, describing our assumptions and explaining the decisions taken during the process.

The design of the dimensional model was developed using the IBM Rational® Data Architect product.

7.1 The project

In the following sections we describe a fictitious project as a case study. We define the scope and objectives, and relate them to the requirements of a fictitious client. This is a good test case to see that a good dimensional model can be achieved by following the DMDL.

7.1.1 The background

In this fictitious case study, the client has been suffering during many years of not having a data warehouse solution. With many systems spread through many countries, and many databases created for ad hoc and local transaction management purposes, the client was not able to effectively consolidate the data in a manner that satisfied the reporting requirements. In order to solve this problem, the client decided to build a data warehouse. After some time, and a few iterations, the client finished implementing a data warehouse. It had all the subject areas that, from the business point of view, represented significant strategic value.

The new data warehouse, depicted in Figure 7-1 on page 335, provided a consolidated view of the company data that they were looking for, and satisfied most of their stated requirements for standard reporting. Initially, the reporting needs were more of a static nature. In other words, once reports were written, they did not need to be rewritten or changed. However, as the number of users of the data warehouse grew, the reporting requirements became more complex in nature. The client soon realized that the reports started changing dynamically.

These changes were primarily because the business realized the potential of data available in the data warehouse and the client wanted to analyze information in a number of new and different ways. In other words, the value of analyzing the data was realized and thus emerged the requirement for more ad hoc reporting. This change in reporting business requirements resulted in more dependence on the reports, and an increased requirement for programmers to develop them. The increase in the number of reports and the increase in the volume of data accessed by them, started to drastically impact the performance of the reports. Upon analysis, the client determined this was primarily because the data warehouse data was in 3NF, and based on an E/R model.

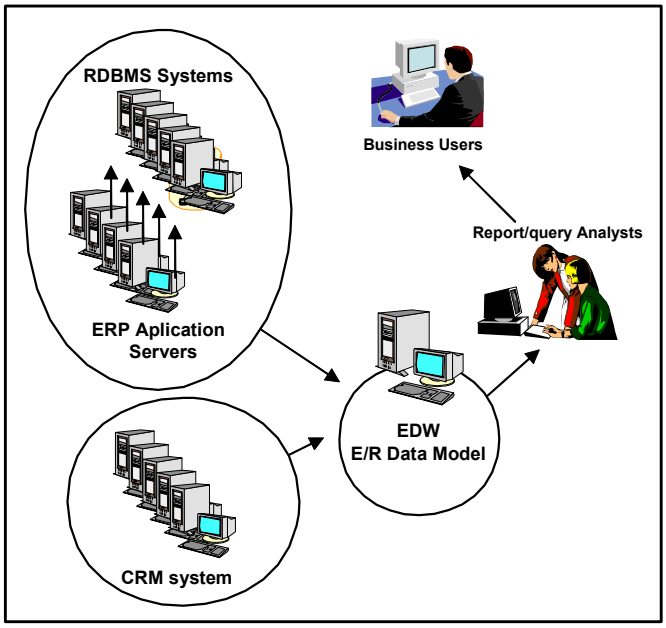


Figure 7-1 Client Data Warehouse Architecture

In addition, because of the complexity of the E/R model, it was extremely difficult for the business users to query the data warehouse on an ad hoc basis.

To solve the performance problem, the client created a new set of denormalized tables in the data warehouse. The purpose was to help the users get a faster response time when accessing their reports. In addition, it was also expected that the users would be able to do ad hoc reporting against this new layer branching out from the warehouse. This process of denormalizing several tables was done on a report-by-report basis. Soon, it was observed that the client ended up with a formal data warehouse, plus a huge number of very highly denormalized tables. These denormalized tables were supposed to provide easy and fast access to the data. However, in reality, it turned out that the data across the tables was not consistent and the tables were still complex, in cases even more complex, to interrelate, link, and maintain, than that in the data warehouse. This made the new data warehouse very inefficient to be used as a data source for reporting purposes.

The client realized the problems of creating denormalized tables for each report and decided to adopt another strategy. Following the same iterative approach that the client had already successfully applied for the development and implementation of the data warehouse, the client decided to start a new project to implement and maintain new data structures in the form of dimensional

databases (also called data marts) that would allow reporting and data analysis tasks in the way the business was demanding. The dimensional models would not only help the clients with ad hoc analysis but also would be easier to read and understand. In addition to this, the dimensional models would give much better performance when compared to the E/R modeled data warehouse tables. The client was confident that the necessary data was not only available but also quality checked in the data warehouse. This important exercise of quality checking the data was done by the data warehouse development team. So now the client was ready to implement dimensional models from the data warehouse.

To help the team learn the proper design skills needed for dimensional design, the client decided to approach the problem in a more efficient way. That meant sending the project manager, a business analyst, a data modeler, the Enterprise IT architect, and a couple of developers to a training center for education on dimensional modeling. In addition to learning dimensional modeling techniques, they became familiar with a methodology called DMDL.

To review DMDL and dimensional modeling technique, review Chapter 5, “Dimensional Model Design Life Cycle” on page 103. There we see that to design a dimensional model, we should first identify a business process candidate. The next step is then to identify the corresponding dimensional models that would be developed and implemented. And that is the methodology we used in this case study. But before we do that, we will try to understand more about the company’s business.

7.2 The company

The Redbook Vineyard was founded in 1843 in Spain, in the region of La Rioja. It started producing and selling its own wine, first locally and then spread through most of Europe. The company specialized in Rioja wine and for many years the sales of this wine continued to represent its primary source of income.

In 1987, Redbook Vineyard acquired a distributor of alcoholic and non-alcoholic drinks and moved the corporate office to Madrid, Spain. Since then, it has also incorporated other non-drinkable items, such as T-Shirts.

The company is number ten in sales of spirits, and number three as a wine producer in Europe, having a market share of 4% and 23%, respectively.

7.2.1 Business activities

The Redbook Vineyard is an old company owned by one family. Since 1843, they have been producing their own wine mainly made with mazuela grapes harvested from different wine locations across the La Rioja region. The wine is

produced at their winery in La Uvilla. The production of wine includes different activities, from buying grapes, though the wine processing, to the distribution of the wine in bottles to customers.

In 1987 the Redbook Vineyard decided to broaden their activity portfolio and started to distribute major brands of wine and spirits to America and Australia. Since then they have established sales branch offices in most of the major European cities.

The Redbook Vineyard has built up very good business relationships with large retail store chains over the years, and its market share in spirits and wine is about 10% of the entire European market.

The Redbook Vineyard is highly customer-oriented. They collect all customer-related information in their Customer Relationship Management (CRM) system at each branch location.

In order to manage its business across different countries, the Redbook Vineyard uses modern IT systems. For their core businesses, they use an Enterprise Resource system (ERP) and a CRM system. In order to support top management for decision-based activities, the Redbook Vineyard has just built a centralized Enterprise Data Warehouse in Spain.

7.2.2 Product lines

As shown in Figure 7-2 on page 338, the company markets several lines of products. However, in reality, the beverages are the products that keep the business going for Redbook Vineyard.

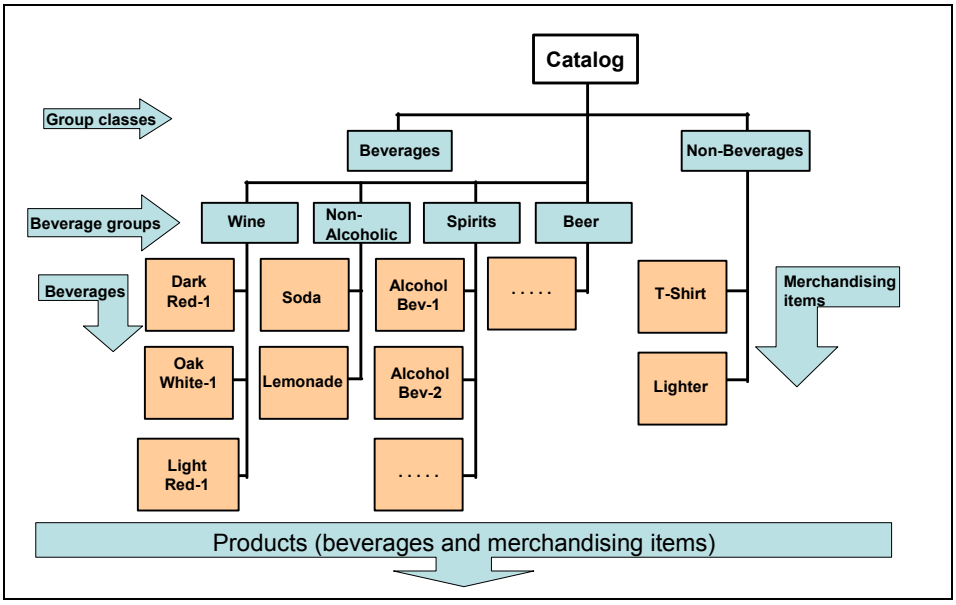


Figure 7-2 Product lines for Redbook Vineyard

The product catalog of the Redbook Vineyard company is divided into two main categories:

- Beverages
- Non-beverages

We discuss each of these product categories in more detail below.

Beverages

Beverages are initially any sort of potable drink. During the last few years, despite the appearance of new types of drinks, such as light drinksand energy drinks, the product structure has not changed. It is however very likely that this structure will need to change very soon. As seen in Figure 7-2, the Redbook Vineyard presently has the following beverage lines: wine, non-alcoholic, spirits, and beer.

We discuss each of these beverage types in more detail below.

Wines

The line of wine covers any sort of wine that has no more than 25% alcohol. That includes, for example, sherry and oporto wine, which has a graduation significantly higher than normal wine.

Non-Alcoholic

The non-alcoholic beverages presently include most types of soda with 0% alcohol contents. It is foreseen that Redbook Vineyard will market stimulant drinks, also called *energy drinks*, in the near future. In addition to this, milk and yogurt drinks will also be experimentally introduced and temporarily distributed to a few chosen resellers. Until now dairy drinks were not introduced because of the extra overhead involved with these short shelf-life drinks.

Beer

Any sort of beer with under 10% alcohol is included in this line. Beers with an alcohol content above 10% are uncommon. Beers with low alcohol content, also called light beers, are also marketed under this product line. However, depending on the legal clarifications, the client is evaluating the possibility of moving the light beers with an alcohol volume of less than 1% into the non-alcoholic product line.

Spirits

Any other marketed alcoholic drink not covered in the beverage lines previously described will be included in this group. Spirits are typically beverages with the highest alcohol content.

Non-Beverages

This is the other product line, apart from beverages. It offers merchandising products, such as T-shirts, glasses, and lighters. The company started these products in order to support marketing the beverage product line. However, objects such as certain jars, or art decó lighters, are very successful and the company is considering marketing them separately.

7.2.3 IT Architecture

The Redbook Vineyard IT infrastructure is distributed across the different branch offices. Every branch office in every country has its own ERP system. This is because each country has different legal requirements. Activities related to the customers are covered by the CRM system. The topology of the architecture of a typical branch office is shown in Figure 7-3 on page 340.

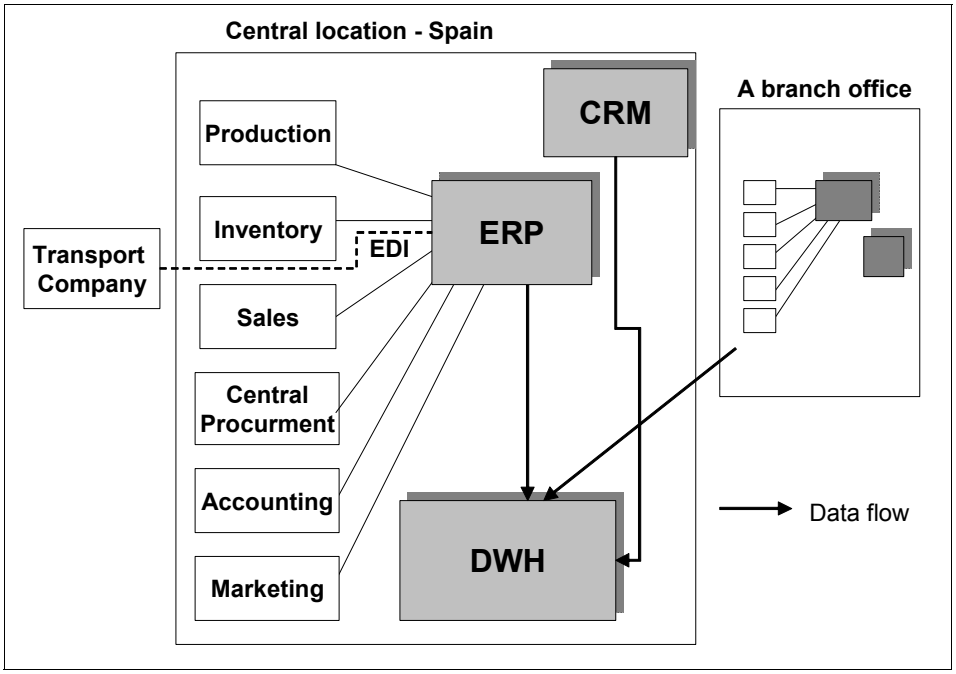


Figure 7-3 Typical branch office IT architecture

The data warehouse component, as shown in the Figure 7-3, is the only IT component that is centralized. The data warehouse component provides services to the Spanish Central location (headquarters) and to the remote users from all other European branch offices via corporate virtual private network (VPN). The data warehouse is refreshed overnight.

The main consumable data source for business users is the company's data warehouse system of record (SOR), also called the Enterprise Data Warehouse. The data in this system is cleansed and reliable. It contains data that originated in the different operational systems used at the Redbook Vineyard.

7.2.4 High level requirements for the project

In the following section, we list the requirements identified for the design and implementation of the business intelligence solution that the Redbook Vineyard is demanding.

User requirements

- The data mart must have a high business strategic value. This is a critical requirement.

- Ideally the system availability should be 24 x 7. This is not a critical requirement because to have the system available during weekdays and office hours would actually suffice.
- The system should be implemented in the shortest period of time possible. This is a critical requirement.
- The data must be consistent, reliable, and complete (*the truth, the whole truth, and nothing but the truth*). This is the most critical requirement.

IT requirements

Some of the IT requirements are:

- There should be one decision support system of choice covering all source systems with business strategic value within the company.
- Complex advanced analysis should be possible against consolidated data from several source systems.
- A generic data warehouse structure should be made available for retail and manufacturing processes to store and retrieve data/information.
- The MIS and primary source systems should be separated. This will help increase performance for the OLTP systems.
- The data warehouse must include detailed history data to avoid overloading OLTP systems when restoring archived tapes for inquiry purposes.
- The data must be standardized by integrating available data standards into the data modeling tool.
- There should be a minimum basic meta data management system easily accessible, with information about the data warehouse components.

OLAP client requirements

- One central OLAP system should be available.
- A user-friendly GUI should be available to do analytical work with minimal assistance. The GUI must permit drag and drop operations to build the OLAP queries without having to type field and table names.

7.2.5 Business intelligence - data warehouse project architecture

In Figure 7-4 on page 342, we show the proposed architecture for the data warehouse, showing the major components. The shading in the figure differentiates between the present architecture and the components needed for completing the data warehouse solution.

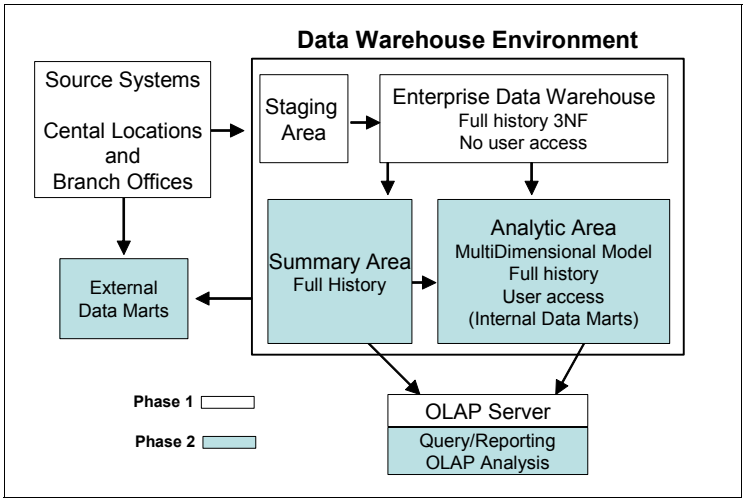


Figure 7-4 High level project data warehouse architecture

The architectural components of the data warehouse, shown in Figure 7-4, are as follows:

- **Source systems:** ERP central systems and CRM systems at each branch office**.**
- **Data warehouse environment:** Consists of the following sub-components:
  - **Staging area.** This area contains all the data consolidated, preformatted and transformed before it is loaded the into System Of Records (SOR) area or the Enterprise data warehouse.
  - **Enterprise data warehouse.** This system of records area stores all detailed information, including all the company history. It is a highly normalized database (designed in 3NF), covering the subject areas with the most strategic business value.
  - **Analytical area.** This includes detailed atomic dimensional models (data marts). The data stored in these dimensional models is highly detailed and atomic.
  - **Summary area.** The summary area includes the aggregate tables based on the detailed analytical dimensional models.
- **Reporting:** The reporting component includes basic static reports (via SQL queries). This will be replaced by a new OLAP/reporting solution.
- **OLAP:** This component includes advanced analytical function with ad hoc capabilities and what-if analysis.

- **External data marts:** This component includes the data marts for mobile Customer Representatives.

Note: The dimensional model or the multidimensional database is in the analytical area of the data warehouse architecture (see Figure 7-4).

7.2.6 Enterprise data warehouse E/R diagram

Figure 7-5 shows an example of an E/R diagram. The present enterprise data warehouse could have one that looks similar. The objective here is not to show a complete E/R diagram that is meaningful. It is simply to demonstrate that the size and number of interrelationships can result in quite a complex diagram.

The primary subject areas of the enterprise data warehouse model are:

- Arrangements (AR): consisting of Orders, Invoices, and Orders Lines
- Transactions (TXN)
- Finance
- CRM

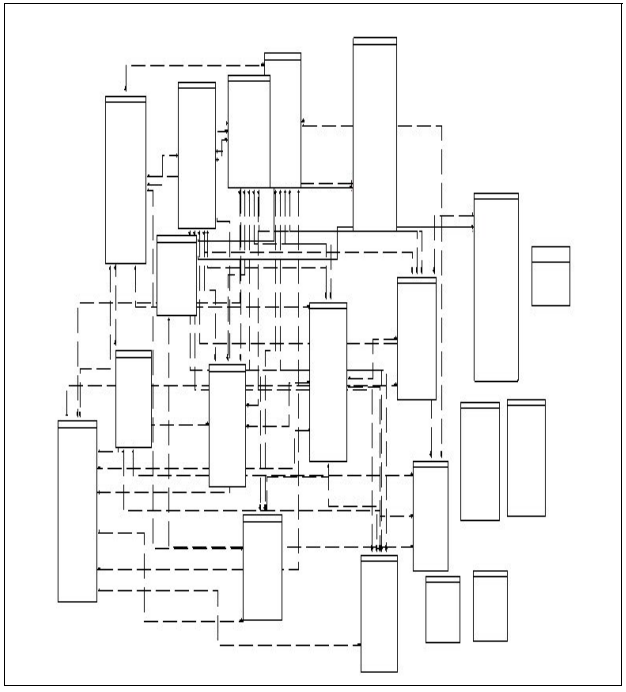


Figure 7-5 Physical E/R model

7.2.7 Company structure

Figure 7-6 shows the company organization structure. The IT and the human resources departments are not included in this case study. The structure is the same as the one the company had before the business expansion of 1987. The company is now considering a reorganization of departments and this chart will certainly change. But that should not influence the results of this case study.

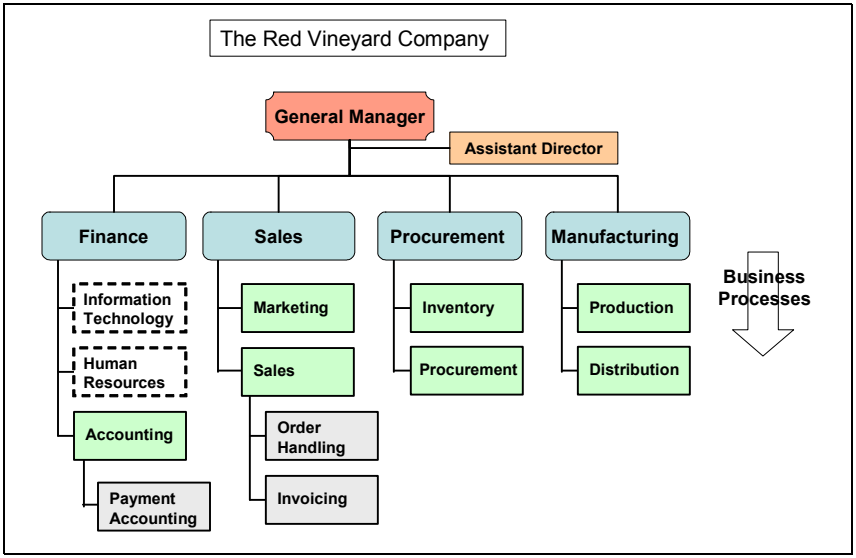


Figure 7-6 Company structure

7.2.8 General business process description

In this section we describe three important aspects of the Redbook Vineyard company. They are the:

- Products
- Customers
- Business processes

We now focus on each of these in more detail below.

Products

We discussed the company products in 7.2.2, “Product lines” on page 337. In our case study, we focus only on the Beverages line of products as shown in Figure 7-7 on page 345, since beverages are the primary source of income for the company.

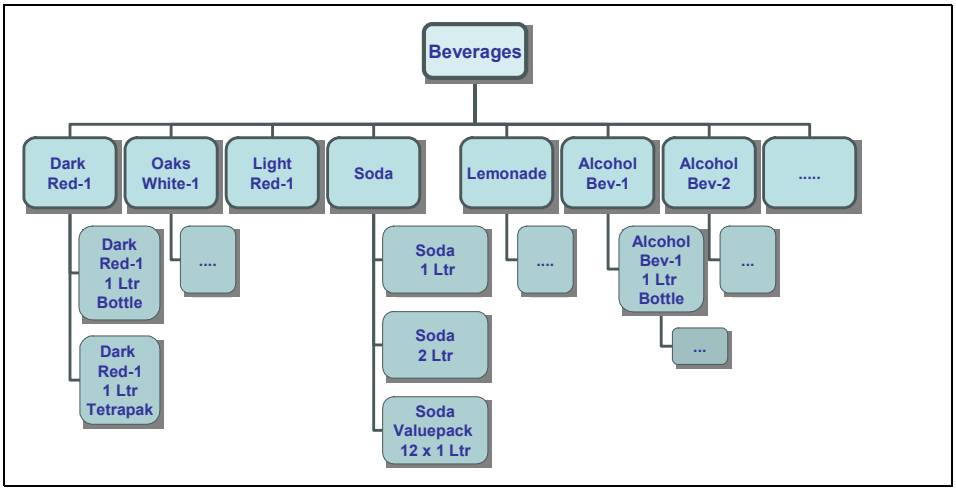


Figure 7-7 Beverage Items

The products we show in Figure 7-7 were valid at the time of creating the diagram. Important points to consider about these beverage items are:

- Products could be ordered and shipped, and therefore they are considered active (`status = active`).
- There are beverages that are no longer produced (`status = discontinued`).
- There could also be beverages that are new and are about to be released. Although not yet released, new beverages can be ordered (status = new).
- Another characteristic of the products is their presentation. A beverage is not always in bottles. Other forms of presentation include tetrabriks, soft plastic bags, and barrels.
- The characteristics of the products normally remain unchanged during their lifetimes. However, the structure itself is subject to change every two or three years.

Customers

We now discuss the Redbook Vineyard Company customers who have been the key to the company's success over the years. An example of the customer structure is shown in Figure 7-8 on page 346.

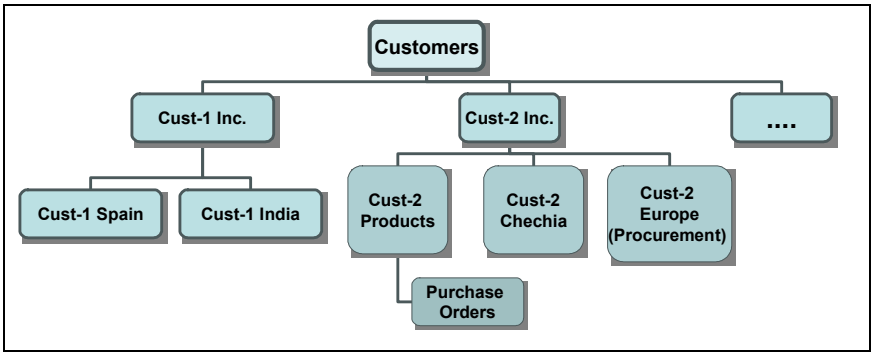


Figure 7-8 Customer structure example

Important points to note about the Redbook Vineyard customers are:

- The typical customer is a medium size company that resells the beverages to smaller companies and to end-consumers.
- Customers are grouped by classes: `"WHOLESALE"`, `"STORE CHAIN"`, `"GAS STATION CHAIN"`, `"RESTAURANT"`, or `"SMALL SHOP"`.
- The customer gets a credit limit and a discount volume depending upon certain factors, such as seniority of the customers, the average order volume per year, and the paying discipline of the customer. The credit limit can be any amount starting from 10 Euros, and the discount can be between 1% and 40%.
- Customers usually have a preferred way of payment, such as money transfers, or credit cards. The company assigns payment terms to customers. The default is 30 days for existing customers and pay upon delivery for new customers. However, this can vary between 5 and 90 days.
- Some customers are small companies, that have just have one department or address from which all the orders come. Other customers are multinationals with presence in several countries. In some of the countries they have only one organization, in others, they have several organizations with divisions with different names, but each of them are actually the same company.
- Customer attributes, such as name or address, usually remain stable and do not change across the life of the customer in the company database. There are, however attributes, such as credit limit (for example, 10 000 dollars), payment terms (for example, 30 days after invoice), payment method (for example, bank transfer), and discount (for example, 10%) that are recalculated and reassigned monthly. For example, during specific phases, such as two months before, and during Christmas, these attributes are recalculated on a weekly basis.

Business processes

In this section we discuss the business processes that are shown in Figure 7-6 on page 344. The business processes were accurately and thoroughly documented at the beginning of the data warehouse project. Most of the data for these business processes was made available in the enterprise data warehouse that the Redbook Vineyard company built. However there are a few exceptions. These exceptions are mentioned below.

- In this section, we discuss the following business processes of the Redbook Vineyard company:

  a. Inventory handling (also called Inventory management)
  b. Procurement
  c. Distribution
  d. Production
  e. Accounting
     i. Payment accounting (also called Credit Controlling)
  f. Sales
     i. Sales Order Handling
     ii. Sales Invoicing
     iii. Sales Marketing/CRM

a. Inventory management

The main function of this process is to manage the inventory. This process is responsible for the following:

- Receiving the wine when it is shipped from the bottling plant. In addition to this, inventory management is also responsible for receiving other beverages that are shipped by external suppliers.
- The placement of received products in their corresponding storage areas.
- The inventory department maintains inventory levels. Also on a weekly basis, the inventory department communicates to the procurement department about inventory levels of products. This communication helps the procurement team to reorder products depending on the need.
- Inventory is also responsible for packing and shipping the ordered products. Although in exceptions where certain items are expensive and need special handling, single bottles of certain spirits are separately packaged and sent. Normally each product is packaged in boxes or containers of several units each. Typically packages will have six, 12, 24, and 100 bottles each. Additionally, for better handling of the goods, the packages can be stored on pallets which may hold up to 100 packages or boxes.

Note: The shipping charges are dependent on the order volume and total weight and not the number of items ordered. However there are a few items, such as very expensive and luxurious champagne bottles, whose packaging and special handling demands that we track the shipping cost per order line rather than per order. Therefore, shipping charges are invoiced at the order line level and not at the order level. Also, products ordered in one order can be shipped from different warehouses.

The Redbook Vineyard customers order the goods throughout the year, and, therefore, the Redbook Vineyard has its own warehouses in each European country. When customers order the product, the Redbook Vineyard fulfills the request from the warehouse in the country where the order was placed. If the product is not available, then it is shipped from the central warehouse or any other close available warehouse. If no warehouse has the ordered goods, then Redbook Vineyard orders the goods from the corresponding supplier responsible for maintaining the warehouse.

b. Procurement

The main function of this process is to manage the purchase orders sent to the suppliers and manufacturing plants. This process, however, has no special business strategic relevance, since the main business income is from the wine produced by the owning company. The data warehouse contains data about this process but at a very summarized level.

This procurement business process is responsible for:

- The coordination with the inventory process for stock level maintenance. We discussed earlier that inventory communicates to procurement about the inventory levels for various products.
- Procurement is responsible for the creation, submission, and tracking of the purchase orders. In addition to this, the process also checks and closes the order after receiving the ordered goods.

c. Distribution

This is a very straightforward and simple process. This process generates high infrastructure costs. It is not part of the core business because the company has outsourced it to an external company called Unbroken Glass Hauler Inc. (UGH). UGH uses the IBM Informix database manager for its operations. Electronic Data Interchange (EDI) is used between the two companies and it has been set up in a way that enables periodic transfer of the minimum data necessary to coordinate and report the product transport operations. The method of goods delivery typically depends on the customer, who will have a preferred method such as by truck, train, or plane.

d. Production

Presently the company produces only wine. This process is very complex because of innovative techniques implemented for the production of wine, and no operational data is fed yet into the data warehouse. It is presently supported by an ERP application, and if data is needed for reporting or analysis, then the ERP application is usually accessed directly without many restrictions. This production process mainly covers:

- Planting and collection of wine grapes.
- Reception, processing of the grapes, and storage in the wine cellars.
- Bottling and shipping of the wine.

e. Accounting

This process covers all the key financial and accounting aspects of the company. However, for similar reasons to the ones pointed out for the distribution process, the company is presently evaluating the possibility of outsourcing this process. Although the main data is fully available in the data warehouse, there are a number of tables that need to be recreated every day and access to them from the data warehouse is periodically restricted. This process manages, among others, the following:

- Maintenance of the General Ledger.
- Control of the completeness, consistency, and accuracy of all financial statements from all the branches and offices worldwide
- Financial reporting to the local authorities

The Accounting process has another sub-process which is called the payment accounting process. This process is discussed in more detail below.

e-i. Payment accounting (also called Credit Controlling)

This is a special process working very closely to the sales business process, especially regarding the customer credit analysis and reclassifying.

This process is responsible for:

- Ensuring that the customer pays on time and does not owe more than their allowed credit limit.
- Analyzing customer debts. The process classifies the customer credit limit according to factors such as purchase volume, payment discipline, and credit history. This input is further provided to the sales process for classifying the customer bonus and discount percentages.

f. Sales

Apart from the production process, which we reviewed earlier, the sales process is one of the most complex. This process heavily and directly impacts the

generation of revenue and is considered to be highly critical for the business. The sales business process encompasses the following:

- The design, execution, and analysis of marketing campaigns
- Analysis of the market for introduction of new products or elimination of non-profitable products
- Responsibility for providing all customer assistance and guidance by a Sales Representative
- The reception, handling, and invoicing of the Customer Orders

The sales process is divided into three separate sub-processes. They are sales order handling, invoicing, and marketing. We discuss each of these processes below.

f-i. Sales Order Handling: This sub-process covers the order handling of the sales orders from receiving the order to shipping. It deals with the following:

- Receiving an order and processing it
- Coordination with inventory for shipping purposes
- Coordination with procurement for ordering purposes
- Overall tracking of the sales order
- Any related customer interaction

f-ii. Sales Invoicing: The invoicing is less complex than the order handling process. But from a reporting standpoint, it is the most important one. This is because the main financial and corporate reports are made against the invoiced amounts. This process manages:

- Printing and sending the invoice data to the customer
- Quantitative analysis (volumes, regions, customers, and products)
- Financial analysis (total sales, discounts, regions, customers, and products)
- Correction or cancellation of invoices for returned or damaged material

f-iii. Sales CRM/Marketing: This sub-process has a objective to analyze to decide on the introduction of new profitable beverages. In addition, this process also tracks the decrease in the acceptance and market demand of existing beverages. Based on this input, a decision is made whether or not to continue these beverages. The marketing process responsibilities include:

- Analysis of market by channel and product.
- Ad hoc screening of the market along with the competitors.
- Tracking and analysis of marketing activity (type of actions, regions, and customers).

- The analysis and segmentation of customer groups to allow a better targeting during marketing campaigns.
- The execution of marketing campaigns around new or existing products to acquire new customers or increase the volume of sales to existing ones.
- Preparing, performing, and analyzing customer contacts by Sales Representatives.

A part of the data needed for the execution of marketing campaigns is delivered (purchased) by external companies specialized in market analysis. This data is not presently part of the data warehouse, and is usually obtained in the form of XML data structures, which, depending on the amount of data, are sent by e-mail or by DVD.

7.2.9 Developing the dimensional models

The Redbook Vineyard has decided to design dependent data marts for several of its business processes. Each of these dependent data marts would get the data from the existing data warehouse. We have understood well the critical business needs for which the Redbook Vineyard wants to move forward with this initiative.

But the next question is, where do you start? What do you do first? To help you in that decision process, we have designed a Dimensional Model Design Life Cycle (DMDL). For a detailed description of the DMDL, see Chapter 5, “Dimensional Model Design Life Cycle” on page 103.

In the remaining sections, we complete the case study by developing a dimensional data model for the company. We use the DMDL as a guide to demonstrate how it can help to develop a robust and complete dimensional model.

In the next section, we start with the first phase which is *Identify the Requirements*.

7.3 Identify the requirements

The first phase of the DMDL starts with identifying the requirements. This phase is heavily dependent on analysis of the business processes inside the organization. This is shown in Figure 7-9 on page 352.

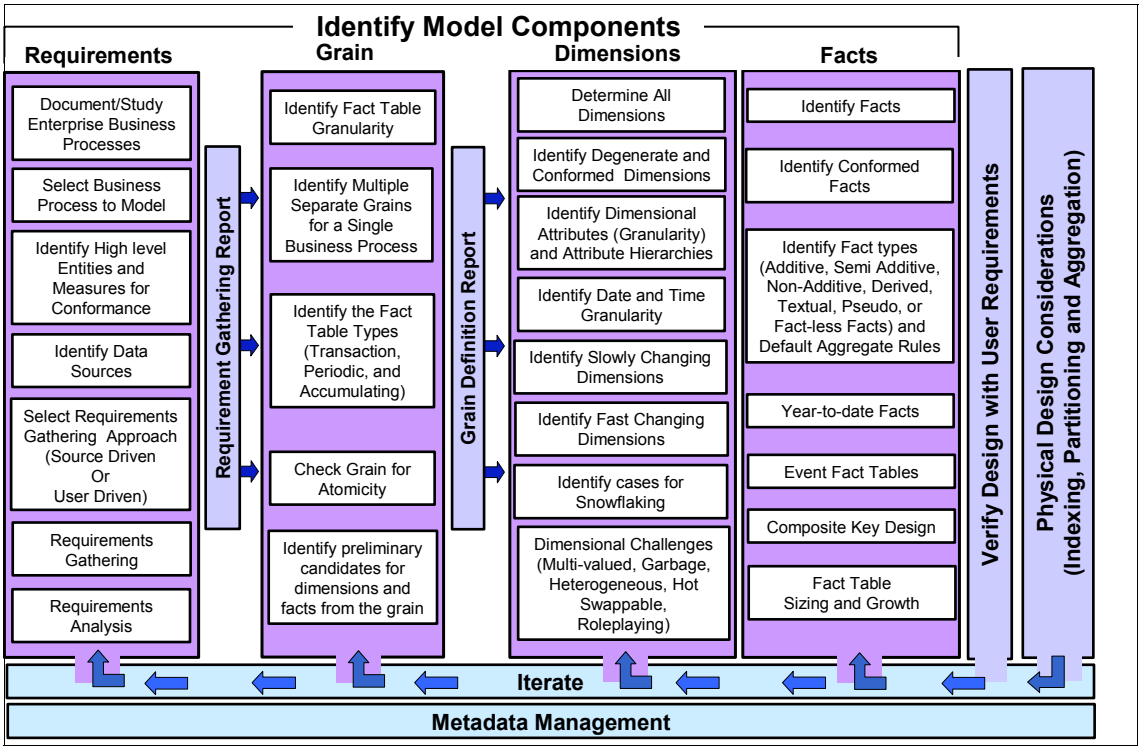


Figure 7-9 Business Process Analysis

The main goals of this phase are to:

- Create an enterprise-wide business process list
- Identify the business process (includes prioritization)
- Identify high level entities and measures for conformance
- Identify various data sources involved
- Identify the requirement gathering approach you would follow
- Gather requirements
- Analyze requirements

7.3.1 Business process list

This is a very straightforward task, assuming that the documentation available from the first phase of the project is still up to date. To recall, the Redbook Vineyard company implemented a data warehouse for its company a few months back. We will not include here the detailed business process descriptions because they are very extensive. We identified important business processes in detail in 7.2.8, “General business process description” on page 344. They are:

- Inventory handling (also called Inventory Management)
- Procurement
- Distribution
- Production
- Accounting
  - Payment accounting (also called Credit Controlling)
- Sales
  - Sales Order Handling
  - Sales Invoicing
  - Sales Marketing/CRM

Business process assessment

In this activity we identify the factors that the company considers most important to evaluate each business process. We also prioritize them and assign weights. These assessment factors are shown in Table 7-1.

Table 7-1 Point table for assessment factors

Assessment factor	Low	Medium	High
Complexity	6	4	2
Data quality and availability	3	6	9
Strategic business	2	4	6
System availability	1	2	3

Note: We have mentioned only a few assessment factors in Table 7-1. However, we encourage you to assess your business processes across various factors while deciding to prioritize a process.

The next step is to relate those factors to each business process and rate each combination accordingly. You can see the results in Table 7-2.

Table 7-2 Business process assessment

Name of business process	Complexity	Data quality and availability	System availability	Strategic business significance
Distribution	Low	Medium	Low	Low
Sales CRM/ Marketing	Medium	Medium	Medium	Medium

Name of business process	Complexity	Data quality and availability	System availability	Strategic business significance
Inventory	Medium	High	High	Medium
Procurement	Medium	Medium	High	Low
Sales Orders	Medium	High	High	Medium
Accounting	Medium	High	Medium	Low
Production	High	Low	High	High
Sales Invoicing	Low	High	High	High

Note: We have rated both the `data quality` and `availability`, and the `system availability` factors as *High* for all the processes whose data is already present in the data warehouse system of record. For all the outsourced processes, or process candidates for outsourcing, we have rated the strategic business significance as *Low*.

Marketing data and system availability has been rated as *Medium* because the data partially comes from external sources.

7.3.2 Identify business process

In this activity, we prioritize the business processes. This means we identify the most and least feasible process for building a dimensional model. In order to do this, we use the Point table for assessment factors as shown in Table 7-1 on page 353, and we turn the priority indicators (`Low`, `Medium`, and `High`) into numbers. Doing so, we get results depicted in Table 7-3.

Table 7-3 Business process prioritization

Business process	Complexity	Data quality and availability	System availability	Strategic business significance	Final points
Distribution	6	6	1	2	15
Sales CRM/ Marketing	4	6	2	4	16
Inventory	4	9	3	4	20
Procurement	4	6	3	2	15
Sales Orders	4	9	3	4	20
Accounting	4	9	2	2	17
Production	2	3	3	6	14
Sales Invoicing	**6**	**9**	**3**	**4**	**22**

Observing Table 7-3 we see that Sales Invoicing has the most points. In other words, this means that Sales Invoicing is one of the important and feasible business processes. We will then use the Sales Invoicing business process to design our dimensional model.

Note: Sales, as a process, is divided into three processes. They are Sales Invoicing, Sales Orders, and Sales Marketing/CRM. However, after assessing the various business processes, we have found that the Sales Invoicing process has the most points.

More on the sales process

The corresponding process detailed description is taken from the data warehouse documentation, which is relatively fresh and easily accessible. The sales process consists of the following processes:

One: Sales CRM/Marketing

The Sales CRM/Marketing process involves the following activities and procedures:

- A Sales Representative contacts a customer or is contacted by a customer. The contact can be established by telephone, e-mail, fax, instant messaging, or in person.
- The Sales Representative assists the customers and guides them through the purchase decision process. The most important priority is the satisfaction of the customer. Next is to generate an immediate sales order.

Two: Sales Order handling

The Sales Order handling process involves the following activities and procedures:

- If the sale evolves positively, the Sales Representative enters the order by means of a mobile computer. Otherwise, the order can be sent in later, by one of the following ways:
  - Online, using the ordering application available through the Redbook Vineyard Web page
  - By sending a printed copy of the order produced via the Redbook Vineyard Web application
  - By sending the order using the postal service, fax, or e-mail

  Note: The customer can access the order form through the Web application, print it, and send it later rather than immediately submit it. Both existing and new customers can also enter an order via the company Web page without having made any previous contact with any Sales Representative.

- The Customer Order is processed by the sales office closest to the customer address.
- Ordered products are consolidated, within each order, according to their product identifiers, so that one product id only appears in one line and no more than in one line of the same order. In other words, if a customer orders the same product twice in the same order, that same product is represented in one line with the total quantity.
- The sales department sends an order acknowledgement to the customer, including the Redbook Vineyard order number and an estimated delivery date.

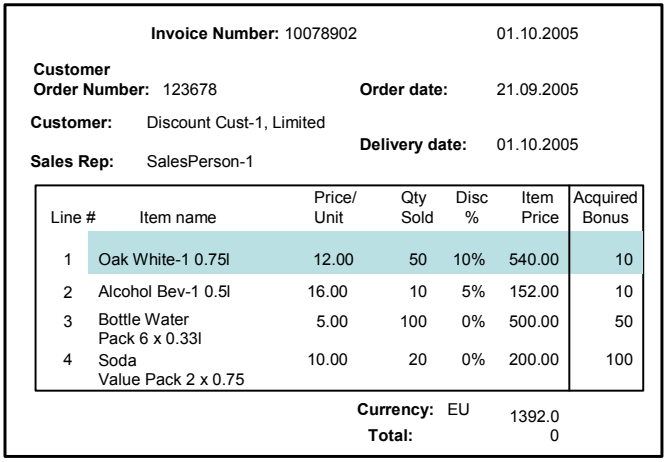
Three: Sales invoicing

Sales invoicing involves the following activities and procedures:

- An order can be partially shipped, in which case it can also be partially invoiced. The following describes partial order handling:
  - **Partial order delivery with partial invoice:**
    This can happen when an item is not in stock, and has to be reordered. In this case, only the items available in stock are shipped. The other activities in partial shipments are:
    - Notify the procurement department of items not available.
    - Items in stock have to be shipped immediately.
    - The inventory transaction and the invoice amount are registered in the General Ledger.
  - **No partial order delivery:**
    In this case the order must be entirely shipped at once.
    - Notify the procurement department of items not available.
    - Order is shipped only when all the items are available.
    - The final price is calculated and adjusted.
    - The invoice is printed and sent to the customer.
    - The inventory transaction and the invoice amount are registered in the General Ledger.
  - **Partial order delivery *without* partial invoice:**
    The instructions are the same as the case described above for “Partial order delivery with partial invoice” except that the invoice can only be produced after the order has been fully shipped.
  - **Closing the order**
    After the order has been fully shipped and invoiced, the order is formally closed in the system.
  - **Returned and damaged material**:
    A printed invoice cannot be changed. In case of damaged or returned material, updates or corrections are performed by issuing a credit note (which is the same as an invoice but with a negative amount).

Note: An order item can be shipped from different warehouses. Different shipments of the same product are allowed, if having ordered more than one unit, the product can only be fully delivered when shipping it from different warehouses. Every shipment corresponds to one line in the corresponding invoice. An invoice line does not consolidate different shipments of the same product.

Figure 7-10 shows an example of an invoice form.



Invoice Number: 10078902 01.10.2005

Customer
Order Number: 123678 **Order date:** 21.09.2005

Customer: Discount Cust-1, Limited **Delivery date:** 01.10.2005

Sales Rep: SalesPerson-1

Line #	Item name	Price/ Unit	Qty Sold	Disc %	Item Price	Acquired Bonus
1	Oak White-1 0.75l	12.00	50	10%	540.00	10
2	Alcohol Bev-1 0.5l	16.00	10	5%	152.00	10
3	Bottle Water Pack 6 x 0.33l	5.00	100	0%	500.00	50
4	Soda Value Pack 2 x 0.75	10.00	20	0%	200.00	100

Currency: EU
Total: 1392.0 0

Figure 7-10 Example invoice

E/R model for order handling and invoicing

Figure 7-11 on page 359 depicts the tables in the E/R model used by the Sales Order handling and Sales Invoicing processes. A dimensional model will be designed for the Sales Invoicing business process.

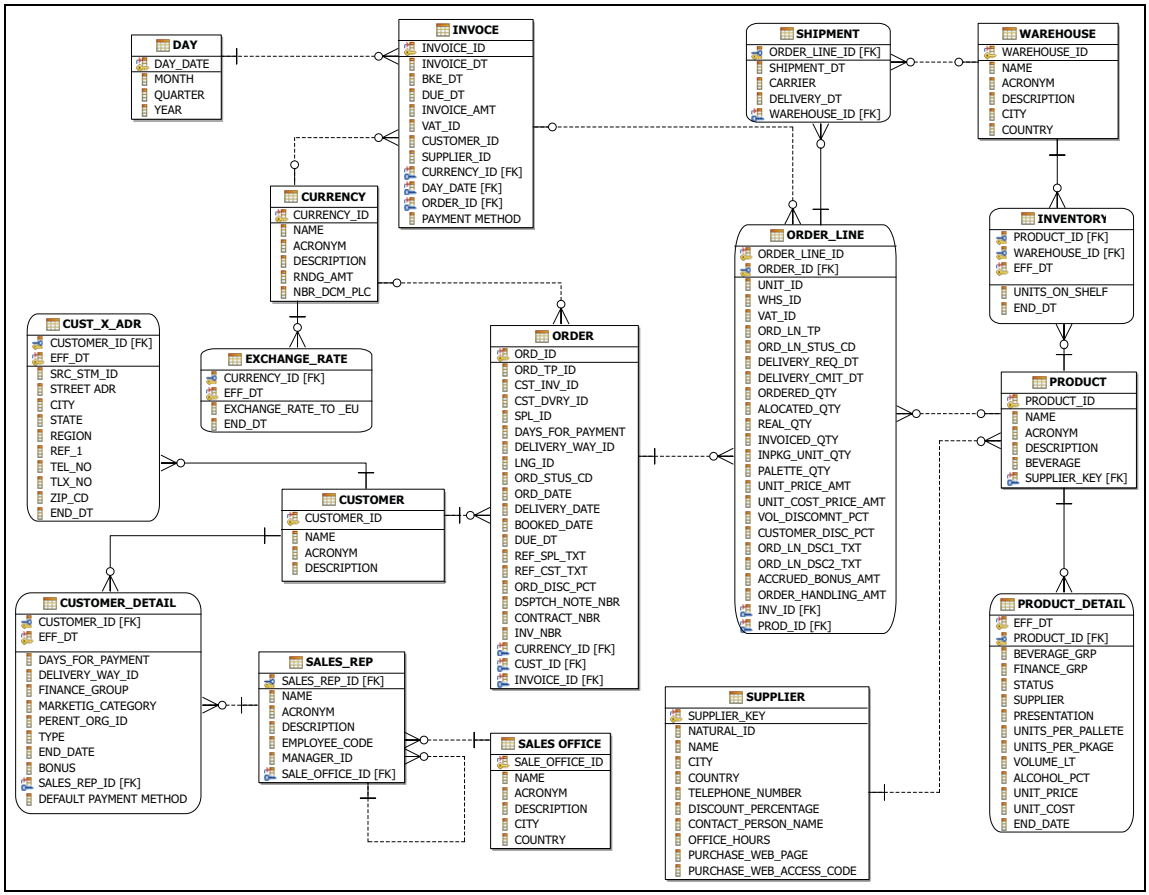


Figure 7-11 E/R model for Sales Process

7.3.3 High level entities for conformance

In this activity we identify the entities and measures that need to be consistently defined and shared across the different business processes. We were able to select the entities from the data warehouse. A sample of those selected are shown in Table 7-4 on page 360.

Table 7-4 High level entities

High Level Entities ---> Business Processes	**Beverage**	**Time**	**Organization**	**Employee**	**Orders + Invoice**	**Warehouse**	**Currency**	**Shipment**	**More**
Sales Invoicing	X	X	X	X	X	X	X	X	
Accounting	X	X	X	X	X		X		
Inventory	X	X			X	X		X	
Sales CRM/Marketing	X	X	X	X		X			
Distribution	X	X	X		X	X	X	X	
Procurement	X	X	X			X	X	X	
Production	X	X	X	X					
More									

For simplicity, we have limited the sample set of entities in Table 7-4 to only those initially needed for the Sales order handling process, rather than all the ones needed for the complete process.

The idea behind this exercise is to find out what entities are common across several business processes. Once we are able to find out these common entities, we want to make sure that all data marts use common or conformed entities in all dimensional models. Each of the business processes are tied together through these common entities.

In order to create conformed dimensions that are used across the enterprise, the various businesses must come to agreement on defining common basic definitions for common entities.

For example, as shown in Table 7-4, the business processes such as Sales, Accounting, Inventory, CRM/Marketing, Distribution, Procurement, and Production should agree on a common definition of the Beverage (Entity). This is because all these business processes have the beverage entity in common. This beverage entity may later become a conformed beverage (product) dimension which will be shared across all the business processes. However, getting agreement on common entities such as Product (beverage) and Customer may be difficult because these entities are subjective and their definition will vary from one business process to another. On many entities it will be easy to reach

agreement for defining a common definition. For example, entities such as stores, dates, and geographies, because these entities and their definitions are are fairly straightforward.

7.3.4 Identification of data source systems

In this activity we identify and document the data sources that support the business processes described in 7.3.2, “Identify business process” on page 355 and 7.3.3, “High level entities for conformance” on page 359. The data sources are documented in Table 7-5.

Other information contained in Table 7-5 can be derived from the Redbook Vineyard company structure and its business process descriptions as described in 7.2.7, “Company structure” on page 344 and 7.2.8, “General business process description” on page 344.

Table 7-5 Data sources for business process

Business process	Data sources	Data owner	Location	Platform	Data update
Sales order handling	Enterprise data warehouse	Sales	Madrid	DB2	Daily
Sales Invoicing	Enterprise data warehouse	Sales	Madrid	DB2	Daily
Accounting	Enterprise data warehouse	Finance	Madrid	DB2	Daily
Inventory	Enterprise data warehouse	Procurement	Madrid	DB2	Daily
Sales CRM/ Marketing	Enterprise data warehouse and external source systems	Sales	Madrid	DB2/ XML file	Daily/ upon request
Distribution	Outsourced ERP	Procurement/ Sales	Madrid	Informix/ DB2	Periodic Real time

Business process	Data sources	Data owner	Location	Platform	Data update
Procurement	Enterprise data warehouse	Procurement	Madrid	DB2	Daily
Production	ERP	Production	Madrid	DB2	Real time

Note: The data owner is not the system owner. For example, the Sales order data available in the Data Warehouse is owned by the Sales order department, but the data warehouse system of record is owned by the IT department.

7.3.5 Select requirements gathering approach

We discussed the source-driven and user driven requirement gathering approaches in detail in 5.2.5, “Select requirements gathering approach” on page 113. For this exercise, we have decided that the requirements gathering approach will be user-driven rather than source-driven. This is because there was no request to include all the source data, so we decided to involve the users in the decision since they best understand the business needs.

Note: The business users typically have clear ideas about what source data is required. So be careful if asked to integrate all data from all existing data sources. Though easy to ask, it may result in a much higher cost and larger effort than expected. Therefore, an accurate analysis of the data sources will minimize the potential for these issues.

7.3.6 Gather the requirements

In this activity we gather business requirements for the Sales Invoicing business process. We perform a series of interviews with top and middle management, and users involved in Sales Invoicing-related reporting activities. The objective is to inquire, collect, and document the different requirements for reporting and online data analysis purposes.

In 5.2.6, “Requirements gathering” on page 116, there was a list of aspects to consider while interviewing the user. Here are more specific considerations to use when gathering requirements:

- Ask for existing reports and for information about queries or programs used to get the required data.
- Ask how often each report or query is run.
- For each requirement, ask about:
  - Specific data needed
  - Selection condition or filtering criteria to apply
  - Consolidation or grouping needed

Note: Questions formulated as follows: “Show me the total sales and total discounts per month” and “Show customers sorted by total sales for a given country” should be encouraged. This enables them to be quickly converted into SQL pseudo statements, such as “`Select A where B grouped by C order by D`” for query development.

In Table 7-6, we show a sample list of the requirements collected during the user interviews. For intermediate filtering, we asked the user to prioritize the requirements by assigning one of these three categories: very critical, necessary, and nice-to-have.

Nice-to-have requirements are excluded from this list. You might decide a different categorization, and cover all the requirements in the first design. The questions have been numbered using two different prefixes: Q for very specific requirements, and G for general requirements.

Table 7-6 User requirement table

Q#	Sales Invoicing business process requirements
Q01	What is the gross profit of product sales grouped by beverage and customer category during a given month?
Q02	Debt analysis by customer: Print active and outstanding debts at a given date, by customer and grouped in ranges or bands of days. The user needs to have a report to list the sum of payments due in the next seven days per customer or per customer country. Also the user needs to report the total invoiced amount with payment due longer than 30 days.
Q03	What is total amount of outstanding debts on a selected date by beverage and by category? And also by customer and invoice number?
Q04	What is the number of units sold and net sales amount per product for non-beverage items, per quarter?
Q05	What are the 10 most profitable beverages by: 1. Largest total sales per year to date and region. 2. Largest gross margin per year to date and region.

Q#	Sales Invoicing business process requirements
Q06	Who are my 10 most profitable customers per region by largest total sales volume per year?
Q07	How is my net profit developing since the beginning of year? (Display profit during current year at the end of month, calculating monthly and accumulated profit per month.)
Q08	How many bottles (containers) and liters were delivered per time period in months compared to the previous three years during the same period?
Q09	List the total sale discount per beverage or product group or customer or customer category per time period compared to previous three years during the same period.
Q10	Organization credit history. For a selected month interval, list the outstanding payments at the end of each month grouped by customer and sorted by outstanding amount.
Q11	Report showing net revenue, gross sales, gross margin, and cost of sales per beverage and month, quarter and year.
Q12	Analysis of delayed invoicing due to partial shipments: Total amount grouped per warehouse, beverage, or beverage group for a selected period where the invoice date is a number of days greater than the first shipping date. **Business reason:** If the customer does not want partial invoicing, the order will be invoiced once it is fully shipped. Partial order deliveries will wait to be consolidated in a full order invoice. Orders may take long time to complete, from the time of the first partial shipment. It is necessary to analyze the impact of the indirect credits that the customer gets in this situation, and the increase in the sales performance if we improve the shipping process. The pertaining data needs to be analyzed only once per month.
Q13	What is the sales order average value and sales order maximum value per month per Sales Representative?
Q14	What customers have placed orders greater than a given value during the last year?
Q15	What are the companies that last year exceeded their credit by more than 20%?
Q16	List gross sales totals and total discounts by Sales Representative for the past year sorted by gross sales amount in descending order.
Q17	List the total discount amount, percentages, and gross margin per Sales Representative and customer sorted by gross margin in ascending order during last year.

Q#	Sales Invoicing business process requirements
Q18	List the number of orders per month containing one or more non-beverage items since the first year available.
Q19	Gross sales per Sales Representative, sales office, month, and currency selected by year.
Q20	Gross sales amount per each method of payment and country during last year.
Q21	Three top week days per month according to average sales by Web orders. The listing must include total gross sales and number of orders. Business reason: For planning system maintenance, is there a day (or days) of the week in which the number and orders increase or decrease? (sales office = WEB)
Q22	Number of orders and average sales order volume per transmission type (fax, telephone, Web) per month and year.
Q23	Revenue, gross margin, and cost of goods during last year per supplier sorted by revenue in descending order.
Q24	For tracking urgent customer orders specially ordered to the supplier, a report is needed listing: Supplier id, Supplier name, Contact name, Telephone number, Discount%, Web page and Web access code from the supplier.
Q25	Monthly report for accounts containing: Month, Customer id, Net sales, Duty, Cost of item handling, Cost of goods.
G01	All sales and cost figures for beverage reports must be available in both reporting currency and invoice currency.
G02	Beverages need to be summarized or added by product group.
G03	Allow tracking of average delivered alcohol liters, maximum and minimum alcohol content values per organization, Sales Representative and/or beverage group **Business reasons:** Restructuring of the existing marketing product groups, and supporting resellers with the legal requirements regarding not selling beverages to the non-adult population.
G04	Gross sales, discount and bonus totals per year per corporation.
G05	Changes to beverage and sales organization structure and all corresponding attributes must be tracked across the time (history needed).
G06	Changes to customer attributes need to be fully maintained and stored.

Note: ***Points to review about the requirements***

- Gross profit = Profit calculated as gross sales income less the cost of sales.
- Gross sales = Total invoiced amount minus total discount including returned and damaged items (in the requirements, Sales means Gross sales by default, unless otherwise indicated).
- Gross margin = Percentage difference between the cost of sales and the gross sales.
- Customer active debt = Amount invoiced to a customer and not yet due for payment.
- Customer outstanding debt = Amount invoiced and due for payment.
- Net sales = Gross sales minus returned and damaged items.
- Net profit = Net sales - Cost of sales - Duty amount - Cost of item handling (Item shipping charges).
- Cost of sales = Cost of the goods that have been sold, tax excluded.

7.3.7 Analyze the requirements

In this activity we analyze the Sales Invoicing business process requirements gathered in 7.3.6, “Gather the requirements” on page 362. In this section we do the following:

- Analyze each business question.
- Identify high level entities and also possible hierarchies associated with each of these entities.
- Identify high level measures.

The high level entities and measures identified here give us an idea about the kind of data with which we are dealing. It is highly possible that the entities identified here could become dimensions or facts in the future phases of the Dimensional Model Design Life Cycle.

Table 7-7 on page 367 shows the analyzed business requirements and the identified entities and measures.

Table 7-7 Identified entities and measures requirement table

Q#	Sales Invoicing business process requirements	High level entities	Measures
Q01	What is the gross profit of product sales grouped by beverage and customer category during a given month?	Beverage, Organization, Time (Month), Organization Category	Gross Profit Amount
Q02	Debt analysis by customer: Print active and outstanding debts at a given date, by customer and grouped in ranges or bands of days. The user needs to have a report to list the sum of payments due in the next seven days per customer or per customer country. Also the user needs to report the total invoiced amount with payment due longer than 30 days.	Organization, Time (Day), Country	Outstanding Debt (Net Sales Amount)
Q03	What is total amount of outstanding debts on a selected date by beverage and by category? And also by customer and invoice number?	Beverage group, Organization, Invoice, Time (Day)	Outstanding Debt (Net Sales Amount)
Q04	What is the number of units sold and net sales amount per product for non-beverage items, per quarter?	Beverage, Product group, Time (Quarter)	Net Sales Amount, Quantity Sold
Q05	What are the 10 most profitable beverages by: 1. Largest total sales per year to date and region. 2. Largest gross margin per year to date and region.	Beverages, Time (Year to Date), Region	Gross Sales, Gross Margin%
Q06	Who are my 10 most profitable customers per region by largest total sales volume per year?	Organization, Time (Year to Date), region	Gross Sales Amount
Q07	How is my net profit developing since the beginning of year? (Display profit during current year at the end of month, calculating monthly and accumulated profit per month)	Time (Year to Date), Time (Month)	Net Profit Amount

Q#	Sales Invoicing business process requirements	High level entities	Measures
Q08	How many bottles (containers) and liters were delivered per time period in months compared to the previous three years during the same period?	Time (Month)	Quantity Sold, Liters Delivered
Q09	List the total sale discount per beverage or product group or customer or customer category per time period compared to previous three years during the same period.	Beverage, Product Group, Organization, Organization Category	Discount Amount
Q10	Organization credit history. For a selected month interval, list the outstanding payments at the end of each month grouped by customer and sorted by outstanding amount.	Time (month), Invoices, Organization	Net Sales Amount
Q11	Report showing Net revenue, Goss sales, Gross margin, and Cost of sales per beverage and month, quarter, and year.	Product Group, Time (Month), Time (Quarter), Time (Year)	Net Revenue Amount, Gross Sales Amount, Gross Margin%, Cost of Goods
Q12	Analysis of delayed invoicing due to partial shipments: Total amount grouped per warehouse, beverage, or beverage group for a selected period where the invoice date is a number of days greater than the first shipping date. **Business reason:** If the customer does not want partial invoicing, the order will be invoiced once it is fully shipped. Partial order deliveries will wait to be consolidated in a full order invoice. Orders may take a long time to complete, from the time of the first partial shipment. It is necessary to analyze the impact of the indirect credits that the customer gets in this situation, and the increase in the sales performance if we improve the shipping process. The pertaining data needs to be analyzed only once per month.	Warehouse, Product group, Shipment, Invoices	Invoice Delayed Amount

Q#	Sales Invoicing business process requirements	High level entities	Measures
Q13	What is the sales order average value and sales order maximum value per month per Sales Representatives?	Time (Month), Employee	Gross Sales Amount
Q14	What customers have placed orders greater than a given value during the last year?	Organization, Time (Year)	Gross Sales Amount
Q15	What are the companies that last year exceeded their credit by more than 20%?	Organization, Time (Year)	Outstanding Debt
Q16	List gross sales totals and total discounts by Sales Representative for past year sorted by gross sales amount in descending order.	Employee, Time (Year)	Gross Sales Amount, Discount Amount
Q17	List the total discount amount, percentages and gross margin per Sales Representative and customer sorted by gross margin in ascending order during last year.	Employee, Organization, Time (Year)	Discount Amount, Discount Percentage, Gross Margin%
Q18	List the number of orders per month containing one or more non-beverage items since the first year available.	Time (month), Product Group	Number of orders
Q19	Gross sales per Sales Representative, sales office, month, and currency selected by year	Sales Representative, Sales Office, Time (Month), Time (Year)	Gross Sales Amount
Q20	Gross Sales amount per each method of payment and country during last year.	Payment Method, Country	Gross Sales Amount
Q21	Three top weekdays per month according to average sales by Web orders. The listing must include total gross sales and number of orders. Business reason: For planning system maintenance, is there a day (or days) of the week in which the number and orders increase or decrease? (Sales office = Web)	Time (Weekday), Sale Office,	Gross Sales, Number of Orders

Q#	Sales Invoicing business process requirements	High level entities	Measures
Q22	Number of orders and average sales order volume per transmission type (fax, telephone, Web) per month and year.	Order transmission type, Time (year), time (month)	Number of orders
Q23	Revenue, gross margin, and cost of goods during last year per supplier sorted by revenue in descending order.	Organization (supplier), Time (Year)	Gross Margin%, Cost of Goods, Revenue
Q24	For tracking urgent Customer Orders specially ordered to the supplier, a report is needed listing: Supplier Id, Supplier name, Contact name, Telephone number, Discount%, Web Page and Web access code from the supplier.	Organization (Supplier)	NA
Q25	Monthly report for Accounts containing: Month, Customer id, Net Sales, Duty, Cost of Item handling, Cost of Goods.	Organization (customer), Time (Month),	Net sales amount, Duty Amount, Cost of Item Handling, Cost of goods
G01	All sales and cost figures for beverage reports must be available in both reporting currency and invoice currency.	Currency, Product Group	All money amount measures requested in the other requirements
G02	Beverages need to be summarized or added by product group.	Beverage - Product group	All measures applicable to beverages
G03	Allow tracking of average delivered alcohol liters, maximum and minimum alcohol content values per Organization, Sales Representative and/or Beverage group **Business reasons:** Restructuring of the existing marketing product groups, and supporting resellers with the legal requirements regarding not selling beverages to the non-adult population.	Organization, employee, product group	Alcohol Content% Spirit liters.

Q#	Sales Invoicing business process requirements	High level entities	Measures
G04	Gross Sales, Discount and Bonus totals per year per corporation.	Organization (customer corporation)	Gross Sales Amount, Discount Amount, Bonus Amount
G05	Changes to Beverage and Sales Organization structure and all corresponding attributes must be tracked across the time (history needed).	Beverages, Product group	N.A.
G06	Changes to customer attributes need to be fully maintained and stored.	Organization	N.A.

High level entities and measures identified

Table 7-8 shows the high level entities (potential future dimensions), and any associated hierarchies with the measures identified in Table 7-7 on page 367.

Table 7-8 High level dimension identified

Entities	Hierarchy
Beverage	Product Group → Beverage
Organization (Customer/Supplier)	Region → Country → City → Organization Organization Category → Organization Corporation → Organization → Division
Time	Year → - Quarter → Month → Day → Time WeekDay → Day → Time
Employee	Region → Country → City → Sales Office → Sales Employee
Warehouse	Region → Country → City → Warehouse
Currency	Country → Currency
Shipment	-
Orders	Order Transmission Type - Order
Invoices	-

Table 7-9 on page 372 shows the high level measures identified from the requirements analyzed in Table 7-7 on page 367.

Table 7-9 High level measures

Measures
Alcohol Content%
Cost of goods
Discount Amount
Discount Percentage
Gross Profit Amount
Gross Sales Amount
Gross Margin%
Gross Sales Amount
Invoice Delayed Amount
Liters Delivered
Net Profit Amount
Net Sales Amount
Net Revenue Amount
Number of orders
Outstanding Debt
Quantity Sold
Revenue Amount
Spirit liters
YTD Gross Sales
YTD Gross Margin
Duty Amount
Accrued bonus
Cost of item handling

7.3.8 Business process analysis summary

The final output of the *Identify business process* phase is the requirements gathering report. This requirements gathering report contains the following:

- Business process listing
- Business process prioritization
- High level entities and measures common between the business processes
- Business process identified for which dimension model will be built
- Data sources listing
- Requirement gathering for the business process
- Requirement gathering analysis
- High level entities and measures identified from the requirement analysis

7.4 Identify the grain

In this phase we focus on the second step of the dimensional model design life cycle, which is *Identify the grain*, as shown in Figure 7-12.

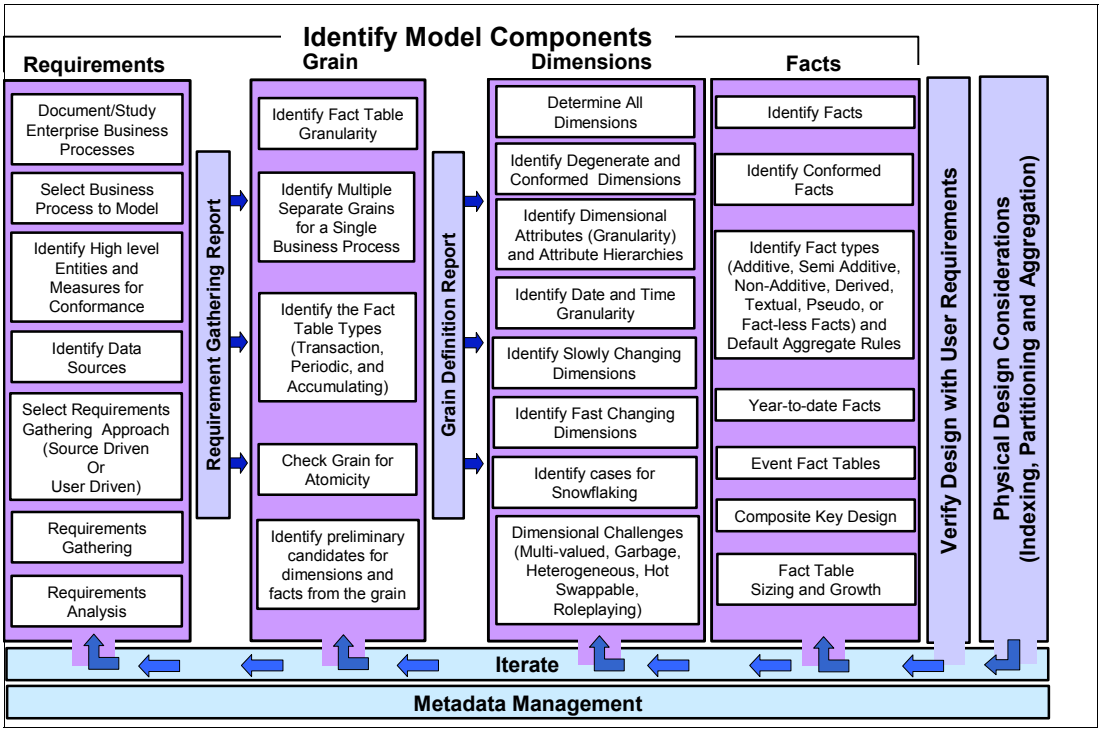


Figure 7-12 Identify the grain

In 7.3, “Identify the requirements” on page 351, we identified the Sales Invoicing business process for which we will design the dimensional model. In this phase we will design the grain definition for the requirements we gathered for the Sales Invoicing process in 7.3.6, “Gather the requirements” on page 362.

The primary goals of this phase are to:

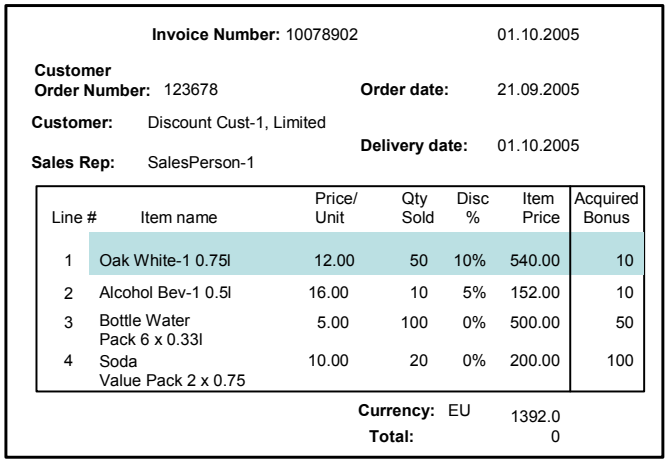
- Define the fact table granularity.
- Identify existence of multiple grains within one business process.
- Identify type of fact table (Transaction, Periodic, or Accumulating).
- Study the feasibility to be able to provide lower grain for fact table than is requested.
- Identify higher level dimensions and measures from the grain definition.

- Produce the grain definition report which contains all the results and final output of this phase.

7.4.1 Identify fact table granularity

We have previously defined the sales process as the action of invoicing an order. The sales process was discussed in detail in 7.3.2, “Identify business process” on page 355. Therefore, in this case the natural candidate to look at when searching for the right granularity is the invoice form. An invoice consists of a header and one or many line items. A sample invoice form is shown in Figure 7-13. The invoice form does not show all associated attributes with the invoice. In this form we find the lowest grain for the fact table.

The grain is identified as *a single invoice line in the invoice form* (highlighted in the example in Figure 7-13).



Invoice Number: 10078902 01.10.2005

Customer Order Number: 123678 **Order date:** 21.09.2005

Customer: Discount Cust-1, Limited

Delivery date: 01.10.2005

Sales Rep: SalesPerson-1

Line #	Item name	Price/ Unit	Qty Sold	Disc %	Item Price	Acquired Bonus
1	Oak White-1 0.75l	12.00	50	10%	540.00	10
2	Alcohol Bev-1 0.5l	16.00	10	5%	152.00	10
3	Bottle Water Pack 6 x 0.33l	5.00	100	0%	500.00	50
4	Soda Value Pack 2 x 0.75	10.00	20	0%	200.00	100

Currency: EU **Total:** 1392.0 0

Figure 7-13 The grain of the sales fact table

Note: The grain you choose determines the level of detailed information that can be made available to the dimensional model.

7.4.2 Identify multiple separate grains

The primary focus of this activity is to determine if there are multiple grains associated with the Sales Invoicing business process for which we are designing a dimensional model.

There will be times when more than one grain definition is associated with a single business process. In such a scenario, we recommend to design separate fact tables with separate grains rather than to forcefully try to put facts that belong to separate grains in a single fact table.

In short, this activity will allow us to decide if there are multiple grains associated with the sales process, and if we need to implement one or multiple fact tables.

In order to decide whether to go for one or multiple fact tables, we suggested criteria in 5.3.2, “Multiple, separate grains” on page 125. Here are those criteria:

- One of the most important criteria that helps us determine the need for one or multiple fact tables are the facts. It is very important to understand the dimensionality of the facts to make the decision of whether the facts belong together in one fact table or in separate fact tables with different grains. For example, let us consider the sales business process.

  We have found two facts Outstanding Debt and Invoice Delayed Amount that belong to the sales process as a whole, but are not true at the grain definition at the line item level we have chosen. We will therefore exclude them now and leave them for a subsequent iteration.

  Multiple granularity: Facts such as *Number of Orders*, *Invoice Delayed Amount,* and *Outstanding Debt* will be implemented in separate fact tables.

- Other criteria to consider is whether or not multiple OLTP source systems are involved. Generally if we are dealing with business processes such as order management, store inventory, or warehouse inventory, it is very likely that separate source systems are involved and hence the use of separate fact tables may be appropriate. However, in our Sales Invoicing business process, we have only one source of data and that is the enterprise data warehouse.
- It is also important to find out if multiple business processes are involved. Multiple business processes involve the creation of multiple separate fact tables. And, it is possible that a single business process may involve creation of separate fact tables to handle facts that belong to different granularity. For our dimensional model, we have chosen only one business process and that is Sales Invoicing.

Note: It is important to understand that had we chosen the sales process as a whole, we would have to design separate dimensional models for various subprocesses inside the sales process. In 7.2.8, “General business process description” on page 344, we discussed the various subprocesses inside the sales process, which were Sales Invoicing, Sales Orders, and Sales CRM/Marketing.

- Another important criteria to consider are the dimensions. If you find that a certain dimension is not true to the grain definition, then this is a sign that it may belong to a new fact table with its own grain definition.

Based on the assessment factors discussed above, we decide that there would be one fact table to handle the Sales Invoicing data. And, facts such as *Number of Orders*, *Invoice Delayed Amount,* and *Outstanding Debt* will be implemented in separate fact tables.

Are multiple grain definitions identified?

Yes, there are multiple grain definitions identified for the Sales Invoicing process. And, they are:

- A single line item on the invoice form: In this chapter, we will be designing the dimensional model for this grain definition only. The other grain definitions defined below will be handled separately in different fact tables.
- Outstanding debt for each customer.
- Single delayed invoice for a customer on an order.

Note: It is extremely important not to merge facts or dimensions which are true at different grains into one fact table. Instead, a new dimensional model should be created with a new grain.

7.4.3 Identify fact table types

In this section we identify the types of fact tables. There are basically three types of fact tables, and they are:

- Transaction fact table: A table which records one row per transaction. A detailed discussion about the transaction fact table is available in “Transaction fact table” on page 231.
- Periodic fact table: Stores one row for a group of transactions. In other words, a periodic fact table stores a single row for a number of transactions over a period of time. A detailed discussion about the periodic fact table is available in “Periodic fact table” on page 232.
- Accumulating fact table: Stores one row for the entire lifetime of an event. For example, the lifetime of a credit card application being sent, until the time it is accepted by the company. Another example is the lifetime of a job or college application being sent, until the time it is accepted or rejected by the job posting company or college. A detailed discussion about the accumulating fact table is available in “Accumulating fact table” on page 233.

We also recommend that you review 5.3.3, “Fact table types” on page 126 for more details.

We now know that the grain of our sales fact table is the invoice line. Invoices are generated once and they never get updated again. If changes are made, they have to be handled through credit notes.

Invoice lines are considered transactions, and one invoice line item will represent one row in the fact table. In other words, each single item on an invoice will form a part of the fact table.

Fact table type: For the Sales Invoicing business process, we have identified three separate grains. However, for this case study, we will be working on the first grain definition which is *a single item on an invoice*. The fact table for this grain is a transaction fact table.

7.4.4 Check grain atomicity

Regarding the Sales Invoicing process, we have already chosen the lowest grain possible, which is the invoice line.

Both the dimension table and fact table have a grain associated with them. To understand the grain of a dimension table, we need to understand the attributes of the dimension table. Every dimension has one or more attributes. Each attribute associates a parent or child with other attributes. This parent-child relationship provides different levels of summarization. The lowest level of

summarization or the highest level of detail is referred as the grain. The granularity of the dimension affects the design such as the retrieval of data and data storage.

Grain adjustment: For our sales invoice grain definition, we have chosen the most detailed atomic grain as a single line item on the invoice. Lowering this grain definition is neither necessary nor possible.

7.4.5 Identify high level dimensions and facts

In this activity, we identify high level preliminary dimensions and facts from whatever can be understood from the grain definition. No detailed analysis is carried out to identify these preliminary dimensions and facts.

For our Sales Invoicing business process, we defined the grain (see 7.4.1, “Identify fact table granularity” on page 374) as a single line item on the invoice. An example of this is shown in Figure 7-14.

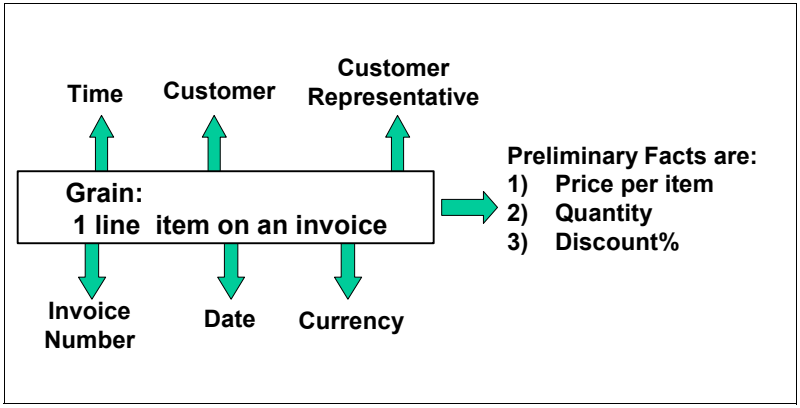


Figure 7-14 Identifying high level dimensions and facts from the grain

From the invoice form shown on Figure 7-14, we have identified several facts and dimensions. Those preliminary facts and dimensions are listed in Table 7-10.

Table 7-10 Preliminary facts and dimensions

Name	Output
Preliminary facts	Quantity, Price per Item, Discount%
Preliminary dimensions	Invoice, Customer, Time, Beverage, Customer Representative, Currency

These preliminary high level dimensions and facts are helpful when we formally identify dimensions (see 7.5, “Identify the dimensions” on page 380) and facts (see 7.6, “Identify the facts” on page 409). The dimensions and facts get iteratively refined in each of the formal phases of the DMDL.

Note: Preliminary facts are facts that can be easily identified by looking at the grain definition. For example, facts such as Price per Item, Quantity, and Discount% are easily identifiable by looking at the grain. However, detailed facts such as cost, manufacturing cost per line item, and transportation cost per item, are not preliminary facts that can identified by looking at the grain definition. Preliminary facts are not the final set of facts. Formal detailed fact identification occurs in the *Identify the facts* phase in 7.6, “Identify the facts” on page 409. The same is true for preliminary dimensions.

7.4.6 Grain definition summary

The output of the *Identify the grain* phase is the *Grain Definition Report,* which contains information such as that shown in Table 7-11.

Table 7-11 Main output of the Identify the grain phase

Name	Output
Fact table granularity	Invoice line
Are there multiple grains within one business process?	Yes, there are multiple grain definitions identified for the Sales Invoicing process. They are: ► A single line item on the invoice: In this chapter, we will be designing the dimensional model for this grain definition only. The other grain definitions will be handled separately in different fact tables. However, they are out of the scope of this chapter. ► Outstanding debt for each customer. ► Single delayed invoice for a customer on an order.
Type of fact table	Transactional (Grain is a single line item on an invoice.)
Check grain atomicity (Necessary to lower it?)	No
Preliminary facts	Quantity, Price per Item, and Discount%
Preliminary dimensions	Invoice, Customer, Time, Beverage, Customer Representative, and Currency

7.5 Identify the dimensions

In this section, we focus on the *Identify the dimensions* phase of the dimensional model design life cycle, depicted in Figure 7-15.

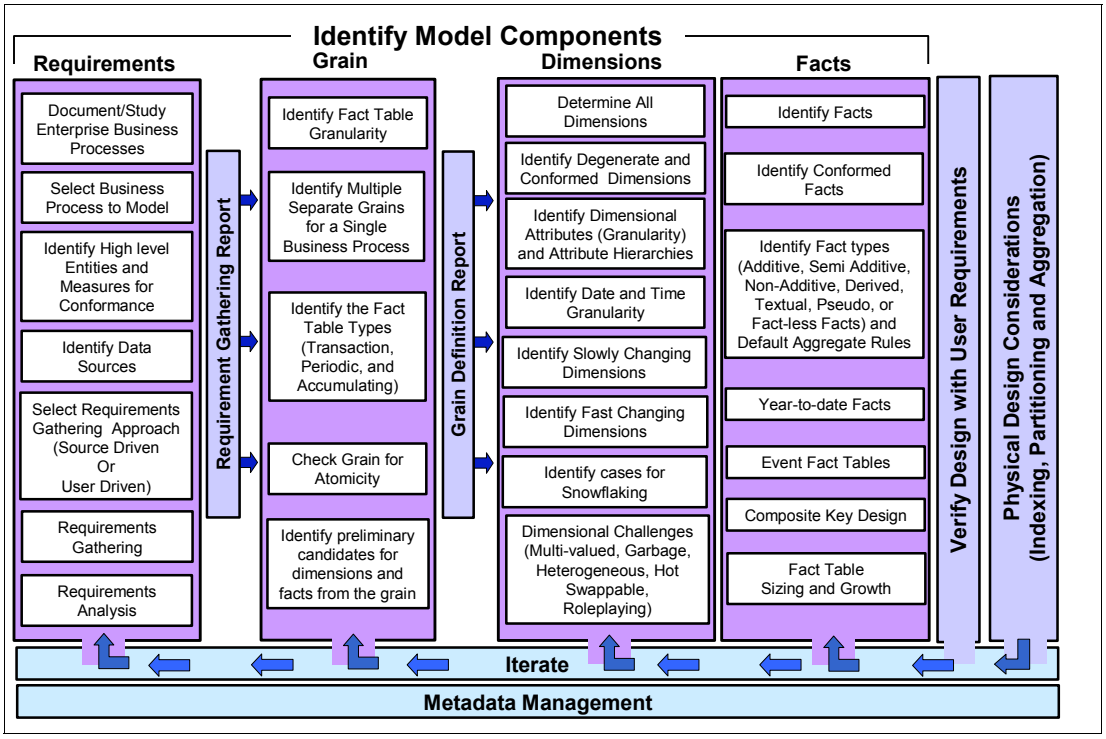


Figure 7-15 Dimensional Model Design Life Cycle

Table 7-12 shows the activities that are associated with the *Identify the dimensions* phase.

Table 7-12 Activities in the Identify the dimensions phase

Seq no.	Activity name	Activity description
1	Identify dimensions	Identifies the dimensions that are true to the identified grain.
2	Identify degenerate dimension	Identifies one or more degenerate dimensions.

Seq no.	Activity name	Activity description
3	Identify conformed dimensions	Identifies any existing shared dimensions present in the data warehouse or other star schemas that may be used for designing the dimensional model.
4	Identify dimensional attributes and dimensional hierarchies	Identifies the various dimension attributes for the dimension. It also identifies the balanced, unbalanced, or ragged hierarchies that may exist in the dimensions. Techniques are suggested to handle these hierarchies in the design.
5	Identify date and time granularity	This activity identifies the date and time dimensions in the dimensional design. Typically it is these dimensions that have a major impact on the overall grain and size of the dimensional model.
6	Identify slowly changing dimensions	Identifies the various slowly changing dimensions in the design. Also three major techniques (Type-1, Type-2, and Type-3) are described for handling slowly changing dimensions.
7	Identify very fast changing dimensions	Identifies very fast changing dimensions and describes ways of handling such dimensions by creating one or more mini-dimensions.
8	Identify cases for snowflaking	Identifies which dimensions need to be snowflaked.
9	Other dimensional challenges are:	Description of other challenges:
	→ Identify Multi-valued Dimensions	Looks for multi-valued dimensions and describes ways of handling such dimensions in the design by using bridge tables.
	→ Identify Role-playing Dimensions	Describes ways of looking for dimensions that can be implemented using the role-playing concept.
	→ Identify Heterogeneous Dimensions	Describes ways of identifying heterogeneous products and implementing them in the design.
	→ Identify Garbage Dimensions	Describes ways to look for low-cardinality fields and use them for making a garbage dimension.
	→ Identify Hot Swappable Dimensions	Describes ways of creating profile-based tables or hot swappable dimensions to improve performance and secure data.

Note: We have listed activities to consider when designing dimensions. The purpose is to make you aware of several design techniques available to use, depending on the particular situation.

7.5.1 Identify dimensions

During this phase we identify the dimensions that are true to the grain selected in 7.4, “Identify the grain” on page 373. We were able to identify preliminary dimensions and facts just by looking at the grain definition in 7.4.5, “Identify high level dimensions and facts” on page 378. However, there are typically several dimensions that are true to the grain but cannot be identified by looking at the grain definition. The same is true for facts. The goal of this phase is to formally identify all dimension tables that are true to the grain definition.

The dimensions identified for the grain definition (a single line item on an invoice) are shown in Table 7-13.

Table 7-13 Dimensions list

Identified dimension	Granularity of the dimension	Source table
Invoice	Identifies each invoice. (Typically invoice numbers are treated as a degenerate dimension. We discussed this in 5.4.2, “Degenerate dimensions” on page 142. However, the Invoice number or invoice id is not treated as a degenerate dimension. We discuss this in 7.5.3, “Identify degenerate dimensions” on page 384.)	INVOICE
Customer	Contains information about the customer. A customer for the Redbook Vineyard is an organization.	CUSTOMER - CUSTOMER_DETAIL
Sales date	Contains all dates on which the products were sold. The date should be stored at the day granularity.	DAY
Beverage (Product)	Contains all beverages.	PRODUCT, PRODUCT_DETAIL
Sales Representative	Contains the information about the Sales Representatives.	SALES_REP
Currency	Contains the currency.	CURRENCY

Identified dimension	Granularity of the dimension	Source table
Warehouse	Consists of information for all warehouses that supply the products.	WAREHOUSE

The preliminary dimensional schema is shown in Figure 7-16. The dimension attributes are notated in the dimensional as TBD (To be Determined).

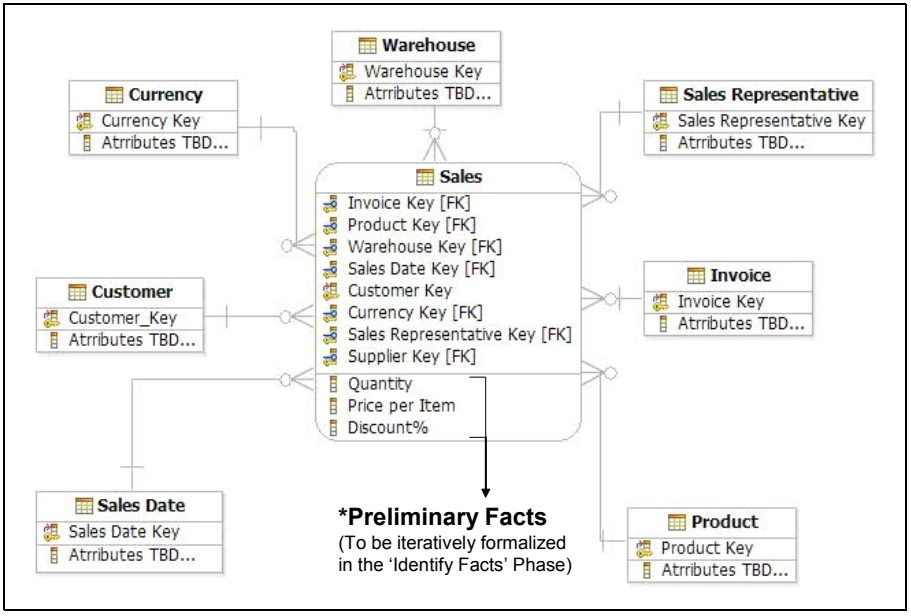


Figure 7-16 Preliminary Sales Invoicing star schema

Note: How to identify dimensions from an E/R model

As previously discussed, the source of a dimensional model can be either the enterprise data warehouse or the OLTP source systems. Both the data warehouse and the OLTP source systems may be based on an E/R model. That is, both may be in third normal form. So, once you learn to create a dimensional model from an E/R model, then you can create dimensional models either from the data warehouse or directly from the OLTP source systems.

The following are the steps involved in converting an E/R model into a dimensional model:

1: Identify the business process from the E/R model.
2: Identify the many-to-many tables in E/R model to convert to fact tables.
3: Denormalize the remaining tables into flat dimension tables.
4: Identify the date and time from E/R model.

For more detail about this process, see 6.1, “Converting an E/R model to a dimensional model” on page 210.

7.5.2 Check for existing conformed dimensions

This step in our case study is relatively simple. This is the first dimensional model implemented in the Redbook Vineyard company, and therefore no other processes exist which could have shared conformed dimension that we might use for our dimensional model. For more details, see 5.4.3, “Conformed dimensions” on page 144.

7.5.3 Identify degenerate dimensions

After we have identified dimensions, the next step is to identify degenerate dimensions (dimensions without attributes). The term degenerate dimension is not actually a dimension, but is often a form of a transaction number, from the OLTP source system, that has been placed the fact table. Degenerate dimensions are discussed in more detail in:

- “Identify degenerate dimensions” on page 384
- “Degenerate dimensions” on page 240

Coming back to our Sales Invoicing case study, we identified the dimensions in Table 7-13 on page 382. We discussed earlier that a transaction number, such as invoice number, is a degenerate dimension which is stored in the fact table as a

number. However, in our case study, we observe that the invoice number is not a number, but has several attributes associated with it. Attributes are:

- Order Number associated with the invoice
- First delivery date
- Order opening date
- Invoice date (note that invoice date is different from sales date)
- And more

We discuss more such attributes associated with the Invoice number next.

7.5.4 Identify dimensional attributes and hierarchies

In this activity we identify:

- Dimensional attributes for the dimensions identified in “Identify dimensions” on page 382
- Hierarchies, such as balanced, unbalanced, and ragged, associated with each of the dimensions

The purpose of this activity is to identify and define the attributes needed for each of the dimensions. These attributes are derived from the requirements collected in 7.3.6, “Gather the requirements” on page 362. More details are available in 5.4.4, “Dimensional attributes and hierarchies” on page 145. In addition to the attributes, we also identify all possible associated hierarchies with each of these dimensions.

In 7.5.1, “Identify dimensions” on page 382, we designed the preliminary star schema shown in Figure 7-16 on page 383.

Now let us fill this preliminary star schema with detailed dimensional attributes for each dimension. But first, let us define the granularity of each dimension table as shown in Table 7-14.

Table 7-14 Dimensions list with granularity

Identified dimension	Granularity of the dimension
Invoice	This dimension identifies each invoice and contains information such as order number and order date associated with a particular invoice number. (Typically invoice numbers are treated as a degenerate dimension. We discussed this in 5.4.2, “Degenerate dimensions” on page 142. However, in this case study, the Invoice number, or invoice id, is not treated as a degenerate dimension. We discuss this in 7.5.3, “Identify degenerate dimensions” on page 384.)

Identified dimension	Granularity of the dimension
Customer	The customer dimension contains information about the customer. A customer for the Redbook Vineyard is an organization.
Sales date	The sales date dimension contains all dates on which the products were sold. The date should be stored at the day granularity.
Beverage (Product)	The beverage dimension contains all beverages.
Sales Representative	This dimension contains the information about the Sales Representatives.
Currency	The currency dimension contains the currency used.
Warehouse	The warehouse dimension consists of information for all warehouses that supply the products.

The dimensions, along with their detailed attributes, are described in the following list:

- **Invoice dimension:** Contains information about the invoices for Customer Orders that have shipped. In other words, the granularity of the invoice dimension table is at single invoice header or invoice number. The invoice dimension is shown in Table 7-15.

Table 7-15 Invoice dimension's attributes

Attribute	Description
Invoice Key	It is a surrogate key which is a system-generated integer number.
Invoice Id	Identifier of the invoice.
Customer Order Id	The order reference number used by the customer to order the respective products. It will normally be a meaningless number such as KJS8374L.
First Delivery Date	Deliver date. When there are partial deliveries, it is the date when the first shipment takes place.
End Buyer Id	The company or person to whom the beverages are resold. This might be a company tax identifier or the name of a person.
Invoice Date	Date when the invoice is printed and sent to the customer.
Order Date	Date the order was opened.
Order Key	Surrogate key uniquely identifying an order dimension instance.

Attribute	Description
Payment Due Date	Date when the payment of the invoice is due. This date can be the same as when the last shipment of the order was or can be later.
Payment Effective Date	Date when the invoice was paid by the customer.
Payment Method	The type of payment method, for example, credit card, cash, bank transfer, and check.

- **Customer dimension**

The customer dimension contains data about the organizations, normally resellers, to whom the company sells. It also describes the hierarchical structure of companies or corporations. In other words, the organizations which buy from the Redbook Vineyard company are its customers. The customer dimension is shown in Table 7-16.

Table 7-16 Customer dimension's attributes

Attribute	Description
Customer Key	A surrogate key, which is a system-generated integer number. This number uniquely identifies each customer.
Name	The name of the organization or the customer. For example, Composed Gupta and Gupta, Inc., or Ballard Halogen Lights, Inc.
Acronym	The acronym or short name by which the company is known, for example, CGGS for Composed Gupta and Gupta, Inc., or BHL for Ballard Halogen Lights, Inc.
Allows Partial Shipment	It specifies if partial shipment is allowed in case the order cannot be fully shipped at once. If the flag is true and one or more of the ordered products is in stock, then the order will be partially shipped.
City	The city where the customer has its main address, for example, New Delhi or Prague.
Comments	Any comment pertinent and of value to the reporting about the customer that cannot go in any of the existing fields.
Company Tax Id	The identifier of the company used for tax purposes. This identifier is unique for every customer and can be used for identifying the customer across its history in the customer dimension table.
Country	The country of the customer. For example, United Kingdom or Australia.

Attribute	Description
Delivery Method	The preferred delivery method for the organization.
Financial Group	The name of the internal financial group within the customer.
Marketing Category	Marketing customer category.
Parent Organization Key	It maps to the surrogate key of the parent organization to which the customer belongs. This is used to store information about the hierarchical structures of corporations. For example, CorpHQ France mapping to the parent CorpHQ, which is the top parent company worldwide, or Buy and Drink Overseas Department, belonging to Buy and Drink, Inc.
Region	The region, as defined by the company marketing organization, where the customer country is located, for example, EMEA for Morocco, or SouthAmerica for Argentina,
State	The state, province, department, or canton, where the customer has their main address. For example, California in USA, Provence in France, or Valais in Switzerland.
Type	The type of the customer. For example, small, medium, or large.

- **Sales Date dimension:** The date on which the product was ordered. This table could be progressively populated as new orders arrive, or we could also make a one time insertion with the all the date-month-year combinations for a particular time period. The granularity of the sales date dimension is a single day.

There are more details about the importance of choosing a correct granularity for the date dimension in 5.4.5, “Date and time granularity” on page 155. The sales date dimension attributes are shown in Table 7-17.

Table 7-17 Sales date dimension attributes

Attribute	Description
Sales Date Key	Surrogate key uniquely identifying a Sales Date dimension instance.
Day	Date indicating an exact day, such as December 1, 2010.
Week Day	Day of the week, such as Monday or Tuesday.

Attribute	Description
Month	Name of the month, such as January or February.
Quarter	The quarter of the year: 1, 2, 3, or 4.
Year	It indicates a year, such as 1994 or 2008.

For the case study on the Sales Invoicing business process, we discussed the sales date dimension in 7.5.6, “Date and time dimension and granularity” on page 399.

- **Product dimension:** Describes the products or beverages sold by the company. It currently only contains the drinkable products also commonly known as beverages. The product indicates what drink is contained in what type and size of container. For example, a soda in a one liter glass bottle, or in a 33cl. aluminum can. The company also sells other non-beverage products, such as T-Shirts and glasses, but they reside in a different sales fact table and are not covered in this process.

  The granularity initially requested in the case study was at the beverage level. For example, just specifying a brand of lemonade or a brand of soda. However, we should always try to go to the lowest level of available granularity to enable handling more demanding requirements that might exist in the future. The product granularity is a single product, which is defined as a particular beverage with a particular volume and a particular presentation form. The product dimension table attributes are defined in Table 7-18.

Table 7-18 Product dimension attributes

Attribute	Description
Product Key	Surrogate key, which is a system-generated integer.
Name	The name of the product.
Acronym	A short name, or initials, for a beverage, such as ATS for Amit Tasty Soda.
Beverage	Identifies the beverage, such as Amit Tasty Soda or RedBook Lemonade.
Beverage Group	The group of drinks to which a beverage belongs, for example, Soft Drinks.
Description	An adequate description of the product with important characteristics that do not have a corresponding field in the product table.

Attribute	Description
Financial Group	Which financial group includes this product.
Presentation	The type of container used, such as a bottle or value-pack.
Product Id	The identifier of the product, commonly known to the users, or as generated in the source system. Normally a number or code such as SOD-40-L, not necessarily with any particular meaning.
Status	The status of the product, such as active or inactive.
Supplier	The name of the supplier of the products, such as Soda Supplier-1.
Units Per Package	The number of units shipped in simple packages.
Units Per Pallete	The number of units of product that can be shipped per pallet or crate.
Vat Code	The VAT or Tax code applied to the product, such as 5%, or 20%.

- **Sales Representative Dimension**

The Sales Representative dimension contains data about the company sales force. The Redbook Vineyard management is keenly interested in tracking the performance of their Sales Representatives. The granularity of the Sales Representative dimension is a single Sales Representative. The Sales Representative dimension attributes are shown in Table 7-19.

Table 7-19 Sales Representative dimension attributes

Attribute	Description
Sales Representative Key	Surrogate key which is a system-generated integer. This number uniquely identifies each employee.
Name	Employee name.
Comments	Any additional comments required about the Sales Representative.
Manager Name	Name of the manager of the sales employee.
Employee Id	A number or code identifying the employee.
Sales Office	Sales office where the customer representative works.

Attribute	Description
City	Sales office city location.
Country	Name of the country where the sales office is located.

- **Warehouse dimension:** This dimension contains information about the warehouse from which the ordered products are shipped. The granularity of the warehouse is a single warehouse at a location. Of course, each single warehouse is actually structured in areas, rooms, and shelves. To better track which product came from which shelf of the warehouse, we could keep the grain of the warehouse dimension to shelf-location within a warehouse. However, this atomic level detail of information was not available in the original ERP system from where the data warehouse was populated. The warehouse dimension attributes are described in Table 7-20.

Table 7-20 Warehouse dimension's attributes

Attribute	Description
Warehouse Key	Surrogate key which is a system-generated integer. This number uniquely identifies each warehouse.
Name	Name of the warehouse.
Acronym	A short name for the warehouse.
City	The name of the city where the warehouse is located.
Country	The country where the warehouse is located.
Description	Text describing additional features and characteristics, or special remarks.

- **Currency dimension:** This dimension contains a list with the currencies used in local orders and invoices. It also identifies which is the currency used for reporting and consolidation purposes. The granularity of the currency dimension is a single currency. The currency dimension table attributes are described in Table 7-21.

Table 7-21 Currency dimension's attributes

Attribute	Description
Currency Key	Surrogate key uniquely identifying a currency dimension instance.
Currency Name	The official currency name, such as Sterling Pound and Swiss Franc.

Attribute	Description
Description	The description of the currency.
Is Corporate Currency?	This flag indicates the currency used for consolidating financial statements corporate and worldwide. Only one currency can be the corporate currency, but it can change across the Sales Date.
ISO Code	The ISO code by which the currency is known. For example, USD for US dollars.

The dimension schema we get after identifying the dimensions attributes is shown in Figure 7-17.

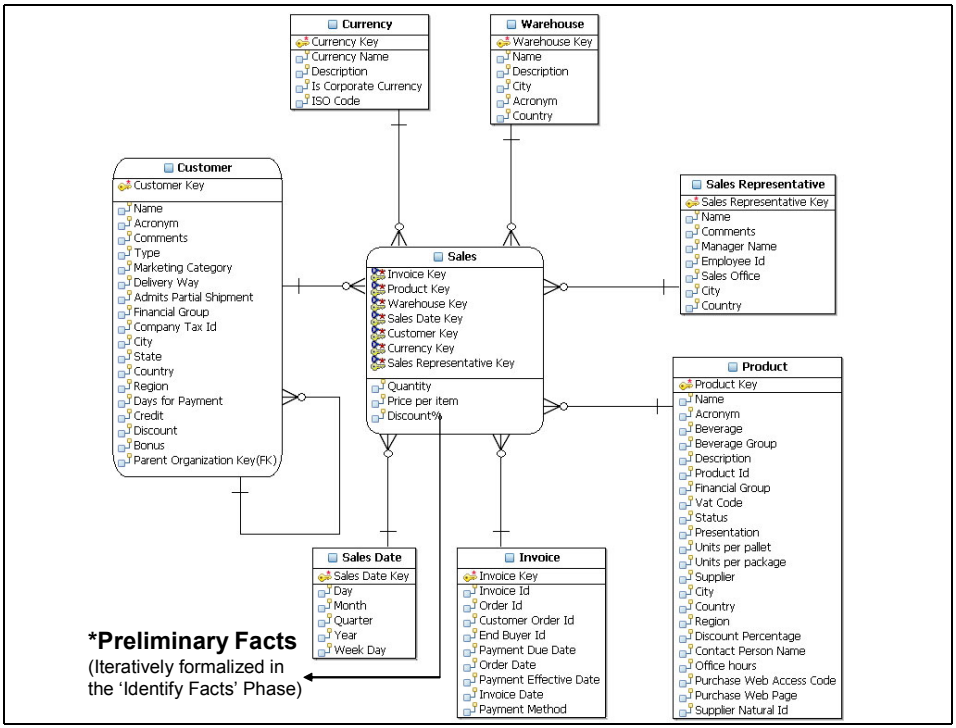


Figure 7-17 Star schema after identifying dimension attributes

In the next activity, we further identify the hierarchies that each dimension has and also identify what attributes of the hierarchies.

7.5.5 Identifying the hierarchies in the dimensions

A hierarchy is a cascaded series of many-to-one relationships. A hierarchy basically consists of different levels, with each level corresponding to a dimension attribute.

In other words, a hierarchy is a specification of levels that represents relationships between different attributes within a hierarchy. For example, one possible hierarchy in the date dimension is Year → Quarter → Month → Day.

There are three major types of hierarchies, and they are described in Table 7-22.

Table 7-22 Different types of hierarchies

S.No	Hierarchy name	Hierarchy description	How is this hierarchy implemented?
1	Balanced	A balanced hierarchy is a hierarchy in which all the dimension branches have the same number of levels. For more details, see “Balanced hierarchy” on page 249.	See “How to implement a balanced hierarchy” on page 250.
2	Unbalanced	A hierarchy is unbalanced if it has dimension branches containing varying numbers of levels. Parent-child dimensions support unbalanced hierarchies. For more details, see “Unbalanced hierarchy” on page 251.	See “How to implement an unbalanced hierarchy” on page 252.
3	Ragged	A ragged dimension contains at least one member whose parent belongs to a hierarchy that is more than one level above the child. Ragged dimensions, therefore contain branches with varying depths. For more details, see “Ragged hierarchy” on page 260.	See “How to implement a ragged hierarchy in dimensions” on page 261.

Note: A dimension table may consist of multiple hierarchies, as well as attributes or columns that belong to one, more, or no hierarchies.

For the Sales Invoicing example, Table 7-23 shows the various hierarchies present in the dimensions.

Table 7-23 Dimensions and hierarchies

Seq. no.	Dimension name	Hierarchy description	Type of hierarchy
1	Sales Date	Year → Quarter → Month → Week Day	Balanced
2	Invoice	None	
3	Product	Financial Group → Beverage Group → Beverage	Balanced
4	Sales Representative	Sales Rep Country → Sales Rep Region → Sales Rep Office	Balanced
5	Currency	None	
6	Customer	Customer Country → Customer Region → Customer City	Balanced
7	Customer	Customer has an unbalanced hierarchy.	Unbalanced
8	Warehouse	Warehouse Country → Warehouse Region → Warehouse City	Balanced

A detailed discussion about handling hierarchies is given in 6.3.4, “Handling dimension hierarchies” on page 248.

Note: Geographical objects, such as cities, countries, and regions, should always be considered as part of a Geography hierarchy.

Unbalanced hierarchy

A hierarchy is unbalanced if it has dimension branches containing varying numbers of levels. Parent-child dimensions support unbalanced hierarchies. For example, the parent-child relationship present in the Customer dimension (see Figure 7-18 on page 395) supports an unbalanced hierarchy as shown in Figure 7-19 on page 396.

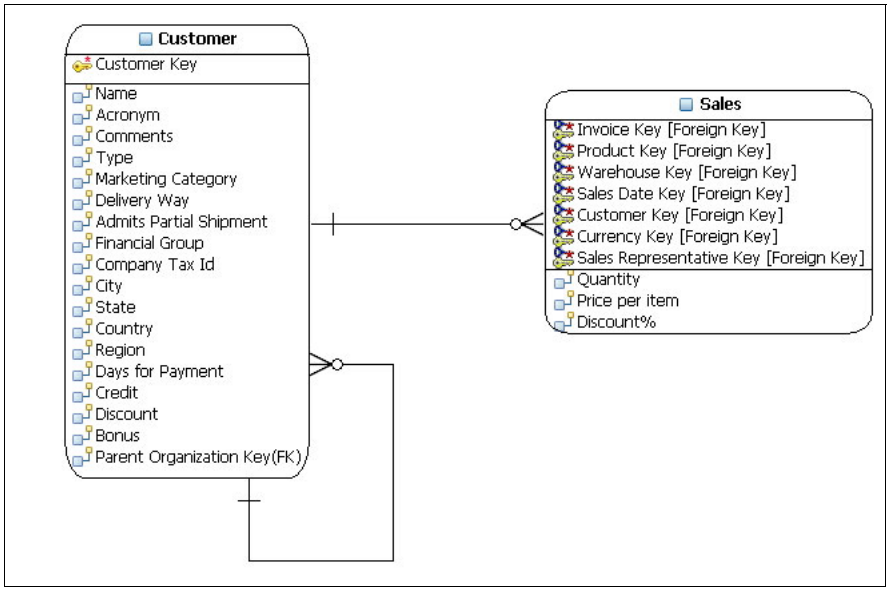


Figure 7-18 Recursive pointer to show parent-child relationship

It is important to understand that representing an arbitrary, unbalanced hierarchy is an inherently difficult task in a relational environment. A common example of an unbalanced hierarchy is one that represents the parent-child companies. A company at a given level may have several smaller companies below it, while others at the same level may have none or a few. If you wish to create a report which computes the sales totals for all companies at a given level (see levels in Figure 7-19 on page 396), you will find this question more efficiently answered with an OLAP (cube-based) reporting system than with a relational database.

However, if your hierarchies are symmetrical, then arguably either type of technology (OLAP Cubes or Relational database) is equally capable of providing the answer. For example, the date hierarchy is completely symmetrical. Each year has the same number of quarters, each quarter has the same number of months, and each month has same number of weeks in it.

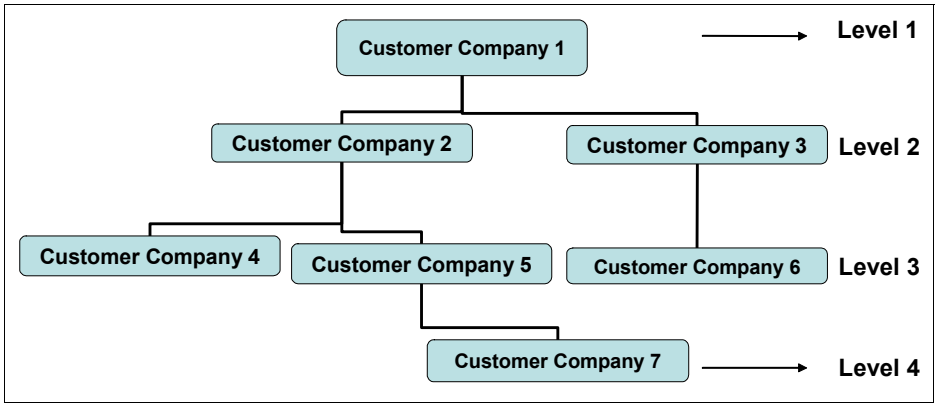


Figure 7-19 Unbalanced hierarchy represented by the Customer table

The customer dimension is an unbalanced hierarchy, as shown in Figure 7-19. It has levels with a consistent parent-child relationship, but that have logically inconsistent levels. The hierarchy branches also can have inconsistent depths. An unbalanced hierarchy can represent a parent-child company.

Let us assume that we want a report which shows the total sales revenue at the node Customer Company 2. What this means is that we want the report to show the total sales made to Customer Company 2, Customer Company 4, Customer Company 5, Customer Company 7, and Customer Company 8. We cannot use SQL to answer this question because the GROUP BY function in SQL cannot be used to follow the recursive tree structure downward to summarize an additive fact such as sales revenue. Therefore, because of this problem, we cannot connect a recursive dimension (Figure 7-18 on page 395) to any fact table.

So, how do we solve the query using an SQL Group-by clause? And, how can we summarize the sales revenue at any node?

The answer is to use a Bridge table between the Customer and SALES table as shown in Figure 7-20 on page 397. The aim of the bridge table is to help us traverse the unbalanced hierarchy.

The customer table will have a hierarchy built in itself. For example, Customer A is a corporation that owns Customer B and C, and Customer D is a division of B. In our dimensional model, the Fact Gross Sales Amount for customer A will not include any of the sales amounts relative to B, C, or D. The same applies to every single child in the hierarchical tree.

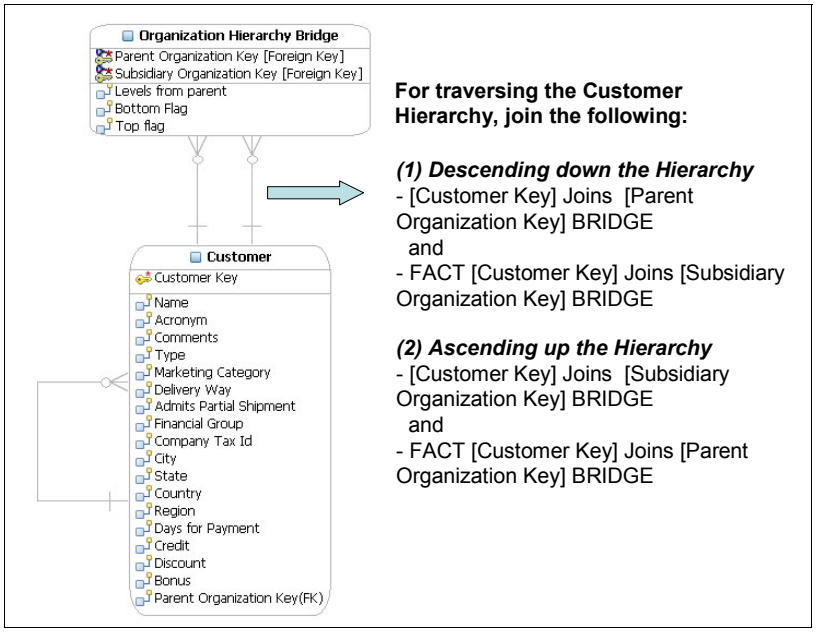


Figure 7-20 Organization bridge table for level navigation

There are two ways in which you can move through the unbalanced hierarchy shown in Figure 7-19 on page 396.

- Descending the hierarchy: For this you need to make joins between Customer, Sales, and Organization_Hierarchy_Bridge as shown in Figure 7-20. We show one example of such a query in Example 7-1 on page 398.
- Ascending the hierarchy: For this you need to make joins between Customer, Sales, and Organization_Hierarchy_Bridge as shown in Figure 7-20.

The bridge table is called this because, when running queries and joining the facts table with the respective dimension, this new table will have to be placed in the middle of the other two tables and used as a bridge, as you can see in Figure 7-21 on page 398.

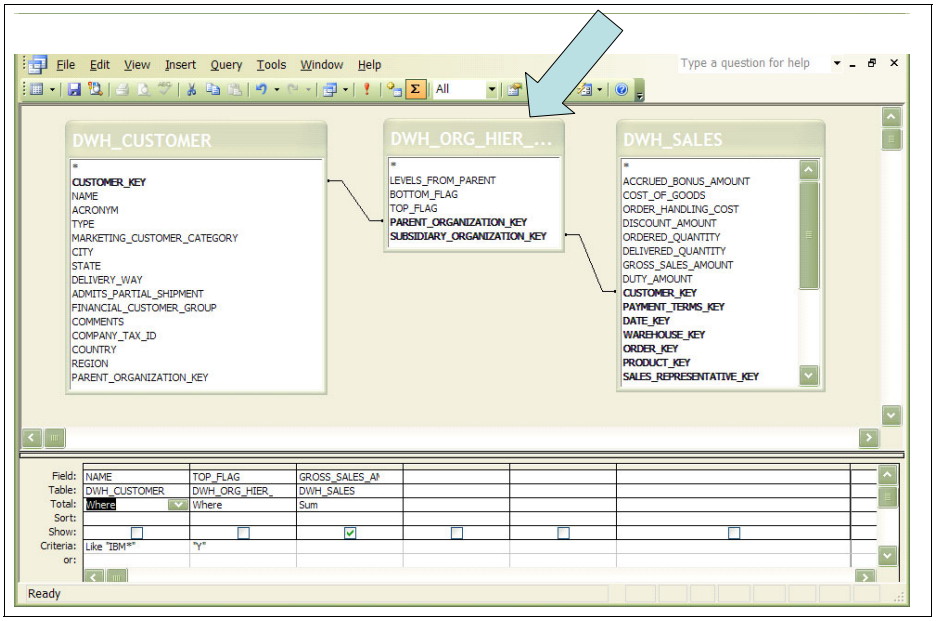


Figure 7-21 Graphical presentation of a bridge table

Figure 7-21 shows a graphical query where we calculate the total number of customer sales to the IBM corporation, including all its branches and owned companies, since the beginning of the data mart history.

And here is the SQL code produced by the tool used to design and run that query.

Example 7-1 SQL for traversing down the hierarchy

```
SELECT Sum(DWH_SALES.GROSS_SALES_AMOUNT) AS Sum_Of_GROSS_SALES_AMOUNT
FROM DWH_SALES INNER JOIN
   (DWH_CUSTOMER INNER JOIN
      DWH_ORG_HIER_BRIDGE
      ON DWH_CUSTOMER.CUSTOMER_KEY=
         DWH_ORG_HIER_BRIDGE.PARENT_ORGANIZATION_KEY)
   ON DWH_SALES.CUSTOMER_KEY = DWH_ORG_HIER_BRIDGE.SUBSIDIARY_ORGANIZATION_KEY
WHERE DWH_CUSTOMER.NAME Like "IBM*" AND DWH_ORG_HIER_BRIDGE.TOP_FLAG="Y";
```

For more detail about how to implement and traverse a parent-child hierarchy as shown in Figure 7-19 on page 396, refer to “How to implement an unbalanced hierarchy” on page 252.

The dimensional model we have developed to this point, after having identified the various hierarchies, is shown in Figure 7-22 on page 399.

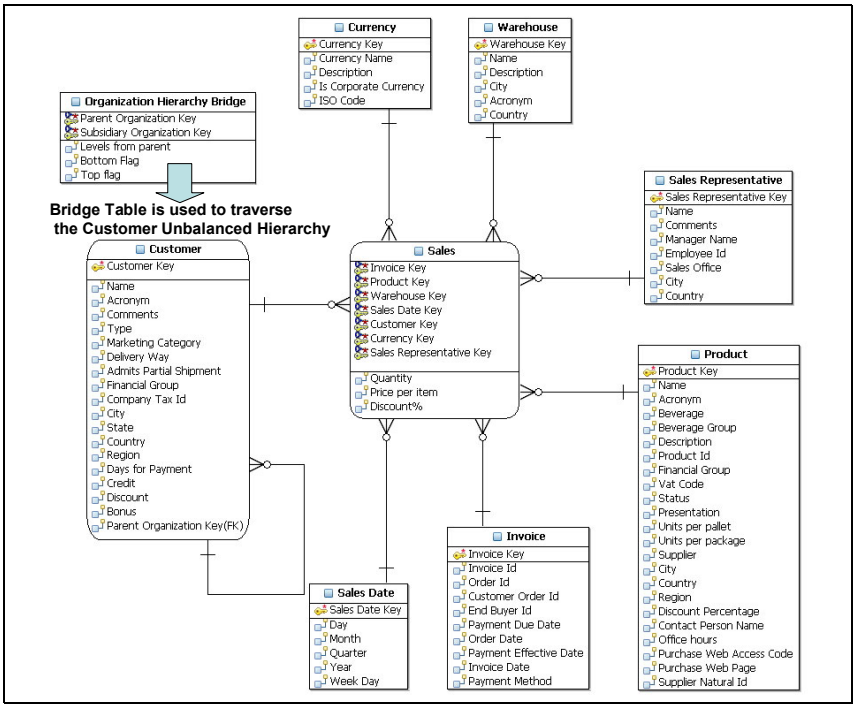


Figure 7-22 Dimensional model developed after identifying hierarchies

7.5.6 Date and time dimension and granularity

The date and time dimensions are considered the most important dimensions in dimensional modeling. The identification of the grain is crucial for the success of the dimensional database. For more details, see 5.4.5, “Date and time granularity” on page 155. The granularity specifically required for the Sales Invoicing dimensional model is a single day. Although a lower granularity would be possible, down to the timestamp level, the information in the data warehouse is at the date level, and a lower granularity is not considered to have any additional business value. However, if our data mart was for the ordering process, then a granularity at a timestamp level could have an additional business value for analysis, for example, for orders received through the Web.

The requirements for the sales invoice process can be fulfilled by the date dimension as shown in Table 7-24 on page 400.

Table 7-24 Example of Sales Date table dimension

Day	Month	Quarter	Week day	Year
3/10/2005	3	1	4	2005
3/11/2005	3	1	5	2005
3/12/2005	3	1	6	2005
3/13/2005	3	1	7	2005

7.5.7 Handling slowly changing dimensions

The Redbook Vineyard stores full history for product, warehouse, customer, Sales Representative, invoice, and currency in the Enterprise Data Warehouse. Slow changing dimensions do not normally represent a problem, and we only need to decide what type of change management strategy we are going to use: Type-1, Type-2, or Type-3.

In the case study, the product, warehouse, Sales Representative, currency, and customer dimensions are slowly changing dimensions, but they have different requirements. The change handling strategies are listed in Table 7-25.

Table 7-25 Slowly changing dimensions

Dimension	Requirements	Change handling strategy
Product	Keep the history on product structure and corresponding attributes.	Type-2
Warehouse	No special requirement. Information can be overwritten.	Type-1
Sales representative	Keep the history on sales structure and corresponding attributes.	Type-2
Invoice	The only changes to this dimension will be the population of some date fields. But once populated, the fields will not change.	Type-1
Sales date	This dimension will not change.	NA

7.5.8 Handling fast changing dimensions

In this section we identify the very fast changing dimensions that cannot be handled using the Type-1, Type-2, or Type-3 approaches discussed in 5.4.6, “Slowly changing dimensions” on page 159.

Fast changing dimensions are not so easy to handle, and normally require a different approach than the Type-1, Type-2, and Type-3 strategies. The information contained in fast changing dimensions is subject to change with a relatively high frequency. Table 7-26 shows the dimension identified as fast changing.

Table 7-26 Fast changing dimensions

Dimension	Requirements	Type
Customer	Keep the history for payment conditions, and credit and discount limits.	Fast Changing Dimension

An approach for handling very fast changing dimensions is to break off the fast changing attributes into one or more separate dimensions, also called mini-dimensions. The fact table would then have two foreign keys—one for the primary dimension table and another for the fast changing attributes. For more detailed discussion about handling fast changing dimensions, refer to 5.4.7, “Fast changing dimensions” on page 162.

We identify the following fast changing attributes for the customer dimension shown in Figure 7-23:

- Days for Payment
- Credit
- Discount
- Bonus

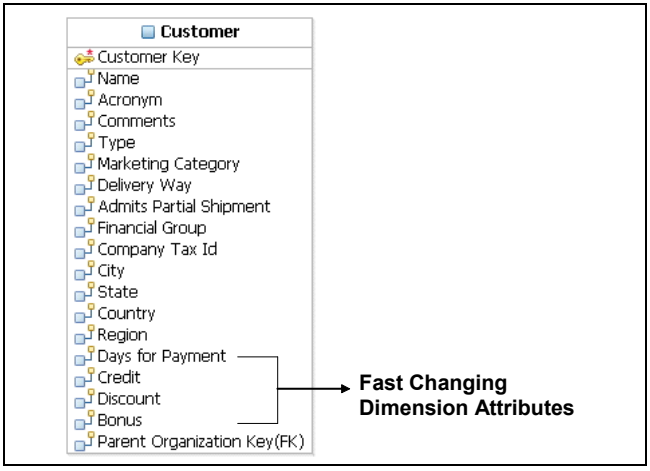


Figure 7-23 Customer with detailed financial conditions

For the fast changing attributes, we create a mini-dimension called Payment Terms. The Payment Terms dimension is shown in Table 7-27 on page 402. In

order to handle the fast growth, we create this mini-dimension with attributes which have a range of values. In other words, each of the attributes has a value in a particular band.

Table 7-27 Payment terms dimension table

Attribute	Description
Payment Terms Key	Surrogate key uniquely identifying a payment term dimension instance.
Credit Band	The credit band groups different ranges of credit amounts allowed for the customer. For example: 1 Low credit: Less than 1000; 2 Medium credit: Between 1000_ and 4999; 3 High credit: Between 5000_ and 10000; 4 Extra high credit: Greater than 10000.
Days For Payment Band	This band groups the different times or plans for invoice payments. For example, immediate: 1 - 30 days; more than 30 days. The customer can arrange a payment period in multiples of 15 days, or arrange for an immediate payment. In theory the customer could have to pay inmediately, in 15, 30, 45, or 60 days.
Discount Band	This describes the bands grouping the different discount ranges. For example, no discount; less than 10%; between 10% and 30%; more than 30%.
Bonus Band	This describes the bands grouping the different bonus ranges. For example: No bonus; less than 5%; between 5% and 10%; and more than 10%.

Figure 7-24 on page 403 shows the new mini-dimension (Payment Terms) with the customer financial conditions and the updated customer dimension.

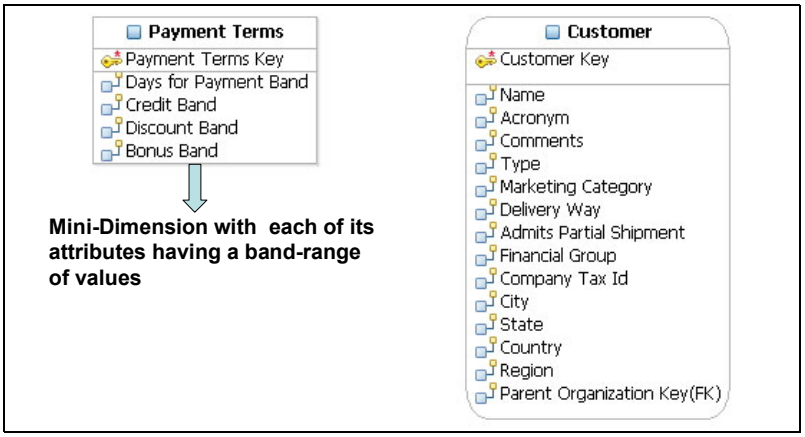


Figure 7-24 Mini-dimension for customer financial conditions

Each of the dimensions (Customer and Payment Terms) are joined separately to the Sales fact table. The dimensional model developed at this point (after identifying slowly and fast changing dimensions) is shown in Figure 7-25 on page 404.

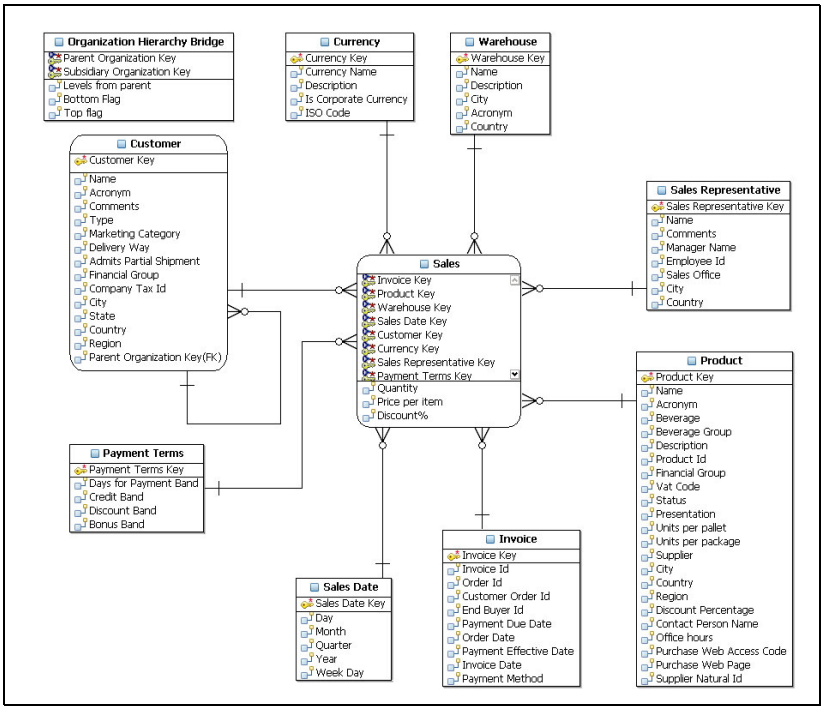


Figure 7-25 Model after identifying fast and slowly changing dimensions

7.5.9 Identify cases for snowflaking

In this section we identify potential cases for snowflaking for all the dimension tables. Table 7-28 shows the snowflakes that we have identified and for what reasons.

Table 7-28 Snowflaking actions per dimension

Action	Objects	Reason
Snowflaking	Supplier from products	There is a significant number of attributes which are strictly supplier dependent. In other words, the supplier has several attributes which are at a different grain than the overall product table. It is very likely that this table will be used as a dimension by the Procurement department with their future data mart, which will have a different fact table, but will share some conformed dimensions. The supplier dimension then would be shared with the procurement business process dimensional model.

Refer to 6.3.7, “Identifying dimensions that need to be snowflaked” on page 277, for more topics on snowflaking, such as:

- When to snowflake
- When to avoid snowflaking
- When to snowflake to improve performance
- Disadvantages of snowflaking

Table 7-26 on page 401 depicts the dimensional model after we have identified the dimensions that will be snowflaked.

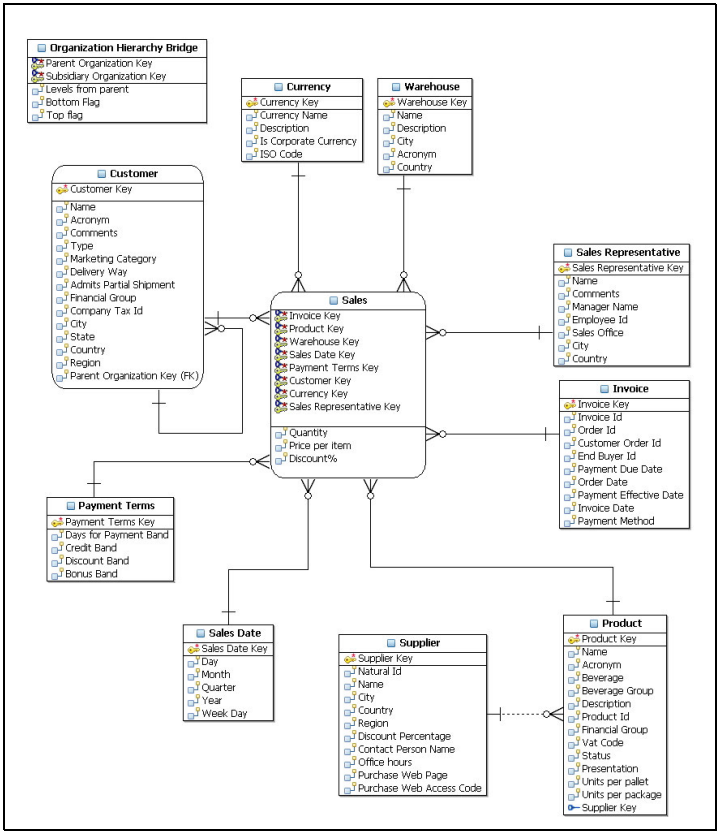


Figure 7-26 Model after identifying snowflake dimensions

7.5.10 Handling other dimensional challenges

In this section we identify special types of dimensions that may be applicable to the Sales Invoicing case study. The special dimensions that we identify are shown in Table 7-29 on page 406.

Table 7-29 Other special dimensions to identify

Seq no.	Dimension type	How is it implemented?
1	Multi-valued dimension	**Description** Typically, while designing a dimensional model, each dimension attribute should take on a single value in the context of each measurement inside the fact table. However, there are situations where we need to attach a multi-valued dimension table to the fact table. In other words, there are situations where there may be more than one value of a dimension for each measurement. Such cases are handled using multi-valued dimensions. **Implementation** Multi-valued dimensions are implemented using Bridge tables. For a detailed discussion, refer to 6.3.10, “Multi-valued dimensions” on page 288. **Sales Invoicing case study** For the case study on Sales Invoicing, we do not have any scenarios with multi-valued dimension. This is primarily because each attribute in each of the dimensions takes on a single value for a particular fact table row.
2	Role-playing dimension	**Description** A single dimension, which is expressed differently in a fact table using views, is called a role-playing dimension. A date dimension is typically implemented using the role-playing concept when designing a dimensional model using the accumulating snapshot fact table. This is discussed in more detail in “Accumulating fact table” on page 233. **Implementation** The Role-Playing dimensions are implemented using views. This procedure is explained in detail in 6.3.9, “Role-playing dimensions” on page 285. **Sales Invoicing case study** For the case study on Sales Invoicing, we do not have any scenarios for dimensions that can be implemented using role-playing.

Seq no.	Dimension type	How is it implemented?
3	Heterogeneous dimension	**Description** Heterogeneous products can have different attributes, so it is not possible to make a single product table. Let us assume that an insurance company sells different kinds of insurance, such as Car Insurance, Home Insurance, Flood Insurance, and Life Insurance. Each type of insurance can be treated as a product. However, we cannot create a Single Product table to handle all these types of insurance because each has extremely unique attributes. **Implementation** Here are ways to implement heterogeneous dimensions: ► Merge all the attributes into a single product table and all facts relating to the heterogeneous attributes in one fact table. ► Create separate dimensions and fact tables for the heterogeneous products. ► Create a generic design to include a single Fact and Single Product Dimension table with common attributes from two or more heterogeneous products. The above mentioned concepts of implementing heterogeneous dimensions are discussed in more detail in 6.3.12, “Heterogeneous products” on page 292. **Sales Invoicing case study** The Sales Invoicing case study has two dimension groups, beverages and non-beverages, that do not share all the data in the Product dimension. For example, the fact table contains measures only applicable to the beverages (such as alcohol content volume and delivered liters). And, the two main groups of products have specific attributes that are applicable to one but not the other. We therefore conclude that the product is a heterogeneous dimension. However, we decided to leave both subtype attributes and measures in the same dimension and facts table. Any specific attributes since they are not needed by the business. Product is a heterogeneous dimension but will be left unchanged.

Seq no.	Dimension type	How is it implemented?
4	Garbage dimension	**Description** A dimension that consists of low-cardinality columns, such as codes, indicators, status, and flags, also referred to as a junk dimension. The attributes in a garbage dimension are not related to any hierarchy. **Implementation** Implementation involves separating the low-cardinality attributes and creating a dimension for such attributes. This implementation procedure is discussed in detail in 6.3.8, “Identifying garbage dimensions” on page 282. **Sales Invoicing case study** For the case study, we do not have any situations involving a garbage dimension.
5	Hot swappable dimension	**Description** A dimension that has multiple alternate versions of itself, and can be swapped at query time, is called a hot swappable dimension or profile table. Each of the versions of the hot swappable dimension can be of a different structure. The alternate versions access the same fact table, but have different output. The versions of the primary dimension may be completely different, including incompatible attribute names and different hierarchies. **Implementation** The procedure to implement hot swappable dimensions is discussed in detail in 6.3.13, “Hot swappable dimensions or profile tables” on page 294. **Sales Invoicing case study** For the case study about Sales Invoicing, we do not create hot swappable dimensions. However, we could if we decide to implement tighter security, or for performance reasons.

7.5.11 Dimensional model containing final dimensions

Figure 7-27 shows the dimension model up to the point of having completed the *Identify dimensions* phase of the DMDL.

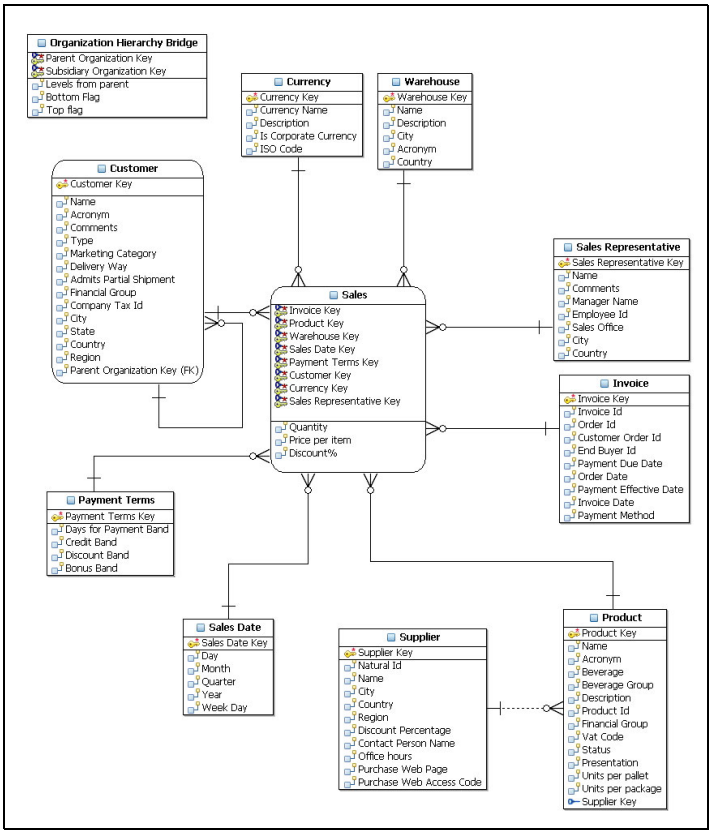


Figure 7-27 Model after "identify dimensions" phase

7.6 Identify the facts

In the "Identify the facts" phase, shown in Figure 7-28 on page 410, the primary goal is to provide more detail on the fact table.

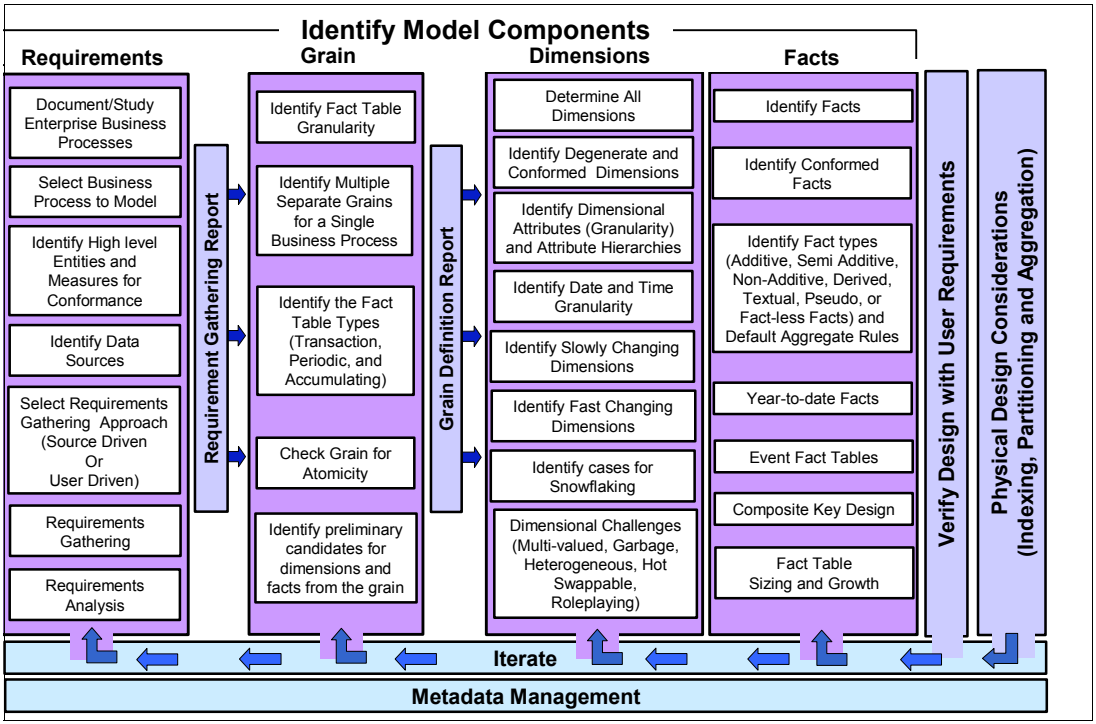


Figure 7-28 Dimensional Model Design Life Cycle

We have identified preliminary dimensions and preliminary facts, for the sales invoice business process, using the grain definition shown in Figure 7-29. In this phase we will iteratively and in more detail, identify additional facts that are true to this grain.

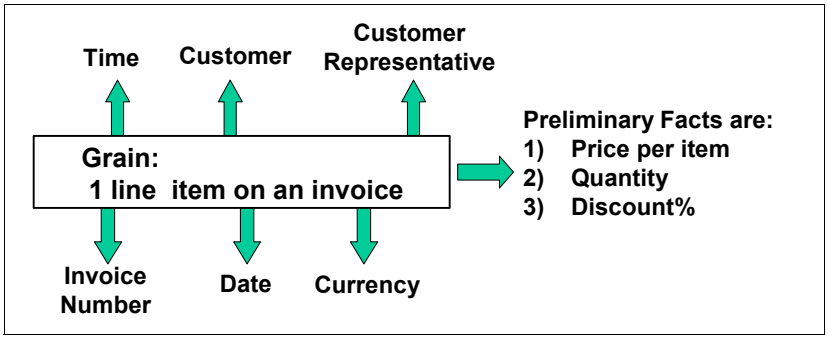


Figure 7-29 Grain definition for sales invoice business

Table 7-30 on page 411 shows the activities associated with the *Identify the facts* phase.

Table 7-30 Activities in the Identify the Facts phase

S no.	Activity name	Activity description
1	Identify facts	Identifies the facts that are true to the grain identified for our Sales Invoicing process. The grain was identified in 7.4, “Identify the grain” on page 373.
2	Identify conformed facts	Identify any conformed facts. That is, in this activity we identify if any of these facts have been conformed and shared across the organization. If yes, we use those conformed facts.
3	Identify fact types	Identify the fact types, such as: - Additive - Semi-additive - Non-additive - Derived - Textual - Pseudo - Factless
4	Year-to-date facts	Year-to-date facts are numeric totals that consist of aggregated totals from start of year until the current date. In this activity we discuss year-to-date facts to make sure such facts are not included in a fact table which includes data at the atomic line item level.
5	Event fact tables	In this activity we describe how to handle events in the event-based fact table, and highlight the pseudo and factless facts that may be associated with such tables.
6	Composite key design	General guidelines for designing the primary composite key of the fact table. We also describe situations when a degenerate dimension may be included inside the fact table composite primary key.
7	Fact table sizing and growth	In this activity we describe guidelines to use to predict fact table growth.

We now discuss each of the activities listed in Table 7-30.

7.6.1 Identify facts

Identifying and documenting the fact types is important for the applications and users of the dimensional model. There are several facts, other than the preliminary facts, that cannot be known by glancing at the grain definition. Such facts are detailed and derived facts that need further analysis to be found.

In this activity we identify all the facts that are true to the grain. These facts include the following:

- The preliminary facts identified in “Identify high level dimensions and facts” on page 378. Preliminary facts are easily identified just by looking at the grain definition or the invoice.
- Detailed facts, such as cost per individual product, labor manufacturing cost per product, or transportation cost per individual product. These facts can only be identified after a detailed analysis of the source E/R model to identify all the facts that are true at the line item grain.

The sales fact table contains information about the customers that ordered the products, the products ordered, the date on which the product is ordered, the warehouse from which the products are shipped, the payment terms applied to the ordered item, and the currency in which the ordered item is invoiced.

Table 7-31 shows the facts identified for the Sales Invoicing business process that are true to the line item grain definition (see Figure 7-29 on page 410).

Table 7-31 Sales fact table

#	Fact	Description
1	Accrued Bonus Amount	An amount of money in the corporate currency indicating the bonus the customer gets for purchasing a product. Accrued Bonus Amount= Quantity sold X Accrued Bonus per Item.
2	Alcohol Duty Amount	The amount in corporate currency to be paid for duty if the product is an alcoholic drink. [Alcohol Duty Amount= Quantity sold X Alcohol Duty per Item].
3	Alcohol Volume Percent	Alcohol by volume (ABV) is an indication of how much alcohol (expressed as a percentage) is included in an alcoholic beverage. This measurement is assumed to be a world standard, although in the United States, the predominant measurement is Alcohol by weight (also known as ABW). Another way of specifying the amount of alcohol is alcoholic proof.
4	Cost Of Goods	Also known as cost of product in corporate currency excluding the taxes. [Cost of Goods = Quantity sold x Cost of good per item].

#	Fact	Description
5	Delivered Liters	Amount of liters shipped with a product. [Delivered Liters = Quantity sold x Delivered Liters per Item].
6	Quantity Sold	The quantity of the product that has been already sold. **Note:** This was a preliminary fact we identified in 7.4.5, “Identify high level dimensions and facts” on page 378.
7	Discount Amount	The amount of money, in corporate currency, the customer gets as a discount. **Note:** This was a preliminary fact we identified in 7.4.5, “Identify high level dimensions and facts” on page 378. [Discount Amount = Quantity sold x Discount Amount per item].
8	Gross Sales Amount	The amount of money in corporate currency already invoiced. **Note:** We identified *Price per item* as preliminary fact in 7.4.5, “Identify high level dimensions and facts” on page 378. Instead of storing ‘Price per item’ as a fact, we store the Gross sales cost for the line item as Quantity sold x Price per item. [Gross Sales Amount = Quantity sold x Price per item].
9	Lc Bonus Amount	The amount of money, in the ordered local currency, indicating the bonus the customer gets for purchasing the product. [Local Bonus Amount = Quantity sold x Bonus Amount per item].
10	Lc Cost Of Goods	The net amount of money in the ordered local currency the product costs. [Cost of Goods = Quantity sold x Cost of good per item].
11	Lc Discount Amount	The amount of money in the ordered currency the customer gets discounted on the product. [Discount Amount = Quantity sold x Discount Amount per item].
12	Lc Duty Amount	The amount in ordered local currency to be paid for duty if the product is an alcoholic drink. [Alcohol Duty Amount = Quantity sold X Alcohol Duty per Item].
13	Lc Gross Sales Amount	The amount of money in ordered local currency already invoiced for this product. [Gross Sales Amount = Quantity sold x Price per item].
14	Lc Order Handling Cost	The amount of money in the order currency for order handling. This information is not kept at the order level because certain kinds of liquors are very expensive luxury articles that need to be treated very carefully and be specially packaged. [Order Handling Cost = Quantity sold x Order Handling Cost per Item].

#	Fact	Description
15	Order Handling Cost	The amount of money in the corporate currency for order handling. This information is not kept at the order level because certain kinds of liquors are very expensive luxury articles that need to be treated very carefully and be specially packaged. [Order Handling Cost = Quantity sold x Order Handling Cost per Item].
16	Spirit Liters	Amount of alcohol in liters per item = Delivered Liters * (Alcohol Volume Pct / 100).
17	Gross profit	Profit calculated as gross sales income less the cost of sales.
18	Gross margin Percent	Percentage difference between the cost of sales and the gross sales.
19	Net sales Amount	The amount of the sale. This is a negative amount, if it is a credit note, due to returned or damaged material.
20	Net profit Amount	Net Sales - Cost of Sales - Alcohol Duty Amount

How to identify facts or fact tables from an E/R model

The following are the steps to convert an E/R model into a dimensional model:

1. Identify the business process from the E/R Model.
2. Identify many-to-many tables in E/R model to convert to fact tables.
3. Denormalize remaining tables into flat dimension tables.
4. Identify date and time from E/R model.

Because of the huge E/R model of the Redbook Vineyard company data warehouse, we do not cover all the steps in detail. For more step-by-step details about how to determine fact tables and facts in the E/R model, refer to 6.1, “Converting an E/R model to a dimensional model” on page 210.

7.6.2 Conformed facts

For the Sales Invoicing case study, we do not have any conformed facts.

7.6.3 Identify fact types (additivity and derived types)

In this activity we categorize facts based on the way they can be added. This categorization is especially useful in order to avoid mistakes when adding up columns of data that cannot be absolutely or partially summed.

The important factors we consider when aggregating data by column are: additivity, semi-additivity, and non-additivity. For a detailed definition of fact types, refer to 5.5.1, “Facts” on page 171.

In Table 7-32 we describe all the facts by:

- Categorizing them as additive, non-additive, or semi-additive. Handling semi-additive and non-additive facts is considered to be an advanced concept. For more on handling such facts, refer to the following:
  - “Non-additive facts” on page 297
  - “Semi-additive facts” on page 299
- Identifying them as derived, when they can be calculated from other facts that exist in the table, or that have been also derived.

Table 7-32 Facts identified for Sales fact table

S#	Fact	Description	Type of fact	Derived
1	Accrued Bonus Amount	An amount of money in the corporate currency indicating the bonus the customer gets for purchasing a product. This fact is true to the line item grain. [Accrued Bonus Amount = Quantity sold x Accrued Bonus per Item]	Additive	No [It would have been a derived fact if Accrued Bonus per Item had also been stored in the fact table].
2	Alcohol Duty Amount	The amount in corporate currency to be paid for duty if the product is an alcoholic drink. This fact is true to the grain. [Alcohol Duty Amount= Quantity sold x Alcohol Duty per Item]	Additive	No [It would have been a derived fact if [Alcohol Duty per Item] had also been stored in the fact table].
3	Alcohol Volume Percent	Alcohol by volume (ABV) is an indication of how much alcohol (expressed as a percentage) is included in an alcoholic beverage. This measurement is assumed as a world standard, although in the United States, the predominant measurement is Alcohol by weight (also known as ABW). Another way of specifying the amount of alcohol is alcoholic proof.	Non-additive	No

S#	Fact	Description	Type of fact	Derived
4	Cost Of Goods	Also known as cost of product in corporate currency excluding the taxes. [Cost of Goods = Quantity sold x Cost of good per item]	Additive	No [It would have been a derived fact if [Cost of good per item] had also been stored in the fact table].
5	Delivered Liters	Amount of liters shipped for a product. [Delivered Liters = Quantity sold x Delivered Liters per Item]	Additive	No [It would have been a derived fact if delivered liters per item had also been stored in the fact table].
6	Quantity Sold	The quantity of product in units that has been already sold. **Note:** This was a preliminary fact we identified in 7.4.5, “Identify high level dimensions and facts” on page 378.	Additive	No
7	Discount Amount	The amount of money in corporate currency the customer gets discounted on the product. **Note:** This was a preliminary fact we identified in 7.4.5, “Identify high level dimensions and facts” on page 378. [Discount Amount = Quantity sold x Discount Amount per item]	Additive	No. It would have been a derived fact if [Discount Amount per item per item] had also been stored in the fact table.

S#	Fact	Description	Type of fact	Derived
8	Gross Sales Amount	The amount of money in corporate currency already invoiced. **Note:** We identified *Price per item* as preliminary fact in 7.4.5, “Identify high level dimensions and facts” on page 378. Instead of storing ‘Price per item’ as a fact, we store the Gross sales cost for the line item as Quantity sold x Price per item. [Gross Sales Amount = Quantity sold x Price per item]	Additive	No [It would have been a derived fact if Price per item had also been stored in the fact table].
9	Lc Bonus Amount	The amount of money in the ordered local currency indicating the bonus the customer gets for purchasing the product. [Local Bonus Amount = Quantity sold x Bonus Amount per item]	Semi-additive (You cannot add different currencies.)	No [It would have been a derived fact if Bonus Amount per item had also been stored in the fact table].
10	Lc Cost Of Goods	The net amount of money in the ordered local currency the product costs. [Cost of Goods = Quantity sold x Cost of good per item]	Semi-additive (You cannot add different currencies)	No [It would have been a derived fact if Cost of good per item had also been stored in the fact table].
11	Lc Discount Amount	The amount of money in the ordered currency the customer gets as a discount on the product. [Discount Amount = Quantity sold x Discount Amount per item]	Semi-additive (You cannot add different currencies)	No [It would have been a derived fact if Discount amount per item had also been stored in the fact table].

S#	Fact	Description	Type of fact	Derived
12	Lc Duty Amount	The amount in ordered local currency to be paid for duty if the product is an alcoholic drink. [Alcohol Duty Amount = Quantity sold x Alcohol Duty per Item]	Semi-additive (You cannot add different currencies)	No [It would have been a derived fact if Alcohol duty per item had also been stored in the fact table].
13	Lc Gross Sales Amount	The amount of money in ordered local currency already invoiced for this product. Gross Sales Amount = Quantity sold x Price per item.	Semi-additive (You cannot add different currencies)	No [It would have been a derived fact if Price per item had also been stored in the fact table].
14	Lc Order Handling Cost	The amount of money in the order currency that the order handling of the product costs. This information is not kept at the order level because certain kinds of liquors are very expensive luxury articles that need to be treated very carefully and be specially packaged. [Order Handling Cost = Quantity sold x Order Handling Cost per Item]	Semi-additive (You cannot add different currencies)	No [It would have been a derived fact if Order handling cost per item had also been stored in the fact table].
15	Order Handling Cost	The amount of money in the corporate currency that the order handling of the product costs. This information is not kept at the order level because certain kinds of liquors are very expensive luxury articles that need to be treated very carefully and be specially packaged. [Order Handling Cost = Quantity sold x Order Handling Cost per Item] (in Corporate currency)	Additive	No It would have been a derived fact if Order handling cost per item had also been stored in the fact table.

S#	Fact	Description	Type of fact	Derived
16	Spirit Liters	Amount of alcohol in liters per item = Delivered Liters * (Alcohol Volume Pct / 100)	Additive	Yes (See description for formula)
17	Gross profit	Profit calculated as gross sales income less the cost of sales.	Additive	Yes (See description for formula)
18	Gross margin Percent	Percentage difference between the cost of sales and the gross sales.	Non-additive	Yes (See description for formula)
19	Net sales Amount	The amount of the sale. This is a negative amount if it is a credit note due to returned or damaged material. Net sales = Gross Sales minus Amount returned and damaged items. 7.3.6, “Gather the requirements” on page 362. When the invoice is created for a customer, the <Amount returned and damaged items> is 0. However, if the customer returns goods which are damaged, a new invoice form is created and sent to the customer with <Amount returned and damaged items> showing a negative Amount.	Additive	Yes (See description for formula)
20	Net profit Amount	Net Sales - Cost of Sales - Alcohol Duty Amount (in Corporate currency)	Additive	Yes (See description for formula)

Note: A fact is said to be derived if the fact can be calculated from other facts that exist in the table, or that have also been derived. You may decide not to include the derived facts inside the fact table and to calculate them inside the front-end reporting application.

7.6.4 Year-to-date facts

We have identified two year-to-date facts: Gross Sales YTD and Gross Margin YTD. These YTD facts are shown in Table 7-33.

Table 7-33 Year-to-Date Facts

S.No	Name of Year-to Date Fact	Description
1	Gross Sales YTD	Gross Sales during the present year
2	Gross Margin YTD	Gross Margin Average during the present year

However these year to date facts are not true to this grain. Our two year-to-date facts will not be physically implemented and will be handled out of the dimensional model by means of **one** of the following of:

- An OLAP application
- RDMS user-defined functions
- An RDMS view
- A user SQL query

For more information about Year-to-Date facts, refer to 5.5.4, “Year-to-date facts” on page 176.

7.6.5 Event facts, composite keys, and growth

For our Sales Invoicing case study, we do not have any event-based fact tables. For more information about event-based fact tables, refer to 6.4.4, “Handling event-based fact tables” on page 311.

The composite key of the sales fact table consists of the following list of keys, all of which are surrogate keys.

- Currency
- Customer
- Invoice
- Payment Terms
- Product
- Sales Date
- Sales Representative
- Warehouse

In this activity, we estimate the fact table size and also predict its growth. It is important to understand that approximately 85-95% of the space of a dimensional model is occupied by the fact tables. Therefore, it is extremely important to understand the future growth pattern to plan for future fact table

performance. For more information about how to predict fact table growth, refer to 5.5.7, “Fact table sizing and growth” on page 179.

7.6.6 Phase Summary

We now have the definitive version of the logical dimensional model. The final dimensional model is shown in Figure 7-30.

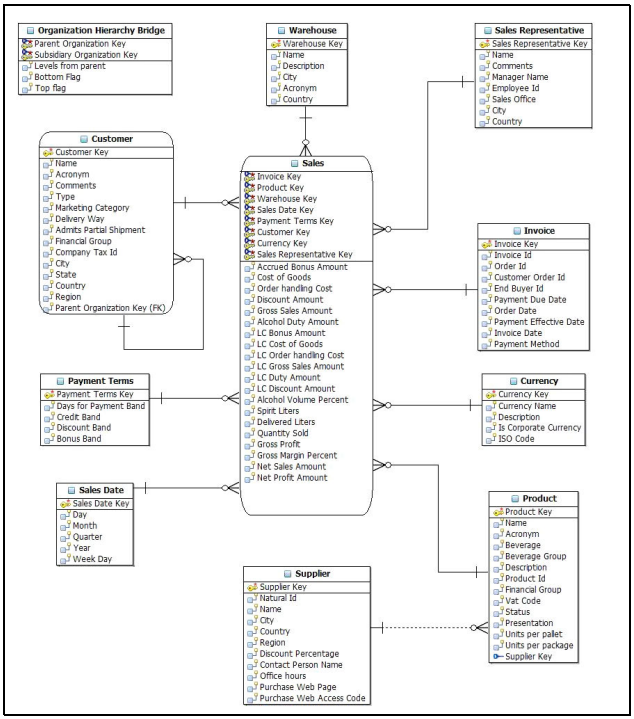


Figure 7-30 Final model for Sales Invoicing business process

7.7 Other phases

We developed a dimensional model for the Sales Invoicing business process, in this case study. We discussed the following phases of the DMDL:

- Identify the business process
- Identify the grain
- Identify the dimensions
- Identify the facts

We now briefly describe how to apply the other phases of the DMDL for the Sales Invoicing case study. The other phases of the DMDL we consider are:

- **Verify the model:** Here we verify the dimensional model (Figure 7-30 on page 421) we made for our sales invoice process. We verify whether the dimensional model designed is able to answer the business requirements identified during the requirements gathering phase in “Gather the requirements” on page 362. For more information about the ‘Verify’ phase, refer to “Verify the model” on page 181.
- **Physical Design Considerations:** In this phase we do the following:
  - Identify Aggregates: In simple terms, aggregation is the process of calculating summary data from detail base level fact table records. Aggregates are the most powerful tool for increasing query processing performance in dimensional data marts. The main goal of this phase is to identify the aggregates that will help improve performance. For more information about creating aggregates, refer to “Aggregations” on page 184.
  - Aggregate Navigation: This is a technique that involves redirecting user SQL queries to appropriate precomputed aggregates. In this phase we design intelligent aggregate navigation strategies. For more information about this advanced topic, refer to “Aggregate navigation” on page 188.
  - Indexing: In this phase we create indexes for the dimensions and fact table. For more information about creating indexes, refer to:
    - “Indexing” on page 190 provides guidelines about creating indexes.
    - “Indexing for dimension and fact tables” on page 324 (A sample star schema to show indexing).
  - Partitions: Partitioning a table divides the table by row, by column, or both. If a table is divided by column, it is said to be vertically partitioned. If a table is divided by rows, it is said to be horizontally partitioned. Partitioning large fact tables improves performance because each partition is more manageable and smaller to enable better performance. In this phase we create partitions. For more guidelines about creating partitions, refer to “Partitioning” on page 195.
- **Meta Data management:** In this phase we manage meta data for the following phases of the DMDL:
  - Identify business process: The output of this phase results in the Requirements gathering report. The report primarily consists of the business requirements for the selected business for which you will design the dimensional model. In addition, the report also consists of the business processes, owners, source systems involved, data quality issues, common terms used across business processes, and other business-related meta data.

- Identify the grain: The output of this phase results in the Grain definition report (See DMDL Figure 7-28 on page 410). The Grain definition report consists of one or multiple definitions of the grain for the business process for which the dimensional model is being designed. Also, the type of fact table (transaction, periodic, or accumulating) being used is mentioned. The Grain definition report also includes high level preliminary dimensions and facts.
- Identify the dimensions: The meta data for this phase includes the dimension name, its hierarchies, update rules, load frequency, load statistics, usage statistics, archive rules, archive statistics, purge rules, purge statistics, attributes, and so on. For more detail, refer to “Meta data management” on page 196.
- Identify the facts: The meta data for this phase includes detailed information about the facts. For more detail, refer to “Meta data management” on page 196.
- User Verification phase: This phase includes meta data gathered after the user verification of the dimensional model.
- Physical design consideration: The meta data for this phase includes aggregate tables involved, aggregate navigation strategy, index names, index descriptions, and partitions involved.

- **Design the next priority business process data mart:** In this activity, we designed the dimensional model for the Sales Invoicing business process of the Redbook Vineyard company. However, the Redbook Vineyard company has several business processes that we identified and prioritized in the “Identify business process” on page 355. The various business processes are shown in Table 7-34. We identified the Sales Invoicing as the process with the highest priority for which the dimensional model should be designed.

  The next step is to select the next high priority business process and use the DMDL to design another mart. Most likely, management will agree on either Inventory or Sales Orders. This is because these have the highest points as shown in Table 7-34. For more information on prioritizing business processes, refer to 7.3.2, “Identify business process” on page 355.

Table 7-34 Business process prioritization

Name of Business Process	Complexity	Data quality and Availability	System availability	Strategic business significance	Final Points
Distribution	6	6	1	2	15
Sales CRM/ Marketing	4	6	2	4	16

Name of Business Process	Complexity	Data quality and Availability	System availability	Strategic business significance	Final Points
Inventory	4	9	3	4	20
Procurement	4	6	3	2	15
Sales Orders	4	9	3	4	20
Accounting	4	9	2	2	17
Production	2	3	3	6	14
Sales Invoicing	**6**	**9**	**3**	**4**	**22**

7.8 Conclusion

In this chapter, we designed a dimensional model for the Sales Invoicing process. The objective was to put to test the DMDL developed in Chapter 5, “Dimensional Model Design Life Cycle” on page 103.

We exercised the DMDL by executing the following phases, and completing a dimensional model of the sales invoice process:

- Identify the business process
- Identify the grain
- Identify the dimensions
- Identify the facts
- Verify
- Physical design considerations
- Meta data management

The important point to understand is that each phase of the DMDL consists of several activities. However, depending upon the business situation and the design need, you may use all, or only some of the activities inside each of the phases shown in the DMDL.

8

Case Study: Analyzing a dimensional model

In this chapter we review an existing design of a dimensional model and provide guidelines to improve the model. So, the flow of this chapter is different because we typically describe how to design a new model from the beginning.

In short, in this chapter the discussions focus on the following topics:

- Study the business of a fictitious company.
- Understand the business needs for creating a dimensional model.
- Review the draft dimensional model.
- Discuss guidelines for reviewing the existing dimensional model.
- Make improvements to the draft dimensional model, and discuss the reasons for doing so.

8.1 Case Study - Sherpa and Sid Corporation

In this section, we briefly describe the fictitious Sherpa and Sid Corporation and its business. In doing so, we describe the reason that the corporation has submitted a request to build a data mart.

8.1.1 About the company

Sherpa and Sid Corporation started as a manufacturer of cellular telephones, and then quickly expanded to include a broad range of telecommunication products. As the demand for, and size of, its suite of products grew, Sherpa and Sid Corporation closed down the existing distribution channels and opened its own sales outlets.

In the past year Sherpa and Sid Corporation opened new plants, sales offices, and stores in response to increasing customer demand. With its focus firmly on expansion, the corporation put little effort into measuring the effectiveness of the expansion. Sherpa and Sid's growth has started to level off, and management is refocusing on the performance of the organization. However, although cost and revenue figures are available for the company as a whole, little data is available at the manufacturing plant or sales outlet level regarding cost, revenue, and the relationship between them.

To rectify this situation, management has requested a series of reports from the Information Technology (IT) department. IT responded with a proposal to implement a data mart. After consideration of the potential costs and benefits, management agreed.

8.1.2 Project definition

Senior management and IT put together a project definition consisting of the following objective and scope:

Project objective

To create a data mart to facilitate the analysis of cost and revenue data for products manufactured and sold by Sherpa and Sid Corporation.

Project scope

The project will be limited to direct costs and revenues associated with products. Currently, Sherpa and Sid Corporation manufacturing costs cannot be allocated at the product level. Therefore, only component costs can be included. At a future time, rules for allocation of manufacturing and overhead costs may be created, so the data mart should be flexible enough to accommodate future changes. IT

created a team consisting of one data analyst, one process analyst, the manufacturing plant manager, and one sales region manager for the project.

8.2 Business needs review

In this section, we review the requirements gathering meta data. In other words, we review what the project team defined, and felt they needed to investigate, in order to understand the business need for the Sherpa and Sid Corporation. The understanding of the business will greatly help in the review of the data mart.

The team identified the following areas of interest:

- Life Cycle of a product
- Anatomy of a sale
- Structure of the organization
- Defining cost and revenue
- What do the users want?

8.2.1 Life cycle of a product

The project team first studied the life cycle of a product. Each manufacturing plant has a research group that tests new product ideas. Only after the manufacturing process has been completely defined and approval for the new product has been obtained is the product information added to the company's records. Once the product information is complete, all manufacturing plants can produce it.

A product has a base set of common components. Additional components can be added to the base set to create specific models of the product. Currently, Sherpa and Sid Corporation has 300 models of their products. This number is fairly constant as the rate of new models being created approximately equals the rate of old models being discontinued. Approximately 10 models per week experience a cost or price change. For each model of each product, a decision is made about whether or not it is eligible for discounting. When a model is deemed eligible for discounting, the salesperson may discount the price if the customer buys a large quantity of the model or a combination of models. In a retail store, the store manager must approve such a discount.

The plant keeps an inventory of product models. When the quantity on hand for a model falls below a predetermined level, a work order is created to cause more of the model to be manufactured. Once a model is manufactured, it is stored at the manufacturing plant until it is requested by a sales outlet.

The sales outlet is responsible for selling the model. When a decision is made to stop making a model, data about the model is kept on file for six months after the last unit of the model has been sold or discarded. Data about a product is removed at the same time as data about the last model for the product is removed.

8.2.2 Anatomy of a sale

There are two types of sales outlets: corporate sales office and retail store. A corporate sales office sells only to corporate customers. Corporate customers are charged the suggested wholesale price for a model unless a discount is negotiated. One of Sherpa and Sid Corporation's 30 sales representatives is assigned to each corporate customer. Sherpa and Sid Corporation currently serves 3000 corporate customers. A customer can place orders through a representative or by phoning an order desk at a corporate sales office. Orders placed through a corporate sales office are shipped directly from the plant to the customer. A customer can have many shipping locations. So, it is possible for a customer to place orders from multiple sales offices if the policy is to let each location do their own ordering.

The corporate sales office places the order with the plant closest to the customer shipping location. If a customer places an order for multiple locations, the corporate sales office splits it into an individual order for each location. A corporate sales office, on average, creates 500 orders per day, five days per week. Each order consists of an average of 10 product models.

A retail store sells over the counter. Unless a discount is negotiated, the suggested retail price is charged. Although each product sale is recorded on an order, the company does not keep records of customer information for retail sales. A store can only order from one manufacturing plant. The store manager is responsible for deciding which products to stock and sell from their particular store. A retail store, on average, creates 1000 orders per day, seven days per week. Each order consists of an average of two product models.

8.2.3 Structure of the organization

It was clear to the team that understanding products and sales was not enough; an understanding of the organization was also necessary. The regional sales manager provided an up-to-date copy of the organization structure, which is depicted in Figure 8-1 on page 429.

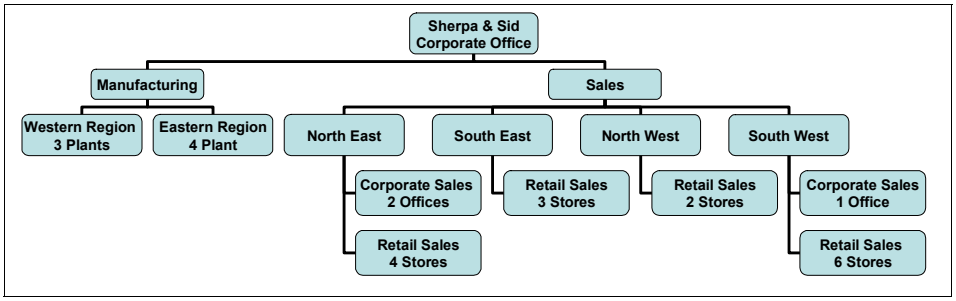


Figure 8-1 Sherpa and Sid Corporation organizational structure

8.2.4 Defining cost and revenue

The project team studied the cost and revenue components from a business standpoint. Their goal was to be able to define cost and revenue, in order to be able to effectively analyze those factors.

For each product model, the cost of each component is multiplied by the number of components used to manufacture the model. The sum of results for all components that make up the model is the cost of that model.

For each product model, the negotiated unit selling price is multiplied by the quantity sold. The sum of results for all order lines that sell the model is the revenue for that model.

When trying to relate the cost of a model to its revenue, the team discovered that once a model was manufactured and added to the quantity on hand in inventory, the cost of that unit of the model could not be definitively identified. Even though the cost of a component is kept, it is only used to calculate a current value of the inventory. Actual cost is recorded only in the company's financial system, with no reference to the quantity manufactured.

The results of this determination were two-fold. First, the team requested that the operational systems be changed to start recording the actual cost of a manufactured model. However, both management and the project team recognized that this was a significant change, and that waiting for it would severely impact the progress of the project. Therefore, and based on the fact that component costs changed infrequently and by small amounts, the team defined this rule:

> The revenue from the sale of a model is always recorded with the current unit cost of the model, regardless of the cost of the model at the time it was manufactured.

8.2.5 What do the users want?

Because the objective of the project was to create a collection of data that users could effectively analyze, the project team decided to identify a set of typical questions users wanted the data to answer. Clearly, this would not be an exhaustive list. The answer to one question would certainly determine what the next question, if any, might be. As well, one purpose of the data mart is to allow the asking of as yet unknown questions. If users simply want to answer a rigid set of questions, creating a set of reports would likely fill the need. With this in mind, the team defined a set of questions, and they are shown in Table 8-1.

Table 8-1 Business questions

S.no	Business question
1	What are the total cost and revenue for each model sold today, summarized by outlet, outlet type, region, and corporate sales levels?
2	What are the total cost and revenue for each model sold today, summarized by manufacturing plant and region?
3	What percentage of models are eligible for discounting, and of those, what percentage are actually discounted when sold, by store, for all sales this week? This month?
4	For each model sold this month, what is the percentage sold retail, the percentage sold corporately through an order desk, and the percentage sold corporately by a salesperson?
5	Which models and products have not sold in the last week? The last month?
6	What are the top five models sold last month by total revenue? By quantity sold? By total cost?
7	Which sales outlets had no sales recorded last month for the models in the top five models list?
8	Which salespersons had no sales recorded last month for the models in the top five models list?

As well as being able to answer the above questions, the users want to be able to review up to three complete years of data to analyze how the answers to these questions change over time.

8.2.6 Draft dimensional model

Based on the business functions and the requirement gathering, the team came to the following conclusion:

- The Sherpa and Sid corporation is tracking the sales of its products (made in different manufacturing plants) to different customers.
- The Sherpa and Sid corporation is basically comprised of two broad operations:
  - Manufacturing products in its manufacturing plants
  - Sales of these products by its sales outlets to customers
- The customers of Sherpa and Sid corporation are either big corporate companies or retailers who buy directly over the counter.
- Each customer purchases one or more products through an order.
- There are two types of seller outlets:
  - Corporate sales office
  - Retail stores
- The products can be bought in the following two ways:
  - In the case of retail (non-corporate) customers, the products are purchased over the counter from retail outlets.
  - In the case of corporate customers, orders can be placed over the phone and goods are delivered directly from plant to the particular corporate office.

Based on the above conclusions, the data mart team chose the grain for the dimensional model to be *a single line item on an order placed*. The draft dimensional model that the team built is shown in Figure 8-2 on page 432.

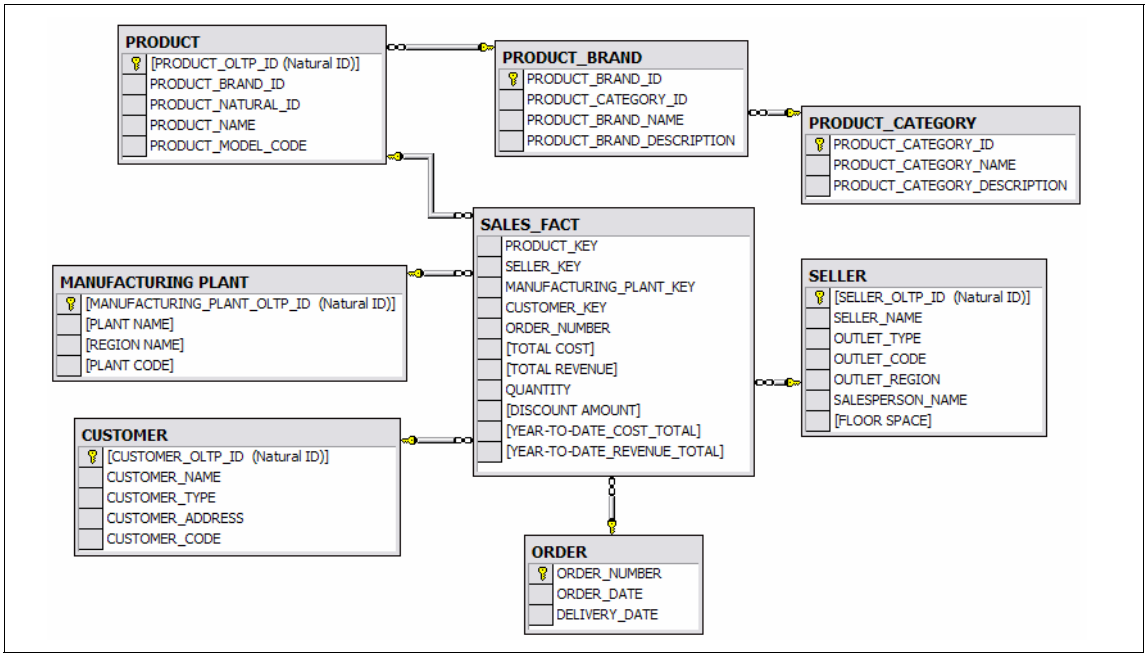


Figure 8-2 Draft dimensional model before review

8.3 Dimensional model review guidelines

In this phase we review the design of the dimensional model shown in Figure 8-2. We perform this review by using the concepts presented in the Dimensional Model Design Life Cycle. See Chapter 5, “Dimensional Model Design Life Cycle” on page 103. The life cycle is divided into different phases. Each phase explains different concepts that help you in the design of the dimensional model.

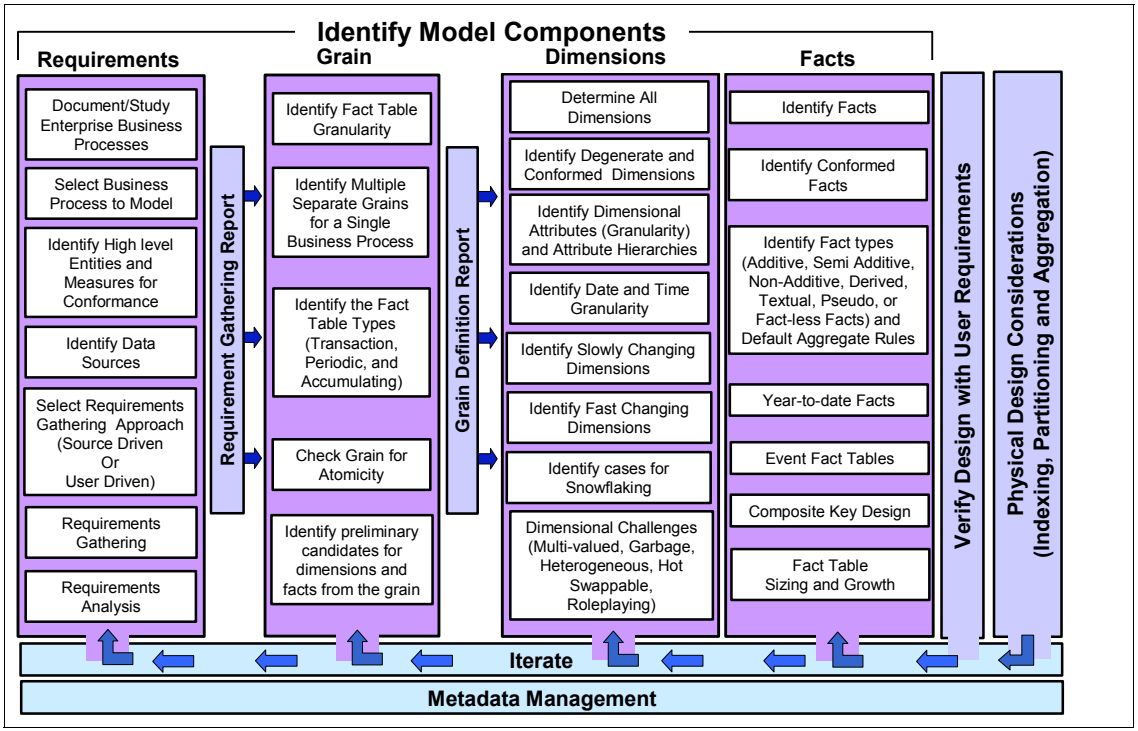


Figure 8-3 Dimensional Model Design Life Cycle

In this case study, instead of designing a new model, we review an existing one. Below we provide general guidelines to consider while reviewing the design of a dimensional model.

8.3.1 What is the grain?

Grain specifies what level of detailed information your star schema will have. The grain is the most important part of the dimensional model. It is the grain definition that sets the stage for development of the entire model. An incorrect grain may lead to the redesign of the entire model. It is very important that enough time be spent on analyzing the grain definition.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, the grain definition is right and appropriate considering the need of the business to track the sales of the different product models on a daily basis (see the question in Table 8-1 on page 430).

8.3.2 Are there multiple granularities involved?

There will be times when more than one grain definition will be associated with a single business process. In these scenarios, we recommend you design separate fact tables with separate grains and not forcefully try to put facts that belong to separate grains in a single fact table.

You can handle differing data granularities by using multiple fact tables (daily, monthly, and yearly tables). Also consider the amounts of data, space, and the performance requirements to decide how to handle different granularities. The factors that can help you to decide whether to design one or more fact tables are discussed in more detail in 5.3.2, “Multiple, separate grains” on page 125.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, there are no separate fact granularities that are involved so there is only one fact table needed. The business requirements (questions 1 to 8) shown in Table 8-1 on page 430 show that only sales of products needs to be tracked for different products. Had the business required knowing the inventory of all sold products in different outlets, we would have had to design a separate fact table to handle inventory in each outlet on a daily basis.

8.3.3 Check grain atomicity

Ideally we would like to design the dimensional model at the most atomic level. The dimensional model has been designed at the most atomic level if it cannot be further divided into detailed information. It is important for the dimensional modeling team not to be shortsighted. Suppose, for example, for now that the data needs to be summarized at the monthly level. The dimensional modeling team may, based on the current business requirements, decide to design the dimensional model with a monthly grain. But what if in future the business asks to see the data at the weekly level? The sooner you ask these types of questions, the better. In other words, try to foresee the business needs. In this case, if it seems likely that they may need to see weekly data in the future, then design the model at the most detailed level. This way you not only give the business what they need (monthly data) right now, but are also well positioned to proactively help them in future with daily data.

For more discussion on the importance of having a detailed atomic grain, you can refer to the following:

- “Check grain atomicity” on page 128.
- “Importance of detailed atomic grain” on page 228

Review results for draft model: For the dimensional model shown in Figure 8-2 on page 432, the grain is a single line item on an order. This is the most detailed atomic definition, so we do not need to go further.

8.3.4 Review granularity for date and time dimension

All dimensional models will surely consist of a date dimension. Some may have a time dimension too. Date and time dimensions play a very important role in identifying the level of detail of information that is going to be available in your model. Guidelines for handling date and time in your dimensional models are as follows:

- Date should be handled separately as its own dimension. We discussed the importance of handling date separately as a dimension in “Date and time granularity” on page 155.
- Date and time dimensions should not be merged together.
- If there are multiple dates involved in the dimensional model, then instead of creating separate physical tables for these dates, use the concept of role-playing to implement several dates as views from one main date table. The concept of role-playing is discussed in “Role-playing dimensions” on page 285.
- Time can be handled in two ways:
  - As a fact inside the fact table
  - As its own dimension

  We discussed this in more detail in 6.3.2, “Handling time as a dimension or a fact” on page 245.
- We may handle date and time across international time zones by storing both the local date/time and the international GMT date/time. This is discussed in “Handling date and time across international time zones” on page 248.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, there are two date dimensions involved. Currently, the order date and delivery date are both incorrectly stored in an order dimension table. The order date and delivery date should be separated into individual order date and delivery date dimensions.

8.3.5 Are there degenerate dimensions?

A degenerate dimension is a dimension without attributes. Degenerate dimensions are typically numbers such as a booking number, order number, confirmation number, receipt number, and ticket number.

How to identify and handle a degenerate dimension

A degenerate dimension exists in the dimensional design if any of the following is true:

- The dimensional design consists of a dimension table which has an equal number of rows as the fact table. In other words, if while loading your fact table with X rows, you need to pre-insert the same number of X rows in any dimension table, then this is an indication that there is a degenerate dimension table in your design.
- If any of your dimension tables have a nearly equal (but not equal) number of rows as compared to the fact table, then in this case there is a possibility of a degenerate dimension.

The concept of identifying and handling a degenerate dimension is discussed in more detail in the following sections:

  - "Degenerate dimensions" on page 142.
  - "Degenerate dimensions" on page 240.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, the order table is actually a degenerate dimension. The order dimension table is storing the order number, order date, and delivery date. We discussed in 8.3.4, "Review granularity for date and time dimension" on page 435 that order and delivery dates belong as separate dimensions and not in the order table. In addition to this, all order-related information, such as products ordered and from which outlet, is stored in separate dimensions such as product, manufacturing plant, and seller.

The order dimension therefore is degenerate and the only information remaining is the Order_Number, which can be stored inside the sales_fact table.

8.3.6 Surrogate keys

Primary keys of dimension tables should remain stable. We strongly recommend that you create surrogate keys and use them for primary keys for all dimension tables. In this section we discuss surrogate keys, and why it is important to use the surrogate keys as the dimension table primary keys.

What are Surrogate Keys?

Surrogate keys are keys that are maintained within the data warehouse instead of the natural keys taken from source data systems. Surrogate keys are known by many other aliases, such as dummy keys, non-natural keys, artificial keys, meaningless keys, non-intelligent keys, integer keys, number keys, technical integer keys, and so on. The surrogate keys basically serve to join the dimension tables to the fact table. Surrogate keys serve as an important means of identifying each instance or entity inside a dimension table.

For a detailed discussion about surrogate keys and their importance, refer to "Reasons for using surrogate keys are:" on page 139.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observed that all the dimension tables are using the natural OLTP id instead of using a surrogate key. It is important that all dimension primary keys strictly be surrogate keys only.

8.3.7 Conformed dimensions and facts

A conformed dimension means the same thing to each fact table to which it can be joined. A more precise definition is that two dimensions are conformed if they share one, more than one, or all attributes that are drawn from the same domain. In other words, a dimension may be conformed even if it contains only a subset of attributes from the primary dimension.

Typically, dimension tables that are referenced, or are likely to be referenced by multiple fact tables (multiple dimensional models) are called conformed dimensions.

Identify whether conformed dimensions or fact exist

If conformed dimensions already exist for any of the dimensions in your data warehouse or dimensional model, you will be expected to use the conformed dimension versions. If you are developing new dimensions with potential for usage across the entire enterprise warehouse, you will be expected to develop a design that supports anticipated enterprise data warehouse needs. In order to determine the anticipated warehouse needs, you might need to interact with several business processes to find out how they would define the dimensions. The same is true for conformed facts.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observed that this is the first effort toward building a data warehouse for the Sherpa and Sid Corporation. Therefore, since this is the first dimensional model of its kind, there are no previous shared or conformed dimensions or facts we can use for this data mart.

8.3.8 Dimension granularity and quality

Dimension granularity specifies the level of detail stored in a dimension. The level of detail of all dimensions helps to determine the overall level of information that is available in the dimensional model. Guidelines for reviewing dimension granularity are:

- Every dimension should be reviewed to see if all attributes are true to the grain.
- Every dimensional attribute should take only one value for a single row inside the fact table. If a dimensional attribute has two or more values for a single fact table row, then you must treat this dimension as a multi-valued

dimension. The concept of handling multi-valued dimensions is explained in more detail in the following sections:

- "Other dimensional challenges" on page 166
- "Multi-valued dimensions" on page 288

Important points to consider when choosing attributes for dimensions are explained as follows:

- Non-key columns are generally referred to as attributes. Every dimension table primary key should be a surrogate key which is not considered an attribute. Because the business users will not use this surrogate key for analysis (because it is simply an integer with no associated information).
- Use a separate field in the dimension table to preserve the natural source system key of the entity that is used in the source system.
- A schema design that contains complete, consistent, and accurate attribute fields helps users write queries that they intuitively understand and reduces the support burden on the organization (usually the IT department) responsible for database management and reports.
- A well-designed schema includes attributes that reflect the potential areas of interest and attributes that you can use for aggregations as well as for selective constraints and report breaks.
- If a dimension table includes a code, in most cases, include the code description as well. As an example, if branch locations are identified by a branch code, and each code represents a branch name, include both the code and the name. Avoid storing cryptic decodes inside a dimensional table attribute to save space.
- Make sure the attribute names are unique within your model. If you have duplicate names for different attributes, use the prime term (entity name) to create a distinction. For example, if you have multiple attributes called *Address Type Code*, one might be renamed *Beneficiary Address Type Code*, and another might be *Premium Address Type Code*.
- The dimension attributes serve as report labels and should be descriptive and easy to understand. For example, in a given situation where you want to store a flag, such as 0/1 or Y/N, it is better to store something descriptive such as Yes/No instead.
- An attribute can be defined to permit missing values in cases where an attribute does not apply to a specific item, or its value is unknown.
- An attribute may belong to more than one hierarchy.
- Use only the alphabetic characters A-Z and the space character. Do not use punctuation marks or special characters, including the slash (/) or the hyphen (-).

- While naming your attributes, do not use possessive nouns, for example, use “Recipient Birth Date" rather than "Recipient's Birth Date."
- Do not reflect permitted values in the attribute name. For example, the attribute name "Employee Day/Night Code" refers to code values designating day shift or night shift employees. Name the attribute to reflect the logical purpose and the entire range of values. For example, "Employee Shift Type Code," which allows for an expandable set of valid values.
- Do not include very large names for building your attributes.
- Properly document all dimensional attributes.

Note: It is important to remember that 85-90% of space in large dimensional models is occupied by the fact table. Therefore, saving space by using codes in dimensional attributes does not save much overall space, but these codes do affect the overall quality and understandability of the dimensional model.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, the following improvements need to be made:

- The Customer dimension stores address in a single column. In this way we are unable to determine the detail about addresses such as city, district, state, and region. We will redesign the address column to include more address details, such as street name, suite, city, district, state, and region.
- The draft dimensional model shown in Figure 8-2 on page 432 consists of codes in mostly every dimension. We need to include the detailed description for these codes also.
- All the dimension attributes have a single value for every fact table row. In other words, we do not need to use multi-value dimensions.

8.3.9 Dimension hierarchies

A hierarchy is a cascaded series of many-to-one relationships. A hierarchy basically consists of different levels. Each level in a hierarchy corresponds to a dimension attribute.

In other words, a hierarchy is a specification of levels that represents relationships between different attributes within a hierarchy. For example, one possible hierarchy in the date dimension is Year → Quarter → Month → Day.

There are three major types of hierarchies that you should look for in each of the dimensions. They are explained in Table 8-2 on page 440.

Table 8-2 Different types of hierarchies

S.No	Name	Hierarchy description	How implemented
1	Balanced	A balanced hierarchy is a hierarchy in which all the dimension branches have the same number of levels. For more details, see “Balanced hierarchy” on page 249.	See “How to implement a balanced hierarchy” on page 250.
2	Unbalanced	A hierarchy is unbalanced if it has dimension branches containing varying numbers of levels. Parent-child dimensions support unbalanced hierarchies. For more details, see “Unbalanced hierarchy” on page 251.	See “How to implement an unbalanced hierarchy” on page 252.
3	Ragged	A ragged dimension contains at least one member whose parent belongs to a hierarchy that is more than one level above the child. Ragged dimensions, therefore, contain branches with varying depths. For more details, see “Ragged hierarchy” on page 260.	See “How to implement a ragged hierarchy in dimensions” on page 261.

Note: A dimension table may consist of multiple hierarchies. Also, a dimension table may consist of attributes or columns which belong to one, more, or no hierarchies.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observe that the product hierarchy (Product Category → Product Brand → Product Name) has been incorrectly snowflaked into separate tables. It is typically bad from a performance standpoint to snowflake hierarchies. The product, brand, and category tables should be merged into one product table.

8.3.10 Cases for snowflaking

Further normalization and expansion of the dimension tables in a star schema result in the implementation of a snowflake design. In other words, a dimension table is said to be snowflaked when the low-cardinality attributes in the

dimension have been removed to separate normalized tables and these normalized tables are then joined back into the original dimension table.

Typically, we do not recommend snowflaking in the dimensional model environment because it has drastic affects on the understandability of the dimensional model and can result in decreased performance because of the fact that more tables need to be joined to get the results.

We need to identify which dimensions need to be snowflaked for each of our dimensions that we have chosen for our dimensional model. There are no appropriate candidate snowflakes identified for the grocery store example.

The following topics on snowflaking are discussed in more detail in 6.3.7, "Identifying dimensions that need to be snowflaked" on page 277:

- What is snowflaking?
- When do you do snowflaking?
- When to avoid snowflaking?
- Under what scenarios does snowflaking improve performance?
- Disadvantages of snowflaking

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observe that the product hierarchy (Product Category → Product Brand → Product Name) has been incorrectly snowflaked into separate tables. It is not appropriate for the product hierarchy to be snowflaked. In addition to this, there are no other tables that need to be snowflaked.

8.3.11 Identify slowly changing dimensions

A slowly changing dimension is a dimension whose attribute or attributes for a record (row) change or vary slowly over time.

In a dimensional model, the dimension table attributes are not fixed. They typically change slowly over a period of time, but can also change rapidly. The dimensional modeling design team must involve the business users to help them determine a change handling strategy to capture the changed dimensional attributes. This basically describes what to do when a dimensional attribute changes in the source system. A change handling strategy involves using a surrogate (substitute) key as the primary key for the dimension table.

There are three ways of handling slowly changing dimensions. They are:

- Type-1: Overwrite the value.
- Type-2: Create a new row.
- Type-3: Add a new column and maintain both the present and old values.

For more detailed discussion on slowly changing dimensions, refer to the following:

- “Slowly changing dimensions” on page 159
- “Slowly changing dimensions” on page 261

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observe that product, seller, manufacturing plant, and customer can be treated as slowly changing dimensions and handled using a Type-2 change handling strategy.

8.3.12 Identify fast changing dimensions

Fast changing dimensions cannot be handled using slowly changing dimension handling strategies. That is, fast changing dimensions cannot be handled using the Type-1, Type-2, or Type-3 slowly change handling strategies. The rate of change of these dimensions is much faster when compared to slowly changing dimensions. It is important that we review each dimension to see if it is a fast changing dimension and handle it appropriately. It may be also possible that a fast changing dimension may have been handled using a Type-2 change handling strategy. Handling a fast changing dimension using a Type-2 approach is inappropriate, does not solve the fast changing problem, and can significantly increase the number of rows in the dimension.

Fast changing dimensions are handled by splitting the main dimension into one or more mini-dimensions. For more detailed discussion about handling fast changing dimensions, refer to the following:

- “Fast changing dimensions” on page 162
- “Handling fast changing dimensions” on page 269

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observe that all dimensions can be handled comfortably using the Type-2 change handling strategy.

8.3.13 Year-to-date facts

The period beginning at the start of the calendar year, up to the current date, is called Year-to-Date. For the calendar year where the starting day of the year is January 01, the Year-to-Date definition is a period of time starting from January 01 to a specified date.

In other words, Year-to-Date facts are numeric totals that consist of an aggregated total from the start of year to the current date. For example, assume that a fact table stores sales data for the year 2005. The sales for each month are additive and can be summed to produce year-to-date totals. If you create a

Year-to-Date fact such as Sales_$$_Year_To_Date, then when you query this fact in August 2005, you would get the sum of all sales to August 2005. In other words, the Sales_$$_Year_To_Date fact stores aggregated values.

Other types of aggregated facts can be Month-to-Date facts and Quarter-to-Date facts. Dimensional modelers may include aggregated Year-to-Date facts inside the fact table to improve performance and also reduce complexities in forming Year-to-Date queries. However, storing such Year-to-Date facts inside the fact table may lead this fact to be incorrectly overcalculated if business users count it twice. Such untrue-to-grain facts should be calculated inside the front-end reports, and should not be stored as a fact, to avoid confusion.

Suggested approaches for handling Year-to-Date facts are as follows:

- Year-to-Date facts can be easily handled by OLAP-based applications.
- Year-to-Date facts can also be defined by using SQL functions in Views or Stored Procedures.

Review result for draft model: For the dimensional model shown in Figure 8-2 on page 432, we observe that the fact table contains a year-to-date fact called Year-to-Date_Total at the line item grain. In other words, the fact called Year-to-Date_Total is untrue to the grain and should not be stored in this fact table.

8.4 Schema following the design review

After reviewing the initial draft schema as shown in Figure 8-2 on page 432, we made a number of improvements, resulting in the new design shown in Figure 8-4 on page 444.

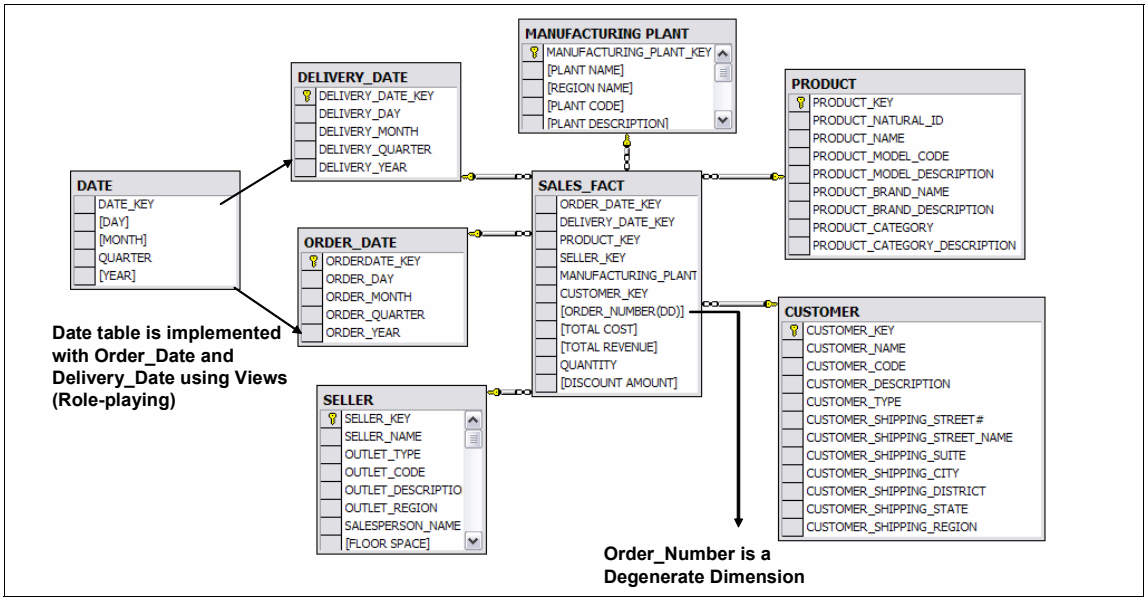


Figure 8-4 Dimensional Model after review

Improvements made to the model include:

- The order dimension table has been dropped and replaced by the Order_Number which is a degenerate dimension. In other words, previously in the draft model shown in Figure 8-2 on page 432, the Order dimension was incorrectly represented as a dimension because it really had no attributes. The attributes, such as order date and delivery date, actually can be well represented as a separate dimension.
- The Order_Date and Delivery_date columns have been removed from the Order table, which has now been made a degenerate dimension. The order date and delivery date have now been represented as separate dimensions using the common date dimension table.This is shown in Figure 8-4. This concept of using a common table to represent different conceptual dimensions is called role-playing. The concept of role-playing is discussed in more detail in “Role-playing dimensions” on page 285.
- The quality of all the dimension tables has been improved by adding descriptions for the following coded columns in Figure 8-2 on page 432:
  - Product_Model_Code
  - Plant_Code
  - Outlet_Code
  - Customer_Code

In addition to the above codes, now the dimensional model shown in Figure 8-4 on page 444 also has descriptive columns that describe these codes. The new descriptive columns added are:

  - Product_Model_Description (To describe Product_Model_Code)
  - Plant_Description (To describe Plant_Code)
  - Outlet_Description (To describe Outlet_Code)
  - Customer_Description (To describe Customer_Code)

- In the draft design, we saw that all dimensions were using OLTP natural keys as primary keys. However, after review, we changed all dimension primary keys to surrogate keys.
- We saw in the old draft design that the product tables hierarchy (Category → Brand → Product name) had been incorrectly snowflaked into two separate tables. We believe that snowflaking hierarchies can be bad for browsing reasons because of the increased number of joins involved when selecting different components of the hierarchy. As shown in the revised design (Figure 8-4 on page 444), we have removed the snowflaked tables and merged the broken hierarchy into a single product table.
- We studied all the dimensions in detail and concluded that all dimensions changing data can be easily implemented using the Type-2 change handling strategy. Also we determined that there were no fast changing dimensions.
- We have removed the two Year-to-Date facts called Year_to_date_cost_total and Year_to_date_revenue_total from the fact table as shown in Figure 8-4 on page 444. This is because these year-to-date facts are untrue to the grain definition which is a single line item on the order.
- In the old draft model, we saw that the customer address is stored inside a single column called Customer_Address. It is difficult to analyze the customers according to their city, district, state, and region if we have only a single address field. This is because all of these details are stored inside the Customer_Address column as a single string. To better analyze the customer address, we split the Customer_Address column in our improved, post-review design (Figure 8-4 on page 444) into more detailed address field columns, such as:

  - Customer_Shipping_Street#
  - Customer_Shipping_Street_Name
  - Customer_Shipping_Suite
  - Customer_Shipping_City
  - Customer_Shipping_District
  - Customer_Shipping_State

– Customer_Shipping_Region

9

Managing the meta data

In this chapter we provide an overview of meta data, specifically for the purpose of implementing a data warehouse or dimensional database.

The most common and simplistic description of meta data is *data about data*. Since this expression was coined, a long time has passed and the data warehouse and business intelligence environments have become bigger and more complex. More complex data warehouses have resulted in more complex data and more problems with managing data. The need to know more about the huge volumes of data stored in organizations has also impacted the definition of meta data. Meta data is now more commonly referred to as *information about data*. In other words, meta data is now considered information that makes the data understandable, usable, and shareable.

Today, companies of all sizes are exposed to quickly evolving business environments that challenge the way computer systems and databases are designed and maintained. One of the most significant parts of this challenge derives from the business intelligence needs of companies today. The volume of data in companies is growing at a rapid rate, much of which is fueled by the Internet. While the Internet, and particularly the Web, grow and reach more and more locations, and computer systems become more and more complex, generating and relying on more and more data, we have to deal with the challenge of efficiently managing and taking advantage of volumes of data, and converting them into useful information. That is where the Business Intelligence and, more specifically, meta data management play their critical role.

9.1 What is meta data?

In order to explain the meaning and usage of meta data, think of an example that refers to a typical IT (Information Technology) environment in a mid-size company where Business Intelligence has not yet been fully exploited.

This company has recently purchased a smaller company in the same branch of business, and that action has resulted in significant redundancy in certain types of data. This company already had a number of separate systems dispersed across various departments. There was some degree of integration between them, but they were designed at different times, and followed different implementation approaches. Sound familiar? Actually, this is a somewhat typical situation in many companies. One of the problems the company now has to deal with, or more particularly the IT-User community, is to understand the data available in each system and its interrelation with the data on other systems. In Figure 9-1, you can see some of the situations mentioned.

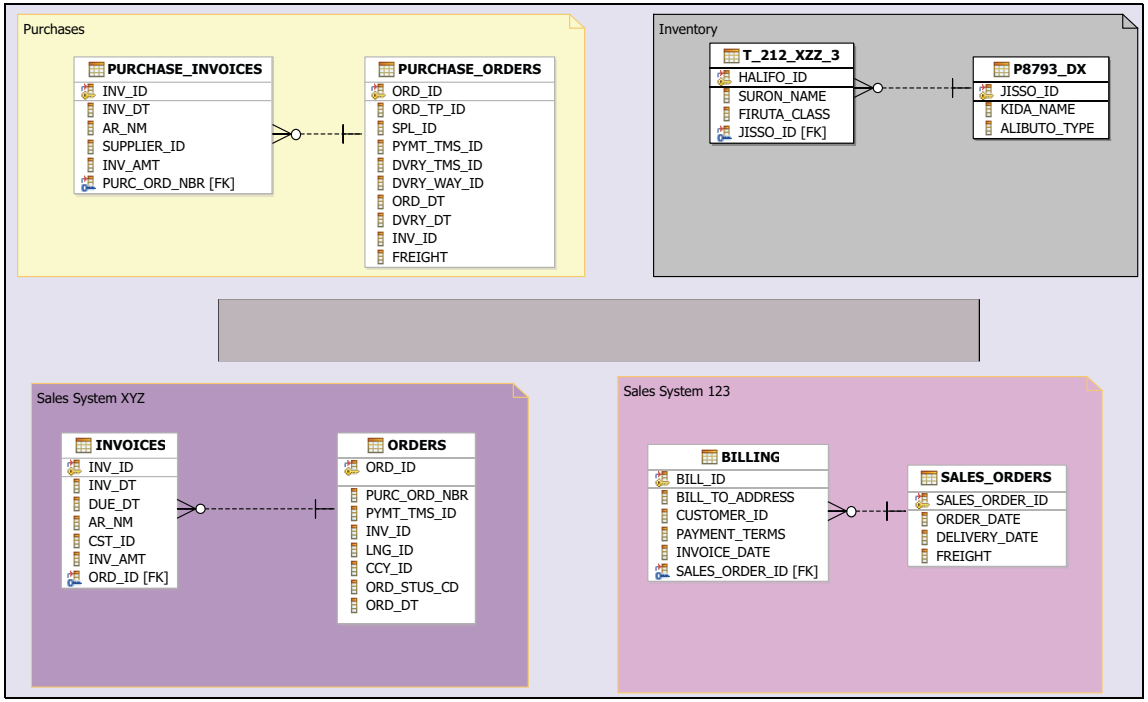


Figure 9-1 Data Structure of Company ITSO-CO

As examples:

- These systems are not thoroughly documented anywhere, which implies that a system developer will have to analyze the database to find out what exactly the format is of any given field.
- Many times the table or field names are not meaningful. As examples, consider names such as T_212_XZZ_3, and P8793_DX as depicted in Figure 9-2. There needs to be documentation indicating such things as the primary purpose of each object, how can it be used, and from where it is populated.

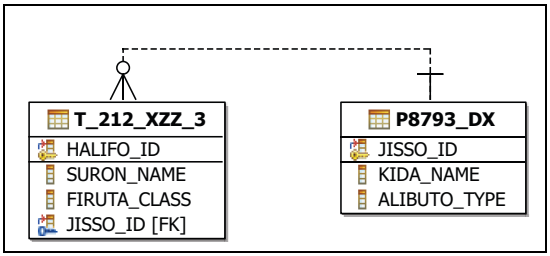


Figure 9-2 Meaningless table names

- In Figure 9-3 there is a field called Purchase_Order_Number (PURC_ORD_NBR) in two different tables. They are *Orders*, in the Sales system, and *Purchase_Invoices* in the Purchasing system. In the Orders table, Purchase Order Number indicates the purchase order number used by the customer to order the goods that are being sold to him. In the Purchasing system, the Purchase_Order_Number indicates the purchase order number used when buying goods from a supplier. In the Sales system the first field, ORD_ID, is more typically called Customer_Order_Number. These are situations where a name in one system is used as synonym for a name in another.

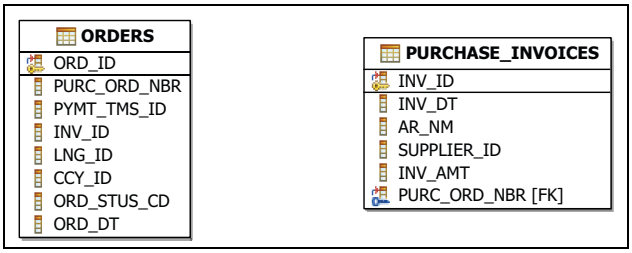


Figure 9-3 Same name with different meanings

- Now consider the Sales systems of the two merged companies. Look for invoice information. In Figure 9-4 on page 450, notice that the INV_ID field in

the INVOICES table of Sales system of the Company ITSO-CO is equivalent to the field BILL_ID of the Sales system of the Company Res-CO. As well, the ORDERS table in the PURCHASES system, does not have the same meaning as in the Sales System.These observations may be more easily seen in Figure 9-1 on page 448.

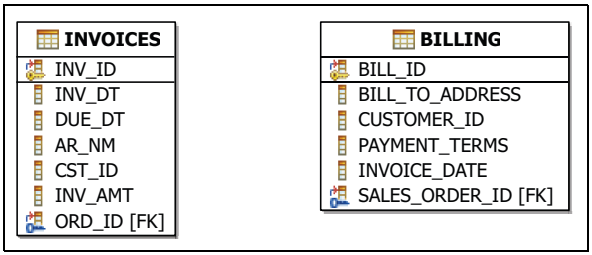


Figure 9-4 Different names with the same meaning

Those familiar with these systems may think that all these clarifications are unnecessary and irrelevant. But think of a new user entering the company. Or, even riskier, a user changing departments who is sure they know the meaning of all the names used in the new department. Perhaps now you can get an idea of the confusion, and misinformation, that can occur in these situations.

Well, the meta data is precisely the information that has to be kept in order to efficiently describe all the data objects in a company. A primary objective is to avoid the huge pitfall of having a highly inconsistent and unreliable methodology for the management of the information about the different data objects being used in the business. Keep in mind that a data object is not necessarily a particular field in a form or a table. In this context, when we say Data Object, we mean such things as fields, tables, forms, and reports. Even entire applications, such as Sales and Purchases, and all their associated data structures, can be considered Data Objects subject to the requirement to be described in the meta data. Keep in mind also that not only business users, but also system developers and modern and conveniently designed applications, need to work and access good meta data for efficient and reliable functioning. This is especially critical when interacting with other users, departments, or systems.

Notice that meta data not only describes structured, database recordable objects, it can also describe any sort of data container, such as e-mails, pictures, sound recordings, memos, books, and disks.

9.2 Meta data types according to content

According to the data described, and the usage, we can distinguish four types of meta data:

- Business
- Structural
- Technical
- Operational

9.2.1 Business meta data

Also called informational or descriptive meta data, business meta data describes the contents of the business objects or any other data object in which the user is interested. We now take a look at typical questions or problems addressed by business meta data:

- In Figure 9-5, we have two tables from Sales Orders and Purchase Orders Systems. In both tables we find the field WEIGHT, which is used for freight calculation purposes with, more or less, the same meaning. However, in one system the weight is given in Kilograms. In the other system the weight is given in pounds.

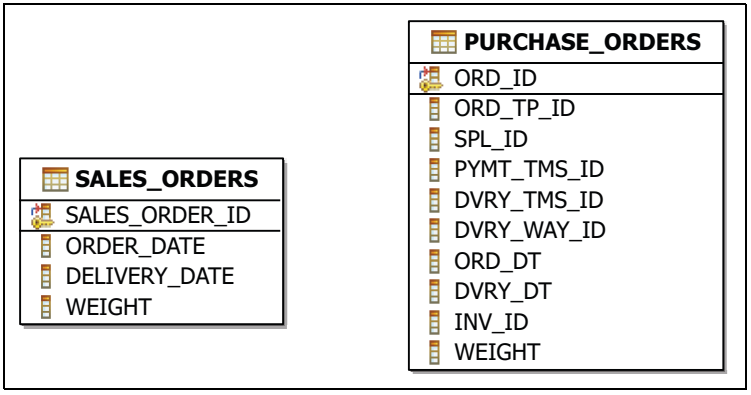


Figure 9-5 Same name, same meaning, different unit of measure

- In the company reporting system, we also find a field called YEAR_TO_DATE_ORDERS. A user wanting to use this field for reporting purposes for the first time will have questions about it, such as:
  - How is the amount given? In dollars (think of an international application)? If it is in dollars, is it in currency units, or in, for example, thousands (K) of dollars?

- Are Order Amounts gross or net? That is, with or without such things as additional costs, taxes, and discounts?
- Does year-to-date include the orders of the present month? Or, the orders of the present week? What about orders from today, until precisely the moment of the inquiry?

Reference meta data

The reference meta data is an important type of business meta data and it describes the lists of values, groups of codes, types, and domains, which implicitly contain business rule information or static external information, such as status, geographical codes, types, and categories.

9.2.2 Structural meta data

It is the meta data that describes the structure of the different data objects. It is useful when implementing navigation and presentation of the data to the user. For example:

- An EMAIL data object is composed of a header and body. The header has TO_ADDRESS, FROM_ADDRESS, DATE and SUBJECT. The body has TEXT, and attachment names or descriptors.
- A BOOK data object is, for instance, comprised of a TITLE, AUTHOR, TABLE_OF_CONTENTS, PREFACE, CHAPTERS, INDEX, and GLOSSARY. Every CHAPTER contains from 0 to many HEADERS.
- A PICTURE data object is described by a FILENAME, the FORMAT of the file (such as BMP or GIF.), and a description.
- A SOUND file can be described by the FILENAME, FORMAT, DATE, HIGHEST_TONE, and LOWEST_TONE, for example.

9.2.3 Technical meta data

This describes all technical data necessary for the development, integration, functioning, and comprehension of the system applications. It defines source and target systems, and their table and fields, structures and attributes, and derivations and dependencies. For example:

- Database field formats and lengths, such as CUSTOMER_NAME VARCHAR(100)
- Table physical characteristics, such as TABLE_TABLESPACE_NAME = TS_MAIN_DATA
- Control fields, such as Record_Creation date or Record deleted

This meta data is useful for Business Intelligence, OLAP, ETL, Data Profiling, and E/R Modeling tools.

9.2.4 Operational meta data

This type of meta data manages the information about operational execution (events) and their frequency, record counts, component-by-component analysis, and other related granular statistics.

It is primarily used by operations, management, and business people.

9.3 Meta data types according to the format

According to the format, or type of container, we can also distinguish two types of meta data:

- Structured
- Unstructured

9.3.1 Structured meta data

Structured meta data is the meta data that can be efficiently stored and systematically accessed. For example, a relational database is a good recipient for structured meta data, since the data will always be accessed in the same way, for example, by inquiring using the same variables through the same paths. XML documents are also adequate for storing structured meta data.

9.3.2 Unstructured meta data

Unstructured meta data is stored in repositories or containers that do not allow systematic access to the information, for example, audio recordings, descriptive written documents without any internal structure, and employees.

9.4 Design

It is very important that the design of the meta data management system is user-oriented. Therefore, one of the first steps when building a meta data system is to conduct a workshop with the users. The goal of this workshop is to ensure that the business needs are satisfied, and that this is done in a comfortable way for the user. When designing a meta data system you should consider the following development methodology and structures:

1. Meta data strategy (Why?)

2. Meta data model (What?)
3. Meta data repository (Where?)
4. Meta data management system (How?)

9.4.1 Meta data strategy - Why?

The first step working with meta data in a Business Intelligence environment is to develop the meta data strategy for this particular solution. It helps respond to the question why is it needed along with a detailed description. The strategy determines the specifics regarding meta data, including:

- Is there an existing meta data strategy and direction in place?
- Why is meta data needed?
- What will be the business value of this information?
- Who will use the meta data (including both tools and people), and for what purposes?
- What type of meta data is to be captured?
- When will that meta data be captured?
- Where will the meta data that is captured and collected get stored?
- How and when will this capture occur?
- How will the captured meta data be made available?

Once these questions have been answered, then the results must be prioritized in order to determine the level of effort that is required. The results of this prioritization allow the work to be directed toward the most important tasks.

9.4.2 Meta data model - What?

The next step is to determine how best to satisfy the prioritized needs identified in the strategy by determining what specific meta data will be captured. In order to answer *what* meta data to capture, an initial identification of the tools that will be used in the data warehouse environment need to be inventoried. These include data modeling, database management, ETL, and user reporting and analysis tools.

Each of the specified types of tools has meta data that is needed as input to begin the specific tasks that will be supported by the tool. In addition, each tool produces meta data, which is needed by another tool at another point in time. The information that should be gathered about the tools is:

- A list of all types of meta data that each tool supports.

- Identification of the meta data required to start the activity supported by each tool.
- Identification of the meta data that is produced by the activity supported by each tool.
- Determination of what specific types of meta data will be captured.

Note: Before attempting to do this identification, check to see if a meta-model has been developed for each tool.

You need the previous activities to determine the technical meta data. But you still need the business meta data, so that users can locate, understand, and access information in the data warehouse and data mart environments.

Out of the analysis of these two meta data types, you should get a E/R logical meta data model, normalized to a reasonable extent, that consolidates the meta data models of the different systems.

History

An important aspect to consider when specifying the meta data structure, is the history of the meta data. To a lesser or greater degree, the database structures of a company evolve. This evolution can take place at an application or database level, where new databases or applications replace old ones. Inside a database, old tables may not be used any longer and will probably be archived and deleted from the database. Data warehouses and data marts, because of their iterative approach, are especially volatile and unstable. For these structures, if the data history is kept during long periods, and the structure of the data changes, it is very important to keep track of the changes. Otherwise, there is a high risk that business information maintained during many years may become useless because it cannot be understood.

9.4.3 Meta data repository - Where?

The next step will be to determine where to collect the meta data that is captured during each activity and/or tool evaluation. This location may be in the tool itself (data models or reports), or collected and integrated into another medium (word document, spreadsheet, or meta data repository). As with any situation where information resides in multiple locations, the redundancy may introduce data integrity and data quality problems. On the other hand, integrating multiple sources of information requires additional time and effort.

Often the decision will be made to merely capture the meta data in each individual tool and then *virtually integrate* this by providing knowledge of where that meta data resides. Most, if not all, of the meta data can be integrated into a single meta data repository. This last method is costly but provides the highest

level of quality and should only be attempted if the organization is committed to maintaining and using the meta data in the future.

9.4.4 Meta data management system - How?

Once the determination of why, what, and where meta data will be addressed in this Business Intelligence effort, the actual steps to do the capture must be integrated into the project plan. This task may be fairly straightforward, as creating database tables and therefore producing the technical meta data for the tables. On the other hand, ETL (extract, transform, and load) tools and user query tools can capture a variety of meta data. A decision must be made as to the appropriate place to capture various types of meta data.

A good example is the business rules. Even though no one would doubt that this information is important and somehow needed, it must first be determined whether the results of this discovery are captured. If the decision to capture is made, then the second decision is where and when that capture will be done. Is it captured in the ETL tool where it is used as part of the load process or in the user query where the results are presented? The answer to these questions will determine whether the meta data captured is considered business (the rule defined in business terms) or technical (the rule expressed as a calculation or edit) meta data, or both.

If the meta data will be captured and maintained in the individual tools, the time to capture and the validation of the effort must be accounted for. If the individual tool meta data is to be integrated into a separate repository, time must be allocated for potential integration and resolution effort.

9.4.5 Meta data system access - Who?

The last step, and the one most often ignored, needs to address who will use the meta data that is captured and how it will be provided to that identified user. The approach to provide business and technical meta data is often handled very differently (if indeed business meta data is even considered at all).

The majority of the technical meta data is captured and maintained in individual tools. This information is most often required by developers or administrators, and access to this information is via use of the individual tools. This access must be identified and made available.

The type of business meta data to be captured and the users of this meta data are much harder to identify, and often ignored. When this business meta data has been identified, a way of accessing this information by the user must be developed. Most recently, APIs (application program interfaces) have been used to capture meta data from where it is stored and used to make it available

through other methods. An example of this is to take the definition of an item on a report from wherever that definition is stored, and make it available through a help facility as the report is being viewed.

One of the key points to remember in this step is that if the usage of meta data cannot be identified, the value of capturing it should be questioned. Data, of any kind, that is not used tends to decay.

9.5 Data standards

Data standards are part of the meta data and can be understood as the rules or guidelines to be considered or followed when naming and coding the data objects, or setting the base of the data structure for every project that creates or handles data structures. These rules would be applicable to every project or activity, independently of the number of people and the size of the project. Here are examples:

Imagine that, in order not to forget your tasks, you write them down on your note pad, and you always write one short word or code with the task description indicating the priority and urgency. You may even document the meaning of the codes somewhere so that you do not forget what they mean. If you do that and do it consistently and systematically, and you always specify a code and a description of the task, and you always use the same codes, then you have designed your own data standards, and, although simple, they are working efficiently for you.

From then on, you will only find good reasons for designing and maintaining (meta) data standards. Whether you work in a small team designing data structures or creating object information, and your work has to be combined and consolidated with the others' work, or whether you consider the Web, a worldwide data structure, where everybody puts information, and from which everybody gets information, you will find that data standards are essential to efficiently create, interpret, and manage all the information.

- The contents of the data
- The naming of the data
- The structure of the data
- The format of the data

9.5.1 The contents of the data

Data standards referred to as structured contents, such as the ones of a database, are equivalent to the reference meta data covered in "Reference meta data" on page 452. In that section we learned that reference meta data contains

information about the business rules, and one of its purposes is, precisely, to establish unique and common values for the data object contents (such as names and codes).

9.5.2 The format of the data - domain definition

A data domain, in the context of meta data, is the definition of a common type of data in your business. In Figure 9-6, you can see an example with the definition of the PERCENT domain. It states that a PERCENT type attribute is defined as a numeric field that can be *nnn.nn* where "." is the decimal separator. In addition to the attributes you see in this example, you can specify that the values this domain can adopt range from 0.00 to 100.00.

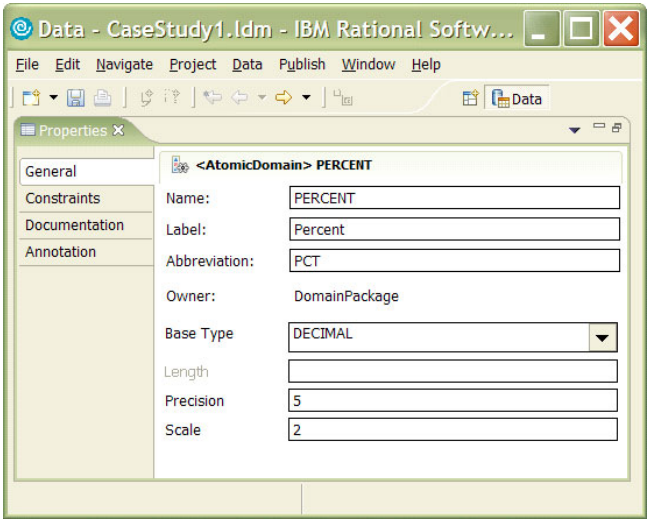


Figure 9-6 Example of a data domain definition

Using data domains, modern case tools allow you to efficiently describe the different percents you have in your database. For example, as you see in Figure 9-7 on page 459, for defining the attributes *AlcoholVolume%*, *TotalSales%*, or *Tax%*, you just need the data domain `PERCENT`.

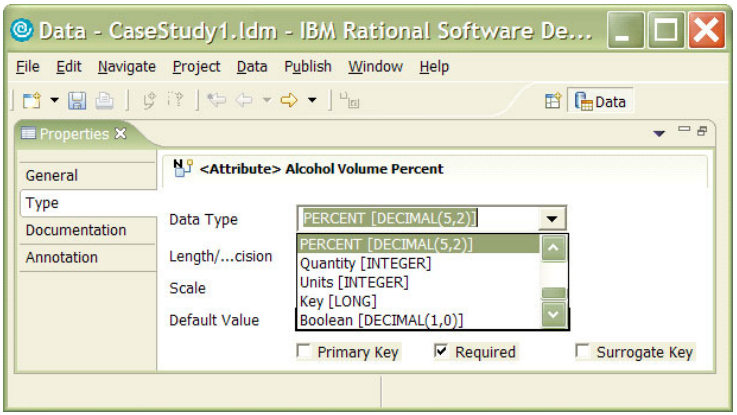


Figure 9-7 Assigning data domains

Domains are used in data modeling to speed up the process of design, and help to efficiently maintain the data model.

Following the previous example, if, after the data model is finished, you need to redefine all the attributes under the same domain, you only need to redefine the domain. The tool cascades the changes of the domain to all the attributes that use that domain.

In the following sections we explain and recommend what to use for common data types.

Dimension keys

For dimension keys and, generally speaking, for data warehouse keys, we recommend using surrogate keys. We explain the benefits and reasons in detail in “Primary keys for dimension tables are surrogate keys” on page 139.

Always choose an integer format. And choose the integer type in your database that provides the largest range. Integer length types vary from database to database. In DB2, for example, LONGINT, goes up to +9 223 372 036 854 775 807.

Tip: If you are concerned about performance when choosing the length, the general rule is:

If you foresee having a maximum of 10^n records, then choose the integer type necessary to contain 10^{2n}. For example, if you are sure that your table will never have more than 100,000 records, then choose a surrogate key format to accommodate 1,000,000,000. With this approach, it is unlikely that you will run out of identifier values.

Date data type

This is a data type that generates much confusion and causes trouble for people and programs when not correctly specified, published, and accepted. Just think, for example, about the date 10/12/05. In Europe that date corresponds to the tenth of December. In the United States, however, that date corresponds to the twelfth of October. And the year? Does it refer to 1905, or to 2005, or to some other century?

Always define dates so that you use them consistently. For example, `mm/dd/YYYY`, where `dd` is the day of the month, `mm` is the month of the year, and `YYYY` is the year in the Gregorian calendar. These are typical values Western countries use. It is good practice not to use a YY format for the year. You may recall the issues caused by a two-digit format when we changed centuries from 1900 to 2000.

Exceptions

Some applications still accept years in YY format. The important thing is that you define appropriate defaults and the year information is stored in the database in four digit format. Depending on the business subject, you should define your defaults in one way or the other. An example of how to process these date types is depicted in Example 9-1.

We do not see the benefit in using a two digit data format, and recommend you to always present the complete four digit year. Unless you are quite convinced about the short life of your application and the evolution and situations of the business are covered, this approach could become unreliable for assigning a wrong century to the years. For example, with regard to birth dates, it is possible to find people born in 2005 and 1905.

Example 9-1 Year format example

```
Date of first ADSL contract in YY format (in pseudocode):
   if YY > 90 then ADSL_DATE = 19YY
      else ADSL_DATE = 20YY
```

Time data type

Time is less confusing than an unspecified date format, but also prone to misunderstandings. Always document the format you want, need, or expect to use in your daily operations, such as hh:mm:ss or hh:mm. Also specify the separator, as ":" or ".", and do not forget to indicate whether it is in 12 hour format (a.m./p.m.) or 24 hour format.

Boolean data type:

This is a type of domain that contains only two logical values: true and false. Unless the database server you use explicitly manages boolean values, we recommend you define this domain as a numeric field admitting the two values 0 and 1. You can say that, for the user, a YES/NO field is better to understand than a number. But to get the user accustomed to it and gain acceptance to this solution is relatively easy. And, the database systems can handle 0 and 1 more efficiently, if appropriately defined. In addition, it makes additive operations straightforward.

For example, assume that you have defined a boolean field named HAS_NO_CAR as numeric, and this field admits the values 0 and 1. If you want to find how many customers have not yet gotten a car, you only define a SUM operation on this field. Whether you are using a modern reporting tool, or you are manually entering the corresponding SQL-SELECT statement, this is a very straightforward operation.

Now imagine that you have defined the field as CHARACTER. The selection of the customers who have not yet gotten a car is still easy. But if you or the user needing this information do not remember the two exact values of the boolean field, and the database or user application makes a difference between upper case and lower case, you can have trouble remembering which of the following variants is the right one: Y/N, y/n, Yes/No, YES/NO, yes/no, or True/False. This variant is also less appropriate because you have to cast between different data types. If you do not know well the database server system and do not know how to use its functions, or the OLAP tool you use does not efficiently support the data type conversion needed, then you have additional work to do when using boolean values for selection or calculation purposes.

Tip: Define your boolean domain as numeric, and use the values 0 and 1 for false and true respectively. This will enhance the data management performance, and simplify the arithmetic operations based on boolean fields.

9.5.3 Naming of the data

In the section 9.1, “What is meta data?” on page 448 we saw how important is to have a common and unique understanding of the names given to every data object. In this section we list and describe options that help you to properly name your data and determine naming rules for your project or data structure.

Historical considerations

In the past, one of the primary characteristics of computer systems was the lack of enough space for comfortably storing all the information handled. That affected the way data was stored, for example, when full postal addressees had to be stored in 20 character fields. But it also imposed limitations when naming system component names, such as, for instance, tables or fields. Because of that, system developers and users entering data had to start abbreviating as many words as possible. And, due to the lack of resources that database administrators and developers had available when working with data objects, often the name given to the data objects had nothing to do with the contents of the data object.

Here is an example taken from a recent project. During the physical design of the database of a new data warehouse, the customer insisted on using the same naming rules for the new data warehouse tables they had for their existing OLTP environment. So, for example, the table containing the Customer Orders was DWHTW023, where DWH was the user or schema owning the table, T indicated that it was a table, “023” was a sequential number, and W had no special meaning as far as we could recall. Here ORDERS would have been a more appropriate name.

Remember that the standards are for helping business operations to work reliably, provided the benefit the standards bring is larger than the overhead they add to the business. It was correct to try to follow the standards, but it would have been much better to have changed and adapted somewhat to the new environments and business area needs, which was, reporting and data analysis activities.

Naming: General considerations

Although it is common to use client tools for OLAP and reporting activities, often the business user is exposed directly to the database tables of the data marts or even to the main enterprise data warehouse tables. It is therefore important to name the table and fields as meaningfully as possible. The name should whenever possible refer to its contents and nothing else. The goal is that by looking at the name of the table or field, the user has no doubt about its contents. In the previous example, none of the parts of the name had any value or meaning to the user.

Tip: Naming **booleans**. Whenever possible, use a verb in third person, such as, HAS, IS, and OWNS. For example, HAS_ALCOHOL, IS_PREFERRED_CUSTOMER, and OWNS_MORE_THAN_1_CAR. This is sufficient to indicate to the user that the contents of the field is of the type true/false.

Abbreviations

Due to system limitations, but also in normal written or spoken communications, people tend to shorten the words for what we euphemistically call *language economy*. At the end we frequently find conversations, communications, and documents with such a high percentage of abbreviations, that somebody new to the business or the subject probably does not understand.

And if there is a tendency to shorten what we verbally say, this tendency, regarding computer systems is even bigger because we have to key it instead of saying it. Fortunately any modern computer applications dealing with data at any level, including OLAP applications, offer easy-to-use drag-and-drop features that allow us not to have to worry any longer about our typing skills or the high probability to misspell a field. In OLTP environments you may still consider it advantageous to use abbreviations for naming data objects, because the user is normally not exposed to the internal data structure. However in a business intelligence environment we highly recommend you avoid the usage of abbreviations. Only names that appear in practically all tables are candidates for abbreviation. For example, Name, Description, Amount, and Number. Otherwise, use full names, unless the database physical limitations force you to abbreviate. And if you use abbreviations, abbreviate consistently following the standard company abbreviation list.

Technical prefixes, midfixes, and suffixes: Substrings containing technical or meaningless information, such as T for table, can be useful in OLTP environments. However, avoid them in data warehouse environments.

In Chapter 7, “Case Study: Dimensional model development” on page 333, we used the standard abbreviation list used in IBM. An example is in Figure 9-8 on page 464.

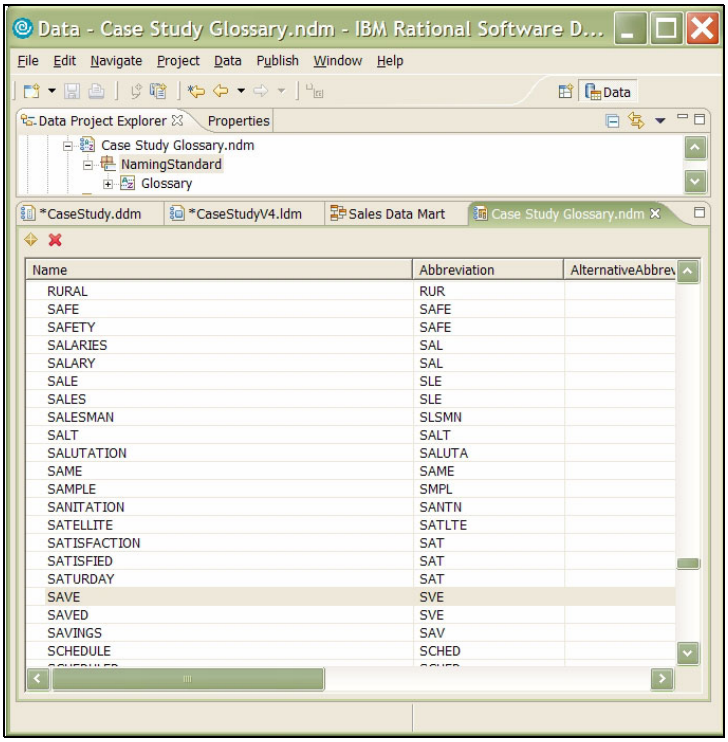


Figure 9-8 IBM abbreviation list

Glossary

Just like abbreviations, companies develop their own language or use acronyms typical of the respective business branch. It often happens that, during business conversations, users working in one department do not understand the users of another department. This situation gets even worse when the listener, or reader, is external to the company. In order to solve or alleviate that situation it is normal to maintain a corporate glossary, or according to the dimension and complexity of the areas covered, specialized glossaries.

A glossary is a document or database, on paper or in electronic format, with a list of all the business words used in the daily activities of the company and their corresponding descriptions.

A company should always maintain a central and unique glossary. Modern tools allow an easy integration and publishing of this data.

9.5.4 Standard data structures

In 9.5.2, “The format of the data - domain definition” on page 458, we explained that a domain is used to describe the structure of an attribute or field for standardization purposes. In the same way, you can use a domain to predefine, or standardize, upper level structures, for example, the fields of a physical dimension table, the attributes of a logical entity, and the relationship between determinate types of tables.

Structure example

When designing a dimensional model, there are several data objects that should be among your common objects. They are depicted in Figure 9-9.

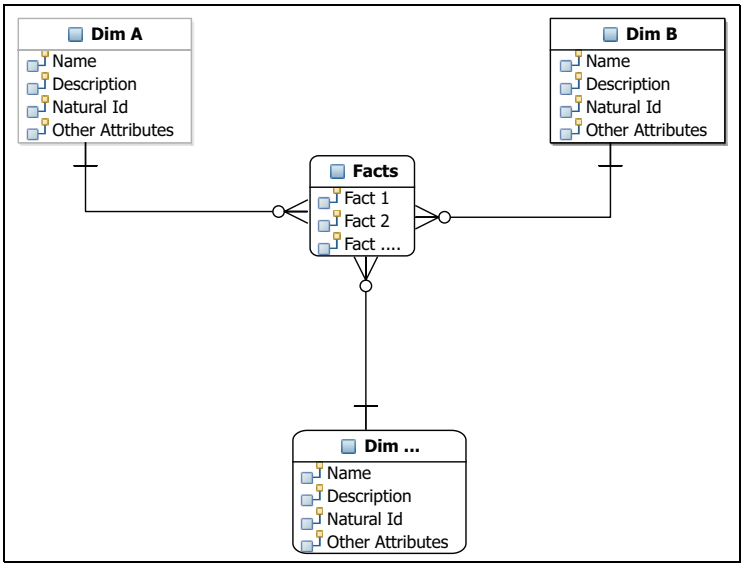


Figure 9-9 Example of a standard basic dimensional design

These are common elements to all the logical entities: “Names”, “Descriptions”, and “Natural Ids”. Keep in mind that this is an example for orientation purposes, and you might need different, or additional, attributes when designing your dimensional model. It is even possible that you may only need one attribute such as “Name” or “Natural Id”, and no “Description”. The need to include one or the other may change according to the business requirements. Nevertheless, at the corporate level, it is still possible, and recommended, to agree and propose a common fixed set of fields, even if not yet requested. Consider this from the perspective of the conformed dimensions, (see 5.4.3, “Conformed dimensions” on page 144), and consider how much will it take to implement a new field into the dimensions, and how big is the probability that a business process needs that information in the near or long-term future.

All the attributes we described above are defined at the logical levels. At the physical level, however, we need additional physical fields, depending on the auditing and technical controls you might need to perform on your data. An example is depicted in Figure 9-10. There we see more fields which have no particular business meaning but are needed for various technical purposes. For example, the surrogate keys (Dimension Keys) and the Record Creation Date, are necessary to know exactly when this information was written into the database.

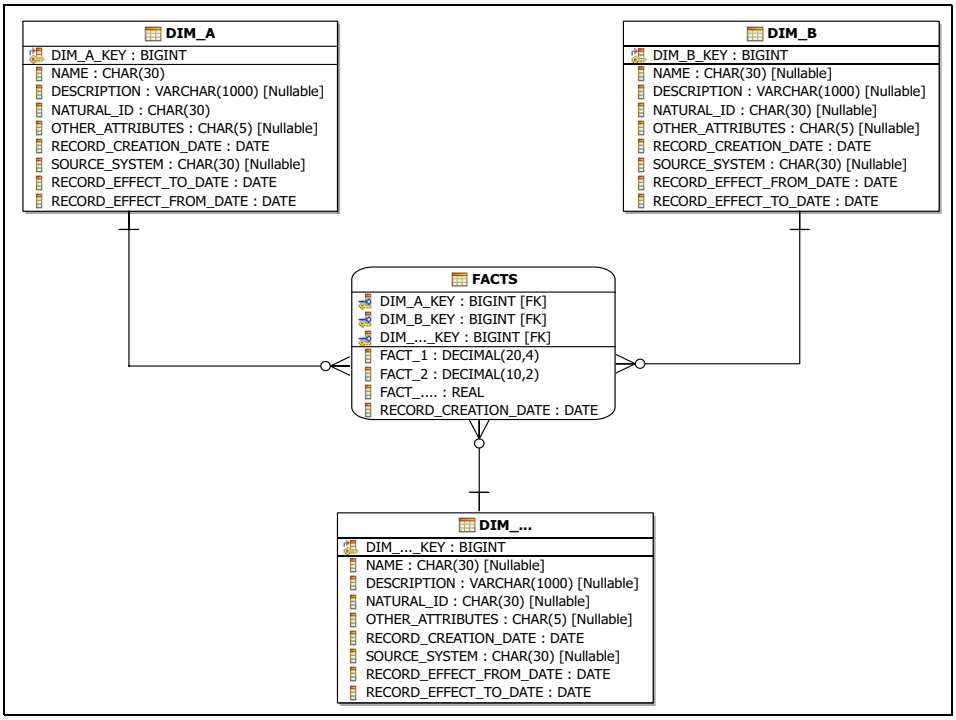


Figure 9-10 Example of a physical structure

Figure 9-11 on page 467 shows an example of a logical representation that can be used as a reference. Especially the data types and formats must be defined and consistently used across the data structure.

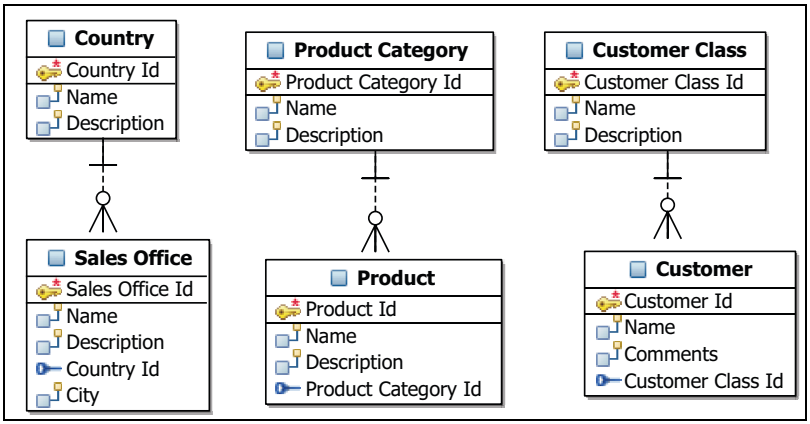


Figure 9-11 Reference meta data examples

Reference meta data structures

Reference tables are usually good for describing business rules, for example, product classes, sale order types, and tax types. In Figure 9-12, entities, including attributes, refer to typical business information, such as Product Categories, and Customer Classes.

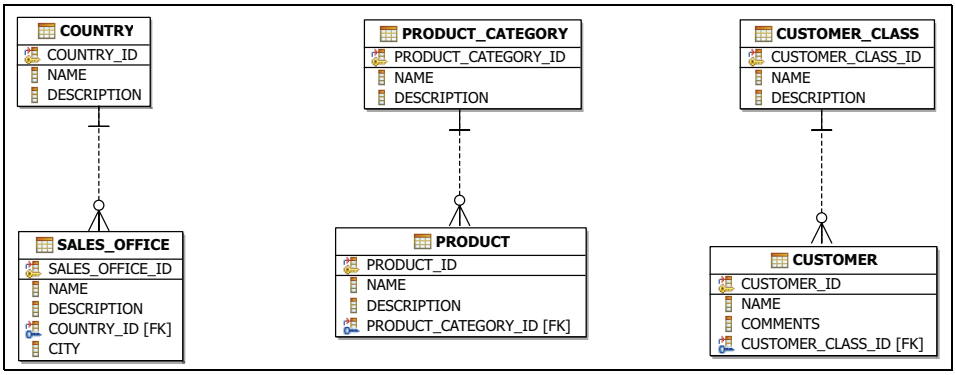


Figure 9-12 One physical table per meta data reference type

Although, at a logical level, every type of reference meta data deserves its own logical entity, when physically implementing there are several approaches. The key common ones are:

- One physical table for each reference meta data type
- Only one physical table for all the reference meta data types

Reference meta data in separate tables

One way to physically implement the reference meta data is by creating one table for every group of reference codes or types. So the physical design to the group above would convert into one such as that represented in Figure 9-13 on page 469. This approach is convenient because of the relatively easy way to join the tables.

And, there is another potential advantage. If you use reference codes whose natural identifiers have significance and they are not foreseen to change either the code or the meaning, you might consider keeping them as identifiers. Suppose you have a Reference Table called “CURRENCIES”, and you have values such as GBP for British Pounds or USD for US Dollars. Although we insist on the usage of surrogate keys for uniquely identifying the information at a table record level, in this case it might be reasonable to use this short code as the primary key identifier. Practically any entity describing information for which ISO codes are available (such as Countries and Currencies) is a good candidate for this *content denormalization*. If considering Figure 9-12 on page 467, you need to select the Sales Offices in a certain country, you will not need to join the country table (assuming you know the short code of the country, such as CA for Canada or FR for France, for example). If you are highly concerned about query performance, you can follow this approach when populating your reference codes.

However, this first approach presents the inconvenience of having to maintain dozens, possibly hundreds, of additional tables. That can be a problem if you expect frequent changes or additions to the reference meta data structure. If you are concerned about overall database performance you may not want to follow this approach, because having to maintain so many small tables will cause, to more or less of a degree, extra maintenance overhead to the database server, compared with the efficiency reached when managing fewer but larger tables.

Reference meta data in one main table

Another common solution is to put all or most of the reference meta data into one main table, as shown in Figure 9-13 on page 469.

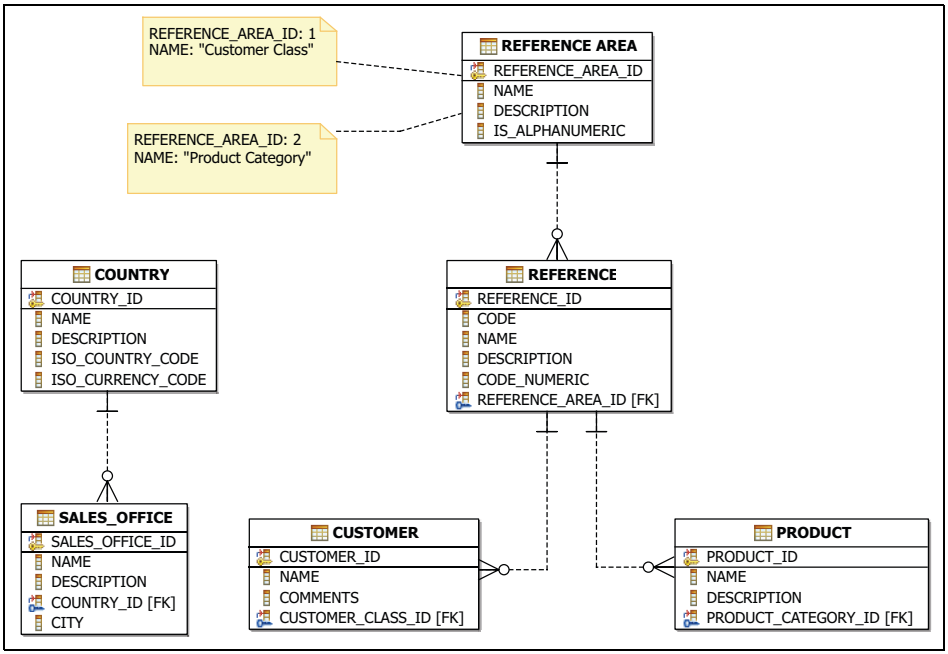


Figure 9-13 One table containing most or all the reference codes

This approach is particularly useful when you have numerous reference tables or you foresee that you will add new ones frequently. In this case with just two tables you can cover as many single reference tables as you need. However the inquiry and usage of the reference data is not as straightforward as when you have one table for each reference data group, or reference area. And in order to be able to sort the reference codes correctly you will need to handle them as numeric or alphanumeric. As you can see in Figure 9-13, the table COUNTRY has been left apart for having a different structure and needing two additional fields: ISO_COUNTRY_CODE and ISO_CURRENCY_CODE.

The use of reference meta data in dimensional modeling

In a dimensional data model you will want to select, filter, or group your data using available reference information, as in the example of the figure below where "`Marketing Customer Category`" can be `WHOLESALER, STORE CHAIN, GAS STATION CHAIN, RESTAURANT,` or `SMALL SHOP`. You may decide that you just want to have those names available in the `CUSTOMER` table. But imagine that you have to perform negative querying, as an example, producing a list of all "`Marketing Customer Category`" names without any sale during the previous year, or select by other attributes that you do not want to or cannot store in the dimension tables. In that case you may consider the use of the formal reference code

identifiers in the dimension tables, and the implementation of reference tables in a snowflake fashion, such as Country in Figure 9-14.

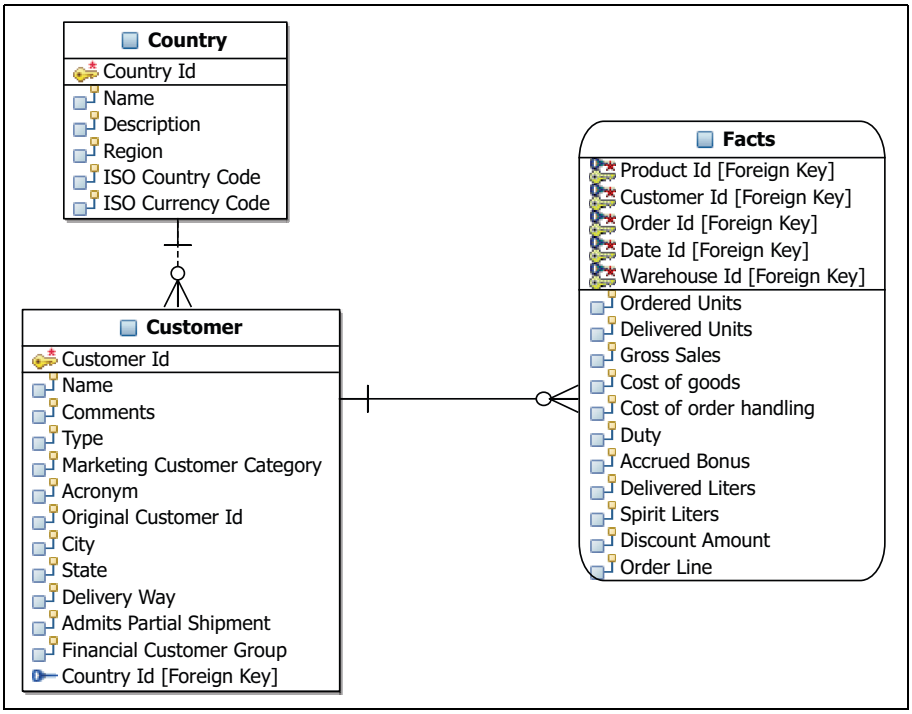


Figure 9-14 Example of denormalized and snowflaked reference data

9.6 Local language in international applications

Companies are challenged by the business globalization that the Internet has triggered, and more particularly, the Web. In order to effectively reach the maximum number of customers, and efficiently work with the maximum number of partners, you need to translate into other languages not only the database contents, but also the interface component labels, such as fields and forms. Typically the translation of the presented information label on the screen is enough, but sometimes it may require additional comments or notes fields, particularly when the field labels can be extended by context help.

An example is depicted in Figure 9-15 on page 471. Here we make the point that, because the students are of a different culture from the teacher, the material is best learned if it is presented in the language of the students.

Figure 9-15 Different cultures, different languages

In Figure 9-16 we show how it would be possible to store multilingual information applied to the data. Here we assume we want to keep translations of names and descriptions for the company products and translation of descriptions for contracts in several languages. This modeling is done at the business data content level. If, for example, you had to translate the names of the labels of the fields you see in the screen, or the fixed headers of a report, then you would have to provide a similar translation data structure within the meta data model so that the names of the meta data objects, and not only the contents, also get translated.

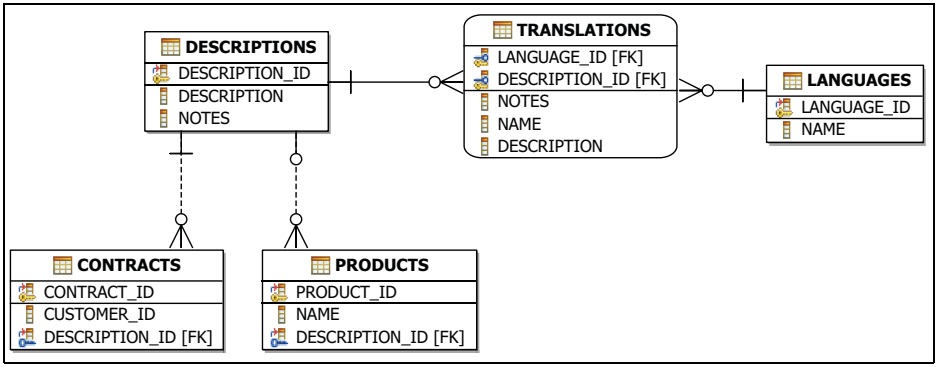


Figure 9-16 Example of structure necessary to store data for different languages

9.7 Dimensional model meta data

In this section, we list and describe all the aspects that have to be considered when modeling or defining the meta data of a dimensional model. We also indicate how they fit into the dimensional model design life cycle (DMDL).

The following meta data is collected during the phases of the DMDL:

- **Identify business process:** The output of this phase results in the Requirements Gathering report. This report primarily consists of the requirements for the selected business for which you will design the dimensional model. In addition to this, this report also consists of various business processes, owners, source systems involved, data quality issues, common terms used across business processes, and other business-related meta data.
- **Identify the grain:** The output of this phase results in the Grain Definition report. This report consists of one or multiple definitions of the grain for the business process for which the dimensional model is being designed. It also contains the type of fact table (transaction, periodic, or accumulating) being used. The grain definition report also includes high level preliminary dimensions and facts.
- **Identify the dimensions:** The meta data documented for this phase contains the information shown in Table 9-1. It includes such items as dimension names, hierarchies, updates rules, load frequency, load statistics, usage statistics, archive rules, archive statistics, purge rules, purge statistics, and attributes.

Table 9-1 Identify the dimensions - meta data

Dimension meta data	Description
Name of Dimension	The name of the dimension table.
Business Definition	The business definition of the dimension.
Alias	Specifies the other known name by which the business users know the dimension.
Hierarchy	The hierarchies are present inside the dimension. They could be balanced, unbalanced, or ragged.
Change Rules	Specifies how to handle slowly changing dimensions (type-1, type-2, or type-3) and fast changing dimension.
Load frequency	The frequency of load for this dimension is typically daily, weekly, or monthly.

Dimension meta data	Description
Load Statistics	Consists of meta data such as: → Last load date: N/A → Number of rows loaded: N/A
Usage Statistics	Consists of meta data such as: → Average Number of Queries/Day: N/A → Average Rows Returned/Query: N/A → Average Query Runtime: N/A → Maximum Number of Queries/Day: N/A → Maximum Rows Returned/Query: N/A → Maximum Query Runtime: N/A
Archive Rules	Specifies whether or not the data is archived.
Archive Statistics	Consists of meta data such as: → Last Archive Date: N/A → Date Archived to: N/A
Purge Rules	Specifies any purge rules. For example, Customers who have not purchased any goods in the past 48 months will be purged on a monthly basis.
Purge Statistics	Consists of meta data such as: → Last Purge Date: N/A → Date Purged to: N/A
Data Quality	Specifies quality checks for the data. For example, when a new customer is added a search is performed to determine if we already do business with this customer in another location. In rare cases separate branches of a customer are recorded as separate customers because this check fails.
Data Accuracy	Specifies the data accuracy. For example, incorrect association of locations of a common customer occur in less than .5% of the customer data.
Key	The key to the dimension table is a surrogate key.
Key Generation Method	This meta data specifies the process used to generate a surrogate key for a new dimension row. For example, when a customer is copied from the operational system, the translation table (staging area persistent table) is checked to determine if the customer already exists in the dimensional model. If not, a new key is generated. The key, along with the customer ID and location ID, are added to the translation table. If the customer and location already exist, the key from the translation table is used to determine which customer in the dimensional model to update.

Dimension meta data	Description
Source	This includes the following meta data:
	→ Name of the source system table: <Table Name>
	→ Conversion Rules: These specify how the insert/update to the dimension table occurs. For example, rows in each customer table are copied on a daily basis. For existing customers, the name is updated. For new customers, once a location is determined, the key is generated and a row inserted. Before the update/insert takes place a check is performed for a duplicate customer name. If a duplicate is detected, a sequence number is appended to the name. This check is repeated until the name and sequence number combination are determined to be unique. Once uniqueness has been confirmed, the update/insert takes place.
	→ Selection Logic: Only new or changed rows are selected.
Conformed Dimension	This specifies if the dimension being used is a conformed dimension.
Role-playing Dimension	This specifies if the dimension is being implemented uses the role-playing concept. Role-playing is explained in 6.3.9, “Role-playing dimensions” on page 285.

Dimension meta data	Description
Attributes (All columns of a dimension)	The meta data for the all the dimension attributes includes the following:
	→ Name of the Attribute: <Attribute Name>
	→ Definition of the Attribute: Attribute definition
	→ Alias of the Attribute: <Attribute Name>
	→ Change rules for the Attribute: For example, when an attribute changes, then use Type-1, Type-2, or Type-3 strategy to handle the change.
	→ Data Type for the Attribute: Data Type, such as Integer or Character.
	→ Domain values for the Attribute: Domain range such as 1-99.
	→ Derivation rules for the Attribute: For example, a system generated key of the highest used customer key +1 is assigned when creating a new customer and location entry.
	→ Source: Specifies the source for this attribute. For example, for a surrogate key, the source could be a system generated value.
Facts	This specifies the facts that can be used with this dimension. Note: Semi-additive facts are additive across only some dimensions.
Subsidiary Dimension	This specifies any subsidiary dimension associated with this dimension.
Contact Person	This specifies the contact business person that is responsible for maintaining the dimension.

The meta data information shown in Table 9-1 on page 472 needs to be captured for all dimensions present in the dimensional model.

- **Identify the facts:** The meta data documented for this phase contains the information as shown Table 9-2.

Table 9-2 Identify the Facts - meta data

Fact table meta data	Description
Name of Fact Table	The name of the fact table.

Fact table meta data	Description
Business Definition	The business definition of the fact table.
Alias	The alias specifies the other known name by which the business users know the fact table.
Grain	This specifies the grain of the fact table.
Load frequency	The frequency of load for this fact table, such as daily, weekly, or monthly.
Load Statistics	Consists of meta data such as: → Last load date: N/A → Number of rows loaded: N/A
Usage Statistics	Consists of meta data such as: → Average Number of Queries/Day: N/A → Average Rows Returned/Query: N/A → Average Query Runtime: N/A → Maximum Number of Queries/Day: N/A → Maximum Rows Returned/Query: N/A → Maximum Query Runtime: N/A
Archive Rules	Specifies whether or not data is archived. For example, data will be archived after 36 months on a monthly basis.
Archive Statistics	Consists of meta data such as: → Last Archive Date: N/A → Date Archived to: N/A
Purge Rules	Specifies any purge rules. For example, data will be purged after 48 months on a monthly basis.
Purge Statistics	Consists of meta data such as: → Last Purge Date: N/A → Date Purged to: N/A
Data Quality	Specifies data quality checks for the data. For example, assume that we are designing a fact table for inventory process. Inventory levels may fluctuate throughout the day as more stock is received into inventory from production and as stock is shipped out to retail stores and customers. The measures for this fact are collected once per day and thus reflect the state of inventory at that point in time, which is the end of the working day.

Fact table meta data	Description
Data Accuracy	This specifies the accuracy of fact table data. For example, assume that we are designing a fact table for the inventory process. We may conclude that the measures of this fact are 97.5% accurate at the point in time they represent. This may be based on the results of physical inventories matched to recorded inventory levels. No inference can be made from these measures as to values at points in time not recorded.
Grain of the Date Dimension	Specifies the grain of the date dimension involved. For example, the date dimension may be at the day level.
Grain of the Time Dimension	Specifies the grain of the time dimension involved. For example, the time dimension may be at the hour level.
Key	The key of the fact table typically consists of concatenation of all foreign keys of all dimensions. In some cases, the degenerate dimension may also be concatenated to guarantee the uniqueness of the primary composite key. This is shown in Figure 5-31 on page 178.
Key Generation Method	The key generation method specifies how all the foreign keys are concatenated to create a primary key for the fact table. Sometimes a degenerate dimension may be included inside the primary key of fact table to guarantee uniqueness. This is shown in Figure 5-31 on page 178.
Source	The meta data for the source includes the following:
	→ Name of the Source: <*Source Name*>
	→ Conversion rules for the source: Rules regarding the conversion. For example, each row in each inventory table is copied into the inventory fact on a daily basis.
	→ Selection Logic: The logic for selecting the rows.
Facts	This specifies the various facts involved within the fact table. They could be: → Additive → Non-additive → Semi-additive → Pseudo → Derived → Factless Fact → Textual
Conformed Fact	This specifies if any conformed facts are in the fact table.

Fact table meta data	Description
Dimensions	This specifies the dimensions that can validly use these facts. Note: Some facts are semi-additive and can be used across only some dimensions.
Contact Person	This specifies the contact business person responsible for maintaining the fact table.

- **Verify the model:** This phase involves documenting meta data related to user testing (and its result) on the dimensional model. Any new requests made or changes to the requirements are documented.
- **Physical design considerations:** The meta data documented for this phase contains the information as shown Table 9-3.

Table 9-3 Physical design considerations - meta data

Physical design consideration meta data	Description
Aggregation	The aggregation meta data includes the following:
	→ Number of Aggregate tables
	→ Dimension tables involved in creating aggregates → Dimension Hierarchies involved in creating aggregation
	→ Fact table and facts involved in creating aggregates
	Information relating to the aggregate tables can include such items as the following: → Load frequency → Load statistics → Usage statistics → Archive rules → Archive statistics → Purge rules → Purge statistics → Data quality → Data accuracy
Indexing	This specifies the indexing strategy used for dimension and fact tables.

Based on all the information listed in this section, we have built the meta data model shown in Figure 9-17 on page 479. It properly describes the relationships between the different components. Notice that it is a logical view, and that, for the

sake of understanding of the data structure, some subtypes have not been resolved.

We have specifically tailored this meta data model to the star schema meta data. However it could be easily integrated in other more generic meta data structures.

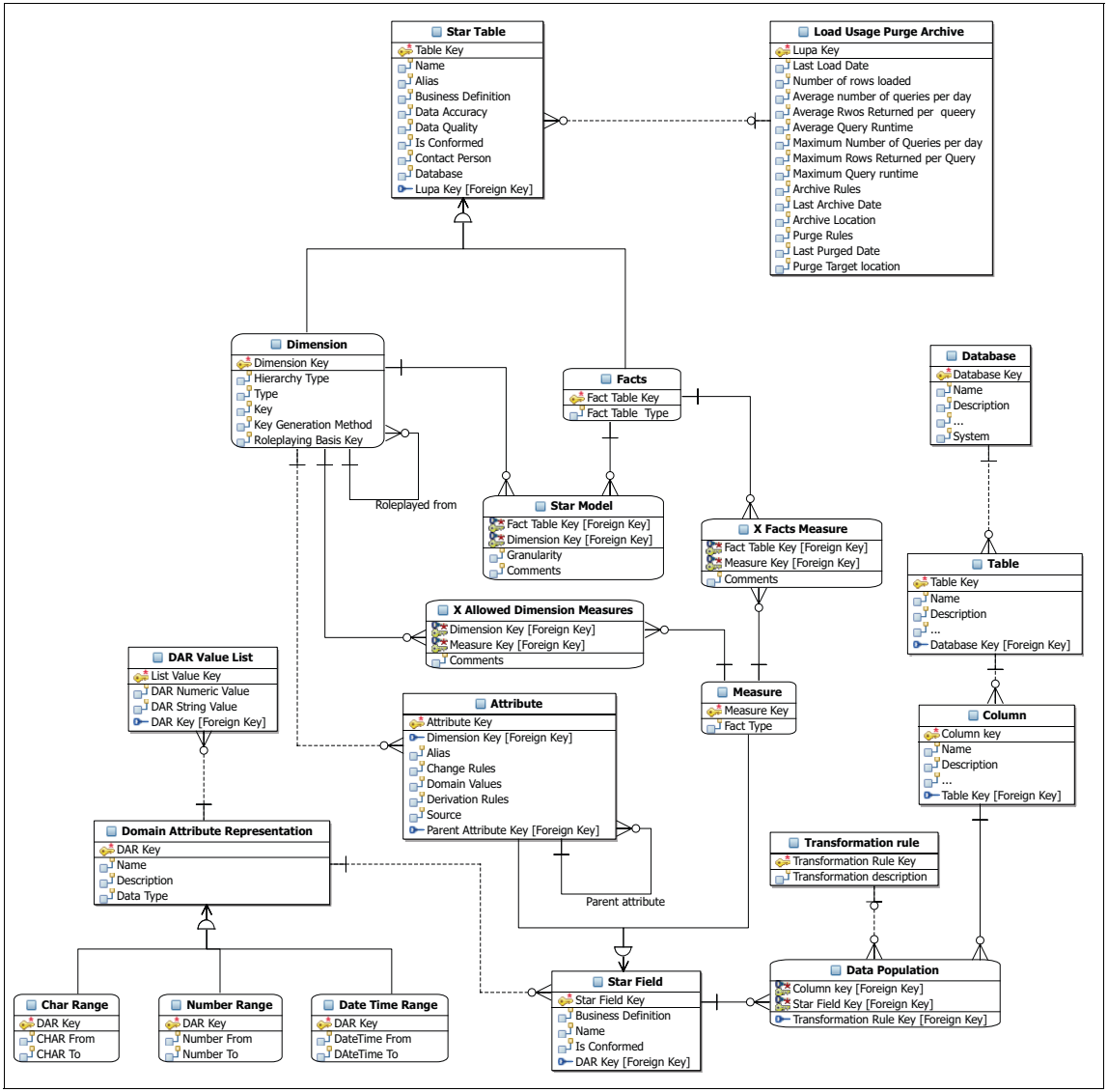


Figure 9-17 Meta Data model

9.8 Meta data data model - an example

In Figure 9-17 on page 479, the data model is focused on the star schema meta data. In Figure 9-18, we show a more generic meta data model which covers not only data stored in databases, but also that stored in other types of data containers. We have chosen a traditional fixed explicit data structure to show the possibility of storing meta data about such types as documents, web pages, and computer office files. If the business you want to model is rather dynamic, and the data structures or business processes change with a relatively high frequency, you might decide to design a more abstract data model where the attributes and entities do not need to be specified in advance.

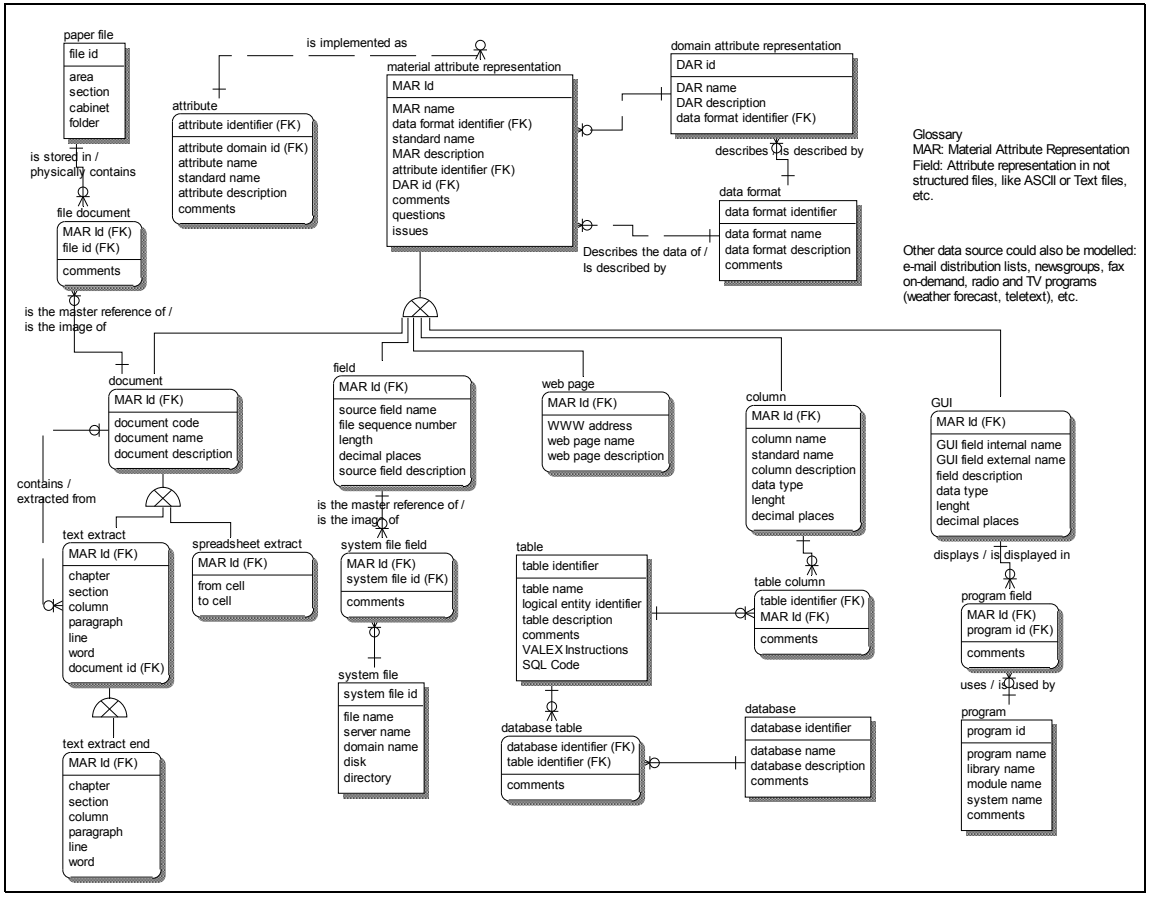


Figure 9-18 Different types of data objects described by the meta data

9.9 Meta data tools

Years ago there were not many types of data management tools, apart from the database management systems. And these systems were the only ones offering a meta data system, which was called the *Data Dictionary*. This was just an externalization of the user data structure descriptions needed by the database management system itself in order to function. And it was covering a limited scope of what we understand today as meta data.

Since then, as data management has evolved, new tool groups, offering to a greater or lesser degree of meta data management functionality, are available. These groups, according to their data management focus, are divided into categories such as ETL (extract, transform, and load), data modeling, data warehouse management, content management, and OLAP.

Today, every data management tool, regardless of the type, normally provides meta data management functions. The problem is that typically the functionality is focused on a specialized area. That is, ETL tools will be very technical and will describe the transformation of data between systems, but often will not implement more abstract business meta data. Or, looking at the other extreme, some OLAP tools will be prolific in mapping and describing business data, but will not describe the meta data present in OLTP systems.

The primary issue resides in the integration of the tool with all the other processes and data structures of the business. Today, almost every company has to deal with information of the types presented in 9.2, “Meta data types according to content” on page 451, that is, business, technical, structural, and reference meta data. Business meta data, in certain companies, not only covers the data but also the processes responsible for the generation and transformation of data at any level. In these situations, it is typical for companies implementing a meta data system to focus on the most relevant area and choose the meta data management tool based on the coverage of that area.

9.9.1 Meta data tools in business intelligence

There are three primary approaches for managing meta data:

- Manually
- Customizing a meta data tool in order to collect information from other systems, or to transfer information by using adequate meta data information exchange protocols.
- Using the available tool modules in order to directly exchange meta data.

Each tool that supports the effort in a business intelligence environment has a need for information to begin. This is often meta data that has previously been collected or produced. How this information is provided (for example, by manual input or tool to tool exchange) must be identified. As the tool is used, additional meta data is created. When the usage of the tool or activity is completed, there is meta data that has been produced as the results of that effort. That meta data should be considered the minimum requirements.

To prevent administrative inefficiency, a mechanism must be found for tools from different vendors to share common information. This can be a very difficult task. For example, each tool defines the same object in a different manner and therefore the objects are not easily shareable. Furthermore, the same meta data is redundantly defined and stored in multiple meta data stores, data dictionaries, repositories, database catalogs, and copy libraries, each with a different API. Typically, the tools cannot exchange meta data among themselves because the meta data is stored in a proprietary format understood only by a specific tool. As a result, changes to meta data in one meta data store are not easily changed in the others.

The work of the Object Management Group (OMG), of which IBM is a major contributor, has concentrated on minimizing these differences through the adoption of a standard meta model for BI efforts and the recommendation of a standard Extensible Markup Language (XML) interface. The results of these efforts should minimize the difficulty in the future.

9.9.2 Meta data tool example

We now look at the information we have been discussing, from a tool perspective. For this discussion, we have chosen MetaStage®, the meta data management tool from Ascential™ Software, an IBM company.

MetaStage introduction

This tool enables integration of data from existing data warehousing processes. A special program module monitors these processes that generate operational meta data. The tool then stores and uses the operational meta data. This enables the collection of detailed information about what has happened across the enterprise.

Most data warehousing products generate their own meta data, with no common standard for exchanging the meta data. MetaStage integrates all the meta data in a centralized directory, from which you can share the meta data with other tools. No additional work is required by the tool vendors. This means you can use unrelated data warehousing tools in a single environment.

By integrating meta data from the entire enterprise, MetaStage can provide answers to such questions as where a piece of data came from, what transformation and business rules were applied, who else uses the data, and what the impact would be of making a particular change to the data warehouse. It also quickly provides detailed hyperlinked reports about items of data and their relationships with other data.

In summary, MetaStage serves as an enterprise-wide tool for many data warehousing functions, such as:

- Synchronizing and integrating meta data from multiple data warehouse tools
- Automatically gathering operational meta data from operational systems
- Sharing meta data, job components, and designs
- Browsing, querying, searching, and reporting on all the meta data of the data warehousing assets from a single point
- Understanding the sources, derivation, and meaning of data in the data warehouse
- Assessing the impact of changes on data warehousing processes

Integrating meta data

You can integrate your own tools and procedures with MetaStage. It enables you to transfer and translate meta data between different external tools, including modeling, design, extraction, transformation, and data analysis tools. For instance, you can collect meta data from these other tools and use it to design data warehouse processes. Or you can export meta data directly to data analysis tools to avoid tedious manual data entry.

MetaStage does not impose a single common model on the meta data you want to share. A common model gives you only the lowest common denominator of the meta data that your data warehousing tools can share. Instead, this tool breaks down the meta data from a data warehouse tool into semantic *atoms* that can then be reconstituted for use by other data warehouse tools. Individual MetaBrokers, described in “MetaBrokers” on page 492, provide an interface between each tool and the semantic units of meta data held in a particular directory.

This tool captures and stores the following types of meta data:

- **Design meta data**, used by designers and developers to define requirements. It includes data models, business meta data, and transformation job designs.
- **Physical meta data,** created, managed, or accessed by tools when they are executed.

- **Operational meta data**, which tells you what happens when a data warehouse activity runs, particularly with regard to the way it affects data warehouse resources.

> **Note:** MetaStage uses MetaBrokers to import and export meta data directly from and to data warehousing tools.

Capturing operational meta data

MetaStage can automatically capture the meta data that describes events that occur when a data warehouse process is running. It then uses this operational meta data to build a view of the relationships between the resources in the data warehouse. By enabling the combination of the operational meta data with the design meta data for these activities, this tool provides you with extensive query capabilities.

Sharing meta data

Meta data can be shared throughout the enterprise with users of the leading data warehousing tools with MetaStage. The major steps to do this are:

1. Create the meta data using an appropriate tool, such as DataStage™ or Platinum ERwin.
2. Import the meta data into this tool via a tool-specific MetaBroker® or capture it with the Process MetaBroker.
3. Publish the selected meta data, so that interested users can subscribe to it and export it for use with their data warehousing tool. Note, this does not have to be the same tool used to create the meta data.

For example, you can import meta data from a database design tool into a particular directory, then export that meta data to DataStage to define the tables and columns to be populated by a DataStage job. Or you might export the meta data for use by a business reporting tool, such as Business Objects. You can also use the *Send to Database* functionality to export data to a set of relational tables.

Analyzing and reporting on resources

After you import meta data into a particular directory, or capture operational meta data from data warehousing activities, you can browse, query, or search the complete meta data structure in the directory, following relationships between the objects. These functions are all accessed from a tool component called the Explorer, which is described in “Explorer” on page 495.

You can do the perform the following functions:

- Inspect the attributes and relationships of one or more objects in the tool directory.

- Specify which person or organization is responsible for an object.
- Associate the object with a business term, the term with a glossary, and the glossary with a business domain.
- Browse from an object, following its relationships to other objects it contains or depends on.
- Run a simple search of a directory.
- Investigate the lineage of data in the data warehouse.
- Investigate the execution history of processes, such as DataStage jobs.
- Use cross-tool impact analysis to investigate the impact of making a change to a data warehouse resource or process.
- Build and run customized and predefined queries for more complex searches.
- Document your meta data by creating reports in a variety of formats, including HTML and XML.

Inspecting objects

This tool makes it easy for you to inspect any object in the tool directory—a DataStage table definition, for example—viewing its attributes and relationships, its overlap in other data warehousing tools, and the queries and templates you can run to generate more information about it.

Browsing from an object

Using the navigation pane in the Content Browser, you can follow the relationships from any meta data object—a DataStage project, for example—to all the objects that populate those relationships, up and down the hierarchy. You can also display the same object from the perspective of a different data warehouse tool—using a different MetaBroker view—and follow the different relationships available in that tools view. An example content browser is depicted in Figure 9-19 on page 486.

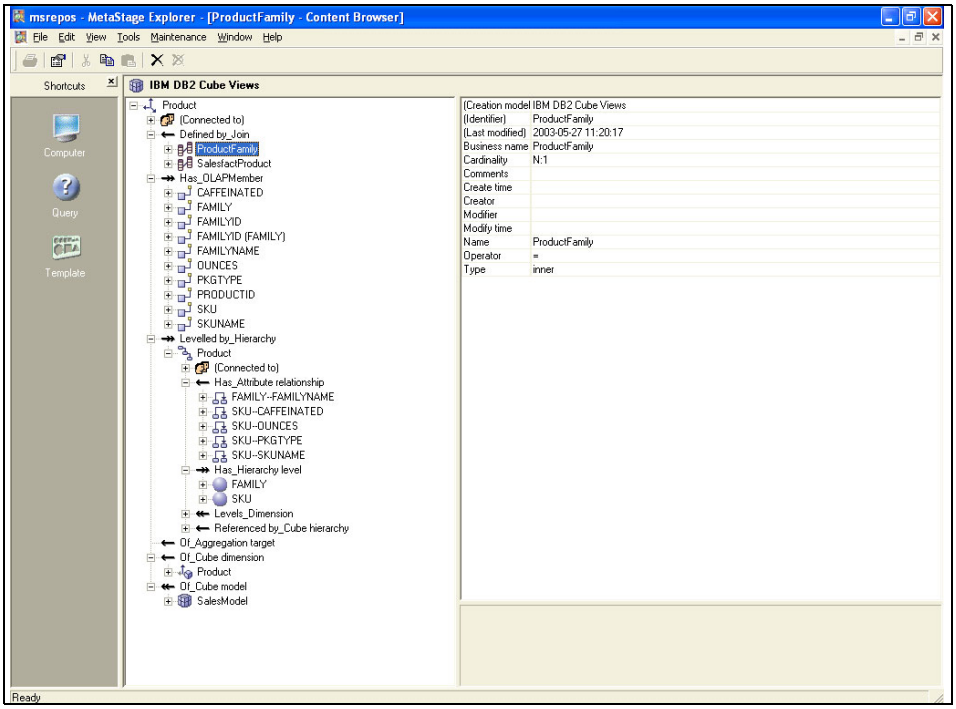


Figure 9-19 Explorer: Content Browser

Performing a simple search

Simple searches let you scan a subset of a particular directory for objects that meet criteria based on name, description, date, and meta data type. You can save the properties of the objects found by the search to a text file that can be read by Microsoft Excel, for example.

Investigating data lineage

Data lineage investigations of operational meta data enable you to find the history of a data item, for example, its source, status, and when it was last modified. When you combine data lineage and impact analysis, you can answer such questions as “From where is a particular column of a particular physical target table derived?” and “When was it last derived?” The tool, Process MetaBroker, gathers the operational meta data to answer questions about data lineage by monitoring activity that affects the data warehouse. It records the data resources affected by, for example, a job run, and whether they were written to or read from. This tool can also import operational meta data from DataStage mainframe jobs. An example of a data lineage report is depicted in Figure 9-20 on page 487.

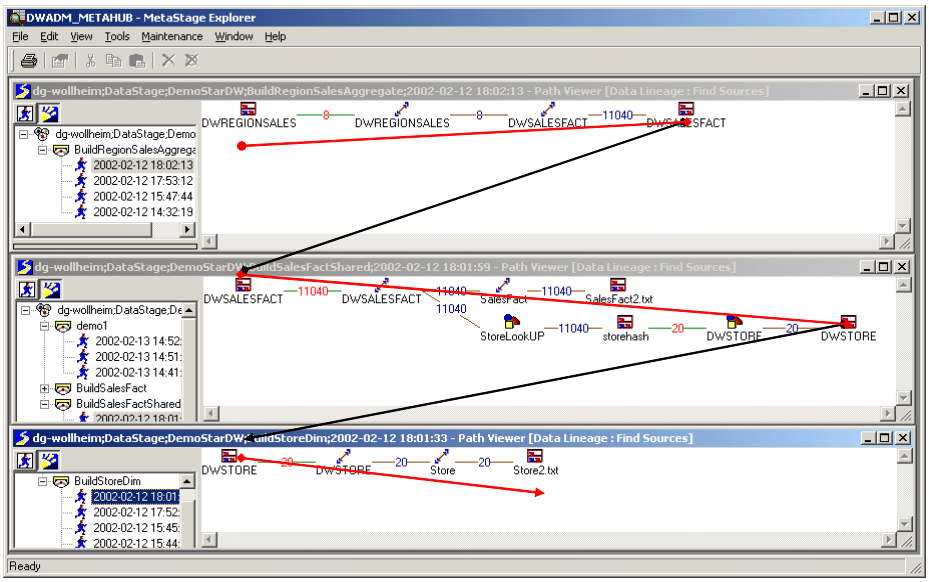


Figure 9-20 Data Lineage report

Performing process analysis

Process analysis investigations are similar to data lineage investigations, but they give information on process execution rather than data movement. Process analysis lets you look at the history of process executions by examining operational meta data. You can answer questions such as “What are the details of the last run of each of these executables?”, “Which parameters were used each of the last three times this process ran?”, and “What information is available about the job that generated this fail event?”

Performing impact analysis

The impact analysis function enables you to answer questions such as “What else will be affected if I make this change?” and “What else is this object dependent upon?” For example, you can find out the effect of deleting a particular transform definition from a particular directory.

This tool provides impact analysis by following relationships within the meta data stored in a directory. The cross-tool support means these relationships can span tools and therefore the whole realm of your data warehouse implementation. For example, you can use this tool to find out not only which DataStage jobs are affected by a change to a routine, but also which CASE diagram is associated with the routine. An example of a cross-tool impact analysis is depicted in Figure 9-21 on page 488.

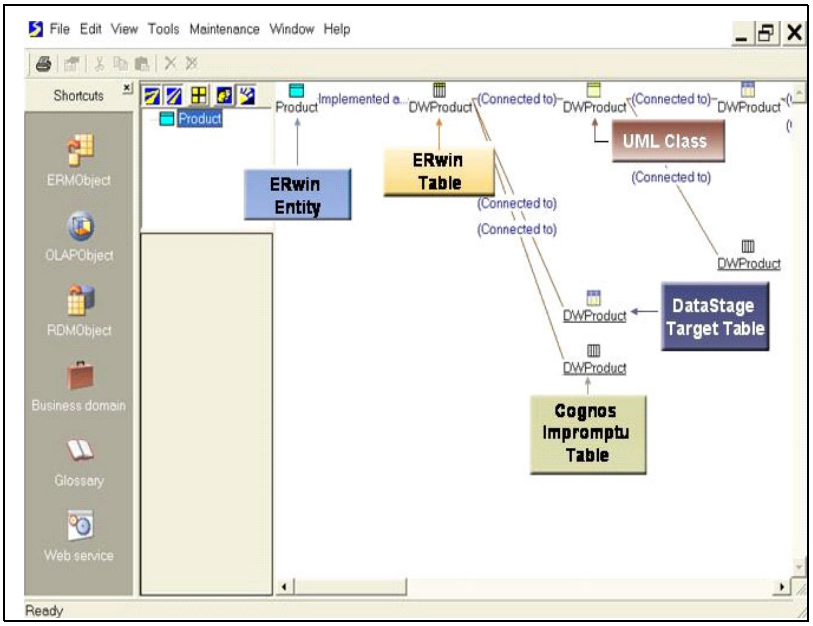


Figure 9-21 Cross-tool impact analysis

Running queries

To fully explore the meta data in a particular directory, you can build your own customized queries using the Query Builder. This lets you answer questions that involve following complex relationships between data warehouse components. The result of the query is a collection of objects, sorted according to your specified criteria. You can refine a search by using the result of one query as the starting point for another.

A query can be saved, then reused to examine different collections of objects. You can also export queries to a text file, from which they can be imported into other tool directories.

Documenting your meta data

This tool lets you almost instantly create complex, hyperlinked reports that show the relationships and attributes of an object or a collection of objects in a particular directory. You can create these reports in HTML, XML, RTF, and text format and print them out or display them in a Web browser, Microsoft Word, or a text editor. An example of the MetaStage Browser is depicted in Figure 9-22 on page 489.

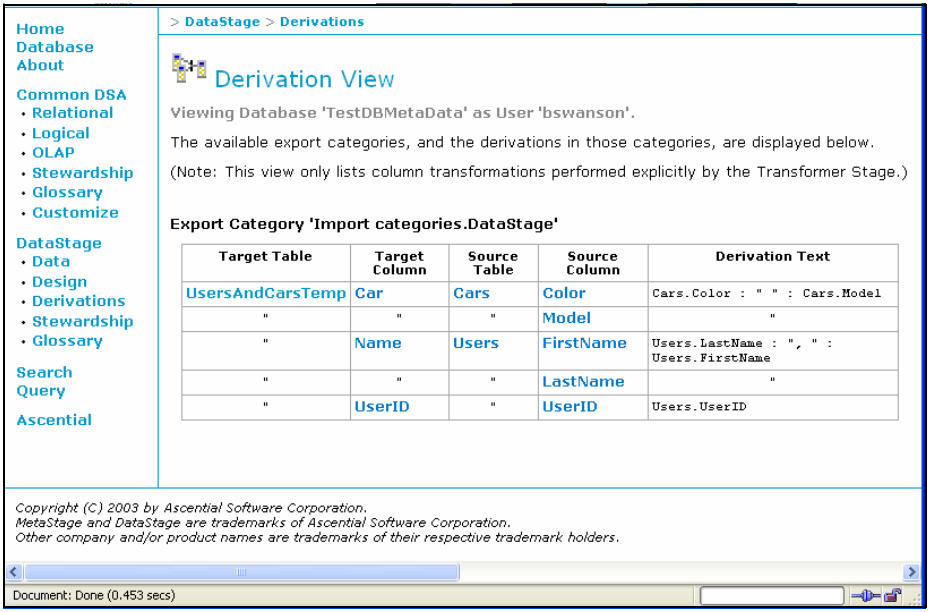


Figure 9-22 MetaStage Browser

MetaStage components

MetaStage consists of client and server components. The primary components, and the flow of the meta data, is depicted in simplified form in Figure 9-23.

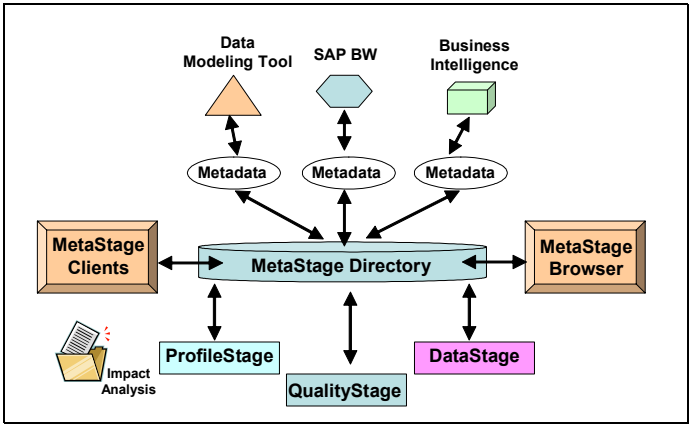


Figure 9-23 MetaStage components

The tool has a client component that lets you browse and search a directory, and provides administration functions. The directory and the Process MetaBroker are

server components. Meta data is added to a particular directory in the following ways:

- Importing meta data from external tools via MetaBrokers
- Using the Process MetaBroker to capture operational meta data generated by data warehousing activities

The sections that follow provide overviews of the MetaStage components.

Directory

This tool holds meta data in a particular directory. You can import static meta data from external tools, such as ERwin or DataStage, and capture operational meta data automatically from data warehousing activities. A directory is a server component. It is built using relational database tables on a Windows® NT or UNIX® server, and is accessed via ODBC. The tool administrator can create more than one directory in a particular system, and users can specify the directory in which their meta data is stored.

Meta data is held in a directory as objects, which are derived from a hierarchy of classes. Objects represent both physical and nonphysical components of the data warehouse. They are cross-linked in a variety of ways, with links being built from imported meta data and from the events that occur when data warehousing activities are executed. The object attributes and relationships enable this tool to provide information about data warehousing processes and assets. Whenever the content of a directory changes significantly, for example as a result of meta data being imported, the tool creates a new version of the directory. Versioning provides a history of directory changes, and lets the tool administrator roll back a directory to a known point if a problem occurs. An administrator can label versions, for example to document the change history of a collection of objects created or modified by particular imports.

MetaStage provides three ways of viewing the structure of objects in a given directory:

- **Relationship hierarchy.** Most objects in a directory, that represent an element in the data warehouse with which you can interact directly, have a variety of potential relationships with other objects. An object can contain other objects, much as a table contains columns, or can depend on another object, much as a column depends on a data element. You can track relationships between objects by browsing this hierarchy of relationships.
- **Class hierarchy.** All the objects in a directory are instances of a class, and many classes are subclasses of other classes. The class hierarchy clarifies what components are in a data warehouse. For example, it can show you a collection of all the computers. The MetaBroker for each external tool has a different class hierarchy— derived from its data model—that determines its

view of the data. The tool itself also has a class hierarchy that encompasses most of the classes in the different MetaBrokers. You can view the objects in the tool directory through the view of the MetaBroker for any tool. The names, attributes, and possible relationships for each object depend on the view. For example, a table object seen in a particular product view, becomes a table definition in the DataStage view and a data collection in the tool view. In this way attributes and relationships an object has can be seen if you export it to another tool.

- **Categories**. This tool uses categories to organize objects in a directory. The category types include:
  - User-defined categories, which let you organize objects in a way that reflects your own view of your data warehousing resources.
  - Import categories, which hold objects imported from external tools.
  - Publication categories, which contain objects that have been made available to export to an external tool.
  - Business Domain and Glossary categories for holding information on meta data objects associated with business terms.
  - Steward categories for designating that an individual or organization is responsible for particular meta data objects.

An example of the category browser is depicted in Figure 9-24 on page 492.

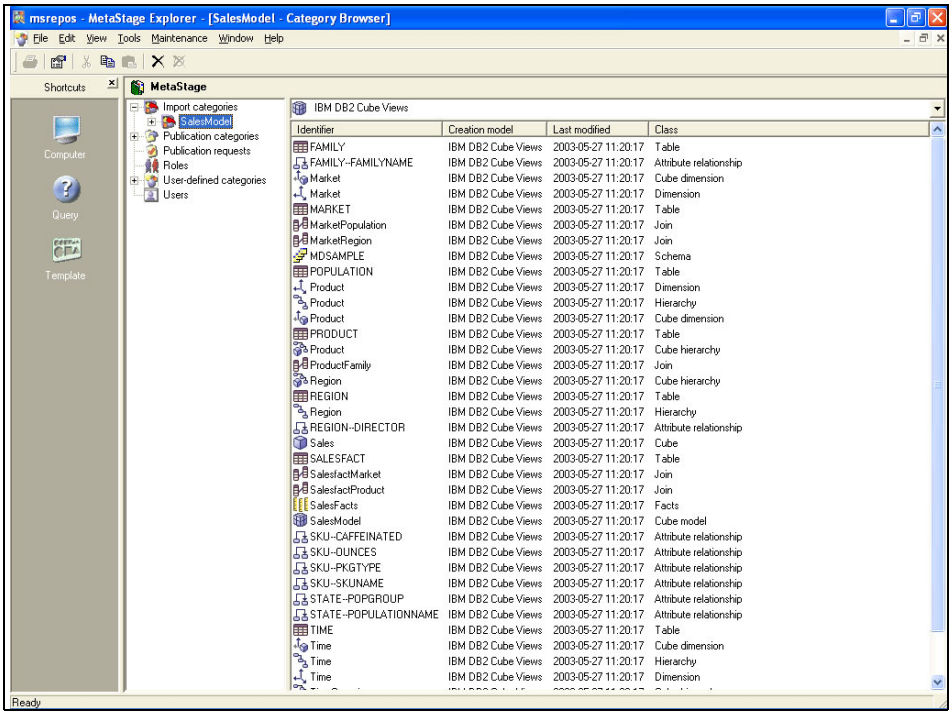


Figure 9-24 MetaStage Explorer: Category browser

MetaBrokers

MetaBrokers provide an interface between a particular directory and the external data warehousing tools. MetaBrokers can also be used with the Ascential tools DataStage and QualityStage.

There is a different MetaBroker for each external tool, or version of that tool. The MetaBrokers are implemented as part of a hub and spoke architecture, in which a MetaBroker and its associated tool are on each spoke and the hub is the directory holding the integrated meta data. For example, this tool can import meta data from Tool A and subsequently export it to Tool B in a form that is meaningful to Tool B. Tool A might be a data warehouse design tool, and Tool B a data extraction and transformation tool. The MetaBroker lets you view data in a directory from the perspective of any installed MetaBroker so the data can be previewed before it is exported to the tool with which that MetaBroker is associated. MetaStage is supplied with a number of MetaBrokers, and others can readily be installed to provide interfaces to additional tools. See Figure 9-25 on page 493.

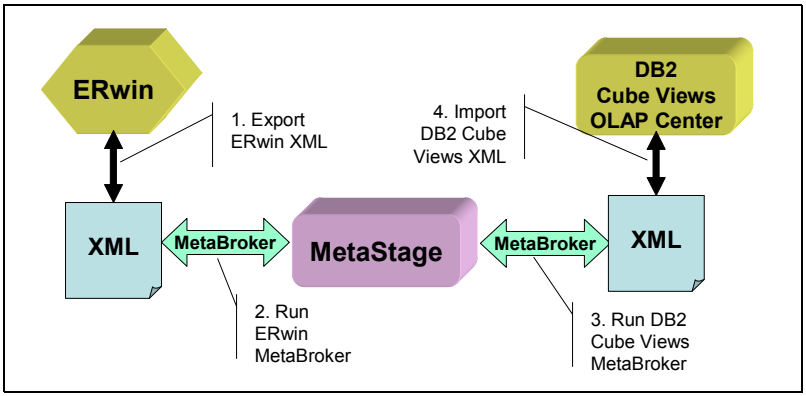


Figure 9-25 MetaBrokers: Meta data flow example

The Publish and Subscribe functions, used with the appropriate MetaBroker, enable movement of meta data from MetaStage to external tools. They can be executed from the Explorer, and provide meta data export functions to enable you to publicize meta data that you want to share with other users. A collection of meta data objects in a directory can be made available to other users by defining it as a publication. A user interested in the meta data in the publication creates a subscription to that publication. The subscription specifies that the user wants to export the meta data through a specified MetaBroker, and perhaps to be informed of any changes to the publication.

The meta data does not have to be exported to the same type of tool from which it was imported. It can be exported to a different tool or to a different version of the same tool as long as there is a MetaBroker for the target tool. In such cases, the target tool has its own view of the published meta data.

DataStage users, for example, might do the following:

1. Import a transform from a DataStage project.
2. Publish the transform.
3. Subscribe to the publication on behalf of another DataStage project.
4. Export the transform from this tool into the project.

The administrator allocates rights to importing and exporting, and so has control over who can share meta data.

MetaArchitect

When you need information from a tool for which a standard MetaBroker does not exist, MetaArchitect® lets you easily create custom MetaBrokers for importing and exporting meta data in CSV or XML format. You can create custom MetaBrokers based on the Common Data Subject Area model, which contains a

full range of OLAP, Entity Relational Model, Relational Data Model, and ETL meta data classes, or is based on the DataStage 7 MetaBroker model, which includes ETL and operational meta data classes. An example of a MetaArchitect is depicted in Figure 9-26.

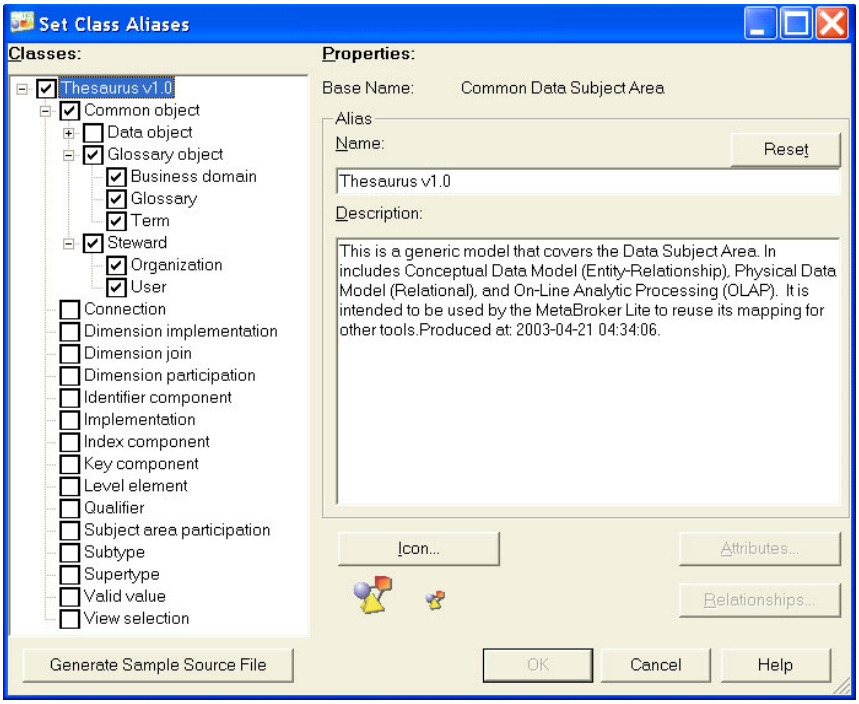


Figure 9-26 MetaArchitect

Process MetaBroker

MetaStage uses a Process MetaBroker to capture operational meta data. A Process MetaBroker can be installed on a UNIX or Windows NT® system.

Typically, data warehouse users may run activities to populate part or all of the data warehouse, or to perform other related tasks such as rollback or postprocessing. The Process MetaBroker keeps track of an activity, gathers operational meta data, and delivers that meta data to a particular directory. The operational meta data includes the times an activity started and ended, its completion status, and the resources that it accessed, such as files and tables.

To capture meta data, users access the Process MetaBroker via a particular proxy. Proxies are UNIX shell scripts or DOS batch files that run data warehousing activities and use the Activity Command Interface to define operational meta data about those activities.

MetaStage contains a template proxy shell script from which you can build customized proxies.

Explorer

The Explorer lets you navigate a particular directory, view the attributes of objects in a directory and, where appropriate, edit the attributes. You can follow the relationships between objects and run queries to explore those relationships in more detail. The Explorer also includes the following components:

- Category Browser, for organizing meta data and launching imports and exports.
- Class Browser, for viewing class hierarchies and overlap between classes and the models for different tools.
- Directory Search tool, which enables you to find objects in a directory that match certain criteria.
- Query Builder, to build more sophisticated queries to locate meta data in the directory.
- MetaArchitect, is for building custom MetaBrokers.
- Object Connector and *connected to* functionality lets you connect objects that are essentially identical but which may have been imported from different tools. For example, a DataStage table definition that was created from an ERwin table.
- The ability to create Data Lineage, Process Analysis, and Impact Analysis paths in order to inspect and analyze the containment structure, dependencies, and sources of each object in the directory.
- *Send to database* functionality, which lets you export meta data to a separate set of tables in the tool directory so you can query these tables using common SQL query tools.
- Online Documentation tool, which lets you create, view, and share detailed documents describing the attributes and relationships of MetaBroker models and objects.

If you have administrative rights, functions in the Explorer let you handle such tasks as configuring the tool options, deleting data from a directory, and defining the access that users have to the tool functions. You can control the interchange of meta data among users by restricting access to the tool functions, and by monitoring user requests to share meta data. You can find examples of Explorer views in Figure 9-19 on page 486 and Figure 9-24 on page 492.

Managing meta data in the data warehouse

Define data once and use it many times is the basic principle of managing meta data. Figure 9-27 illustrates a recommended overall flow of meta data among tools as you build the data warehouse.

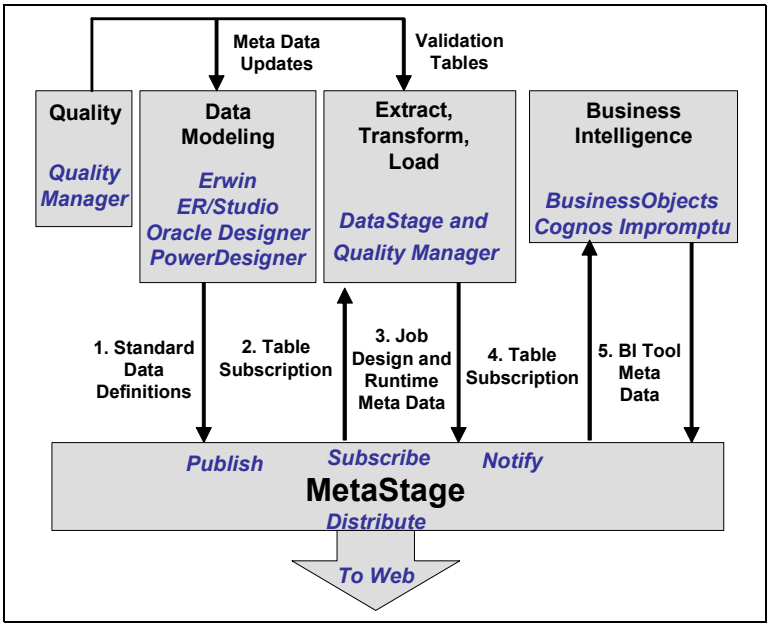


Figure 9-27 Managing meta data

Start with the data model of the databases that will be populated by the ETL (extract, transform, and load) processes. After importing meta data to this tool, publish the table definitions from the physical data models and then run subscriptions to export the table definitions to DataStage.

Load the definitions into the DataStage job stages from the DataStage Manager to preserve meta data continuity once you import the job designs into this tool. You can also export the published table definitions to a business intelligence tool repository, such as a Business Objects Universe or Cognos Impromptu Catalog. If necessary, you can import Universe and Catalog meta data back into MetaStage.

And, to finish, you can publish any collection of meta data—data models, table definitions, impact analysis and data lineage reports— to the Web in a variety of formats.

10

SQL query optimizer: A primer

In this chapter, we discuss a specific Structured Query Language (SQL) and relational database server program capability. In particular, we discuss the role, behavior, and design algorithms (the technology) of query optimizers. This is an important topic, particularly for those who are developing and implementing a data warehousing environment to support your BI initiatives. Why? Because BI is heavily dependent on query and analytic applications, typically operating on huge volumes of data, for supporting data analysis.

We need to understand the optimization process to enable acceptable performance and response times. And, there are other components and technologies involved that also must be understood. As examples, the data warehousing environment and the data models used to define it.

Data warehousing is all about getting your valuable data assets into a data environment (*getting the data in*) that is structured and formatted to make it fast and easy to understand what data is stored there, and to access and use that data. It is specifically designed to be used for decision support and business intelligence initiatives. This is vastly different from the more traditional operational environment that is primarily used to capture data and monitor the execution of the operational processes.

The emphasis with BI is on *getting the data out* of the data warehouse and using it for data analysis. As such, you will typically be working with huge volumes of data. You can *get data out* by using stored applications or ad hoc access. In either case, the technology, and methodology, most used for this purpose is the query.

Therefore, there is an emphasis on developing queries that can get the data out as fast as possible. But, having fast queries is dependent on such things as how and where the data is stored and accessed, which is determined by the data model. And they are dependent on the capabilities of the relational database being used, in the form of the query optimizer.

Though the primary focus of this redbook is on dimensional modeling for BI, we have included information on query optimizers. It is through understanding these two areas, and enabling their integration, that you can develop a robust business intelligence environment.

How do you develop such an environment? You start by understanding dimensional modeling to enable you to create the appropriate data environment. Then you read this chapter to help you understand query optimizers so you can make best use of the data environment. Understanding the optimizer technology is the focus of part one, and in part two, we focus on how to apply it.

In Chapter 11, “Query optimizer applied” on page 599, we discuss the *application* of this technology This is to enable you to perform more accurate data modeling and develop more performant data models. In fact, it can enable you to determine the performance characteristics of your data model before you even create it. This will save significant time, and money, in such activities as data model creation, physical data placement and loading, index creation, and application development. The objective will be to “get the data model right” before building on, and investing in, it.

Not only is it important to know the facts and figures of a given technology, in this case query optimizers, it is also important to know how to work with it. After a brief introduction, we present a real world query optimizer example. This example is presented in a narrative style, and we discuss information as it was discovered during the resolution of an actual customer problem.

10.1 What is a query optimizer

Before we define the term query optimizer, there are other supporting terms that we need to clarify. The term *server* is often used in an imprecise manner. It is used to refer to both a hardware server and software server packages and systems. Examples of software server packages and systems include relational database servers, electronic mail servers, print servers, and Web servers. The IBM pSeries®, formerly known as IBM RS/6000®, is one example of a hardware server.

Process, memory and disk components

Much like any computer program, a software server has a *process component*, a *memory componen*t, and a *disk component*, as depicted in Figure 10-1.

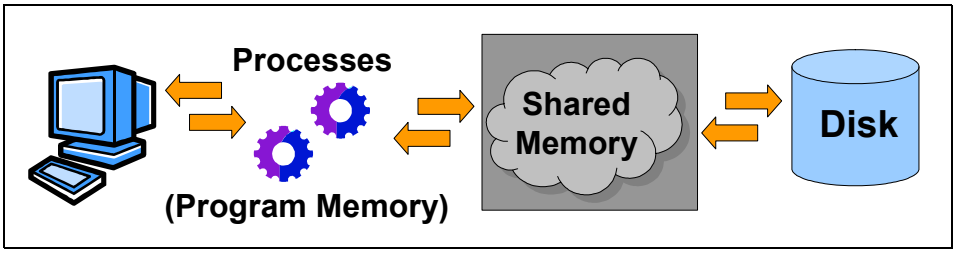


Figure 10-1 Software server architecture

In the case of a relational database (software) server, the components are defined as follows:

- The **disk component**, or disk architecture, refers to those structures that the relational database server software maintains on disk. You might initially think this is the whole point of a relational database server. However, in the case of a relational database server, we are typically discussing entities such as dbspaces, tablespaces, chunks, extents, and pages. The SQL hierarchy of objects, such as SQL tables and SQL indexes, reside inside these disk component structures.
- The term *disk component* refers to that portion of the server which occupies or refers to persistent storage. The term *disk architecture* refers to the design of that portion of the software server.
- The **process component**, or process architecture, refers to the collection of computer programs that execute the workload on behalf of the software server. The two most common relational database process architectures are a two-process architecture, and a multi-threaded process architecture.
  - A relational database server with a **two process architecture** maintains a number of persistent daemon processes which generally listen for new

database connection requests, manage the shared memory structures (the buffer pool), and so on. The phrase *two process* refers to user processes. Each active user has a dedicated back-end server process. If there are 200 users, then there are 200 dedicated database server processes (plus the daemon processes). If eight additional users connect to the server, then you have 208 connections, one connection per user. IBM DB2 UDB and Oracle 9/10g use a two process architecture.

  - A relational database server with a **multi-threaded process architecture** also maintains a number of persistent daemon processes for tasks such as new connections requests and buffer pool management. User activities, however, are handled by a *finite number* of constant and dedicated server processes. Based on the precise database server process architecture, this number of processes might be related more towards the number of (hardware) server CPUs than the number of users. For example, 100 users and four CPUs may be handled by three to four user server processes. However, 200 users and four CPUs may still be handled by three to four user server processes. After all, a hardware server can only execute as many concurrent processes as there are CPUs.The users are represented by threads, in a pool, that are then executed on an available user server process. IBM/Informix IDS, IBM/Informix XPS, and Sybase use multi-threaded process architectures.
  - IBM/Informix IDS and IBM/Informix XPS specifically use a non-blocking, pipelined parallel, multi-threaded process architecture. This means that the server processes dedicated to user activities do not call for program interrupts and can execute with 100% (or very nearly 100%) efficiency, in other words, the non-blocking part. The pipelined part means that one thread can send work to the next thread to perform activity asynchronously.

- The **memory component**, or memory architecture, refers to those structures that the software maintains in memory. Since servers can execute operations in parallel, this memory is most often shared memory, or, memory that is available to numerous and concurrent server processes. Shared memory is the conduit between processes, and between process and disk.

Note: You might wonder why all of this attention to memory and processes architectures in a data modeling book. The reason is performance. You could create the most accurate and elegant data model possible, but if the supported business application system does not perform adequately then the project may not be successful. One other point to consider is that modern relational database servers observe read cache ratios in the high 90% range. This means that 95 to 98 times out of 100 that a data page needs to be read, the page is already located in a buffer pool and does not need to be retrieved from disk. Modern parallel database servers, in addition to being disk management systems, are CPU and data calculation engines.

The relational database server multi-stage back-end

Given all of this, what is a query optimizer? A query optimizer is a subsystem in the process architecture of a relational database server. Most relational database servers employ a multi-stage back-end, as shown in Figure 10-2.

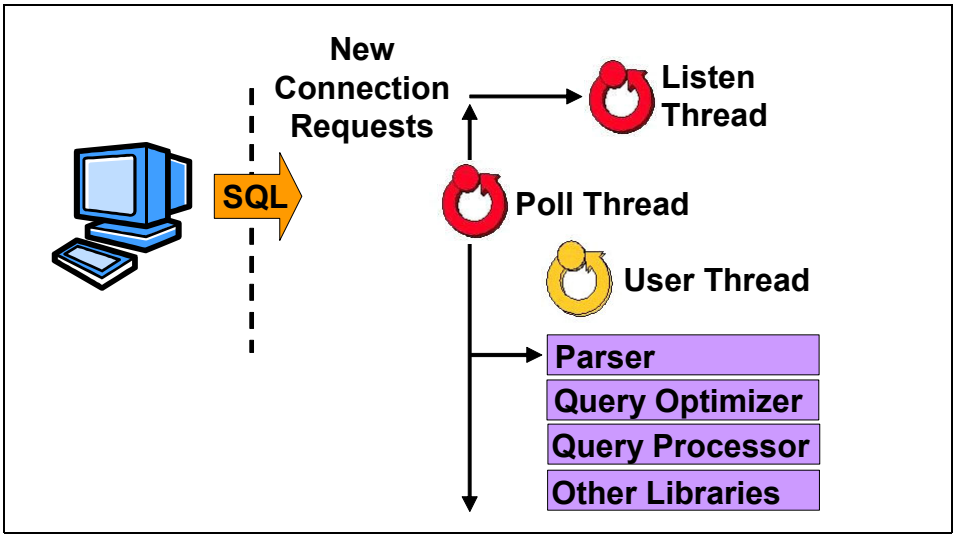


Figure 10-2 Relational database server, multistage back-end

The relational database server multistage back-end is delineated in Figure 10-2 by the four boxes entitled; Parser, Query Optimizer, Query Processor, and Other Libraries. While a given relational database vendor may state that they have a four stage back-end or a seven stage back-end, each database is performing the same work regardless of how these stages are drawn into various boxes.

The work performed in the multistage back-end is detailed in the following list:

- **Parser:** Typically, the first stage in the back-end is a command parser. Each inbound communication request received from the user, and each request for SQL service, must be identified. The inbound command is read character by character until it is known to be, for example, an SQL CREATE TABLE request or an SQL SELECT (table) request. This parser may also be associated with an inbound SQL statement command cache, as a possible source of performance tuning.

  There are two types of SQL commands that are observed by the parser, those that need to examine existing data records in some table and those that do not. Examples of SQL statements that do not examine existing data records include CREATE TABLE, INSERT (new record), and CREATE SCHEMA. The three most common examples of SQL statements that examine existing data records are the SELECT (zero or more records), UPDATE (zero or more records), and DELETE (zero or more records).

  SQL statements that need to examine existing data records move on to the next stage in the multi-stage back-end, the query optimizer. SQL statements that do not need to examine existing records can skip the query optimizer and query processor stages. In the diagram depicted in Figure 10-2 on page 501, this is the area entitled “Other Libraries”.

- **Query optimizer:** Inbound communication statements, requests for SQL service, that arrive here in the multi-stage back-end are those that are somehow examining existing records in a single or set of SQL tables. SQL SELECT statements read data records, and their need to examine existing records is obvious. SQL UPDATE and SQL DELETE statements also read, but they are reading before the expected writing, so that they may find the records they are to UPDATE or DELETE. After UPDATE and DELETE find (read) the records they are to process, they write.

  In each of these cases, the relational database server automatically determines how to access and locate the records that need to be read. This means that the invocation and use of the query optimizer is automatic.

  Note: The query optimizer is a relational database server subsystem, residing in the process architecture, that automatically determines the most optimal method to retrieve and assemble the data records that are required to process a request for SQL service.

  Part of what the query optimizer determines is the *(table) access method* per each SQL table involved in the request. That is, should the given table be read sequentially or via an available index? In a multi-table query, the query optimizer also determines the table order and table join method to be used. The product of the query optimizer stage is a query plan. This is the plan for executing the request for SQL service that is being received and processed.

- **Query processor:** The query processor executes the query plan that was created by the query optimizer. In some advanced relational database servers, the query plan is dynamic and may change as the query processor detects errors in the query plan, or the opportunity for improvement.

  The query processor may also make other run-time decisions. For example, if a given subset of data records is too large to sort in the available memory, these records may be written to a temporary table which is used as a form of virtual or extended memory.

  While the query optimizer and query processor are two distinct stages in the multi-stage back-end, you can treat them as one logical entity, which is what we do in this chapter.

- **Other Libraries:** If nothing else, the remaining stages of the relational database server multi-stage back-end contain the shared libraries and software product code. These designs are often proprietary and there is little value in the current context to offer or review any particular library structure.

10.2 Query optimizer by example

At this point in this chapter, we have only defined the term query optimizer. We have not discussed much or anything about how a query optimizer behaves. Still, we are now going to review a real world example involving a query optimizer. In this real world example, a customer is migrating from a proprietary hardware and software system and is observing poor (unacceptable) relational database server performance. Why discuss a technology in an example without first understanding the technology?

- It has been proven that persons retain new information better, and with better comprehension, when this information is presented in narrative versus statistical presentation. That is, you learn better *by example*.
- The value of query optimizer knowledge is in the application of this knowledge, so you need to know how to use this information. Therefore, we present query optimizer information *by example*.

10.2.1 Background

Note: This example is real. The business description, use, and intent of the tables have been changed to protect the privacy of the actual customer. This example uses three SQL tables. The described purpose of these three tables *may be variably changed* in the example text to make specific points.

The real world example we are reviewing happens in the time just before Internet applications. This company has a toll free call center, where their customers can place new sales orders via the telephone after reading from a printed catalog to determine what they want. A human sales agent listens on the phone, interacts with the customer, accesses various parts of a business application to gather data for the new sales order form, and generally works to complete and then place the order. Approximately 70 customer sales agents are active on the system, 10-12 warehouse workers are using the system to perform order fulfillment, and two or so additional users are running reports.

The proprietary hardware and software system is being replaced to reduce cost. A new hardware and software system has been purchased from an application vendor who has focus on (sells) this type of retail sales application. This packaged application had been modified to a small extent to meet whatever specialized needs the given company required. The new hardware and software system was many times the size of the older proprietary hardware and software system, with many years of capacity planned for growth.

After months of planning and modification, sales agent training, and so on, a weekend cutover was used to move data from the old system to the new. On Monday morning, the new system was fully in use for the first time. After the first few new sales orders were received, it became painfully obvious that the new system was performing very poorly. If the customer on the phone asked to buy a computer peripheral, for example a mouse, the drop down list box offering the various mice for sale would take minutes to populate with data, where it needed to be populated in under a second.

After three to four hours of panic, the problem is assumed to be caused by the new relational database. The relational database software vendor is called to identify and repair the defect in the database product that is causing this poor performance.

10.2.2 The environment

The problem observed seems to be one of performance. At this point, it is not known whether the source of performance is the (hardware) server, the network, the operating system, the (user) business application itself, the relational database server, unreasonable user expectations, or some other source. Here we take the role of the database vendor. Since we have already been called in, we begin to investigate the database server.

Note: There are two areas of relational database server tuning. One involves tuning the relational database server proper, its processes, memory and disk, and the other tuning the user (SQL programming) statements that run inside that relational database server. Tuning user (SQL programming) statements could mean tuning the actual SQL commands, or the run time (data model and indexes) that supports execution of these commands.

Tuning SQL database servers

As mentioned, there is tuning the actual relational database server, and then there is tuning the user SQL statements that operate within the relational database server. Tuning the processes, memory, and disk of a relational database server is a relatively well known procedure. In our example, a database administrator with six months of experience could validate the performance tuning of a single (non-clustered, non-distributed) relational database server system with 500 GB of disk storage and 8 CPUs (just to give you an idea of the size of the system), in well under two hours of work. Among other things, you check the following:

- The read and write buffer pool cache ratios.
- The disk operation request queues, wait times, and high water marks.
- Disk layout. Is the disk I/O activity balanced across devices? Are data segments nice and contiguous or highly fragmented? and other.
- Data server events; Is activity being held or delayed for tape backup operations or related activities?
- User status lists; Are users waiting on locks or other resources?
- A few sanity checks; Can we connect to the database server and select data with reasonable performance?
- Other procedures, as determined by your relational database vendor.

Managing customer expectations

After basic checking to review adequacy of the relational database server tuning, there does not appear to be a relational database server tuning problem. The buffer pool cache is fine, all other observed statistics are fine, yet the user is getting performance that is unacceptable. In the actual example we reviewed, there were many barriers that prevented the reviewing engineer from diagnosing this problem in an efficient manner.

As examples of these barriers, there was a undue amount of tension in the work area, and the environment discouraged rational thought. Due to these pressures, the customer was asking for the problem to be immediately solved, and asking for this to become a training event for his personnel. Even before the source of

the problem had been identified, the customer was looking to assign blame. The following is a list of actions that may aid in such an environment:

1. Information Technology (IT) engineers sometimes rush off and begin solving the wrong problem. Repeat the observations and assumptions back to the customer. For example; “If I can make the performance of this subroutine go from five minutes to two seconds, would that solve your problem? Is this the task you would like me to work on first?”
2. Every IT issue has three elements; technical, business, and emotional. Technical elements are those such as, “What protocol do we use to communicate between a client application and the server?” Business elements are those such as, “How much will this solution cost and what is the expected return on investment?” Emotional elements are those such as, “Bob is stupid and should not be trusted to solve this problem.”We recommend that you use these three indexes in communications with the customer to help quantify and prioritize the tasks that need to be completed. In our actual case, we chose to focus on the technical challenge of identifying the source of the performance problem.
3. Work to set reasonable expectations. In this actual case, it is paramount that we solve the performance problem as quickly as possible. In most cases, performing training while solving the problem will slow your resource down by a factor of three of four. So do not use this time for training, that is not the primary goal. The primary goal at this time is solving the performance problem quickly.
4. In our actual case, there was no test system on which samples or trial solutions could be explored. All trial solutions had to be performed on the live, production system. We would lose credibility if we made the problem worse or brought the production system down. Advise the customer of the additional challenges these various conditions cause, and ask for permission to proceed as the current environment best allows. In our actual case, we created a second database within the production server so that we could partially isolate the impact of our investigations from the production system. The database server software allowed for this option.

10.2.3 Problem identification and decomposition

At this point in the example, we have:

- Performed a basic review of the database server tunable parameters and found no obvious issues. We did this work with little or no prompting because the task was easy, and is a likely source of error, capable of producing a quick solution or return.
- Worked to start confirming assumptions, asking for and then setting priorities, and setting customer expectations.

With more attention and detail than before, we ask the customer to demonstrate the problem. In our actual case, an operation performed inside the business application is to populate a drop-down list box (a specific graphical control within a data entry screen form) with data. The customer sales agents logged in to the database with distinct user names. This made identifying and tracing the specific SQL commands being executed by a given user and at a specific time, an easy task. When users log on to the database using a connection concentrator, or database session pooling mechanism, this task becomes much more difficult. When pooling is in effect, distinctly named users are recognized by the database server software by a single or smaller set of shared names. Figure 10-3 displays some of the information that is discovered at this time.

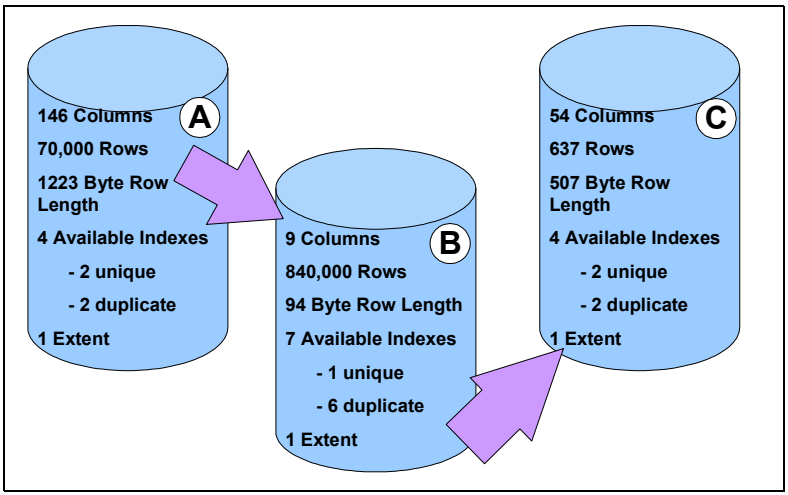


Figure 10-3 Results of investigation at this time

Gather statistics on tables involved in the query

By using the database server software diagnostic utilities, we confirmed that opening the graphical user application program drop-down list box executes a three table SQL SELECT statement. We captured the specific SQL SELECT statement syntax. Then we executed the same SQL SELECT statement on a non-graphical, non-networked, terminal window and observed the same poor performance. This was a significant achievement. We removed many variables from the test case, and captured a repeatable test. Our test produced the same data in the same amount of time as the users. Add the following:

- The actual table names and column names will be obscured from this example. The actual table and column names are those such as “dsxggprtst11”. Names that made no sense to a person who has no experience with this application.

- The number of records (rows), columns, indexes, and physical disk space allocations (extents) is displayed in Figure 10-3 on page 507. This is all data we were able to discover, given the SQL SELECT statement, the table names, and other information.
- The query returned about 700 records in an average of three to five minutes, which equals two to three records a second. Given the small table sizes, two to three records a second is very poor performance.

Note: *Page Corners*. All relational database vendors have the concept of page corners, although the specific term used by a vendor for this condition may vary. All vendors recognize some term which represents the smallest unit of I/O they will perform when moving data to and from memory and disk. We call this unit a *page*. Generally, a vendor will place as many whole data records on one page as will fit. The data record has a length in bytes, the page has a length in bytes, and only so many whole records can fit on a page. The vendor will not unnecessarily split a single record across two pages to make maximum use of (disk) space. Whatever space is left unused due to this condition is called the *page corner*. Why not split data records across pages unnecessarily? That would subsequently require two physical I/O operations to retrieve a record when only one could have sufficed. The answer is to preserve system performance (over disk consumption).

In an SQL table with fixed length records (no variable length columns), the page corner will be the same on all full pages. We say full pages, because a page with unoccupied record locations will have additional unused space.

Figure 10-3 shows that table-A has a record length of 1223 bytes. If the page size for this system is 2 KB (2048 bytes), then 800 or so bytes are left unused on every data page allocated for that table. If there were 300 or so fewer commonly used bytes in a table-A record, we could possibly gain some performance by expelling those bytes to a second table, and observing more table-A records per page on the more commonly used columns. This physical data modeling optimization is generally referred to as *expelling long strings*.

Variable length records have issues too. Some vendors will leave as much free space on a data page to allow for any single data record on the page to achieve its maximum length. If the page size is 2 KB, and the maximum length of a given record is also 2 KB, you may get one data record per page even if the average variable record length is only 80 bytes.

Reading a text presentation of a query plan

Most relational database software includes graphical and/or character-based administrative interfaces to retrieve the query plan of a given SQL statement.

Example 10-1 displays the query plan from the current three table SQL SELECT that is being reviewed.

Example 10-1 Character-based administrative interface showing this query plan

```
0001
0002 --  This file has been edited for clarity.
0003
0004 QUERY:
0005 ------
0006 select
0007       ta.*, tb.col001, tb.col003, tc.*
0008    from
0009       table_a ta,
0010       table_b tb,
0011       table_c tc
0012    where
0013       ta.col001 = “01”
0014    and
0015       ta.col002 = tb.col001
0016    and
0017       tb.col003 = “555”
0018    and
0019       tb.col003 = tc.col001
0020
0021 OPTIMZER DIRECTIVES FOLLOWED:
0022
0023 OPTIMZER DIRECTIVES NOT FOLLOWED:
0024
0025 Estimated Cost: 407851
0026 Estimated # of Rows Returned: 954
0027 Temporary Files Required For:
0028
0029   1) Abigail.ta: INDEX PATH
0030
0031     (1) Index Keys: col001 col002 (Serial, fragments: ALL)
0032         Lower Index Filter: Abigail.ta.col001 = ‘01’
0033
0034   2) Abigail.tb: INDEX PATH
0035
0036         Filters: Abigail.tb.col003 = ‘555’
0037
0038     (1) Index Keys: col001 col002 col003 (Key-Only)
             (Serial, fragments: ALL)
0039         Lower Index Filter: Abigail.ta.col002 = Abigail.tb.col001
0040 NESTED LOOP JOIN
0041
0042   3) Abigail.tc: INDEX PATH
0043
```

```
0044     (1) Index Keys: col001 (Serial, fragments: ALL)
0045         Lower Index Filter: Abigail.tb.col003 = Abigail.tc.col001
0046 NESTED LOOP JOIN
0047
```

By reading the query plan displayed in Example 10-1 on page 509, we can see the following:

- Lines 006-0019; The SQL SELECT statement, which we captured earlier, is repeated.
  - table_a.col002 joins with table_b.col001, line 0015.
  - table_b.col003 joins with table_c.col001, line 0019.
  - All three tables join, A with B, and then B with C. That is good because there are no outer Cartesian products, a condition where some tables are not joined.

Note: It is hard to find a real world application of an outer Cartesian product, since this results in a costly projection of one data set onto another without relation or care. For example, an outer Cartesian product of the Valid-Instructors table onto Valid-Courses table returns Instructors with Courses they do not have any relation to. This is a good way to generate voluminous data that is of little use. Bob cannot teach the Home Economics course, because Bob is a Computer Software Instructor.

Outer Cartesian products *can* exist for valid reasons, but their presence is extremely, extremely rare. They are so rare, that their presence should be considered a possible indicator of an error in the data model. And, a relational database product could show an application error upon receiving an outer Cartesian product. Outer Cartesian products generally occur between two tables, and are *optimized* by creating a third table to act as an intersection. For example, a Courses-Taught-by-these-Instructors table indicating which Instructors can teach which Courses, from the example in this block.

  - Lines 0029, 0034, and the 0042 give us the **table join order**; that is, that table A will be processed first, then table B, then C.
  - Tables A and B both have filter criteria. For example, WHERE Ship_date equals TODAY, WHERE Customer_num = ‘100’.
  - The table A filter is displayed on line 0013. The table B filter is displayed on line 0017. Both filters are based on equalities.
  - Filters and/or joins could be based on equalities, ranges, or other set operands (strings and wildcard meta-characters for one example). A range criteria would be similar to: WHERE Ship_date > “09/02/1974” AND

Ship_date < “09/09/1974”. Generally, equalities are preferred over ranges because fewer data records satisfy the criteria, and therefore fewer records are processed.

- Lines 0021-0023; **query optimizer directives**.
  - Be default, the query optimizer uses the algorithms it contains, and certain information about the tables and resultant data sets involved to form the query plan. This work is automatic.
  - Optionally, the relational database vendor may offer the ability to override or influence query plans by a technology generally referred to as *(query) optimizer directives*, or (*query) optimizer hints*. Some vendors offer the ability to promote and/or discourage certain query paths, other vendors can only promote or discourage, but not both. Optimizer hints are used during those few occasions when the automatically determined query plan just does not perform, and a human operator can specify a better plan.
  - Generally, optimizer directives are specified by embedding specifically formatted strings inside the SQL command text.
  - The topic of optimizer directives is expanded upon in “Query optimizer directives (hints)” on page 584.
  - No query optimizer directives were in effect for the execution of this query, as you can see by the absence of observed text between lines 0021-0023.
- Lines 0025-0026 display the estimated cost and number of rows (records) returned. Generally, if the cost goes down while you are tuning a specific single SQL statement, that is good. It is often less productive to try and compare costs between differing SQL statements or attempt to calculate expected run times from this data.
- Line 0027; **temporary objects**.
  - Generally, each relational database vendor has two categories of temporary objects. Here we define these objects to be temporary tables and temporary files.
  - We define *temporary tables* to be those temporary structures which are *explicitly* requested. For example, CREATE TEMPORARY TABLE, or SELECT ... INTO TEMP TABLE.
  - Temporary tables tend to be private in scope, meaning they are viewable (readable, writable) within the single user session which created the object. Temporary tables are generally released when the user session is terminated (closed), but the capability of a specific relational database vendor may vary here.
  - We define *temporary files* to be those temporary structures which are *not explicitly requested.* The most common use of temporary files tends to be for sorting. For example, a given user session needs to sort 1 GB of data,

but is only allowed to use 5 MB of memory, so data overflow will occur to some sort of non-memory related structure. That non-memory structure has to be a disk space allocation. We are calling these allocations temporary files, although your specific vendor's name for this resource may vary.

  - Sorting data is generally the cause of temporary files (implicit disk space allocation used for query processing).
  - Sorting inside an SQL statement can occur for many reasons.

- Line 0029 lists the first table to be processed in the execution of this query, table A. Table B is second, line 0034, and table C is last, line 0042. This condition is referred to as the **table order**.
- Line 0031 details how table A is accessed, and this data is referred to as the *table access method (for table A)*.
  - The two most common **table access methods** are sequential scan and indexed retrieval.
  - The text "Index keys: col001 col002" means that this table is being accessed by a SQL index which exists on columns col001, col002 in this table. This index happens to prevent duplicate key values, as detailed in Figure 10-3 on page 507, above.
  - Database servers can perform operations using multiple concurrent processes (threads) or not. Non-parallel operations are called serial operations. Access of this table, table A, is done serially as denoted by the presence of the "Serial" keyword on line 0031.
  - **Parallel operations** are not automatically faster than serial operations. In the case of this query, the data sets as so small (as determined by the small records counts), it is likely this query would not run faster if executed in parallel. This could however be an area to research in the solution of this problem.
  - Relational database vendors can also physically place records on the hard disk via a more optimal algorithm than random placement. This specific vendor can create distinct, pre-organized allocations of data records by key value and other schemes. This vendor calls these individual physical allocations fragments. **Fragment elimination** is a term for the nearly zero cost ability to reduce the effective physical size of the table, by determining that given fragments do not satisfy the query criteria by the rules in effect for the storage of that table. For example, given a list of world citizens organized by continent where each continent's list is stored on a separate disk (fragment), we need only examine one of seven fragments if we know the records we need reside in the Czech Republic (Europe). The database server knows these constraints without having to

examine actual data records. The database server knows this by the rules it maintains.

> **Note:** The term *query criteria* refers to those conditions in the WHERE CLAUSE of an SQL statement, both the joins and the filters.

  - Line 0013 contains the text, “fragments: ALL”, which means that organization of data for the table did not make use of this feature, or that no fragments could be, or needed to be, eliminated based on the query criteria.
  - Indexes contain key columns, the members or columns which reside in that index. Indexes contain data that is presorted in ASCII collation sequence, for example, alphabetically or in number sort order. The data in the table is generally assumed to be in random sequence. Indexes are always presorted based on an algorithm determined by the index type.
  - Indexes may be read in ascending or descending order. Line 0032 states that this index is to be read in ascending order. That specific point is of little value.
  - Line 0032 also states that this index is being read in order to satisfy a filter criteria equal to “ta.col001 = ‘01’”.
  - From the data in Figure 10-3 on page 507, we know that the SQL table column ta.col001 is of data type CHAR(02). Generally, relational database servers join or evaluate strings (character columns) by performing two byte character comparisons for the length of the string. This operation is measurably less CPU efficient than say, joining integer values. If ta.col001 can be restricted to containing integer data over character, a degree of performance could be gained when performing certain operations. Having to join character data versus integer data adds to the **join cost**.

- Line 0034 begins the description as to how table B is accessed, also a table access method. This next list of subpoints mentions only new technologies, those which were not discussed when reviewing table A.
  - Line 0038 contains the text, “**Key-only**”, and this refers to a modifier as to how the given index on table B is being used.
  - Generally, indexes are read to find an index key value (Customer_num = ‘101’, for example). This single entry in the index data structure also contains the physical address on disk of the associated full data record in the SQL table proper, (where the remaining columns of this full record reside for Customer_num 101). In most cases, the index is read to find the address of the remainder of the data record, and then the full data record is read.

- The term key-only means the query processor did not need to perform this work. All of the columns needed for the given task were members of the given index, the query processor does not need to go to the table proper to get (other) columns.
- Obviously key-only (index) retrievals are more efficient than standard index retrievals, as less work needs to be performed. The work in question is a reduced amount of physical or logical disk I/O.

- Line 0040 contains the text “NESTED LOOP JOIN”.
  - Thus far we have discussed table access methods and said there is a sequential scan disk access method, and an indexed table access method. There are also other table access methods, discussed in “All table access methods” on page 537.
  - A *table access method* details how the given records contained within *one SQL table* are processed.
  - Line 0040 displays the **table join method**. A *table join method* details how records from *one table are then matched with their counterparts in a second, or subsequent table.*
  - From our current example, and in a nested loop (table) join method, table A is read via an indexed table access method. Although it could have been indexed or sequential, it does not impact the table join method in this context. As each row is read via the index on table A, the filter criteria is applied (line 0013). As each row passes this filter (test), the row is then matched to any and all rows that match in table B, based on the join criteria line 0015, however table B is being accessed.
  - The nested loop table join method gets its name from the software language programming construct which is used to create this program code, namely a recursive (nested) loop.
  - Table A is joined to table B using a nested loop join method, as stated on line 0040. Table B is joined to table C also using a nested loop join method, as stated on line 0046.
  - Generally, a given relational database vendor has three or more table join methods. Nested loop join method is the oldest, and under the proper conditions, the fastest. Sometimes nested loop is not the fastest, which is documented later in “All table join methods” on page 541.
- Line 0042 begins the listing for table C. No new technology is displayed after line 0042.
- To summarize, here is the primary data we need to track:
  - The tables are processed in the order of table A, then B, then C.

– Tables A and C are accessed (table access method) via a unique index scan. That is, records in tables A and C are located via a unique index and an equality in the SQL SELECT WHERE clause.

– Table B is also accessed via an index, although that index is an index that permits duplicate values. As a modifier, the indexed retrieval into table B is *key-only.* A key-only retrieval is one where all of the necessary columns (all of the columns we are being asked to read) are members (elements, attributes) of the given index. Normally, an index retrieval accesses the key columns of an index, in order that we may locate the full width data record in the table proper. A key-only retrieval has all of the columns it needs in the index itself, thus avoiding a table lookup.

Create a query plan diagram

Figure 10-4 provides a graphical depiction of the query plan for this example. Only the most important elements of this query are displayed. It would be an important skill to be able to draw a diagram of this type from your specific relational database server query plan display.

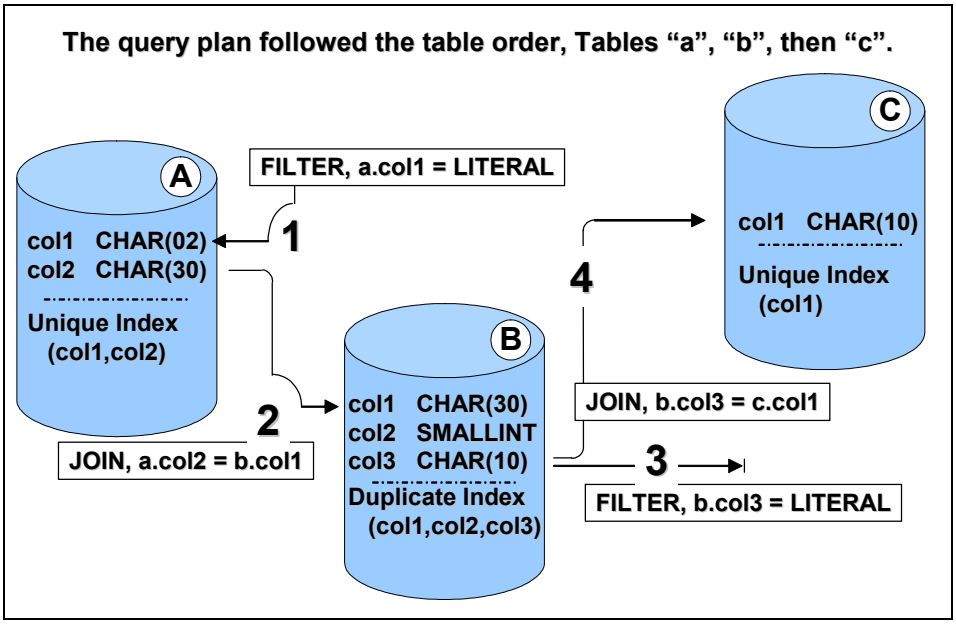


Figure 10-4 Graphical depiction of query plan

Thus far we have determined:

- The query has three tables of small size. (Finding the query through administrative tools, reading the tables’ sizes, specific structure, and disk space allocations through administrative tools.)

- All three tables are joined. (Reviewing the syntax of the SQL SELECT.)
- All three tables are accessed using an indexed retrieval. (Reviewing the query plan, specifically the table access methods.)

Note: We could check to ensure that these indexes are not corrupt, by a number of means. Generally, corrupt indexes do not occur in modern relational database server systems. Most commonly, corrupt indexes give incorrect results, or never return from processing. Corrupt indexes do not display poor performance and correct results. One easy way to check to see whether or not an index is corrupt is to perform a specific query using a sequential (non-indexed) retrieval. Then compare these results to those using an indexed retrieval.

- We see that there are not temporary files or tables being used, which can delay query processing. Absence of these structures observed from the query plan.
- We have executed the query in a simple manner (from a character-based terminal program). We have observed the server returns on average 700 rows in three to five minutes, in other words, two rows per second, on average, from very small tables.
- We have already eliminated the likelihood that slow network traffic is to blame. If there were remote SQL tables, we would copy all of the tables to a single, local database server and repeat the perform tests. (The tables were small and we have available disk.) Also, the test was, as stated above, run from the hardware server console.
- We have already eliminated the likelihood there is multiuser delay due to locks, lock waits, or dead locks, by viewing the database server administrative status reports.
- We still think we can do better.

Note: When you have something that is not working, remove conditions from the test case until something changes. In our case, we have a three table SQL SELECT that is not performing adequately. The third table, table C, provides little value. For experimentation only, remove table C from the query and see what happens.

10.2.4 Experiment with the problem

At this point we are going to experiment with conditions in the run-time environment, specifically those called for in the syntax of the SQL SELECT statement. Because this is an operational and production system, and because

these tables are small (we have the available disk space), we create a copy of these three tables somewhere outside the current database. (The server product allowed for numerous and concurrently operating databases inside one software server. Before proceeding, we re-execute the test case (the SQL SELECT) and compare data results, query plans, and execution times to help indicate that we copied this example run-time environment correctly.

Drop criteria from the query

The current problem is that a small three table SQL SELECT seems to be performing below expectations. We do not know what is causing the problem, so we are going to change the SQL SELECT statement and see what happens. Table C is joined to the products of table A and B, via an equality on a single column that has a unique index placed upon it. Table C is processed last and has only 600-700 records. As a result, table C will be removed from the query for the purpose of research. If we cannot get the two table SQL SELECT of just tables A and B to perform adequately, then we have little hope of making the original three table SQL SELECT perform.

Figure 10-4 on page 515 shows the query plan as it currently executes, and with the final criteria it needs to logically contain. Figure 10-5 displays the query plan after table C was dropped for investigatory purposes.

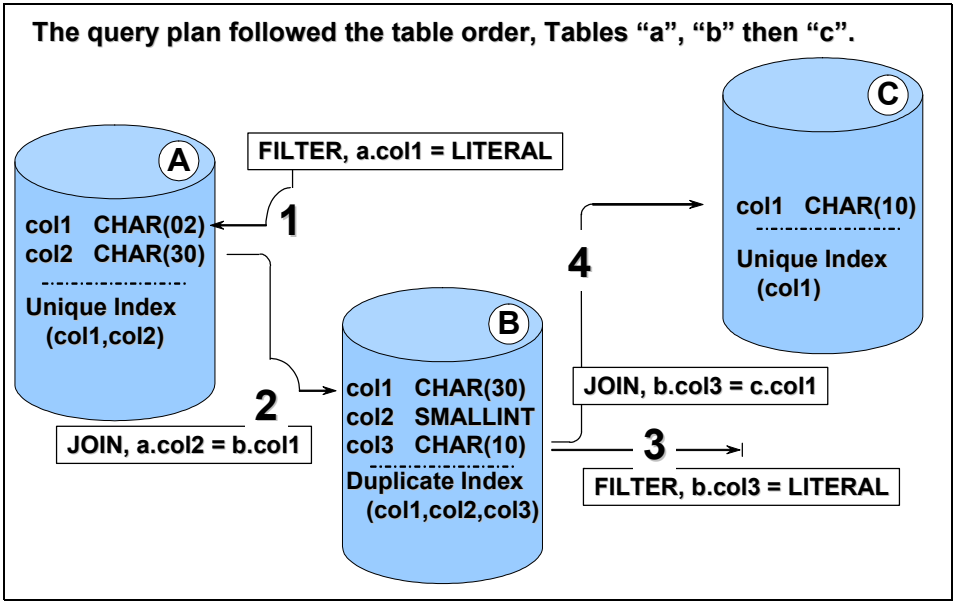


Figure 10-5 Altered query, just tables A and B

As expected, the following was observed:

- Table C is truly low cost. The total query execution time went from an average of five minutes, to four minutes and 40-50 seconds. This is not a significant change, considering the customer needs sub-second performance.
- The table order remained table A then table B. This may also be useful. If the table order changed to table B first then table A, and also produced an acceptable performance result, then we may have trouble recreating this behavior when we reattach table C to the query. One could join table B to table A, then join this product to table C. With several (ten or more) tables and several discontiguous table join orders, this may encourage the implicit use of temporary files which would negatively impact performance.
- What we have now is a two table SQL SELECT that does not perform adequately, perhaps a simpler problem to research.

Measure filter selectivity

With the more simple two table SQL SELECT, we are going to measure the tables separately and then measure them together. From the data presented in Figure 10-3 on page 507, we already have some knowledge of table A. Figure 10-6 displays what is learned when we measure the selectivity of the filter on table A.

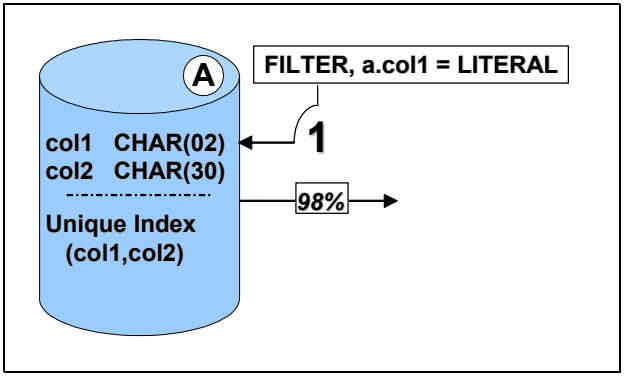


Figure 10-6 Measuring just the filter on table A

Table A is listed as having 70,000 rows. This can be confirmed by many means including an SQL SELECT similar to:

```
SELECT COUNT(*) FROM table_a;
```

Figure 10-6 displays what we learned from executing a SQL SELECT equal to:

```
SELECT COUNT(*) FROM table_a WHERE col001 = '01';
```

Basically, we just copy the predicates (filters) which are in effect on table A, and change the column list to perform a count. This query informs us that the single

filter on table A is not very selective, returning on average 98% of the entire contents of table A.

Index utilization guidelines

As we mentioned previously, part of what the query optimizer determines is table order. In this example, should the query processor read table A first, then B, then C, or perhaps tables C, then B, then A, or another choice? Part of what the query optimizer considers when determining table order is selectivity of the filters. Table order is affected by many things. However, in the area of just selectivity of filters, read the following:

- Here we are using the example of a telephone book, a sorted listing of persons. It contains last name, first name, gender, their telephone number, and perhaps other data.
- One can sequentially read the telephone book, page after page, to find the entry needed. This is a sequential scan, one of the table access methods.

Note: In this context, *index utilization guidelines* does not refer to when the data modeler should use (create) indexes. Index utilization guidelines refer to the conditions that allow for or prevent the query optimizer from using indexes.

- Data in the phone book is sorted, and for this example we will imagine this condition of having the presorted data to be our index. Most phone books are sorted by last name, then first name, and so on. For this example, we will state that the phone book can only be sorted on one column of data.
- If the phone book is sorted on last name, but not any other column (because we are imposing a single column sort order limit), we would locate a male named “Bob Jones” in a manner similar to:
  - This phone book is many hundreds of pages thick. Knowing that the last name is Jones, and that this phone book is sorted by last name, we open the phone book somewhere in the middle. We know the letter J is close to the middle of the alphabet.
  - Paging not by single pages, but by dozens of pages we locate the start of the J page. Even then, the last name starts with “Jo” (Jones), not “Ja”, or “Jb”, so we still page many pages at a time to find the listing for all persons with a last name of Jones.
  - This phone book is sorted only by last name, so while we are at the start of a list of male and female persons with the last name of Jones, the name we seek could be last in this list. This list is not then subsequently sorted by first name or by gender. We perform a sequential scan within Jones to find Bob, and then Bob Jones, a male.

  - In an actual SQL database, we have used the index to position at the start of the list of all persons named Jones, then we had to go to the SQL table proper to read each first name and gender. In this example, the first name and gender were not members of the given index.
  - While this phone book contained 1,000,000 persons total, there were only 10,000 persons with a last name of Jones. By using the last name index, we avoided some amount of workload between sequentially scanning 1,000,000 records, *or* performing an indexed read of 10,000 index key columns, and then looking up the remaining (non-key) columns in the table proper by their absolute disk address.

- Now let us change the example to one with a different selectivity.
  - Let us say that the phone book is not organized by last name but is organized by gender (male or female). We know Bob is a male so we can position at the start of the list of males, then go to each record sequentially and see if the name is Bob Jones.
  - This option would give us measurably more work to perform. As a filter, the gender code is measurably less selective than last name. The amount of physical or logical disk I/O to perform on a list that is sorted (indexed) by gender, a less selective filter, is higher than a list that is sorted (indexed) by last name.
  - We need to tie selectivity of filters and their effect on table order with other concepts which are not defined here. This work will be done shortly. At this time we can state that *for now, all things being equal, preference is given to the filter with the greatest, most unique, selectivity.*
- If you wish to state that the phone book example is too simplistic, or too rigid, to be of real value, consider the following:
  - Why not just create a phone book which is sorted by last name, then first name, then gender (this is called a concatenated, or multi-column index)? Concatenated indexes have issues too. Consider a phone book that is sorted by last name, then first name, then whatever else. To find a person in that phone book whom you only know has a first name of Bob, (you don't know the last name), you lose the index and have to perform a sequential scan.
  - In short, concatenated indexes do not solve every problem. Concatenated indexes have limits equal to or greater than non-concatenated (single column) indexes.

As stated in the note box above, the subsection title "index utilization guidelines" does not refer to when you should create indexes. It refers to when the query optimizer can make use of indexes.

Note: The specific condition that was just discussed, (finding a person in a phone book by first name when the phone book was sorted (indexed) by last name, then first name), is called ***negation of index, non-anchored composite key***. The last name, then first name index can not serve a first name only filter criteria because this index is not headed (anchored) by first name. Imagine trying to find all first names of Bob in a phone book. You would have to sequentially scan the entire book. A negation of an index means that an otherwise usable index is disallowed due to a condition.

Another negation of an index is **negation of index, non-initial substring.** That is, given a phone book sorted by last name try finding everyone in that list where the last name ends with “son”, such as in Hanson, Johnson, Williamson, or Ballardson. To perform this search, you would have to sequentially read the entire phone book. The index (in this case sort) column is not organized by the last three characters of the last name. It is anchored by the leading character of this string. When you lose the leading portion of an indexed string, you negate the effectiveness (use) of the index.

This is another reason not to violate the first rule of third normal form, which is do not concatenate related columns. If the last three characters of this column have meaning, then it is a separate entity attribute (column) and should be modeled as such. As a separate column, it can be indexed. (To be fair, some relational database vendors allow you to index a non-initial substring of a given column. Still, this feature often needs to reference a static range. For example, index character positions five through eight, for example; not the last three characters of a variable length data value.)

B-tree+ indexes defined

We have been discussing index use, without really defining what indexes truly are or how they function. There are several types of indexes, and thus far we have been discussing a type of index called a b-tree+ index without bothering to label it as such. Figure 10-7 on page 522 provides a graphical depiction of a b-tree+ style index.

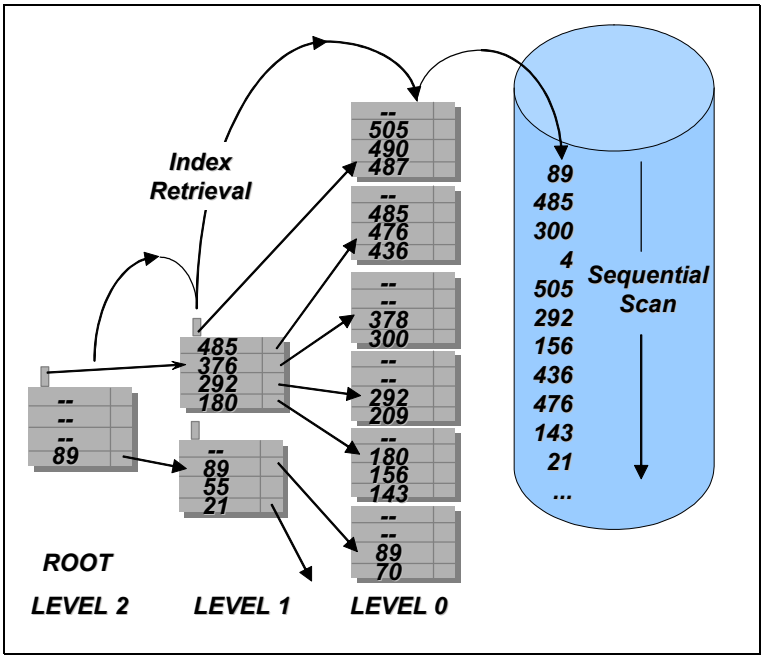


Figure 10-7 Graphical depiction of a b-tree+ type index

B-tree+ style indexes resemble a tree, and represent a standard hierarchy. All access of a b-tree+ style index begins at the root level. B-trees use the analogy of natural wooden trees and have terms such as root, branch, twig, and leaf (nodes), as in the following list:

- Given a list of numbers in the range of 1 to 1,000,000, the root would contain as many entries as can fit on the (index data) page. The root node would say, for example, rows with a key value of 1 to 100,000 are located next at some address, rows 100,001 to 200,000 are located at some other address, and so on.
- Generally any entry in a b-tree+ index contains a single set of column values which are the keys, and an absolute disk address of the structure to which this entry in the index refers.
- Prior to the final (leaf) level of a b-tree+, entries point to further deepening entries in the tree; roots, then branches, then twigs, and then leaves.
- By traversing a b-tree+ index, we eventually arrive at the leaf node, which then points to the address of a row in the SQL table proper.
- The SQL table proper contains all columns found to exist within the table, not just the non-indexes column. In this manner, the table can also be sequentially scanned to arrive at any row or column.

- As any single page in the index structure becomes full, it cannot accept additional entries. At the time of this event, the page will split leaving two pages where one existed previously. Consider the following:
  - By default, a single full page may split into two equally (half) full pages, leaving room for growth on both.
  - If however, you are only adding new records at the end of a range of data (you are only adding increasingly larger numbers), you would be leaving a wake of half full and less efficient index data leaf pages.
  - All modern relational database servers are intelligent enough to detect index data leaf pages that are split in the middle or at the end of a range of data, and to behave optimally. This specific capability is referred to as having a *balanced* b-tree+.
- Because of the cost of having to perform logical or physical disk I/O on the b-tree+ with its many levels, and then still having to go to the SQL table proper to retrieve the remaining columns from the table, the query optimizer considers the selectivity of the index when deciding whether or not to use the index or to discard the index use going instead with a sequential table scan.
- As a very rough guideline, if the query optimizer believes that the query criteria (join or filter) of an indexed column will return one-third or more of the contents of a table, that index is not used in favor of a sequential scan. This is a working guideline only.
- The “+” in b-tree+ refers to a capability that the index can be read in both directions, both ascending and descending.
  - B-tree+ index is either sorted in ascending or descending order at the leaf level.
  - Many years ago, an ascending index could only be read in ascending order, not descending. Descending indexes could read in descending order, not ascending.
  - Index data leaf pages contain a pointer (address) listing the next leaf page in the contiguous chain of index data.
  - Many years ago, these index data leaf pages only contained a pointer to the next leaf page. Now they contain a pointer to the next and previous index data leaf page. This capability adds the “+” to the b-tree+ index name.

Note: The specific condition that was just discussed, (not using an index based on its poor selectivity relative to the query criteria), is called ***negation of an index, (poor) selectivity of filter***. An otherwise usable index may not be used because it would be more efficient to sequentially scan the SQL table proper.

Data distributions

We have seen that the table access method for table A is indexed retrieval. And we have seen that the selectivity of that filter is poor, meaning that better performance would be observed if the index were not used in favor of a sequential scan. We can force a sequential scan of table A using (query) optimizer hints, but that still will not solve our overall problem. We seek to get the total execution time of this query from three to five minutes to sub-second. A sequential scan of table A will save some time, it will not satisfy our objective.

Why did the query optimizer make this error? The query optimizer uses general rules, and observed statistics about the involved tables to determine the query plan. The query optimizer generally gathers these table statistics *on command, not automatically.* Gathering these statistics can be costly. While statistics had been gathered prior to executing the query in our test case, we were gathering the minimal amount of statistics. Because of the cost of gathering statistics, most relational database vendors will gather increasingly complete levels of statistics. Figure 10-8 provides a representation of data by distinct value and the count of records per value.

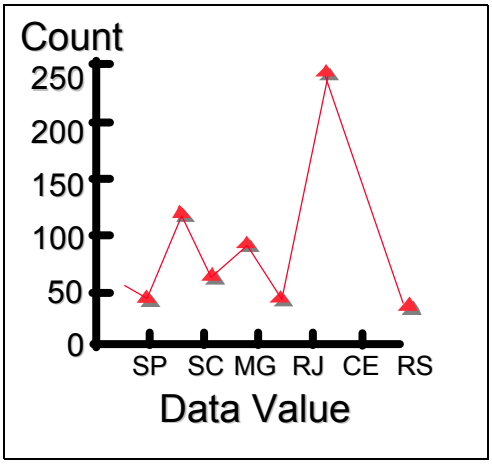


Figure 10-8 A distribution of data by value and count

From the diagram in Figure 10-8, we see that the data value “CE” carries the majority of data by count, while the data value “SP” carries little or no data by count. Refer to the following:

- If we were to evaluate a filter criteria equal to “WHERE value = ‘CE’”, that application is not very selective. If we were to evaluate a filter criteria equal to “WHERE value = ‘SP’”, that application is very selective.

- When query optimizer statistics are gathered at the default level in this vendor's relational database server product, it gathers, among other things, the following:
  - The number of records in a table.
  - The number of unique values in each index.
  - The maximum, second maximum, minimum, and second minimum value in each index key value range.
- Under default conditions, the vendor would know the total number of records, and the number of distinct values. In this case, six.
- Under default conditions, most relational database server vendors assume an even distribution of data. Meaning from our example, we would expect that each of the values in Figure 10-8 on page 524 carries 1/6 of the total number of records on the table. Assuming an even distribution of records is fine if the records are evenly distributed.
- ***Data distributions*** is a relational database server capability where the data is actually sampled to determine the spread (selectivity) of values across a range. Most vendors allow the administrator to determine the percentage of data records that are read to gather a distribution.
- In this example, the server could know in advance to use a sequential scan table access method when receiving a "WHERE value = 'CE'", and possibly to use an indexed retrieval when receiving WHERE value = 'SP'", or at least, given the choice of which filter to execute first, which is more selective.
- Data distribution can be gathered on any column, whether or not it is an element of an index.

Query optimizer statistics

In the three table SQL SELECT example, table A is accessed via a unique index. That is, one that does not permit duplicate values. Table A contains 70,000 records, so this index contains 70,000 unique values. The entire index is unique, yet the SQL SELECT, counting the filter criteria displayed in Figure 10-6 on page 518, returns 98% duplicate values on an equality. How is this possible? The filter criteria for this SQL SELECT accesses a subset of this composite index, only the leading single column of a two column index. Column one of this index is 98% duplicate. The entire width (both columns) of this index is unique.

This vendor default query optimizer statistics gathering behavior looks only at the uniqueness of the entire width of a composite index. Some vendors look at as many as six column deep into a composite index by default. In either case, a query optimizer statistics gathering that examines the data distribution of the leading column of the index solves the problem of whether to use an index or go sequential on the current query plan for this query.

Note: There are many specific applications of data distributions, to include:

- To examine the uniqueness of a subset of a composite (multi-column) index, greater than what the default query optimizer statistics will gather.
- To examine the uniqueness of columns used in filters and joins. For joins, this uniqueness is a large indicator of the cardinality observed between the two joining tables. These columns may, or may not, be indexed.

Measuring the cardinality between tables that are joined

Currently we know that the table access method being used for table A could use improvement. Still, that correction alone will not solve the overall problem. Therefore, we continue to perform additional investigation. We could either measure the selectivity of each of the remaining filters, or simply follow the table join order as the SQL SELECT would be executed by the query processor. Both activities are valid in either order. Since the join may also restrict record counts (there may not be a matching record found to exist in table B that matches with table A), we will follow the query plan table join order.

To measure the cardinality between two tables, execute a SQL SELECT in the manner shown in Figure 10-9.

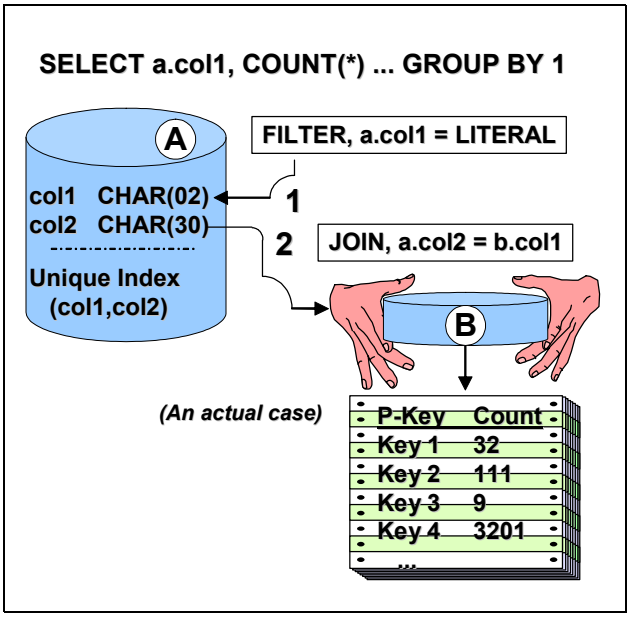


Figure 10-9 Query to measure cardinality between two SQL tables

Given states or provinces within a given country, the SQL SELECT displayed in Figure 10-9 on page 526 would return the state/province primary key, and a count of cities in that state/province. This is a basic construct to measure cardinality between two tables.

In our current example, table A contains 70,000 records, and table B contains 840,000 records. This record counts are accurate from the real world example. In the real world example, each table A join column value would match with somewhere between 8 and 4000 records from table B, producing an intermediate product set, the product of joining table A to table B, of millions and millions of records. If millions and millions of records are produced from this join, and the final query returns only 700 records, where do these records go? Where are they discarded? Figure 10-10 answers that question.

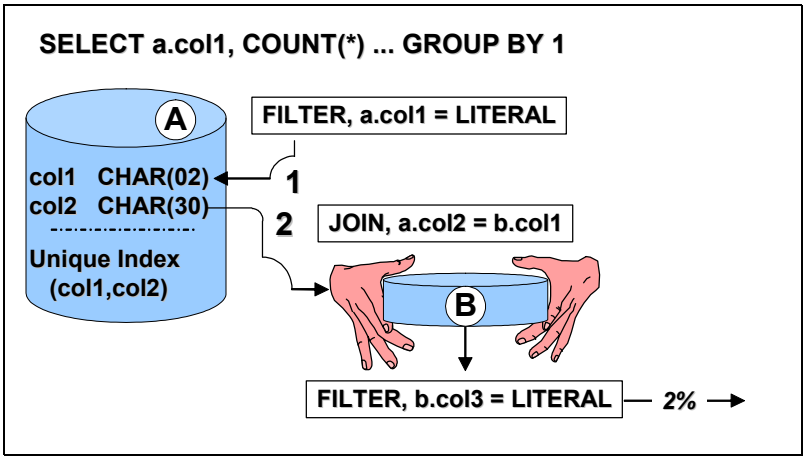


Figure 10-10 Count of both tables with all filters

Figure 10-10 displays the two table query, with tables A and B joined, and with all filters applying to tables A and B being present. The filter on table B is very selective and reduces the number of records produced when joining tables A and B to a final result size of 700 records. The filter on table B is definitely the filter that should be executed first, before table A. Beginning the query processing with a table order of table B before table A, reduces the number of total records processed by several orders of magnitude.

Note: It is important to understand that the query optimizer made the optimal query plan given the conditions that exist in the run-time environment. The primary fault of the query execution time is **related to the data model**.

Before moving on to a solution, Figure 10-11 displays an interesting behavior to this query that was not previously discussed.

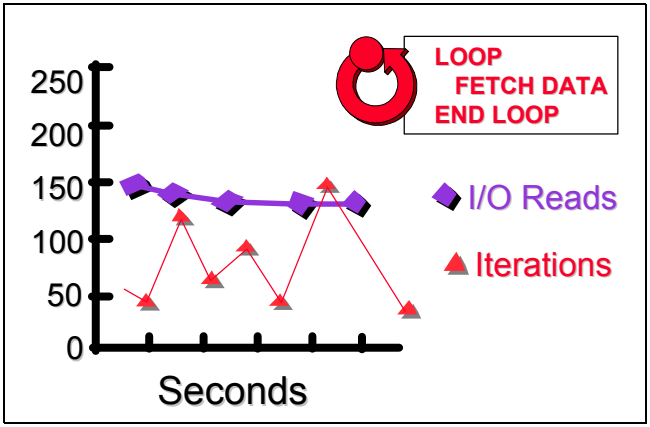


Figure 10-11 Disk I/O versus record retrieval graph

Explaining misleading server tuning behavior

As part of the earlier investigations, we had examined disk I/O rates and queues. The system in this real world example had eight (count) SCSI type III hard disks. All three of the tables in this example were located on one disk. We could have achieved a small amount of improvement in performance by spreading these tables across several disk drives or controllers.

The database server software was reporting a very consistent rate of 150 physical disk I/O operations per second, which is average for a SCSI type III drive. What was odd was another behavior. We created a ten line computer program to execute a SQL FETCH loop for this three table SQL SELECT. This SQL FETCH loop would return a few records and then pause for many seconds or sub-seconds; the loop returned one record at a time as SQL FETCH loops do. Until the data gathered in Figure 10-10 on page 527, we were at a loss to explain this stop and start SQL FETCH loop behavior.

The query was executing at a constant and fast rate. The delay in query processing (as witnessed by the delays in the iterations of the SQL FETCH loop), was evidence of the number of records examined, filtered, and joined between tables A and B, only to be discarded by the filter on table B. We are processing hundreds of thousands of records only to produce 700. This is our major optimization that will reduce the total query execution time from minutes to sub-second.

10.3 Query optimizer example solution

Here we present the solution to the query optimizer example that began in 10.2, “Query optimizer by example” on page 503. Figure 10-12 once again displays the original query plan for this example.

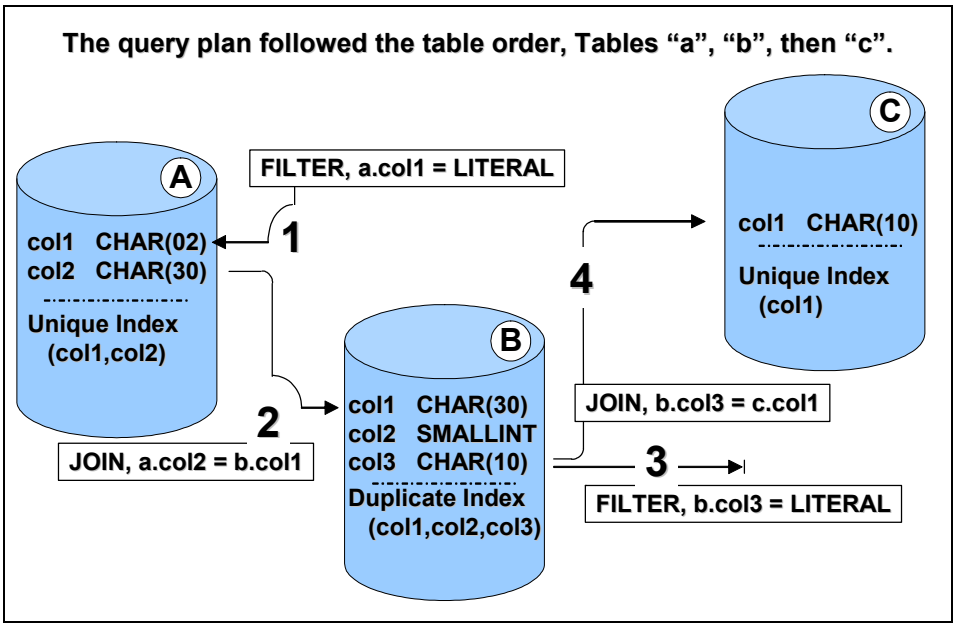


Figure 10-12 Original query plan from example

It is the filter on table B which should be executed first based on the selectivity of that filter. This same filter greatly reduces the number of records that need to be joined from table B into tables A and C. The query optimizer is prevented from taking this query plan because of the run-time environment.

Note: The query optimizer will do nearly anything so that it may complete your request for SQL service. One thing the query optimizer will not do is sequentially scan two or more tables inside one SQL SELECT, nor will it sequentially scan the second or subsequent tables in the table order. Sequentially scanning the second or subsequent tables in a SQL SELECT means that table, or tables, would be sequentially scanned numerous (repetitive, perhaps hundreds of thousands or millions of) times. That is generally not happening, because performance would be grossly unsatisfactory.

The join criteria from table B to table A is on table A col003. Table A col003 is not indexed. If the table join order in this query were table B, then table A, then each of 700 records in table B would cause a sequential scan in table A. *In order for the query optimizer to consider a table B then table A table join order, the join to table A would have to be served via an indexed retrieval.* Currently, there is no index on table A that supports this behavior, and that is why the query optimizer will not choose table B first.

10.3.1 The total solution

The total solution to the three table SQL SELECT example begun in 10.2, "Query optimizer by example" on page 503 contains the following:

- The table A, concatenated index on col001, col002 is next to useless. This index is removed and replaced with an index on table A col002, then col001. This new index preserves the unique data integrity constraint that this index serves.
- Because of the nearly total lack of unique values in col001 of this index, we have little fear in removing this index. Any other portion of the system that was using this index for processing will run faster without this index.
- This concatenated index leads with col002 to support the join from table B.
- This correction is displayed in Figure 10-13.

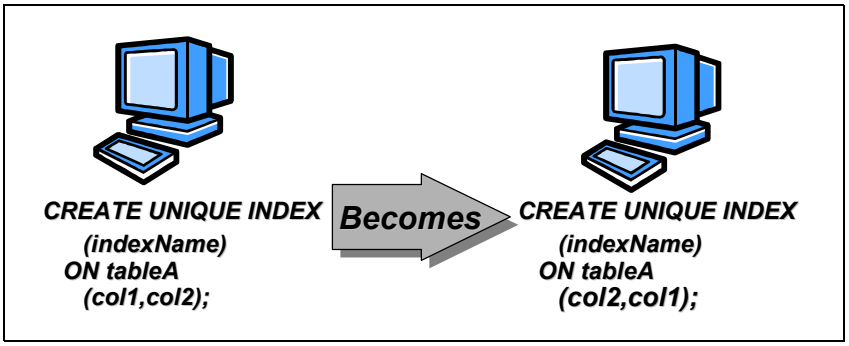


Figure 10-13 Correction to table A

- The table B index on col001, col002, col003 is also reversed to contain col003, col002, then col001, as displayed in Figure 10-14 on page 531.

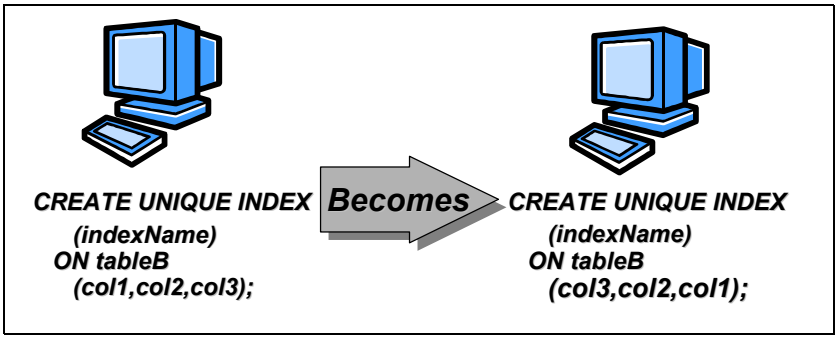


Figure 10-14 Correction to table B

- There was never anything wrong with table C.
- The new query plan is displayed in Figure 10-15.

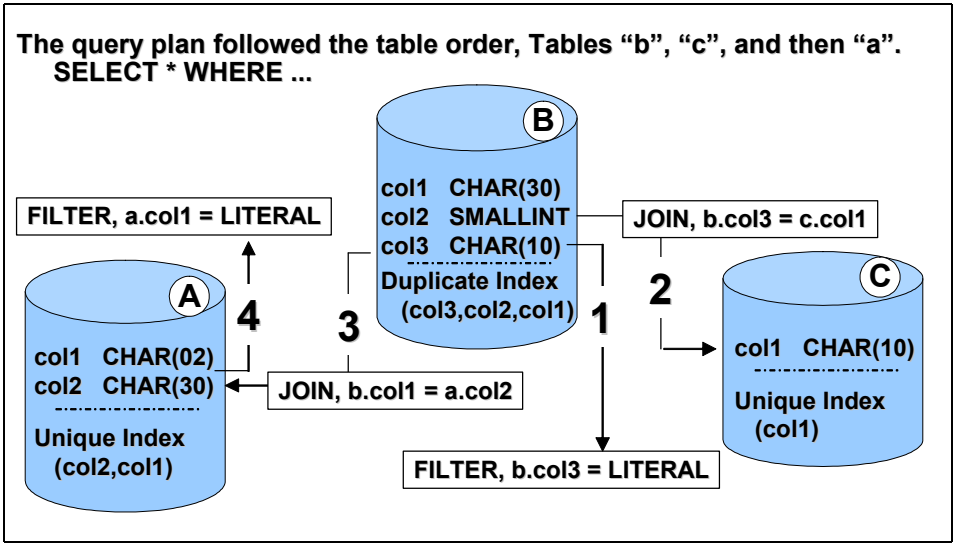


Figure 10-15 Optimal query plan for this example

The old query plan processes hundreds of thousands of records as table A is joined with B, and then the restrictive filter is applied. The new query plan begins by reading table B, retrieving only 700 records which are then joined with tables A and C. The new query plan is detailed below:

- Table B is processed first using a duplicate index, but in a key-only retrieval.
- Then the join is done to table A via the unique index on table A.
- The join to table C is done as before, via a unique index via an equality.

- There are still no temporary files.
- The query execution time completes in under one second.

Before releasing this solution to the customer, we must check both data sets. The one that was produced by the current (under performant) production system, and the one which is created by the (performant) test system. Because the two data set sizes were manageable, we unloaded the results to ASCII text files and used operating system utilities to compare these results and prove them to be identical.

10.3.2 Additional comments

Further comments on this example are contained in the list below:

- Upon critical examination, we see that the index on table B contains three columns; col001, col002, and col003. Col002 is never mentioned in the example. Although col002 is never referenced, it was left as a member in the new solution index in the event that other portions of the application system needed this column in this index.
- The original index on table A could definitely be removed; it was useless.
- You should generally lead a concatenated index with the most unique, most selective columns.
- The original index on table B may have been left in place in the event some other portion of the system needs it. A new second and corrected index could have been added. But, indexes are not free resources. In addition to the disk space they consume, they must be maintained. Here are a few examples, for instance:
  - An index is like a mini, vertical slice of a table. It contains a subset of columns and all rows.
  - When a table has (*n*) indexes, a single new record insertion must be recorded in (*n*+1) places. This event must be recorded the SQL table proper, and then in each index. If a table has four indexes, there is the SQL table proper and four vertical slices of that table, the indexes, that must receive the new record insert.
  - This (*n*+1) guideline is also true for record deletes.
  - Record updates affect the table proper, and then any indexes containing columns that are changed.
- In many of the program code listings and diagrams in this example, the column list of the SQL SELECT statement lists all columns. Retrieving all columns from table B would have prevented a key-only retrieval, since all columns from that table are not members of the index being used.

- Table B has nine total columns and seven indexes. The original data modeler had violated the first rule of third normal form entity relationship modeling, because there was a repeating group of columns that were folded into table B. These columns should be exported as elements of a fourth table. The reason for so many indexes was to overcome index negation, non-anchored composite key:
  - An easy example to describe what was taking place would be an order header table, and a detail table to the order header with the various order line items. The order header lists the unique attributes of order date, customer number, and others. The order line items lists that dynamic count of items which may appear on the order, such as the stock unit identifier, the item count, and cost.
  - This modeler was new to relational database and SQL and could not get over the cost of having to join tables, compared to older database technologies which model data differently.
  - To reduce the number of join operations, this modeler folded the order line item (child) table into the order header (parent) table. This not only limits the number of line items that can be on one order (the column list is now hard coded as a repeating set of columns), it causes a number of performance issues.The most obvious performance issue raised is non-anchored composite key.
  - When order line items appear as a repeating set of columns in the order header table, it becomes very hard to answer the question “which orders contain item number ‘444’, for example. This data could be in any of a repeating set of columns. Not only do you now require a larger number of indexes to support that query, the SQL SELECT syntax is also unduly complex.
- Why was the leading column in the table A index on col001, col002 a duplicate:
  - As mentioned, this application (and its associated data model) was sold to the customer. This is a packaged application and was sold door to door, to dozens of consumers.
  - Basically, the table A column col001 stored the activity location identifier, similar in use to a warehouse code. This customer only has one warehouse.
  - We need not have changed the columns in the table for this one customer, but we could have changed the index plan. Changing columns will likely cause the application program code to change. No one would want this, since it increases cost and allows for divergent code streams in the application. Changing the index plan does not call for the application program code to change.

- For those who are big on maintaining separate logical and physical data models, perhaps the index plan should always be part of the physical data model; at least when a given data model is delivered to numerous physical implementations.

10.4 Query optimizer example solution update

At the beginning of 10.2.1, “Background” on page 503, we stated that this real world example takes place several years ago. Since that time, relational database servers have increased their program capabilities. The current release of the database server software product referenced in this example would likely still not serve this three table SQL SELECT example in sub-second time, given the errors in the run-time environment. Read the following:

- We define the run-time environment as, the SQL SELECT, the available indexes, the version of relational database software, and other conditions.
- The original run-time environment with its older software executes the query in five minutes, on average. (Remember the query needs to execute in sub-second time.)
- Just updating the version of relational database software improves execution time of this query from five minutes to five or 10 seconds.
  - While the query execution time is greatly improved, it is still not sub-second as required.
  - The improvement in execution time is accomplished by a better query plan. This better query plan is still not as good as that provided in the solution in “Query optimizer example solution” on page 529.
  - The new query plan uses a lot more memory and CPU resource to accomplish the improved execution time.
- We can use query optimizer hints to force the updated relational database software to execute the original (least optimal) query plan. Purely as a result of the newer and improved software, query execution time moves from five minutes to 10 or 20 seconds. In other words, just updating the software allows for most of the performance gain, not the new query plan.

The current release of this product would move execution time from five minutes to somewhere around five or 10 seconds using new technology. The modern query plan, from the same run-time environment is listed in Example 10-2 below:

Example 10-2 Query plan of same three table SQL SELECT and run-time environment, with newer release of database server software

```
0000
0001  -- This file has been edited for clarity.
```

```
0002
0003  QUERY:
0004  ------
0005  select
0006        ta.*, tb.col001, tb.col002, tb.col003, tc.*
0007     from
0008        table_a ta,
0009        table_b tb,
0010        table_c tc
0011     where
0012        ta.col001 = “01”
0013     and
0014        ta.col002 = tb.col001
0015     and
0016        tb.col003 = “555”
0017     and
0018        tb.col003 = tc.col001
0019
0020  DIRECTIVES FOLLOWED:
0021
0022  DIRECTIVES NOT FOLLOWED:
0023
0024  Estimated Cost: 1608720
0025  Estimated # of Rows Returned: 57142
0026  Temporary Files Required For:
0027
0028    1) Abigail.tb: INDEX PATH
0029
0030          Filters: Abigail.tb.col003 = ‘555’
0031
0032       (1) Index Keys: col001 col002 col003
              (Key-Only)  (Serial, fragments: ALL)
0033
0034    2) Abigail.ta: SEQUENTIAL SCAN
0035
0036          Filters:
0037          Table Scan Filters: Abigail.ta.col001 = ‘01’
0038
0039
0040  DYNAMIC HASH JOIN
0041      Dynamic Hash Filters: Abigail.ta.col002 = Abigail.tb.col001
0042
0043    3) Abigail.tc: INDEX PATH
0044
0045       (1) Index Keys: col001   (Serial, fragments: ALL)
0046           Lower Index Filter: Abigail.tb.col003 = Abigail.tc.col001
0047  NESTED LOOP JOIN
0048
```

10.4.1 Reading the new query plan

By reading the query plan displayed in Example 10-2 on page 534, we see the following:

- Lines 005-0018 list the very same SQL SELECT we have been reviewing throughout these last sections. All of the joins are the same, and all of the filters are the same.
- Lines 0020-022 lists the query optimizer directives in effect. There are none.
- Line 0024 lists the estimated cost of this query. This estimated cost (1,608,720) is higher than the cost from Example 10-1 on page 509, which was (407,851). The amount of resource, both memory and CPU, is higher which will be seen in this list, but the execution time (which is not displayed) is much lower.
- Lines 0028, 0034, and 0043 display the table order for this query plan.
  - This query plan accesses the tables in the order of B, then A, then C, which is our preferred order.
  - We wish to process table B first because of its highly selective filter criteria. (Remember that the filter criteria on table B returns 700 records, on average, from a total record count of 840,000. This means we only have to perform 700 joins to both tables A and C, not 840,000 joins from table A to B.)
- Line 0028 lists the table access method for table B.
  - The first operation we wish to perform on table B is application of the filter criteria. The filter criteria on table B effectively reduces the size of this table, as records are rejected when they do not match the filter,
  - `WHERE col003 = '555';`
  - The table access method on table B would normally have been a sequential scan, because no usable index is headed with col003 (the column referenced in the filter). The concatenated index on table B columns col001, col002, col003 would be disallowed because of a negation of index, non-anchored composite key.
  - However, we can use the index on table B, col001, col002, col003 because of an exception. In table B, we are only reading columns col001 and col003 which are both members of the col001, col002, col003 index. The only **exception to the negation of index, non-anchored key** behavior is when that index may be used in a key-only indexed retrieval.

– An index is a vertical slice of a table, containing all records but only a subset of the columns. The index on table B, columns col001, col002, col003 will be read sequentially from top to bottom, because of the non-anchored condition. Reading this index is still more efficient than sequentially scanning the SQL table proper, as the index is presorted and measurably more compact.

Note: Another technology, not previously mentioned, will also be employed here. That is *read ahead scans,* also known as *pre-fetching.* When the database server detects (SQL table proper, data) sequential scans, or a full range (all records, sequential) index leaf page scan, which is mentioned above, or a bounded (data range driven) scan of leaf pages within an index, the query processor reads numerous pages in one physical I/O call, knowing in advance that these pages are needed and will be consumed.

In this example, we are sequentially scanning 840,000 records in an index. Because the index is small (3 columns, 44 or so bytes, with duplicate entries which reduce the overall size, perhaps 14 MB, per our example), it is *possible* we will read this table in as few as 8-20 physical I/O calls because of read ahead scans.

- Line 0034 lists the table access method for table A, and there is much new technology to discuss here.
  - It was mentioned many times that the original query plan table order could not be table B then table A, because the join to the second table, table A, is not served via an index. No query optimizer would allow a second or subsequent table in the processing of a query to be read sequentially. If this were allowed, it would mean that each of 700 records from table B would cause a sequential scan into table A. That absolutely would not perform well.
  - But, the new query plan does read table B, then table A, and without preexisting support of an index in place on table A. How is this allowed? The query processor will build a temporary index to support an indexed retrieval into table A. An indexed retrieval will be the access method for table A, even though no index had previously existed.

We are going to pause the review of the new query plan to complete a discussion of table access methods.

10.4.2 All table access methods

Most vendors recognize four or five table access methods, which include:

- **Sequential (table) access method** is used when reading the records data page after data page, and applying filters as required by the query criteria. Read ahead scans are generally applied during sequential scans, unless record locking would discourage this behavior (you do not want to lock an unnecessary number of records and hurt multi-user concurrency and performance.)
- **Indexed (table) access method** is used when reading a single or set of matching key values from an index structure, and then possibly retrieving the remaining (non-indexed) table columns from the SQL table proper.
  - **B-tree+ indexes** are most common. B-tree+ index technology is discussed briefly with Figure 10-7 on page 522.
  - **Hash indexes** (also known as hash tables) are another indexing algorithm. Actually, hash indexes are many times more performant than b-tree+ indexes *under the proper conditions.*
  - Hash indexes lose nearly all performance if any significant amount of the index data changes, which is common in relational database servers. While b-tree+ indexes perform poorly (with all of their hierarchy levels, branch and twig overhead, page splitting, and so on), when compared to hash indexes, b-tree+ indexes do not lose performance with volatile data sets.
  - During the execution of a SQL SELECT (or any query, which is meant to include the read phase of SQL UPDATE and SQL DELETE), the data is non-volatile; that is, the data is read as a moment in time. Under these conditions, a hash index may be used and will be more performant than a b-tree+ index.
  - Most relational database server vendors have hash indexes, but not permanent hash indexes. They use hash indexes during query processing, and nowhere else.
  - Perhaps the quickest way to describe hash indexes is to compare them to an arithmetic modulus operand; given (*n*) records, and a known distribution of key values, you can determine where a given key value is in this list via a calculation. A small amount of (index leaf page data) scanning may be necessary to correct for any error in this calculation. Hash indexes have essentially a single leaf level, to use the b-tree+ index analogy. This is why hash indexes do not handle volatile data well; they have no branch and twig levels to split for new data insertion in the middle of their bounded ranges. Volatile data would soon cause the entire hash index data structure to have to be rewritten.
  - **Bitmap indexes** are another indexing algorithm. Bitmap indexes best serve highly repetitive demographic type data, and are used often in decision support applications.

- Perhaps the quickest way to describe bitmap indexes is to envision a two dimensional matrix of all SQL table proper record physical disk addresses (the x-axis dimension), and then all possible key values (the y-axis dimension). Any point in the two dimensional graph contains a single bit (a very small amount of storage) which is boolean; a true or false condition to answer, is this record a member of this key value group?
- The performance of bitmap indexes lies in the manner that complex groupings of boolean logic (AND and OR conditions) that bitmap indexes can process with highly efficient bit-wise boolean software operands.
- Bitmap indexes can handle volatile or static data.
- **R-tree (Region tree) indexes, or Bounded box indexes**, are used often for geographic data to form associations between near or like data.
- Perhaps the easiest manner to describe R-tree indexes is to envision a single point on a map, and then a listing of every other point located within every gradient of one kilometer. There would be a list of every other point within 1 kilometer, 2 kilometers, and so on. The R-tree data can also encode direction, such as North of the given point and 1 kilometer away.
- **Star indexes** are not a specific term we define here, but more a grouping of index technologies to which various relational database vendors give many names. Generally, star indexes are those used to list key columns from two or more SQL tables; representing a pre-joined set of records to serve a frequently observed query.
- Star indexes can have great difficulty with volatile data, since they represent records from numerous tables, that have also to be evaluated for a join criteria. (Add a new record to one table in the joined table list, and you have to perform the joined select to see which new records have to be added to this multi-table index; each record add calls for execution of a joined SQL SELECT.)

- **Remote path (table) access method** is used when one or more of the involved tables is located in a remote database server. Given a two table SQL SELECT, where the tables are joined and each table has a single filter criteria, here is what is likely to happen:
  - The portion of the query involving the remote SQL table will be sent to the remote server for processing; that is, just the filter criteria is applied to the remote table.
  - The filtered (remote) result set is copied to a temporary table on the local database server.
  - Query processing will continue as though the remote table were local, which it essentially now is.

- Most vendors will not attempt to join records over a network as the communication latency would prove too slow to the overall execution of the query.
- A good query optimizer will have knowledge of the remote table, and based on selectivity of filters and cardinality of the join condition, may filter the local table, export it to the remote server, do the join, then copy the results back. Obviously with that additional communication overhead, the local database server would require strong evidence to take this query plan. The IBM DB2 UDB query optimizer excels at this capability.

- **Auto-index (table) access method** involves something similar to the three table SQL SELECT example we have been examining.
  - As mentioned, the query optimizer will generally disallow having a sequential table access method on two or more tables inside a single SQL SELECT.
  - Given a two table SQL SELECT where neither side of the join pair is indexed, what can the query optimizer do. Years ago, the query processor would create a full multi-user mode b-tree+ style index on the larger of the two tables, then sequentially scan the smaller table, and join to the now indexed (what was largest) table. This index was then discarded immediately after query processing. (The index was discarded because it was built at a moment in time, like a hash index. That is, the index was not maintained during query processing.)
  - Because creation of b-tree+ style indexes is very expensive, especially during query processing, this behavior has been changed (improved). The query optimizer will create a hash index, versus a b-tree+, to perform the query processing.
  - While technically a table access method, creating a hash index to perform a table join is often recorded as a table join method. (If we do not have a join, we do not auto-index.)
- **ROWID (table) access method:** ROWID is generally a term used to refer to the special column in each table which is the absolute disk address of a given record on the hard disk. A ROWID access method is the most efficient since the physical location of a given record is known. As a table access method, the term ROWID is rarely seen; its use is implied when query processing through the given tables and records.

10.4.3 Continue reading the new query plan

Now that all table access methods have been fully defined, we continue on line 0034 in Example 10-2 on page 534, to review that access method on table A:

- As displayed, the query processor will perform a sequential scan on table A, *but only so it may build a hash index* on table A, column col002. The table access method on table A is actually (hash) indexed retrieval; that is, as soon as the index is built.
  - The query optimizer detects that a filter is in effect on column col001 of table A but that this filter is not very selective, and would be better served by a sequential scan (to build the index).
  - The hash index created on table A col002 may fit entirely in memory, or may be large enough that it will overflow to temporary storage on disk. While the query plan contains a section for temporary files (objects), it often does not report temporary files required for hash index creation. Another administrative tool is needed to uncover this activity.
  - After the hash index on table A is built, query processing continues joining table B to table A.
  - The newly created hash index contains the table A col002 key value, in ascending key value sort order, along with a physical disk address of the remainder of the table record in the SQL table proper.
  - Line 0037 displays the filter on table A.
  - Lines 0040-0041 display the join method between tables B then A. The join method listed is dynamic hash join, but it is more commonly referred to as a hash join method.
- Based on the level of query optimizer statistics gathered, we were seeing indexed retrieval on table A to build the hash index, or then in other query plans, sequential. Even though we do not like the current index on table A, it is a (small) vertical slice version of table A and may be a more performant method to retrieve the data for the hash index.
- Lines 0043-0047 detail table C.
  - Table C is accessed via an indexed retrieval and joined via a nested loop.
  - Because the join to table C is via an equality to a single column in a unique index, table C can only return one record per join. For this reason, no read ahead scan behavior is needed for table C.

10.4.4 All table join methods

Most relational database software vendors have at least three or four (table) join methods. Listed below are comments regarding (table) join methods:

- **Nested loop (table) join method:**

- Nested loop (table) join method is the oldest join method. The newer (table) join methods have only arrived in the last five to seven years with parallel process architecture relational database servers.
- The nested loop (table) join method is dependent on at least one of the two join tables having a preexisting index. If one of two tables is indexed, the non-indexed table is read first via a sequential scan, while qualifying rows are read from the (second) table via an indexed read.
- If both join tables lack an index, then one of two tables will have a hash index created on the join columns, of the pre-filtered records; this behavior is called an auto-index. Technically this is an indexed table access method and a hash index (table) join method.
- Nested loop (table) join method is often viewed as the most efficient join method based on total execution time of smaller data sets, but this is somewhat misleading. The nested loop (table) join method depends on indexes (a workload with an associated execution time) that are created and maintained before query execution.
- The nested loop join (table) join method is probably less CPU and disk I/O efficient than other (table) join methods when larger-sized data sets are in effect. This is somewhat dependent on index efficiency, selectivity of filters, and other run-time conditions.

- **Hash index (table) join method:**
  - The hash index (table) join method is somewhat like a nested loop (table) join method in function; an iterative (reentrant) software programming loop is employed. However, the hash index is so much more efficient based on its function and smaller size, that this comparison is rarely made. (Efficient once the hash index is created.)
  - Few if any relational database server vendors have permanent hash indexes, and prefer instead to use them exclusively during query processing.
  - A hash index table join method is reviewed in the query optimizer example detailed in “Query optimizer by example” on page 503, and will not be expanded upon further here.
  - Use of a hash index has two phases. During the build phase, the hash index is created. During the probe phase, a second table uses the hash index to perform the join.
- **Sort merge (table) join method:**
  - Sort merge (table) join method is definitely the most technically advanced (table) join method. It is generally most used in the decision support application area.

– To better understand sort merge (table) join method, first we are going to offer a little more detail regarding the nested loop (table) join method so that we may compare and contrast. Figure 10-16 on page 543 displays a simplistic rendering of a nested loop (table) join method.

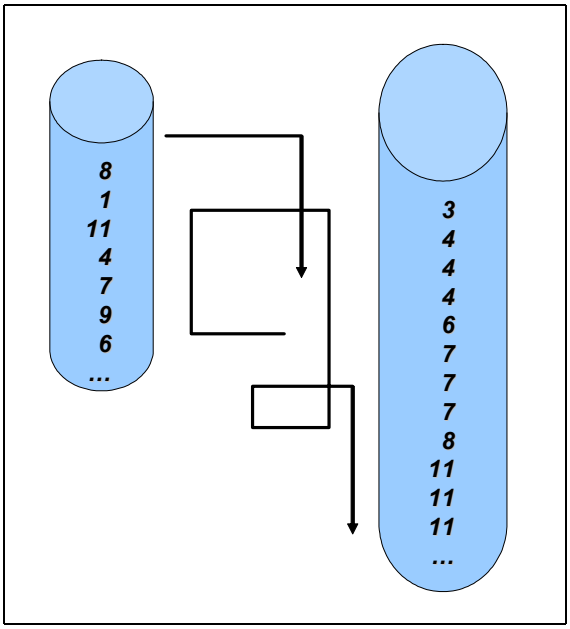


Figure 10-16 Graphical depiction of nested loop (table) join method

Concerning Figure 10-16:

- The query processor only joins two tables at a time, not three or more. To join three or more tables, first two tables are joined, and then the third table is joined to this two-table (now joined) product, and so on until the query is complete.
- It has been stated that the query optimizer will not allow for a query plan wherein the second table is accessed via a sequential table access method. If there is no index to support accessing the second table, one is built during processing of the query. Following this idea, the table on the right of Figure 10-16 is shown to have data in sort order, where the data in the table on the left may be in random or ordered sequence.
- The sort order present in the second table is meant to imply some form of indexed retrieval.
- The lack of sort order in the first table is meant to imply that this table may by read via a sequential scan. (Data in an SQL table is generally not presorted, and is assumed to be in random sequence.)

- The table on the left will be read one time only, from top to bottom via whatever table access method is in effect. If any filter criteria is in effect for the left table, this filter criteria will be applied. Only those records which satisfy the filter criteria are then joined to table two. Each single row from table one which meets the filter criteria will cause table two to be accessed. If 20 records from table one satisfy the filter criteria, then table two will be accessed 20 times. If each record from table one joins with 10 records in table two, 20 accesses of 10 records each equals 200 total accesses of table two.
- The table on the right is accessed via an indexed retrieval. The arrows between the two tables in Figure 10-16 on page 543 are meant to signify program flow control; that is, a recursive loop starts by positioning within table one, and then each row in table one meeting the filter criteria can access all or a subset of table two.
- Given two tables with all other factors being equal (table size, selectivity of the filters being applied to each table are equal or nearly equal, and certainly other factors), the query optimizer follows a convention of *worst table first;* that is, given two operations, one which is efficient and one inefficient, and one of these operations drives the second which must be repeated, lead with the inefficient operation which causes a repetition of an efficient operation. In other words, given one indexed table and one which is not, lead with the non-indexed table, suffer through the sequential scan (which is done once), and repeat an operation which is a small, targeted (indexed) read of the second table.

Concerning Figure 10-17 on page 545:

- The nested loop (table) join method from Figure 10-16 on page 543, has no startup costs; we can just begin by sequentially scanning table one, and join into table two via an indexed read. If neither table is indexed, then one table will be indexed during query processing using, most likely, a dynamic hash join into table two. However, that is a hash join, not a nested loop join.
- A sort merge (table) join method is generally used in business intelligence and/or when the following run-time conditions are in effect:
  - Neither table from the join pair is indexed, *or* such a large volume of these two tables is to be read that indexed (table) access methods would prove too costly. Filters, while present, are still expected to produce a larger percentage of records from the tables than not.
  - This raises a general condition of business intelligence (BI) queries over those seen in online transaction processing (OLTP); that is, BI queries read sweeping and voluminous amounts of data with little or no support from indexes, while OLTP read few records and via known indexed table access methods.

- Also, BI queries are often accompanied with set operands; that is, aggregate computations on received rows, and computations such as AVG (average). These aggregate calculations require the data set that they report on to be sorted by several columns which are often not indexed.

- The query from Figure 10-16 on page 543 was expected to process 20 times 10 records (200 records). The query in Figure 10-17 is expected to process all or most of both tables (many hundreds of records, and then with a heavy data sorting requirement).

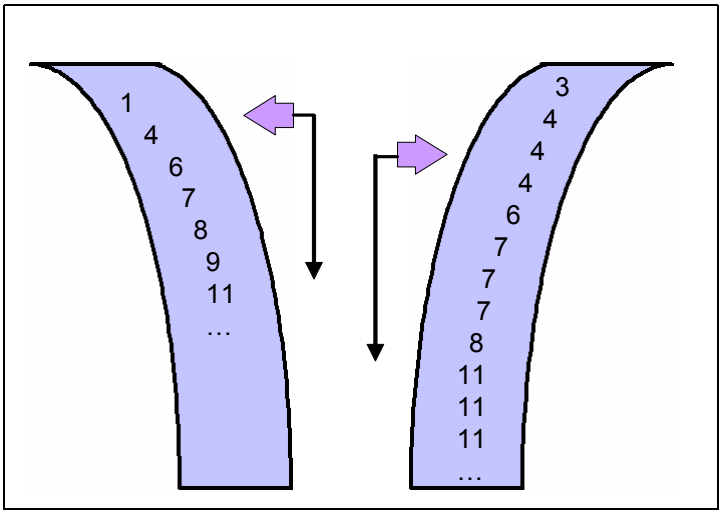


Figure 10-17 Graphical depiction of sort merge (table) join method

- Figure 10-17 displays two data sets, table one data and also table two data. Table one data is read sequentially, has any filter criteria applied, and then sorts the data by join column key value. The same is done to table two, either concurrently or in sequence.

- Then, table one is read from top to bottom one time only. As a single record is produced from table one, it is joined to any matching records in table two. As the join pairs from table two are exhausted (we have reached the end of the range of records in table two which do in fact join), processing returns to table one. The primary point here is, table one is read from top to bottom one time only, and the same is true for table two.

- The sort merge (table) join method has measurably larger startup costs; that is, filtering and sorting both tables before any processing is able to be performed. However, after this preparatory processing is complete, the sort-merge (table) join method is very fast, measurably faster than a nested

loop (table) join method, since significantly less post-preparatory work needs to be performed.

The final comparison on nested loop versus sort-merge (table) join methods is listed below:

- If all run-time conditions are met, a nested loop join method will produce the first row first; meaning, nested loop has little or no preparatory work, and will output a joined record before sort-merge (table) join method.
- If all run-time conditions are met, a sort-merge (table) join method will produce the entire data set first; meaning, even with all of the preparatory work to perform, sort-merge join is measurably more CPU efficient than a nested loop (table) join method. Nested loop (table) join method accesses the second table multiple times, sort-merge (table) join method accesses the second table once. It is a trade-off of waiting for preparatory work to complete, versus not waiting.

Note: If you have seen query optimizer influencing statements that call for *first row* first or total data set (*all rows*) performance; this is an area where these types of statements are considered.

- There are times where the run-time environment does not support a nested loop (table) join method, where you must employ a dynamic hash (table) join method or sort-merge (table) join method.

10.4.5 Continuing the list of all table join methods

The fourth, and perhaps most interestingly named, table join method is the **push down semi-hash table join method**. The push down semi-hash join is used almost exclusively in the area of business intelligence and decision support. The run-time environment that calls for the execution of a push down semi-hash join is listed below:

- As a decision support (business intelligence) query, the tables being referenced in this type of query are expected to be data modeled using the dimensional data modeling; a large central fact table containing millions of records which represent some event or intersection of data. A number of small dimension tables contain the indexes to this data. (Index as in demographic vector; indexes such as time, location, and product categorization. Index in this context does not mean database index, as in hash or b-tree+.) Figure 10-18 on page 547 displays this kind of query.

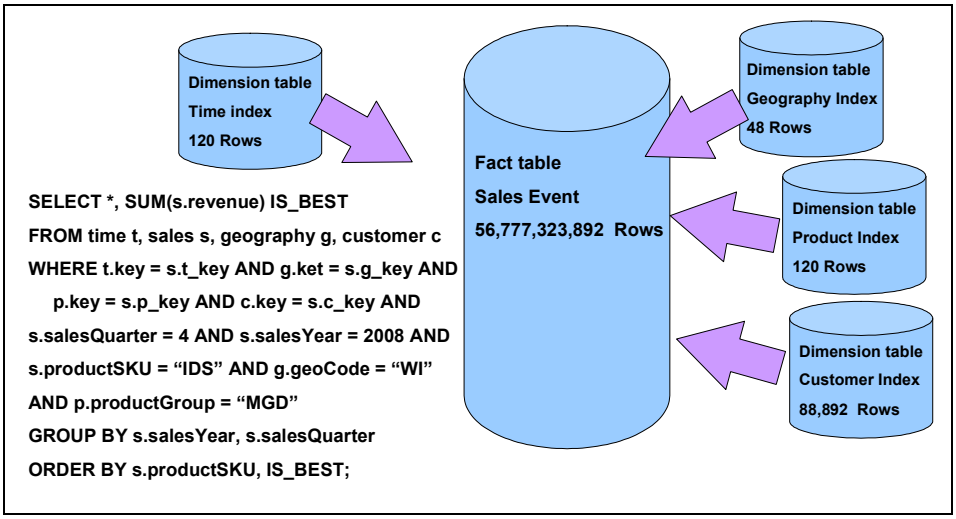


Figure 10-18 Example query referencing dimensional modeled tables

- In Figure 10-18, the Sales table has nearly all of the records by count, and that number is massive compared to the remaining tables. All of the dimension tables, in this case there are four, join to the Sales (central fact) table, which is typical in a dimensional model. The problem is, any normal query processing is going to hit the Sales table within the first or second table, and processing Sales is going to produce a result set of thousands or millions of records, thousands or millions of records which then must be joined to the three remaining dimension tables; (this results in lots of looping and CPU consumption.)
- Figure 10-19 on page 548 is a graphical depiction of a nested loop join method of this query. If Product-A represents the join of the first two tables, Customer and Sales, this result set will have millions of records; millions of records which then have to be joined to the three remaining tables; Product-B will have millions of records, Product-C, and so on.

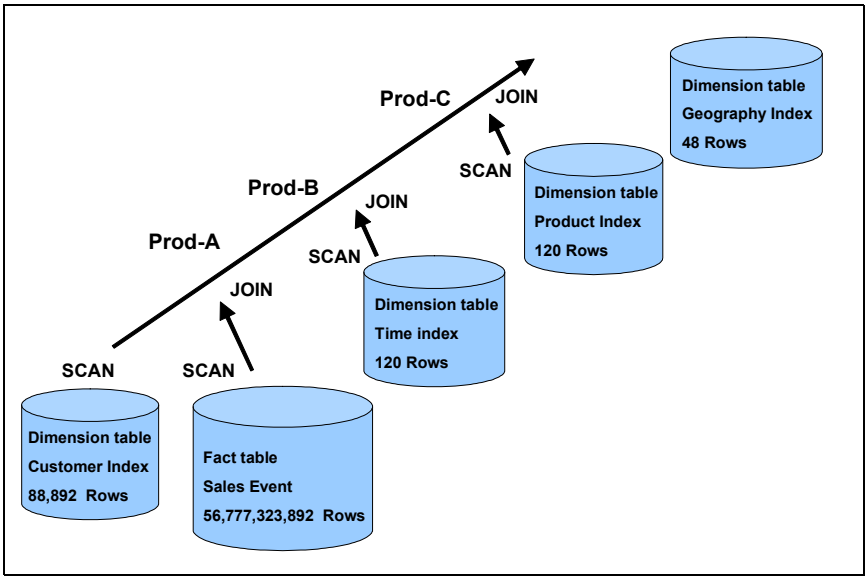


Figure 10-19 Nested loop join of earlier query, also referred to as a left deep tree join

A push down semi-hash join is designed to significantly reduce the number of joins that have to be performed by *processing the (large) central fact table last.* How to do this is not immediately apparent, because all of the dimension tables join to the single fact table, the dimension tables do not join to themselves.

A push down semi-hash join accomplishes this goal (processing the fact table last) in the following manner:

- Using the transitive dependency query optimizer query rewrite capability, copy as many of the filter criteria to the dimension tables as possible. The query in Figure 10-18 on page 547 displays five filter criteria. Example as shown:
- `WHERE s.salesQuarter = 4 AND s.salesYear = 2008 AND`
- `s.productSKU = “IDS” and g.geoCode = “WI” AND`
- `p.productGroup = “MGD”;`
- The two filters on s.salesQuarter and s.salesYear can be copied and applied to the Time table, thus reducing at least a small percentage of records that will need to be processed.
- *The phrase push down refers the following;* join all of the dimension tables into one large result set, applying any applicable filters as this is done. Since the dimension tables do not relate to one another, join these tables as an outer Cartesian product. In other words, create a large intersection data set of

these four tables. What is remaining then is a standard two table join, the single table which is the intersection of all of the dimension tables, and the original fact table as it existed.

- These are two options:
  - Standard nested loop; 88,892 records joined to 56,777,323,892, then 120, 48, and 120, resulting in several billion joins.
  - Or, push down semi-hash join; join 88,892 records to 120, 48, and 120, *then* join to the large 56,777,323,892 record table, resulting in fewer joins.
- The term hash in push down semi-hash join refers to the hash index that is built on the outer Cartesian result set of the dimension tables.
- In a clustered database environment (the database is spread across several physical hardware nodes operating in a single cluster), the outer Cartesian product of the dimension tables is often broadcast to each node in whole, so that all joins may be performed locally on each separate node.

Example index needed for push down semi-hash join

Consider the SQL SELECT listed below. It is a typical business intelligence, star schema type SELECT:

```
SELECT SUM(o.price)
FROM orders o, customers c, suppliers s, product p, clerks c
WHERE o.custid = c.custid AND o.suppid = s.suppid AND
o.prodid = p.prodid AND o.clerkid = c.clerkid AND
c.zipcode = “77491-6261” AND s.name = “KEENE” AND
p.type = “WEB DATABLADE” AND c.dept “ISV”;
```

The optimal query plan is to perform a push down semi-hash join on; Customers, Suppliers, Product, and Clerks. The query optimizer has rules when to consider a a push down semi-hash join. One rule is related to having filters present on some or all of the detail tables (such as Customer and Suppliers). Or, the rules may entail maximum record counts that may exist in any of these (detail) tables. Further, in order for the push down semi-hash join to be available, there needs to be a specific index on the fact table, in this case, Orders. The Orders table requires an index on the Orders table join columns:

```
CREATE INDEX i44 ON orders (custid, suppid, prodid, clerkid);
```

A more verbose depiction of a nested loop join method

Consider the SQL SELECT statement listed below:

```
SELECT c.custno, o.orderno
```

```
FROM customer c, orders o, items i
WHERE c.custno = o.custno AND
o.orderno = i.orderno;
```

The first thing to discuss is that there are several more table join orders than you may realize:

- Join table Orders to Customer, then join the result to Items
- Join Customer to Orders, then Items
- Join Order to Items, then Customer
- Join Items to Orders, the Customer
- (Not likely, but still briefly considered) Join Customer to Items, then Orders
- (Not likely, but still briefly considered) Join Items to Customer, then Orders

The program logic for a table order of Customer to Orders, then Items, is:

```
FOR EACH ROW IN customer TABLE DO
 READ THE ROW INTO c
 FOR EACH ROW IN orders TABLE DO
  READ THE ROW INTO o
  IF o.custno == c.custno THEN
   FOR EACH ROW IN items TABLE DO
    READ THE ROW INTO i
     IF i.orderno == o.orderno THEN
      RETURN THIS ROW TO THE USER APPLICATION
     END IF
   END FOR
  END IF
 END FOR
END FOR
```

From this sample program code, perhaps you can see the CPU intensive nature of the nested loop (table) join method. Also, the ability for the nested loop (table) join method to return the first record after just a few statements can also be seen (three record reads, and two IF statements).

10.5 Rules and cost-based query optimizers

After the example query reviewed in 10.2, “Query optimizer by example” on page 503, through 10.4, “Query optimizer example solution update” on page 534, now we are going to perform a step-by-step detailed review of the all query optimizer algorithms. We cover single and simplistic examples for each data point. After this section, you might then apply all of the points covered here towards the previous example.

The very first query optimizers relied more on (heuristic) program logic to determine the most optimal query plan. As relational database software products became more capable, this logic was enhanced with data related to the tables themselves, and the system which contains them to include; the distinctiveness of duplicate key values, the expected physical performance of the hard disks these tables reside on, the current CPU load or network transfer rates, the amount of available sort memory, and more. As you can see, the query optimizer considers more than table size or the syntax of the SQL SELECT to determine query plans.

When software vendors added more and more awareness of the run-time environment to their query plan calculation, they began to call their query optimizers *cost-based query optimizers*. Cost-based query optimizers did not so much replace *rules-based query optimizers* as they did enhance them. Rules-based query optimizers are based on sound logic. A cost-based query optimizer adds more knowledge to the already sound algorithm of a rules-based query optimizer. Without being overly simplistic, all query optimizers result in a large case statement (with perhaps tens of thousands of lines of code) which determine the query plan.

While different vendors may list more or fewer primary query rules, the list below is generally what most vendors recognize. The primary rules of a query optimizer (which are supplemented with cost-based data) include:

- Rule 1: Outer table joins
- Rule 2: (Non-outer) table joins
- Rule 3: Filter criteria
- Rule 4: Table size
- Rule 5: Table cardinality

Note: As a given SQL SELECT is received by the query optimizer, these five rules are applied in the order listed above until, among other things, a table order is determined; in other words, the order in which the tables will be accessed and then joined.

Each of these rules is expanded upon in the sections that follow.

10.5.1 Rule 1: Outer table joins

If you execute a two table SQL SELECT, and the tables are joined, a record from one table is only returned where there is a matching record in the second table; that is, if you join the Customer and Customer-Order tables, no records from Customer are returned if a given customer has never placed an order. This can be both useful and not so useful. If you need to send a holiday card to each customer thanking them for the (*n*) orders they placed last year, no card is going to the customer who has yet to place orders.

An outer join is one that returns records from joined SQL tables where the stated join criteria exists, and returns only the dominant table record columns where the relationship does not exist; that is, where there is a relationship between two tables, the outer acts like a normal join. When there is not a matching subordinate record, those columns are populated with SQL NULL values.

The term outer join is not entirely industry standard. Some relational database vendors may refer to an outer join as a left-handed join, right-handed join, conditional join, and other similar terms.

Example 10-3 displays an outer join, a join between a Customer and (Customer) Orders tables.

Example 10-3 Example of SQL SELECT with an outer join

```
0000
0001 QUERY:
0002 ------
0003 select
0004    *
0005 from customer t1, outer orders t2
0006 where
0007    t1.customer_num = t2.customer_num
0008
0009 Estimated Cost: 8
0010 Estimated # of Rows Returned: 28
0011
0012   1) Abigail.t1: SEQUENTIAL SCAN
0013
0014   2) Abigail.t2: INDEX PATH
0015
0016     (1) Index Keys: customer_num   (Serial, fragments: ALL)
0017         Lower Index Filter: Abigail.t1.customer_num =
0018         Abigail.t2.customer_num
0019 NESTED LOOP JOIN
```

0020
0021

In an outer join, one table is referred to as the **dominant table**, while the other is subordinate. The dominant table is the one that will return records whether or not a matching record exists in the other **subordinate table.** Consider the following:

- There are two table join orders to the above query; table one then two, or table two then one.
- If the dominant table is read first, then we have all of the records we need from table one to satisfy the query. We join to the subordinate table and return joined records when found. If a table two join record is not found, then one record from table one is returned with SQL NULL values filling the columns that normally come from the subordinate table matching record.
- *If we read the subordinate table first,* we process the join as though performing a normal (non-outer) SQL SELECT; reading the subordinate table first gives us join column key values as they as found to exist in the subordinate table. However, after query processing, we somehow have to produce those rows (the join column key values) from the dominant table that do not exist in subordinate table. We would have to execute a query similar to:

  ```
  SELECT dominant-table-primary-key
  FROM dominant-table
  WHERE dominant-table-primary-key NOT IN
  (SELECT subordinate-table-foreign-key FROM subordinate-table);
  ```

  The above would be a very expensive query; a NOT IN to a nested (one SQL SELECT embedded within another) SQL SELECT can produce many, many disk I/O operations.

Note: In an SQL SELECT with an outer join criteria, the dominant table is always processed before the subordinate table. Obviously, the mere presence of an outer join criteria greatly affects table join order, which can then greatly and negatively affect overall query performance. If you can avoid use of the outer join criteria, do so.

From the text presentation of the query plan in Example 10-3 on page 552:

- Line 0005 displays that the Customer table is dominant, because the outer keyword appears before the Orders table; the presence of the outer keyword before the Orders table makes it subordinate.
- Line 0012 displays that the dominant table is read first, as expected.

- Because there is no filter criteria on t1 (the Customer table), t1 is processed with a sequential table access method; line 0012. (We are returning every record from table t1. It is more efficient to access this table sequentially than via an index; that versus using an index and then a retrieval of the full record from the SQL table proper.)

Example 10-4 displays the same query as in Example 10-3 on page 552, however, the outer keyword is applied to the Customer table, not the Orders table as before:

Example 10-4 Example of outer join with outer keyword having been moved

```
0000
0001 QUERY:
0002 ------
0003 select
0004    *
0005 from outer customer t1, customer t2
0006 where
0007    t1.customer_num = t2.customer_num
0008
0009 Estimated Cost: 8
0010 Estimated # of Rows Returned: 28
0011
0012   1) Abigail.t2: SEQUENTIAL SCAN
0013
0014   2) Abigail.t1: INDEX PATH
0015
0016     (1) Index Keys: customer_num   (Serial, fragments: ALL)
0017         Lower Index Filter: Abigail.t1.customer_num =
0018         Abigail.t2.customer_num
0019 NESTED LOOP JOIN
0020
0021
```

From Example 10-4:

- The table order has changed with the movement of the outer keyword. With t2 now being the dominant table, that table is processed first.
- Without the presence of filters and other query criteria, nearly every aspect of Example 10-3 on page 552 and Example 10-4 remain the same; the costs, and others. In a more realistic SQL SELECT, the affect of the *outer* keyword on query cost and amount of resource to process the query would become evident.

The next example, Example 10-5, displays what occurs if we attempt to use query optimizer directives to force processing of the subordinate table before the dominant table. (If allowed, this query would perform very poorly relative to the other choice for query plans.)

Example 10-5 Use of query optimizer directives and outer join

```
0000
0001 QUERY:
0002 ------
0003 select --+ ordered
0004    *
0005 from outer customer t1, customer t2
0006 where
0007    t1.customer_num = t2.customer_num
0008
0009 DIRECTIVES FOLLOWED:
0010 DIRECTIVES NOT FOLLOWED:
0011 ORDERED  Outerjoin nesting not compatible with ORDERED.
0012
0013 Estimated Cost: 8
0014 Estimated # of Rows Returned: 28
0015
0016   1) Abigail.t2: SEQUENTIAL SCAN
0017
0018   2) Abigail.t1: INDEX PATH
0019
0020     (1) Index Keys: customer_num   (Serial, fragments: ALL)
0021         Lower Index Filter: Abigail.t1.customer_num =
0022         Abigail.t2.customer_num
0023 NESTED LOOP JOIN
0024
```

From Example 10-5:

- The query optimizer directive we used was *ordered*, as shown on line 0003. The ordered query optimizer directive keyword states that the table join order must follow the names of the tables as they are listed in the SQL FROM clause.
- Line 0011 shows that the query optimizer directive was not followed, because it is expected that execution of this query plan would not be performant.

As stated, the query optimizer applies these five rules in order until a table processing order is determined. In each of the three queries above, rule one fully determined table processing order. The query optimizer did not need to refer to rules two through five to determine table order.

10.5.2 Rule 2: (Non-outer, normal) table joins

Outer joins are not that common. Out of 100 queries, you can see outer joins two or four times out of 100. Joins (non-outer or standard joins) usually are expected to be everywhere. In any given query, each table is expected to be joined to at least one other table. If any given table is not joined to at least one other table, this results in an outer Cartesian product.

Earlier, when discussing nested loop (table) join methods, we introduced the concept of worst table first. Given a two table SQL SELECT, where the tables are joined, and both sides of the join pair are supported via an index, make the first table processed to be the least efficient of the two choices; that is, generally make the second table (the table access operation that is performed numerous times), the more efficient of the two.

Example 10-6 displays three queries, two with two tables and the last with three tables.

Example 10-6 Three examples of queries with joined tables

```
0000
0001 QUERY:
0002 ------
0003 select
0004    t1.*, t2.*
0005 from customer t1, orders t2
0006 where
0007    t1.customer_num = t2.customer_num
0008
0009 Estimated Cost: 7
0010 Estimated # of Rows Returned: 22
0011
0012   1) Abigail.t2: SEQUENTIAL SCAN
0013
0014   2) Abigail.t1: INDEX PATH
0015
0016     (1) Index Keys: customer_num   (Serial, fragments: ALL)
0017         Lower Index Filter: Abigail.t1.customer_num =
0018         Abigail.t2.customer_num
0019 NESTED LOOP JOIN
0020
0021
0022 QUERY:
0023 ------
0024 select
0025    t1.*, t2.*
0026 from orders t1, items t2
0027 where
```

```
0028    t1.order_num = t2.order_num
0029
0030 Estimated Cost: 9
0031 Estimated # of Rows Returned: 64
0032
0033   1) Abigail.t1: SEQUENTIAL SCAN
0034
0035   2) Abigail.t2: INDEX PATH
0036
0037     (1) Index Keys: order_num   (Serial, fragments: ALL)
0038         Lower Index Filter: Abigail.t1.order_num =
0039         Abigail.t2.order_num
0040 NESTED LOOP JOIN
0041
0042
0043 QUERY:
0044 ------
0045 select
0046    t1.*, t2.*, t3.*
0047 from customer t1, orders t2, items t3
0048 where
0049    t1.customer_num = t2.customer_num
0050 and
0051    t2.order_num = t3.order_num
0052
0053 Estimated Cost: 15
0054 Estimated # of Rows Returned: 62
0055
0056   1) Abigail.t3: SEQUENTIAL SCAN
0057
0058   2) Abigail.t2: INDEX PATH
0059
0060     (1) Index Keys: order_num   (Serial, fragments: ALL)
0061         Lower Index Filter: Abigail.t3.order_num =
0062         Abigail.t2.order_num
0063 NESTED LOOP JOIN
0064
0065   3) Abigail.t1: INDEX PATH
0066
0067     (1) Index Keys: customer_num   (Serial, fragments: ALL)
0068         Lower Index Filter: Abigail.t2.customer_num =
0069         Abigail.t1.customer_num
0070 NESTED LOOP JOIN
0071
```

You can easily witness outer joins and their impact. Non-outer (normal) joins are perhaps less obvious because their presence is largely expected. From the first query in Example 10-6 on page 556, lines 0000 through 0020, see the following:

- Rule 1, an outer join is not present, so query optimization continues with rule 2, normal (non-outer) joins.
- The two tables are joined, as shown on line 0007. With just two tables, the presence of the join criteria alone is not enough to determine table join order. Actually, a variation of rule 3 determines table join order for this query.
  - Rule 3 is discussed below. For now, we will state that table join order was determined by the principle of worst table first.
  - On lines 0017 through 0018, the join columns are stated to be:

    ```
    t1.customer_num = t2.customer_num
    ```

  - T1 is the Customer table, which has a unique index on customer_num (not shown in the example). T2 is the Orders table, which has a duplicate index on customer_num (not shown in the example).
  - As a general rule the better index is the unique index, since the duplicate index permits numerous values. It is better to process the duplicate table first, and use the unique index to process the repetitive operation. The worst table is that with the duplicate index.

From the second query in Example 10-6 on page 556, lines 0022 through 0041, see the following:

- The second query is much like the first, a unique index column joined to a duplicate indexed column. The query plan is essentially the same between the two queries.
- Evaluating the selectivity, or uniqueness of an index is, of course, a heuristic (greedy, general guideline) type of rule. These rules apply when all other criteria are essentially the same. See the following:
  - Unique indexes generally are more selective (return fewer records) than duplicate indexes.
  - Duplicate indexes with more columns (a composite index) are generally more selective than a duplicate index with fewer columns.
  - The query optimizer considers the lower cost of joining integer columns over character columns, which are more expensive.
  - Given two indexes, both of equal selectivity, the query optimizer will give preference to any index which may also satisfy the SQL ORDER BY clause. Example as shown:

    ```
    SELECT * FROM one_table WHERE
    ```

    ixcol1 = 10 AND ixcol2 = 15

ORDER BY ixcol1;

Using the index on ixcol1 to process the query also returns the records in a presorted manner, and is subsequently more efficient.

From the third query in Example 10-6 on page 556, lines 0043 through 0071, see the following:

- This query gives us the first true glimpse of rule 2.
- The first table to be processed in this query is t3, the Items (order line items, a detail table to customer orders) table, determined by rule 3.
- The remaining table order is then determined by rule 2; table 3 joins with table 2 (Orders, the customer orders table), then table 2 with table 1 (Customer).

10.5.3 Rule 3: (Presence and selectivity of) Filter columns

Because outer joins (Rule 1) are rare, and all tables are expected to be joined (Rule 2), Rule 3 is often the rule that determines most table orders in the calculation of a query plan. Generally, the data regarding the selectivity (cost-based) of these filters is the most important determinant in the creation of the query plan; that is, which filter, if executed first, will reduce the largest number of records to be processed, which then reduces query cost.

Previously we were using the phrase *worst table first,* and that concept still applies (if all other options and costs are equal). However, with the introduction of filters to the query, we will now use the phrase, *as efficiently as possible, reduce the size of the tables that are being examined.* In other words, give preference to the best filter to effectively reduce the size of the tables being examined (reduce the amount of disk I/O to be performed).

Further, we also introduce the acronym, **FJS (filter, then join, then sort)**, which is the high level summary of processing and processing order that occurs in the execution of a query.

The following queries include a number of criteria meant to explore Rule 3, filter criteria, as well as reinforce Rule 1, outer joins, and Rule 2. These queries are depicted in Example 10-7.

Example 10-7 Several queries, demonstrating Rules 1 through 3

```
0000
0001  QUERY: the first query
0002  ------
0003  select
0004     o.order_num,
0005     sum (i.total_price) price,
0006     paid_date - order_date span
```

```
0007  from
0008     orders o,
0009     items i
0010  where
0011     o.order_date > '01/01/89'
0012  and
0013     o.customer_num > 110
0014  and
0015     o.order_num = i.order_num
0016  group by 1, 3
0017  having count (*) < 5
0018  order by 3
0019  into temp t1
0020
0021  Estimated Cost: 27
0022  Estimated # of Rows Returned: 1
0023  Temporary Files Required For: Order By  Group By
0024
0025    1) Abigail.o: SEQUENTIAL SCAN
0026
0027           Filters: (Abigail.o.order_date > 01/01/2089 AND
              Abigail.o.customer_num > 110 )
0028
0029    2) Abigail.i: INDEX PATH
0030
0031       (1) Index Keys: order_num   (Serial, fragments: ALL)
0032           Lower Index Filter: Abigail.o.order_num = Abigail.i.order_num
0033  NESTED LOOP JOIN
0034
0035
0036  QUERY: the second query
0037  ------
0038  select
0039     c.customer_num,
0040     c.lname,
0041     c.company,
0042     c.phone,
0043     u.call_dtime,
0044     u.call_descr
0045  from
0046     customer c,
0047     cust_calls u
0048  where
0049     c.customer_num = u.customer_num
0050  order by 1
0051  into temp t2
0052
0053  Estimated Cost: 5
0054  Estimated # of Rows Returned: 7
```

```
0055  Temporary Files Required For: Order By
0056
0057    1) Abigail.u: SEQUENTIAL SCAN
0058
0059    2) Abigail.c: INDEX PATH
0060
0061       (1) Index Keys: customer_num   (Serial, fragments: ALL)
0062           Lower Index Filter: Abigail.c.customer_num =
               Abigail.u.customer_num
0063  NESTED LOOP JOIN
0064
0065
0066  QUERY: the third query
0067  ------
0068  select
0069     c.customer_num,
0070     c.lname,
0071     o.order_num,
0072     i.stock_num,
0073     i.manu_code,
0074     i.quantity
0075  from
0076     customer c,
0077     outer (orders o, items i)
0078  where
0079     c.customer_num = o.customer_num
0080  and
0081     o.order_num = i.order_num
0082  and
0083     manu_code IN (‘KAR’, ‘SHM’)
0084  order by lname
0085  into temp t3
0086
0087  Estimated Cost: 22
0088  Estimated # of Rows Returned: 28
0089  Temporary Files Required For: Order By
0090
0091    1) Abigail.c: SEQUENTIAL SCAN
0092
0093    2) Abigail.o: INDEX PATH
0094
0095       (1) Index Keys: customer_num   (Serial, fragments: ALL)
0096           Lower Index Filter: Abigail.c.customer_num =
               Abigail.o.customer_num
0097  NESTED LOOP JOIN
0098
0099    3) Abigail.i: INDEX PATH
0100
0101           Filters: Abigail.i.manu_code IN (‘KAR’ , ‘SHM’ )
```

```
0102
0103      (1) Index Keys: order_num   (Serial, fragments: ALL)
0104          Lower Index Filter: Abigail.o.order_num = Abigail.i.order_num
0105  NESTED LOOP JOIN
0106
0107
0108  QUERY: the fourth query
0109  ------
0110  select
0111     *
0112  from
0113     stock
0114  where
0115     description like '%bicycle%'
0116  and
0117     manu_code not like 'PRC'
0118  order by
0119     description, manu_code
0120  into temp t5
0121
0122  Estimated Cost: 4
0123  Estimated # of Rows Returned: 13
0124  Temporary Files Required For: Order By
0125
0126    1) Abigail.stock: SEQUENTIAL SCAN
0127
0128          Filters: (Abigail.stock.description LIKE '%bicycle%'
              AND Abigail.stock.manu_code NOT LIKE 'PRC' )
0129
```

The first query in Example 10-7 on page 559 is contained between lines 0000 and 0034. See the following:

- This query has two tables, as shown on lines 0008 and 0009. The tables are joined on line 0015.
- Rule 1 (outer joins) is not in effect, since there are no outer joins.
- Rule 2 (table joins) does not help determine table order since there are only two tables, and both sides of the join pair can be served via an indexed table access method (there is an index on o.customer_num and i.order_num, not shown).
- The Orders table has two filters, as shown on lines 0011, and 0013. The Items table has no filters.
- There is also a filter on an aggregate expression, as shown on line 0017. If all things are equal, this filter has a lower priority since we must filter and then join these two tables before we can consider a filter on an aggregate expression.

- This filter is on the cardinality between the two tables. That is, how many join pairs have fewer than five records. We must fully process this query in order to satisfy execution of this filter criteria.
- If the aggregate filter were on a MIN() or MAX(), or an indexed join column or similar filter criteria, there would then be more processing choices that the query optimizer could consider since MIN() and MAX() can be known before the join *under the very best conditions.*

- Line 0025 displays that the table join order begins with the Orders table, (table “o”). If table o has 100 records, if the filter on table o reduces these 100 records to just 10 records, and if table i has 100 records, effectively the two choices were:
  - Read table “o” at 100 records, and apply the filter to produce 10 records. Then join 10 filtered “o” records to 100, an operation of 1000 records.
  - Or join table “i” at 100 records to table ‘o” at 100 records, an operation of 10,000 records, only to filter 10% of table “o” to produce the final 1000 records.
- Due to rule 3, the presence of filter records on table “o”, the table join order is table “o” then “i”.
- Because there are two filters on columns in table “o”, and these columns are not both members of one (composite) index, this table is read sequentially. Other choices were:
  - Read table “o” via an index on order_date, assuming one is present, and apply the filter criteria for order_date. But, any qualifying records in table “o” must also meet the criteria on customer_num. Since customer_num is not a member of any order_date index, we would have to access the record in the SQL table proper to evaluate the filter criteria for customer_num.
  - The converse of the above is read any index for customer_num, then process the filter for order_date.
  - Or, read two indexes, one for customer_num and one for order_num. Compare both sets of results and produce an intersection, in other words, return rows which are common to both result sets. Creating an intersection of both result sets means both lists would have to be sorted, then compared key by key, and so on.
  - For cost reasons, the query optimizer chose to sequentially scan the Orders table. (There was no index on order_date. The query optimizer will create indexes to join tables. The query optimizer will not create indexes just to process filters; creating an index requires reading the SQL table proper, which would then allow for filter criteria examination without the overhead of building the index.)

- The table access method in table one (Orders) is sequential, line 0025.
- A nested loop method, line 0033, joins table two (Items) to table one (Orders).
- The access method on table two (Items) is indexed, line 0029.
- A temporary file is required to process the SQL GROUP BY clause. A SQL GROUP BY clause needs to sort records to perform its data sub-grouping.
- The SQL ORDER BY clause is on a differing column set. Therefore, a second sort operation will be performed to process the SQL ORDER BY. This too requires a temporary file.
  - If the SQL GROUP BY and SQL ORDER BY were on the same columns and column orders, this second sort is not required.
  - The SQL GROUP BY clause is on columns 1, then 3. The SQL ORDER BY clause is on column 3. If the SQL HAVING clause had a condition where column 1 equaled a single value, then the value in column 1 essentially equals a (new) constant, whatever that value is. For example:

    ```
    SELECT * FROM t1 ORDER BY "A", column2;
    ```

    is logically equivalent to:

    ```
    SELECT * FROM t1 ORDER BY column2;
    ```

    This is true because “A” is constant, it is unchanging. Consider the following:

    ```
    SELECT * FROM t1 WHERE column5 = 10 ORDER BY column5, column6;
    ```

    is logically equivalent to:

    ```
    SELECT * FROM t1 WHERE column5 = 10 ORDER BY column6;
    ```

    From the example query, line 0017, we see that the aggregate expression column can have values in the range of 1 to 4, which is not a single constant value. Therefore, the SQL ORDER BY clause provides different sort criteria than the SQL GROUP BY clause, two different sorts.

Note: The query rewrite capability of the query optimizer would detect these types of logical equalities and rewrite the query to have the SQL GROUP BY clause match the SQL ORDER BY clause, thus reducing a now redundant sort operation. Example as shown:

- These two queries are logically equal because column5 is constant.
- As specified by the filter criteria “column5 = 10”

```
SELECT * FROM t1 WHERE column5 = 10 ORDER BY column5, column6;

SELECT * FROM t1 WHERE column5 = 10 ORDER BY column6;
```

This specific case of the query rewrite capability is called the **pseudo order-by clause**.

- Line 0019, the retrieve results into a temporary table is used only for the benefit of the Redbook authors. This call did not affect query processing. All of the queries in this section are retrieved into a temporary result sets table.
- Rule 3 determined the table order for this query.

The second query in Example 10-7 on page 559 is contained between lines 0036 and 0064. See the following:

- There are two tables in this query, as shown on lines 0046 and 0047.
- The tables are joined, line 0049, and there are no filters on either table.
- An SQL ORDER BY clause exists on the Customer table, customer_num column. Even though an index exists on this column, and that index is used to perform the join into table two, rows are still not being returned in sort order. The first table to be processed is cust_calls via a sequential access method; therefore, rows are read in random sequence.
  - Why not read the Customer table first via customer_num to match the requested SQL ORDER BY clause sort order and thus avoid a sort operation? That is certainly an option. Returning to the acronym FJS, the query optimizer is concerned first with filtering and joining records before sorting.
  - Not having any filters on any of the tables (no selectivity) meant that it would be measurably more efficient to read at least one of the two tables sequentially. The *worst table first* principle has the query optimizer read cust_calls (Customer Calls) first, and use the unique index into Customer to process the join. The amount of physical I/O to read Customer or cust_calls (Customer Calls) via an index when we are returning every row in the table is prohibitive relative to a sequential scan.

– After the optimal table order was determined, the SQL ORDER BY clause could not be served via the same index used to process the query.

Note: Another specific query rewrite capability is **transitive dependency**. Transitive dependency is an algebraic term. In effect,

```
IF A = B, AND B = C, THEN, BY DEFINITION A = C
```

From the example above, this means,

- These two queries are logically equal, based on the principle of transitive dependency.

  SELECT * FROM t1, t2 WHERE t1.col = t2.col ORDER BY t1.col;

  SELECT * FROM t1, t2 WHERE t1.col = t2.col ORDER BY t2.col;

- This query was determined largely by Rule 3; the expected uniqueness of the join columns made the Customer table more desirable as a second join table, and the lack of filter criteria called for a sequential scan of the worst table first, the cust_calls (Customer Calls) table.
- A temporary file is required to support the SQL ORDER BY clause.

The third query in Example 10-7 on page 559 is contained between lines 0065 and 0106. See the following:

- Query three references three tables, as seen on lines 0076 and 0077.
- Customer is the dominant table in an outer join, line 0076.
- The Orders table and Items table are both subordinate to Customer, line 0077. Although, tables Orders and Items have a normal join relationship between themselves.
- Based on Rule 1, the Customer table will be processed first, as seen on line 0091.
- Because Customer has no filter criteria, this table will be read sequentially. (Absence of filter criteria on Customer between lines 0078 through 0083.)
- Order was processed next due to Rule 2; it is joined directly with Customer.
- A temporary file is required to process the SQL ORDER BY clause, which is on Customer.lname. There was no index on this column, and certainly not one that was already in use to process Customer.

The fourth query contained in Example 10-7 on page 559 is contained between lines 0108 and 0129. See the following:

- This is a single table query, so table join order and join methods are not a concern. The remaining element to determine is table access method.

- There are two filters on this table, as shown on lines 0115 and 0117.
  - There is no index to support the filter on Stock.description, so this filter will not call for an indexed table access method.
  - While there may have been an index on Stock.manu_code, this column is evaluated as an inequality; return everyone from the phone book whose name is not Mary Alice. That results in a lot of records and not a very selective filter. If there were an index on Stock.manu_code, it would not be used to process this filter based on lack of selectivity, (negative of index, poor selectivity of a filter).
- This leaves a table access method of sequential.
- A temporary file is required to process the SQL ORDER BY clause.

10.5.4 Rules 4 and 5: Table size and table cardinality

Most often, Rule 3 in the query optimizer determines table order. Even if no filters are present in the query, table joins are expected to be in the query. The selectivity (cardinality) of the join relationships will determine table order.

Rules 4 and 5 of the query optimizer are rarely observed. See the following:

- If Rules 1 through 3 cannot determine table order, then Rule 4 will be invoked. (Remember the rules are applied sequentially, until table order is determined.)
- Rule 4 is table size. Assume a two table join with all other factors equal, one of these two tables will be processed first based on the size of the two tables.
  - Which table is read first is based on cost. It used to be true that the larger table would be processed and have an index built on it to process the join. Today, the smaller table is likely to be the one processed and have an index built upon it; because the smaller table produces a smaller index, an index which is more likely to fit in cache.
  - The points above assume that neither table is supported via an index on a join column, which is most often the case.

Rule 5 is table cardinality. The word cardinality is seemingly used in a variety of ways in this IBM Redbook. Consider the following:

- Technically the *term cardinality is defined to mean the number of elements in a set or group.*
- In this IBM Redbook, we also discuss the cardinality between two SQL tables; that is, a one to many style relationship between a parent and then detail table, and others. Cardinality in this context means the numerical relationship between two data sets. If each state or province in a nation contains exactly 200 cities, then the cardinality between province and city is 1:200.

- In Rule 5 of the query optimizer, the phrase *table cardinality* refers to the table's named order in the SQL SELECT FROM clause. Do not be overly concerned that table name order is a determining factor in the creation of the query plan. If a given query is determined by Rule 4 or 5, a number of other errors exist in the run-time environment, and it is likely that all queries are performing poorly.

The query optimizers from the various relational database software vendors operate via a set of guidelines that are based on sound engineering principles. The rules of a query optimizer are based on observed facts; facts such as indexed retrieval is generally faster than sequential table access, unless (...). Each vendor also uses algebraic formulas to assign costs to each operation, decision, and so on. These exact algebraic formulas are not published, and can be considered the proprietary and trade secret property of each vendor. Thus far, we have discussed the elements of these costs and formulas, and the logic and application behind these ideas. In the next section, we discuss various other query optimizer features that involve the idea of query costing.

10.6 Other query optimizer technologies

Thus far in this chapter, we have completed the following:

- Reviewed a reasonably complex and real world SQL SELECT example as a means to learn about query optimizers *by example*. This review included the most common query optimizer algorithms and tasks.
- Then we reviewed the five rules of a query optimizer with a number of simplistic SQL SELECT examples.
- Through all of this, we considered the idea of costing, which is the underlying foundation of all query optimizer behavior.

In this section, we are going to introduce (or review) various query optimizer topics including; query rewrite, pipelined sorts, (detailed) query optimizer hints, and other topics.

10.6.1 Query rewrite

During the query optimization phase, the query optimizer may change the syntax of the given SQL SELECT to one which is logically equivalent; meaning a query which returns the same data set, but with different syntax allowing for query processing in a more efficient manner. Thus far in this chapter, two query rewrite scenarios have been introduced. At this time, we review these query rewrites and a few new ones.

Pseudo order by clause

Given a table, t1, with a composite index on column1 and column2, consider the following:

```
SELECT * FROM t1 WHERE column1 = 55 ORDER BY column2;
```

Since every record returned by this query has a column1 value of 55, this query could be rewritten as:

All queries in this block are logically equal:

```
SELECT * FROM t1 WHERE column1 = 55 ORDER BY column1, column2;
SELECT * FROM t1 WHERE column1 = 55 ORDER BY “55”, column2
SELECT * FROM t1 WHERE column1 = 55 ORDER BY “X”, column2;
```

These queries are logically equivalent; meaning they each return the same data set. The query optimizer will detect this condition and choose to use the concatenated (single) index on columns; (column1, column2). Since the data is read in collation (index sort order) sequence, no sorting needs to be performed on these records. They are automatically presorted and cost less to process.

Knowing that the query optimizer observe this conditions, and rewrites the query for better processing, should the application programmer manually write the query to explicitly make use of pseudo order by? Ten years ago the answer to this questions was yes. Today, perhaps nearly every query optimizer should catch this condition. Manually rewriting queries for optimization is a good idea certainly, unless their rewrite may make the syntax or purpose of the query unclear.

Note: Generally, query processing has three logical phases; FJS, filter, then join, then sort. If the same (single) index used in the filter and join phases can also serve the group by and order by clause, then the query processor sort package does not need to be invoked; records are already being processed in sort order. This is referred to as a *pipelined sort operation*.

More commonly, queries do their filters and joins, and then have to be gated (held) and sent to the query processor sort package. This condition is known as a *non-pipelined sort operation*.

The first record produced in a pipelined query can be sent immediately to the application, since records are presorted. A non-pipelined query has to hold all records, and sort them before returning the first record. This is because the last record produced by the query during filtering and joining may sort as the first record to be returned to the application. Pipelined sorts can obviously give the appearance of a faster query, because of their *first row first* capability.

Few group by or order by clauses can be pipelined, as the need to filter and join rarely align themselves with the group by and order by. *There are strategies to increase the likelihood of observing a pipelined query;* for example, ensure that the columns in the order by clause match those of a composite index used to filter and join. (Add columns to the index to match the needs of filters, joins and sorts. The column order must match in both the index reference and order by clause.)

Transitive dependency

Generally, every index contains columns from just one SQL table. Multi-table indexes, (indexes containing columns from two or more tables) are available in better relational database servers; these multi-table indexes are sometimes referred to as star indexes. Multi-table indexes are rare, and are often read only.

Consider the following:

These two queries are logically equal:

```
SELECT * FROM t1, t2 WHERE t1.col1 = t2.col1 AND
t1.col2 = t2.col2
ORDER BY t1.col1, t2.col2;
SELECT * FROM t1,t2 WHERE t1.col1 = t2.col1 AND
t1.col2 = t2.col2
ORDER BY t1.col1, t1.col2;
```

The first query above lists columns from two or more tables in the ORDER BY clause. But, t2.col2 equals t1.col1, as stated in the WHERE clause. Therefore, the query can be rewritten as shown in the second example. The second query more clearly displays that a single index can process the order by clause, and hopefully, will match in an index used above for the join condition resulting in a pipelined sort.

Be algebraic definition, transitive dependency is represented as,

```
IF A = B, AND B = C, THEN, BY DEFINITION A = C
```

In the application of queries, consider the following:

These two queries are logically equal:

```
SELECT * FROM t1, t2
WHERE t1.col1 = t2.col1 AND t1.col2 = t2.col2
AND t1.col1 = 65 AND t2.col2 = 14;
SELECT * FROM t1, t2
WHERE t1.col1 = t2.col1 AND t1.col2 = t2.col2
AND t2.col1 = 65 AND t1.col2 = 14;
```

The message here is that the query optimizer can use transitive dependency to consider alternate, but logically equal, query plans. If an index supports t2.col1 and t2.col2 but not both t1.col1 and t1.col2 (the index supports only t1.col1), transitive dependency may uncover a better query plan.

Semantic equality

Consider the following:

```
-- colPK is the primary key for table t1
SELECT DISTINCT colPK FROM t1;
SELECT colPK from t1;
```

Because the column list is known to produce a unique product set, either by unique column constraints or check constraints, the call for the DISTINCT keyword in the first SQL SELECT above is redundant and may be removed.

How would an example like the first SQL SELECT above ever occur? Very often user graphical reporting tools allow for operations to be checked (called for), when the given operations are unnecessary. (The user just checked the visual control because it was present.)

Another example is shown. Consider the following.

These two queries are logically equal:

```
SELECT * FROM t1 WHERE t1.col1 > ALL (SELECT col2 FROM t2);
SELECT * FROM t1 WHERE t1.col1 > (SELECT MAX(col2) FROM t2);
```

In the second SQL SELECT statement, only one comparison is done for every record in table t1.

Decoding of SQL VIEWS

Consider the following,

```
CREATE VIEW v1 AS SELECT * FROM t1 WHERE code = "CB";
SELECT * FROM v1 WHERE code = "CB";
```

One of the tasks of the command parser or query optimizer is responsible for the decoding of any SQL VIEWS which the given SQL command may reference. In the example above, a redundant filter criteria is produced; the SQL VIEW already restricts (filters) the resultant data set as is requested in the SQL SELECT.

Some time ago, early in the technology cycle, SQL VIEWS were executed and had their results placed into a temporary table before the target SQL command would be processed. Example as shown:

The following example is no longer current (true)

```
CREATE VIEW v1 AS SELECT state.*, COUNT(*) FROM states, cities
WHERE state.stateCode = city.stateCode
GROUP BY state.stateCode;
```

The query above returns the columns from the state table, along with a count of the number of cities that state contains. Given a query equal to:

```
SELECT * FROM v1 WHERE stateCode = "IN";
```

The query processor would first calculate the data set represented by the SQL VIEW; that is, it would read, group, and calculate counts for all 50 states. Then the query processor would read from this temporary table to process the SQL SELECT, a read of just one state, not all 50.

The above is no longer an accurate representation as to how a modern relational database server would process the example query. This example is mentioned as another example of the operations of query rewrite, and the benefit of decoding SQL VIEWS as part of parsing and query optimization.

Encoding of SQL VIEWS

Generally, SQL VIEWS are logical structures; they represent SQL SELECT strings that front a number of physical SQL tables. In addition to providing a macro, or shorthand manner to access a number of tables, SQL VIEWS can also be used to provide unique security schemes. While SQL security allows the database administrator to restrict access on the table or column level (vertical security), SQL VIEWS that returns a subset of records within a table can be used to provide horizontal security. For example, Bob can only see records from a subset of states or provinces within a country, not all states and provinces.

Recently, SQL VIEWS have also become physical structures. Some relational database vendors call these structures *materialized views, materialized query tables*, or other similar names. The syntax of a given relational database vendor for this capability may be SQL CREATE MATERIALIZED VIEW, SQL CREATE TABLE {name} AS (SELECT ...), or another command. Below is an example of this type of command, this one is an SQL CREATE TABLE,

Approximate syntax

```
CREATE TABLE bad_account AS
   (SELECT customer_name, customer_id, a.balance cu
   FROM account a, customers c
   WHERE status IN ('delinquent','problematic', 'hot')
   AND a.customer_id = c.customer_id)
DATA INITIALLY DEFERRED REFRESH DEFERRED ) BY DB2;
```

Since materialized views are real, physical storage structures, the query optimizer query rewrite feature can use these definitions to reduce processing load. If a given SQL SELECT calls to read from the native Account and Customers tables, as listed above, the query rewrite capability redirects the query to make use of the materialized view, thus saving the associated join costs.

Rewrite nested queries as joins

Given the following nested query:

```
SELECT * FROM t1 WHERE t1.col IN
   (SELECT colx FROM t2);
```

One of two conditions will be present in the query above:

- The nested (inner) query will return zero or one records; a cardinality of M:1 (many to one).

- The nested (inner) query will return more than one record; a cardinality of 1:M (one to many).

Note: The case of an IN or EXISTS clause is referred to as a *semi-join*. To process the (join), the record in the nested table needs merely to exist, it is not actually attached to the record of the outer table.

If the query above has a cardinality of many to one, then the query can be rewritten as a standard join. Example as shown:

```
SELECT t1.* FROM t1, t2 WHERE t1.col = t2.colx;
```

If the query above has a cardinality of one to many, then a join between these tables can produce numerous t1 records where previously only one was produced. The query optimizer will consider the choice of:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.col = t2.colx;
```

Why consider the query rewrite at all? The data set produced inside the inner query, table t2, may be massive and may force the production of a temporary table. (Consider a more complex inner query with filters and possibly other table joins, removing the option of just reaccessing table t2 for every record in table t1, the outer table.)

Correlated subqueries

The query above was a nested query, (also referred to as a subquery). In short, the inner query needs to be executed only one time, (before processing the outer query). The inner query needs to be executed only one time, because the resultant data set is unchanging; it is the same for every record in the outer query.

A correlated subquery is a modification to a standard subquery. Correlated subqueries are prohibitively expensive, although they are a legal operation.

Note: Correlated subqueries are so costly that some relational database server products will return a special diagnostic code to the application when a correlated subquery is received. (The thought is that the application may then want to cancel the request for this operation.)

Listed below is an example of a correlated subquery:

```
SELECT * FROM t1 WHERE t1.col IN
   (SELECT colj FROM t2 WHERE t2.col4 = t1.colz);
```

Because the inner query makes reference to a column value from the out portion of the query, (in this case the inner query refers to column t1.colz), each record in the out portions of the query causes the inner query to be entirely re-executed.

The query optimizer query rewrite capability will attempt to change a correlated subquery to a standard join. However, in most cases, the correlated subquery needs to be removed by a modification to the underlying data model (a modification that we do not discuss here).

Due to query rewrite, below is an example of a correlated subquery which can be rewritten as a (non-correlated) nested query,

These two queries are logically equal

```
SELECT * FROM t1 WHERE t1.col1 = 10 AND t1.col2 =
   (SELECT MAX (t2.col2) FROM t2 WHERE t2.col1 = t1.col1);
SELECT * FROM t1 WHERE t1.col1 = 10 and t1.col2 =
   (SELECT MAX(t2.col2) FROM t2 WHERE t2.col1 = 10);
```

The example above is also an example of transitive dependency.

OR Topped queries

Consider the following:

AND topped query

```
SELECT * FROM t1 WHERE col1 = 4 AND col2 = 9;
```

OR topped query

```
SELECT * FROM t1 WHERE col1 = 4 OR col6 = 15;
```

The listing above shows two distinct queries; differing goals and differing resultant data sets. The difference between AND topped queries and OR topped queries is shown below:

- These are both single table queries. The primary task of the query optimizer is to determine the table access method.
- In the case of the first query, a single concatenated index on both t1.col1 and t1.col2 would allow for an indexed table access method. This is true because of the AND condition.
- In the case of the second query, the OR condition initially prevents the use of an indexed table access method.
  - A single concatenated index on both t1.col1 and t1.col6 will not serve the query. Because of the OR condition, a record may have a t1.col1 value of

4 *or* a t1.col6 value of 15. The OR condition makes these two filter criteria to be separate conditions.

  - A single index can serve the t1.col1 filtering needs, *or* the t1.col6 filtering needs, but not both.
  - A single index can not serve both filter criteria because of the negation of index, non-anchored composite key. In effect, the OR topped query says, find everyone in the phone book with the last name of “Schwartz” or with the first name of “Arnold”. The phone book is not sorted (anchored) on first name.

- One solution to an OR topped query is to perform an **OR-union transfer**. Example as shown:

```
SELECT * FROM t1 WHERE col1 = 4

UNION

SELECT * FROM t1 WHERE col6 = 15;
```

- A UNIONED SELECT allows for two or more distinct SQL SELECTS to be executed as though they are just one simple SQL SELECT. These two or more SQL SELECTS can be from the same or different tables. About the only requirement is that each participating SQL SELECT must return the same number of columns, with compatible data types (numeric to numeric, and character to character).
- An OR-union transfer should be considered when the following is observed:
  - Each participant in the transfer makes use of an indexed table access method.
  - The final query runs faster.
- By its syntax, a UNIONED SELECT will remove duplicates from the final data set; that is, from the example above, a record may be both t1.col1 = 4 AND t1.col2 = 15. However, that (single) record cannot be returned to the application twice just because of a query rewrite operation.
- A UNION ALL SQL SELECT will not call to remove duplicates from the participating queries and, therefore, is more efficient than a standard UNIONED SELECT. IF it is determined that each participating query produces a distinct data set, they by all means issue the query as UNION ALL.

Note: One question is whether or not a relational database supports a **multi-index scan.** That is, can the relational database natively use two or more indexes as a table access method for one table? And, under what circumstances is it allowed? Multi-index scan needs to have the ability to remove duplicate records from the (single) source table.

Redundant join elimination

Consider the following,

```
SELECT * FROM t1, t2, t3 WHERE t1.col = t2.col
AND t2.col = t3.col AND t3.col = t1.col;
```

There are redundant join criteria in the example SQL SELECT above. Since t1 joins with t2, and t2 joins with t3, the query processor does not need to join table t3 to table t1. The query rewrite capability of the query optimizer will consider the table t3 to table t1 join path, but will remove at least one of the join paths since it is unnecessary.

Shared aggregate calculation

Consider the following:

```
The first query has 2 SUM() and 1 COUNT()
The second query, produced by the query optimizer
query rewrite, has 1 SUM() and 1 COUNT()

Remember that an AVG() is a SUM() / COUNT)()
```

Original query

SELECT SUM(col) AS s, AVG(col) AS s FROM t1;

Query rewritten by query optimizer

SELECT s, s / c FROM (SELECT SUM(col) s, COUNT(col) c
FROM t1);

The query optimizer query rewrite capability will detect the shared aggregation opportunity and rewrite the first query above, as displayed in the second query above. This example also displays an optimal application of dynamic table reference in the FROM CLAUSE to a SQL SELECT. This technique can be used extensively in business intelligence style queries.

Operation movement

Consider the following,

```
SELECT DISTINCT * FROM t1, t2 WHERE t1.col = t2.col;
```

If the column t2.col is known to be unique (it has a unique index constraint placed upon it), then the DISTINCT set operand can be performed on table t1 alone.

The DISTINCT set operand need not be performed on the larger (wider) data set which is the product and tables t1 and t2 having been joined.

In this manner, DISTINCT set operands can be pushed lower into the execution of the query plan, or higher; whichever results in a lower total query cost.

Predicate translation

Consider the following:

```
Code Fragment 1
WHERE NOT (col1 = 10 OR col2 >3);
Code Fragment 2
WHERE col = YEAR("1990-04-01");
```

The first example can be rewritten as:

```
WHERE col1 <> 10 AND col2 <= 3;
```

This query optimizer query rewrite operation removes the OR topped nature of the query and increases likelihood of an indexed table access method.

The second example is rewritten as:

```
WHERE col = 1994;
```

This rewrite removes the function invocation on every record that is processed; further, the result of this given function was an integer data type which is more efficient than the DECIMAL(16) or structure that often represents SQL DATETIME column types.

Consider the following:

```
SELECT * FROM t1 WHERE integerColumn = "123";
```

The query optimizer rewrite capability translates a literal expression into an equal data type whenever possible; “123” is a numeric data type and is converted to an INTEGER to facilitate the join with greater efficiency. The query optimizer query rewrite capability will encrypt literal value filter criteria to compare with encrypted column values *when the encryption key is set at the table level.* This operation is impossible for systems or applications that encrypt column values at the record level.

Note: Most encrypted data types take 33% or more disk storage to record than non-encrypted data. Omitting the password hint that accompanies individual column values can eliminate as many as 50 bytes per record.

10.6.2 Multi-stage back-end command parser

Technically, in this entire chapter we discuss the query optimizer (which also implies discussion of the query processor). The multi-stage back-end is headed by a command parser, and there are certain technologies you can employ at the parser phase that will also improve SQL performance.

Bypassing the command parser

Consider the following:

Mixture of application program code and SQL

```
FOR i = 1 to 1000
   INSERT INTO t1 VALUES (i);
END FOR
```

The program code fragment above contains a mixture of application program code and embedded SQL commands, as many SQL based applications do. On a given system with a given performance, imagine that the program loop above executes in 60 seconds. Now consider the following:

Mixture of application program code and SQL

```
PREPARE s FROM "INSERT INTO t1 VALUES (?)";
FOR i = 1 TO 1000
   EXECUTE s USING i;
END FOR
```

While the first loop executes in 60 seconds, the second loop may execute in as little as 12 or 20 seconds. Why this is occurring is detailed in the list below:

- The application program will contain a mixture of program code and SQL statements. As a given SQL statement is encountered, this SQL statement is passed to the relational database server for processing via the agreed upon communication protocol. Only SQL commands are passed to the relational database server. The relational database server has no knowledge of non-SQL program code. The relational database server has no idea, for example, that the program has entered a looping construct and is about to execute the very same command 1000 times in a row.
- In the first loop listed above, each of 1000 SQL INSERT commands is sent to the SQL command parser as though the database server has never spoken to the application before. Each command has the following performed:
  - The statement is parsed for syntax errors.

- Any referenced tables and columns are checked to see if they exist, and to see if the given user (or role) has permission to perform the requested operation.
- Any semantic (data) integrity constraints are retrieved, as well as any triggers, check constraints, default column constraints, primary and foreign key definitions, and others.
- And more.

- Whether the loop moves from 60 seconds to execute to 12 seconds or 20, is dependent on the amount of time that was spent in the command parser, or was due to communication overhead.
- The command parser can be bypassed second and subsequent times, by registering SQL commands with the command parser for future use.
  - The SQL PREPARE statement registers the given command with the command parser.
  - At this point, the original INSERT is invoked via the SQL EXECUTE statement to the proper statement id.
  - The location of host variables is preserved via the question mark place holders, and then filled via the USING clause on execute.

Note: Most modern relational database server products offer SQL statement cache; that is, a listing of the 100 or so most recently received SQL commands. Given this feature, is it still necessary to use SQL PREPARE for optimization? Yes. Any newly received SQL command has to be parsed and compared to those which are resident in the cache. The SQL PREPARE statement bypasses this operation also.

SQL INSERT cursors (buffered inserts)

Most application programmers encounter an SQL CURSOR the first time they need to execute a SQL SELECT statement that returns more than one record from the database server. In this case, the SQL CURSOR manages the transport of the multi-row result set from the database server to the application client. The SQL CURSOR is like a gondola transporting records from the bottom of the mountain to the top. But, gondolas go both directions. SQL CURSORS can also be used to send records from the application to the database server.

Consider the following:

```
PREPARE s FROM “INSERT INTO t1 VALUES (?”);
DECLARE c CURSOR FOR s;
OPEN c;
```

```
FOR i = 1 to 1000
   PUT c USING i;
END FOR
FLUSH c;
CLOSE c;
```

An SQL INSERT CURSOR is one that fronts a SQL INSERT statement, as opposed to the standard (expected) SQL SELECT statement. An SQL INSERT CURSOR uses the SQL command verbs, PREPARE, DECLARE, OPEN and CLOSE, much like a standard SQL SELECT CURSOR. New commands include PUT and FLUSH.

All SQL CURSORS are specified to be a given size, which determines how many records it can contain. The SQL PUT verb places records in a buffer on the application client side of this Client/Server topology. As the communication buffer associated with this SQL CURSOR fills, the buffer is automatically flushed; meaning, that all records are sent from the client to the server at once. This is how a SQL INSERT CURSOR achieves its performance gain; in effect, the SQL INSERT CURSOR reduces the number of transports between the client and the server. Because this communication buffer may be left partially full after the completion of the looping construct, the buffer is manually flushed to ensure that the last, potentially partially full buffer is sent to the server.

The most difficult aspect of using SQL INSERT CURSORS is the error recovery. When records are sent one at a time, error checking is rather simple. When records can be sent variably and in larger numbers, error recovery is just a bit harder, but still very manageable.

From the example above, the original 60 second SQL INSERT loop could be moved to two or three seconds of execution time, an admirable gain.

Note: Not every relational database vendor supports SQL INSERT CURSORS, or they may use different terminology and commands to accomplish the same effect.

Multi-statement prepare

Similar to the first example of bypassing the command parser above, prepared statements can actually contain numerous SQL commands. Consider the following:

```
PREPARE s FROM “INSERT INTO t1 VALUES (?);
   UPDATE t2 SET (col4) = (?) WHERE col1 = (?);”
```

```
EXECUTE s USING v1, v2, v3;
```

In the example above, two separate SQL commands are expected to be executed together; that is, the application always needs these two distinct statements to run one after another. A reduction in the number of communications to the relational database server can be observed by preparing a multi-statement.

More on SQL statement cache

SQL SELECT statements that return more than one record, must be managed with a SQL CURSOR. The SQL SELECT associated with the SQL CURSOR is then registered with the command parser, and second and subsequent uses of this SQL CURSOR will bypass the command parser. SQL SELECT statements that do not return more than one record would include:

- Basically select only aggregates with no GROUP BY:

  ```
  SELECT COUNT(*), MAX(col), MIN(col) FROM t1;
  ```

- Select on equality to primary key column:

  ```
  SELECT * FROM t1 WHERE colPK = 17;
  ```

  ```
  SELECT * FROM t1 WHERE colPK = 19;
  ```

Note: Of the two SQL SELECTS shown above, use of the first SQL SELECT is preferred. While the primary key on table t1 may be colPK today, that may change. This often happens when two companies merge; the new primary key in the database is then pre-appended with a code for company, origin, or similar. While this may seem an unlikely event, the task of having to recode (port) thousands or millions of lines of SQL program code is not desirable.

From the perspective of the SQL statement cache subsystem, the last two SQL SELECT statements are unique, and do not allow for statement reuse (the ability for the statement to be recalled from the cache). We recommend you instead use host variables as shown in the following example:

- Host variable H set before SQL invocation

  ```
  SELECT * FROM t1 WHERE colPK = :H ;
  ```

10.6.3 Index negation

In general, index negation refers to the condition of having a perfectly useful and on line (pre-existing) index data structure, but for some reason that index may not be used to process a filter, join, or other task. Consider the following:

- **Negation of index, non-initial substring**; given the example,

```
SELECT * FROM phoneBook WHERE lastName LIKE "%son";
```

If an index exists on the phoneBook.lastName column it cannot be used because of a non-initial substring condition. Imagine trying to find every person in the phone book where the last name ends in "son", such as Hanson, Johnson, and Williamson. While the phone book is sorted by last name, it is not sorted by the last three characters of the last name. You would have to read the phone book sequentially and in its entirety to satisfy this query.

- **Negation of index, non-anchored composite key.** Consider the example:

  ```
  SELECT * FROM phoneBook WHERE firstName= "Chuck";
  ```

  While the phone book is sorted by the concatenation of last name, then first name, it is not sorted by first name alone. If you need to find every person in the phone book by first name, you have to read the entire phone book.

  The only exception to a non-anchored key condition is when all columns in the column list are members of the concatenated index. The phone book example fails to carry this point. Since an index is a vertical slice of a table, (it contains a subset of columns and all rows), the non-anchored read of just first name is better processed by sequentially scanning the (last name, first name) index; better than sequentially the SQL table proper which is larger (wider).

- **Negation of index, (poor) selectivity of filter.** Consider the examples:

  ```
  SELECT * FROM phoneBook WHERE lastName > "Ballard";
  ```

  ```
  SELECT * FROM phoneBook WHERE lastName != "Comianos";
  ```

  Both of these queries will return 90-98% or more of the phone book. Better to suffer through sequentially scanning the phone book, than to use the index with the physical disk I/O that incurs, and still return 98% of the phone book.

  The NOT often removes any selectivity of a filter of join criteria, by the nature that it too negates.

Partial index negation

A query similar to that shown below will cause some vendor relational databases to use only the leading column of a concatenated index. For example:

- SQL CREATE INDEX of concatenated (multi-column) index:

  ```
  CREATE INDEX i1 ON t1 (col1, col2);


  SELECT *
  FROM t1
  WHERE col1 > 50
  ```

```
AND col2 = 4;
```

Because of the range operand on the leading column in the concatenated index, the index would be used to process the filter criteria on column col1 (if selectivity of this filter calls for this operation). Then the data page from the SQL table proper is fetched. Then the filter criteria for column col2 was processed from the data page. This behavior can be detected by reviewing the index and filter utilization in the text representation of the query plan. Basically not having the capability was just a lack of functionality in the relational database software product of a given vendor.

10.6.4 Query optimizer directives (hints)

Most modern relational database vendors offer the ability to influence the query plan that the query optimizer produces. Some vendors offer the ability to promote (call for, or force) certain elements of the query plan, or just demote (prevent) certain elements, or the ability to both promote and demote. Most queries that you will encounter which require manual tuning (creation of optimizer directives) will need help because the query plan is making a poor choice. *We recommend that you use query optimizer directives to demote poor query plan paths. Do not fall into the practice of writing query optimizer directives that promote the (currently) more optimal path.* As the run-time environment changes (improves), new more optimal query plan choices may arrive.

Note: In the query optimizer case study in 10.2, “Query optimizer by example” on page 503, we reviewed a canned application system; that is, one where the source code to the application may not be accessible to us (the database administrators). Some relational database vendors offer the ability to set query optimizer directives in newly created files (or system catalogs) which are external to any canned application system. Needless to say, this is a very useful feature when supporting canned applications.

You can also embed query optimizer directives inside SQL VIEW definitions; perhaps leave the base table in place, and create a SQL VIEW entitled, “v_faster_table”, which accesses a given table while demoting poor query plan choices.

Query optimizer directives can be organized into categories which, not surprisingly, equal the basic categories of operations that a query optimizer considers. These types of optimizer directives include:

- Table join order
- Table access method
- Table join method
- Optimization goal

- Query rewrite directives

Generally, query optimizer directives are embedded inside an SQL command statement as comments; in other words, the query optimizer directives are not part of the declarative portion of the SQL command. Consider the example:

```
SELECT --+ This is a single line comment, begins with "--+"
* FROM t1;
SELECT { This is a multi line comment,
   extending for multiple lines and surrounded by
   curly braces }
* from t1;
SELECT /* This is also a multi line comment,
   surrounded by Java and C language style markers */
* FROM t1;
```

While different relational database vendors may use differing and specific query optimizer directives syntax, below is a representative offering of the standard tags and their associated effect.

ORDERED, INDEX, AVOID_INDEX, FULL, AVOID_FULL

A call to "ORDERED" states that the table join order should equal that as stated in the SQL FROM clause of the SQL SELECT statement. "FULL" states that the named tables, enclosed in parenthesis should use a table access method of sequential scan, while "AVOID_FULL" states the opposite. "INDEX" and "AVOID_INDEX" do the same for indexes. Example as shown:

```
SELECT --+ ORDERED, AVOID_FULL(e), INDEX(i2)
*
FROM employee e, department d
WHERE e.deptno = d.deptno AND e.salary > 100000;
```

The query optimizer directive above states that the SQL tables should be joined in order of Employee to Department. Also a call is made to use the index entitled "i2", whichever table maintains that index. If the named index does not exist, that specific query optimizer directive will be ignored.

Multiple entries inside the parenthesis are separated with commas. If necessary, the table names should be prefaced with the owner or schema name.

Combination of both “AVOID_FULL” and “AVOID_INDEX” on a given SQL table can be used to encourage a hash join. (Remember we prefer to demote bad choices, not explicitly promote the currently best choice.)

INDEX_ALL

“INDEX_ALL” is the query optimizer directive to invoke multi-index scan. For example:

- OR topped query, would benefit from multi-index scan

```
SELECT --+ INDEX_ALL (ix_on_col1, ix_on_col2)
*
FROM t1
WHERE col1 = 5 OR col2 = 15;
```

USE_NL, AVOID_NL, USE_HASH, AVOID_HASH

“NL” is an abbreviation for the nested loop (table) join method. Each of these query optimizer directives are for table access methods. (The current vendor we are reviewing did not document query optimizer directives for a sort-merge table join method, or a push down semi-hash join. This vendor did offer other controls to promote or demote these table join methods, but they are not documented here.)

In the case of a hash table join method, an SQL table must be identified as the SQL table upon which the hash index is built, or should it be the table to access the hash index execute to join? The “BUILD” label marks the table upon which the hash index is built. The “PROBE” label marks the table that reads the hash index. For example:

- Hash index will be built on Dept table

```
SELECT --+ USE_HASH ( dept /BUILD )
name, title, salary, dname
FROM emp, dept, job
WHERE loc = “Almaden”
AND emp.deptno = dept.deptno
AND emp.job = job.job;
```

This vendor also has a “/BROADCAST” label for a hash join used in a clustered database environment. A call to “/BROADCAST” will copy tables built with hash indexes under 100 KB to all participating nodes in query. This is a useful feature to place (copy) smaller dimension tables to all nodes, thus encouraging co-located joins. (Co-located join is a clustered database concept. In effect,

co-located joins are those where both sides of the join pair are local to one (physical) node. This eliminates or reduces internodal traffic for joins.)

“USE_NL” or “AVOID_NL” call for the nested loop table join method to be used, or not used, with regards to the named tables.

FIRST_ROWS, ALL_ROWS

Operations such as creating a hash index to access a table, or sorting a given table to allow for a sort-merge table join method, require an amount of up front query processing that is time spent before any records may be returned to the user application. A “FIRST_ROWS” query optimizer directive, will avoid query plans with these types of up front costs, and instead give preference to query plans that return records more quickly, if only initially (first record, versus total data set performance).

From the discussion on pipelined sorts versus non-pipelined sorts, held earlier, this category of query optimizer influencer will not be viewable to the user application if records have to be gated (held), to complete a sort operation. Under these conditions, the query optimizer may negate this specific query optimizer directive.

Following are the results a query received with no query optimizer directives:

- This output has been edited

```
SELECT * FROM t1, t2
WHERE t1.col = t2.col
Estimated Cost: 125
Estimated # of Rows Returned: 488
1) Abigail.t2: SEQUENTIAL SCAN
2) Abigail.t1: SEQUENTIAL SCAN
DYNAMIC HASH JOIN
   Dynamic Hash Filters: Abigail.t2.col = Abigail.t1.col
```

This same query is then executed with a query optimizer directive of “FIRST_ROWS”. The following are the results:

- This output has been edited

```
SELECT --+ FIRST_ROWS
* FROM t1, t2
WHERE t1.col = t2.col
Estimated Cost: 145
```

```
Estimated # of Rows Returned: 488
1) Abigail.t1: SEQUENTIAL SCAN
2) Abigail.t2: INDEX PATH
   (1) Index Keys: col
   Lower Index Filter: Abigail.t2.col = Abigail.t1.col
NESTED LOOP JOIN
```

We see that the cost is higher for the second query, because of the (CPU) intensive nested loop (table) join method. While the cost is higher on the second query, and the total data set performance may take more time, the first record returned will occur measurably faster than that in the first query, which must first build a hash index.

NESTED

By default, the query optimizer will attempt to eliminate nested queries (subqueries) and correlated subqueries. The query optimizer directive “NESTED” disables that capability. “NESTED” is one query optimizer query rewrite directive.

Optimizer directives not shown

While query optimizer directives are measurably complete, there are still a few query plan-related decisions that are not set via query optimizer directives. As examples:

- Whether a sort operation happens entirely in memory or overflows to disk is a function of the amount of shared or program memory that is allocated to the query. This, generally, is not set via query optimizer directives.
- The degree of parallelism is determined automatically by most vendors based on:
  - Whether records are locked as part of this query. When locking, most vendors will demote parallelism to prevent locking too many records concurrently, and hurting overall system throughout and concurrency.
  - The number of source and target drives. If you have 14 CPUs, but all of the tables are being read from one hard disk, how much parallelism do you actually require? If you are performing an unload of data to a single ASCII text file, how much parallelism do you actually require?
  - And as implied above, if you only have one CPU, how much parallelism do you require beyond parallel table scans?
  - The presence of SQL SELECT TRIGGERS can prevent not only parallelism, but also the transfer of numerous records per each buffer transport inside the SQL CURSOR.

– The number of database disk drives designated (allowed) for sort operations; SQL SELECT DISTINCT, SQL SELECT ... UNION ..., SQL GROUP BY, SQL ORDER BY, hash index creation, and others.

Example query plan with query optimizer directives

Consider the following:

- This file has been edited for clarity:

```
SELECT { +ORDERED,
   INDEX ( emp ix1 ),
   FULL ( job ),
   USE_HASH ( job /BUILD ),
   USE_HASH ( dept /BUILD ),
   INDEX ( dept ix3 )
   }
   j.*, d.*
FROM emp e, job j, dept d
WHERE e.location = 1
AND e.jobno = j.jobno
AND e.deptno = d.deptno
AND d.location = “MILWAUKEE”

DIRECTIVES FOLLOWED
ORDERED
INDEX ( emp ix1 )
FULL ( job )
USE_HASH ( job /BUILD )
USE_HASH ( dept /BUILD )
INDEX ( dept ix3 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 963
Estimated # of Rows Returned: 2
```

```
1) Abigail.emp: INDEX PATH
   Filters: Abigail.emp.location = "Milwaukee"
   (1) Index Keys: empno jobno deptno location (Key-Only)

2) Abigail.job: SEQUENTIAL SCAN

DYNAMIC HASH JOIN
   Dynamic Hash Filters: Abigail.emp.jobno = Abigail.job.jobno

3) Abigail.dept INDEX PATH

   (1) Index Keys: location
   Lower Index Filter: Abigail.delp.location = "Milwaukee"

DYNAMIC HASH JOIN
   Dynamic Hash Filters: Abigail.emp.deptno = Abigail.dept.deptno
```

Additional comments:

- With the ORDERED query optimizer directive, table order is equal to that listed in the SQL FROM clause; Emp, then Job, then Dept.
- Hash indexes are built on Job and Dept as called for in the USE_HASH() directive.
- With no filter on table Job, the hash index is built on every record in the table; therefore, the sequential table access method. (This does not qualify as a second table being sequentially scanned inside of one SQL SELECT, since this table access method is used only to build the hash index.)
- An index is used to build the hash index on table Dept, as called for in the "INDEX (dept ix3)" directive.
- The table access method on table Emp is indexed, for which it was called. This access method is also key only, which the query optimizer detected the opportunity to do so. (All necessary columns from table Emp were part of a usable index, the index used to access that table.)

10.6.5 Data distributions

Data distributions was presented in “Data distributions” on page 524. Here we present additional information.

Filter (predicate) correlation also known as predicate inference

Consider the following:

- Query one, distinctiveness of filters can be multiplied

```
SELECT * FROM autos WHERE make = “CarMaker” and model =
“CarModel”;
```

- Query two, distinctiveness can not be multiplied

```
SELECT * FROM addresses WHERE cityName “Smithtown”
   AND streetName = Main”;
```

In an Autos table with 100,000 records, 10% of which are CarMakers, and only CarMaker makes the CarModel (one of six models made by CarMaker), then the filter on table Autos can be expected to return 1/10 times 1/6 of 100,000 records. These two filters have a strong correlation.

In an Addresses table with 100,000 records, and 10% of the cities are named “Smithtown”, and 1/6 of the streets are named “Main”, one should not make any assumption about the correlation of these two filters; nearly every town in America has a street named “Main” (or “High” street if you prefer).

This is an area where data distributions can help. Data distributions can be gathered on indexed or non-indexed columns, individual columns, or THEN groupings of columns. Generally, the query optimizer will assume that it can multiply the selectivity of predicates, as shown in the first SELECT example above with Autos. A data distribution on both the cityName and streetName columns in Addresses would correctly inform the query optimizer that it should not consider adding the selectivity of these two filters.

Other applications of data distributions

By default, most query optimizers know the number of records in a table, and the number of distinct values in an index. Given a table with 1000 records, and a unique index, that index contains 1000 distinct values (by definition). The optimizer also knows the number of distinct values in a duplicate index. (The optimizer also knows the minimum and maximum values of each index, as well as the second minimum and second maximum.)

Most query optimizers, by default, assume an even distribution of key values within a given range. That is, it assumes that an equal number of persons exist on every continent (land mass) on planet earth. Given the following:

```
SELECT * FROM worldPopulation
   WHERE continent IN (“Antarctica”, Australia”);
SELECT * FROM worldPopulation
   WHERE continent in (“Asia”, “Europe”);
```

The query optimizer would, by default, consider the selectivity of these two queries to be equal. Since the second query returns a very large percentage of the entire contents of the Continent table, this query would be better served by a sequential scan table access method (negation of index, poor selectivity of filter). The first table should use the index.

Data distributions inform the query optimizer of the selectivity of specific values in an indexed or non-indexed column. This additional information helps the query optimizer choose better query plans.

Another application of data distributions is for concatenated indexes. By default, the query optimizer knows the uniqueness of the entire index (or perhaps only a small number of leading columns). Not having this data was one contributor to the error in the query plan reviewed in 10.2, “Query optimizer by example” on page 503.

Note: Some relational database software vendors count every distinct value in a column, or samples therein. Some vendors organize this counting into distinct ranges, sometimes called *quantiles*.

The selectivity of filters in general, is calculated using guidelines similar to those below (selectivity is represented by the value (F), and *EXPRESSIONS* are represented by the bulleted condition):

- indexed_column = LITERAL_VALUE

  indexed_column IS NULL (which is a a literal value)

  F = 1 / (number of distinct keys in index)
- t1.indexed_olumn = t1.indexed_column

  F = 1 / (number of distinct keys in the larger index)
- indexed_column > LITERAL_VALUE

  F = (2nd-max - LITERAL_VALUE) / (2nd-max - 2nd-min)
- indexed_column < LITERAL_VALUES

  F = (LITERAL_VALUE - 2nd-min) / (2nd-max / wnd-min)
- non_indexed_column = LITERAL_VALUE

  F = 1 / 10

- non_indexed_column > LITERAL_VALUE

  F = 1 / 3
- non_indexed_column LIKE {STRING EXPRESSION}

  F = 1 / 5
- EXISTS {subquery}

  F = 1, if subquery estimated to return more than zero rows, else 0
- NOT {EXPRESSION}

  F = 1 - F {EXPRESSION}
- EXPRESSION1 AND EXPRESSION2

  F = F (EXPRESSION1) union F (EXPRESSION2)
- EXPRESSION1 OR EXPRESSION2

  F = F (EXPRESSION1) + F (EXPRESSION2) -

  (F (EXPRESSION1) union F (EXPRESSION2))
- non_indexed_column IN LIST

  Treated as an OR, OR, OR for each item

The primary purpose of offering the list above is to demonstrate the coarseness of query optimizer statistics regarding selectivity of filter and join criteria without data distributions.

10.6.6 Fragment elimination, multidimensional clustering

For all that SQL provides, it does little in the area of the physical storage of data; in other words, how tables and indexes are physically organized on the hard disk. SQL tables and indexes are logical structures, that, at some point, become physical bytes on a collection of disk pages. How are these pages to be organized?

The earliest relational database server products placed both (normal) table data and index data inside the same physical disk space allocation. Modern relational database server products place normal table in a physical allocation by itself, and then each individual index in its own separate physical disk allocation. This newer convention allows for like data to be contiguous; that is, index data from a given index is contiguous, allowing for read ahead scans. The data from the table proper is also contiguous, also allowing for read ahead scans. And all of the behavior increases the physical I/O cache rates.

Records in the SQL table proper are allowed to be in random sequence. While not explicitly stated, this is a base assumption to the governing of SQL tables.

Most vendors offer a clustered index operation; that is, generally in a one time only operation, the data in the table is physically sorted (written) in the collation sequence of one of the indexes found to exist on that table. (A given list of data can only be truly sorted in one manner, based on one hierarchy.)

Splitting a tall table

Given a table with three years of data, but having the majority of queries executed against data created in the last 90 days, creates an interesting opportunity. Many years ago, the database administrator would implement a design pattern, referred to as *splitting a tall table* (tall, meaning having many data records).

When splitting a tall table, one table contains the data from the last 90 days. As these records age, they are deleted from the 90 days table and inserted into the history table. Queries targeting the last 90 days of data read from just one table. Queries that need data from the full date range execute a SQL UNION ALL query. For example:

- `UNION automatically removes duplicate records, but UNION ALL does not. Since records are either in one table below but not both, we can execute a UNION ALL query which is more efficient:`

```
SELECT * FROM recentTable
UNION ALL
SELECT * FROM historyTable;
```

You can instruct modern relational database servers to perform this type of activity automatically.

Table partitioning

The original phrase given to the practice table partitioning was *fragmenting a table.* Generally, and in the area of disk storage, fragmentation is a term with negative connotations. Still, once a phrase is released it is often hard to recapture. The relational database industry calls this idea both *table partitioning* and *fragmenting tables.*

With table partitioning, a table can be physically organized by the value of a column or set of columns. These columns are expected to be those which are of importance to critical queries. For example:

- `Sample SQL CREATE TABLE and SELECT:`

```
CREATE TABLE t1
   (
   col1 INTEGER,
```

```
   col2 INTEGER
   )
FRAGMENY BY EXPRESSION
   col1 < 100 IN driveSpace1,
   col1 >= 100 IN driveSpace2,
   col1 >= 200 IN driveSpace3;


SELECT * FROM t1 WHERE col1 = 150;
```

A “driveSpace” is a non-SQL object. In this case, driveSpace is the concept of a distinct disk storage location. Relational database vendors call these entities *chunks, volumes, containers*, and perhaps other terms.

Given the SQL CREATE TABLE statement above, the relational database server automatically places records in a specific driveSpace determined by the value in col1. When the query is received as shown above, the relational database server (the query optimizer) can perform fragment (partition) elimination; that is, we do not need to access any records in “driveSpace1” or driveSpace3”, because they contain no records with a col1 value of 150.

These are not indexes. This is similar to a free and omnipresent natural index. You can automatically reduce the amount (table size) of the records to processed based on how the table is organized on the hard disk.

Most vendors allow data in a table to be partitioned, as well as partitioning each index. See the following:

- Partitioning is allowed on numerous columns, not just one column.
- There are expression operands to determine driveSpace, as well as hash and others.
- Specifying expression operands can be wordy, but they serve both filter equalities and ranges. Hash value partitioning does not serve range filters well at all since the column values are essentially rotated through the driveSpaces.
- Under the best circumstances, these partitions can be added and removed from the table in full or nearly full multi-user mode. Since unique indexes have to provide a unique data integrity constraint, often these are the culprit as to whether these partition attach operations are zero or low cost.

The query optimizer automatically calculates which table partitions have to be examined or not, and these costs are considered in the determination of the final query plan. This basic operation is referred to as *fragment elimination*.

Given table partitioning and an SQL CREATE table as shown below, consider the following:

```
CREATE TABLE t1
   (
   col1 INTEGER,
   col2 INTEGER
   )
FRAGMENY BY EXPRESSION
   col1 < 100 AND col2 < 100 IN driveSpace1,
   col1 < 100 AND col2 >= 100 IN driveSpace1b,
   col1 >= 100 AND col2 < 100 IN driveSpace2,
   col1 >= 100 AND col2 >= 100 IN driveSpace2b,
   col1 >= 200 AND IN driveSpace3;


SELECT * FROM t1 WHERE col2 = 150;
```

Given the above run-time environment, the query optimizer must call to examine driveSpaces 1b, 2b, and 3, to find records with a col2 value of 150.

Multidimensional clusters

One vendor in particular has a different take on table partitioning. It was stated earlier that any given data set can have only one concurrent sort order. This is true. Multidimensional clusters are like a multi-level table partitioning implementation. Multidimensional clustering uses specific column values to automatically store records in distinct drive areas. A multidimensional cluster is created with syntax similar to:

```
CREATE TABLE t1
   (
   col1 INTEGER,
   col2 INTEGER
   )
ORGANIZE BY (col1, col2);
```

The relational database server will automatically create and enforce a distinct disk space allocation for each unique combination of col1 and col2 values. That means that each of the data pairs (1,1), (1, 2), and (2, 2) go into their own distinct disk space allocation. (Three separate disk space allocations.)

Note: The term *dimensional* in the phrase multidimensional clusters has no relationship to the phrase *dimensional* data modelling.

Standard table partitioning is viewed by some to be too much work, with its precise syntax determining record location by value expression or hash. Multidimensional clustering involves much less work, but better serves columns with few distinct values, since each distinct value causes a new disk space allocation.

Table partitioning, SQL VIEWS, and query rewrite

Consider the following SQL VIEW:

```
CREATE VIEW v1 (col1, col2, col3, col4)
AS
SELECT a, b, c, d
    FROM t1
    WHERE ...
UNION ALL
SELECT e, f, g, h,
    FROM t2
    WHERE ... ;
```

Given any data integrity check constraints or table partitioning expressions that may be in place on these underlying tables, the query optimizer query rewrite feature may remove specific filter criteria from any SQL SELECT accessing this view, since these filter criteria are known to be redundant (unnecessary).

10.6.7 Query optimizer histograms

Query optimizer histograms are the idea that the query optimizer could form a query plan with its associated cost, and record this plan and expected cost for later reference. If the given query plan did not perform as expected, then the query optimizer might later experiment and try a different query plan to see if query performance can improve. This is not artificial intelligence since no new algorithms are being created. This is merely the query optimizer recording costs to add to that data which it considers in the creation of a query plan.

While query histograms might someday be delivered, thus far they have proven too costly to include in current relational database server products. Having also to consider histogram data in the calculation of a current query plan takes

additional time and resource. Without histograms, the current query optimizer makes the most optimal choice 98% or more of the time, it seems. If not, you still has query optimizer directives to rely upon.

10.7 Summary

In this chapter we have presented significant detailed information about the topic of query optimizers. This can help you understand how to create better data models and how to better format and store your physical data. You are working to “get your data in” to the data warehouse, but doing so while you keep in mind that it must be done in such a way that enables you to maximize you capability to efficiently and effectively “get your data out” of the data warehouse to support your BI environment.

To get started on that path, this is an excellent time to continue on with Chapter 11, “Query optimizer applied” on page 599. Here you will find help how to put the knowledge from this chapter into action. So, on you go.

11

Query optimizer applied

In Chapter 10, we reviewed a business application system to analyze the SQL statements used. In that application, there was a query with an SQL SELECT statement that executed in five minutes, when the requirement was for sub-second response time.

This is a situation typically faced by application developers. In this chapter we offer direct and specific techniques that you can employ to help you avoid issues such as this.

The query and the SQL statement, were analyzed and the problems were recognized and changed to meet the application requirement. This example is from an actual application system analysis, where it took the application analyst two hours to solve this problem. The following are comments about some of the issues encountered:

- It took two hours to solve this problem, in part because we had never seen this exact problem before. As it turns out, we saw this same condition weeks later, and were able to solve the problem in less than thirty minutes.
  - While we solved this problem relatively quickly the second time we saw it. But, there are always new problems.
  - In Chapter 10, we stated that a database administrator with six months of experience can performance tune a single (non-clustered, non-distributed) relational database server system with 500 GB of disk storage and 8 CPUs in well under two hours. Performance tuning relational database

server systems is a generally well understood process and is typically well documented.

  - The problem in the example is not one of *relational database server system tuning*, it is one of *SQL statement tuning*. While it takes two hours to tune a server, it regularly takes two hours to tune just one poorly performing SQL statement. In this example, there were dozens of SQL statements that were performing poorly. That means dozens times two hours.

- In our example, we needed to change the indexes on the table to solve the problem. Many times, however, the structures of the tables have to be changed to solve the problem. And, changing table structures often means application program code also has to change, unit test and then system tests have to be performed, and so on. The net result is that solving SQL statement tuning problems is very costly and takes a lot of time.

The techniques we describe in this chapter should better enable you to deliver performant SQL-based business application systems on time and at lower cost. They will also improve your ability to maintain previously developed SQL-based business application systems.

11.1 Software development life cycle

A software development life cycle (SDLC) is defined as the methodology you uses to develop, support, deliver, and maintain computer applications. As a phrase, SDLC is logical; it defines a categorization (of physical methodologies). One specific (and perhaps the most common), SDLC methodology is the Waterfall Method. Figure 11-1 displays a Waterfall Method SDLC.

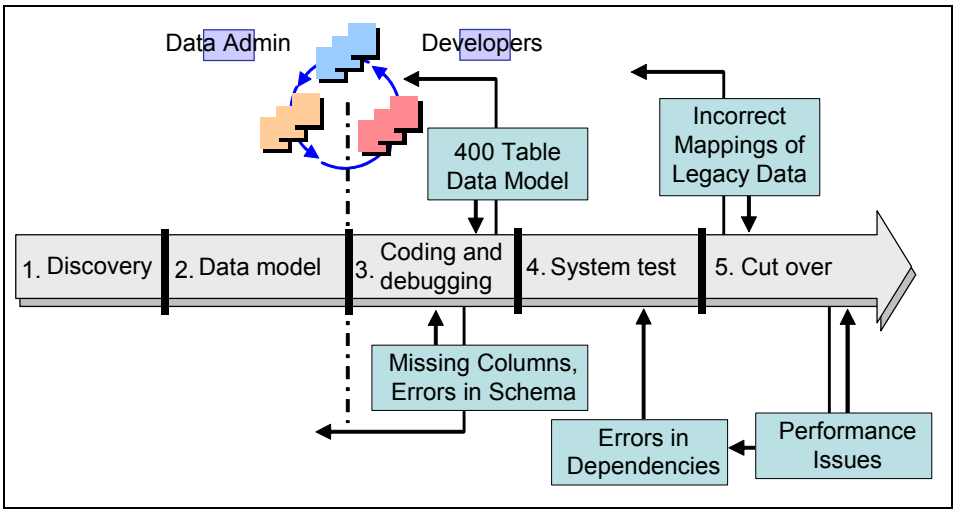


Figure 11-1 Example Waterfall method (SDLC)

While any specific representation of a Waterfall Method SDLC might display five, seven, or another number of phases, each representation shares these basic ideas:

- There are iterative steps that are performed in each phase. An error or other conditions can cause you to revisit, or at least partially revisit, prior phases.
- Generally, you move from:
  - **User discovery:** Gathering the requirements of the application:
    - Generally, the product of the user discovery phase is a requirements, costing, and staffing document of some form.
    - Persons involved include business area leaders (persons who understand and can articulate the business goal to be achieved with the new system), project managers, data analysts, operating system administrators, network administrators, and perhaps others.

– **Data modeling**: Includes both logical and physical data modelling. Here, we assume we are creating a business application system that sits atop a relational database.
  - Generally the product from this phase includes the Structured Query Language Data Definition Language (SQL DDL) statements that enable creation of the SQL tables, SQL indexes, SQL data integrity constraints, and all that is needed to support this application on a relational database server.
  - Persons involved in this phase include product managers and data modelers.
– **Coding and debugging:** In this phase the application program code is created along with unit test, early quality assurance, and program documentation.
  - Persons involved in this phase include the application programmers.
  - The application programmers write program code in whatever language they choose, such as Java, Visual Basic®, C/C++, and COBOL. Each of these languages requires a specific programming language skill set, in addition to expertise in delivering useful user interfaces and systems.
– **System test:** In this phase, the unit deliverables are assembled to create a working system. Generally, the product of this phase is correction and proof.
  - Persons involved in this phase include the application programmers.
– **Cutover**: Here the application is put into production. To support the new system, data is migrated from the existing system to the newly created one.
  - After a final trial period, the application is considered completed, and enters the maintenance phase (not displayed).

11.1.1 Issues with the life cycle

The question discussed in this subsection is, “Are there any specific strengths or capabilities you can make use of when developing application that sits atop a relational database server?” For example, “are there any specific optimization best practices provided?” Or, “Is there some other opportunity that is being overlooked?” In addition to the phases in the execution of a Waterfall Method SDLC, Figure 11-1 on page 601 also depicts some commonly observed sources of contention. For example:

- **Unit development and unit test are often performed on test or staged data**. If unit development and unit test for example with the month end

summary report, are performed with 1000 records in the SQL table, when the production system will have millions of records, this enables performance issues and application errors to remain hidden until late in the development and delivery cycle.

Note: Fixing any issue (such as program logic errors and performance) later in the software development life cycle costs measurably more time and energy to repair, when compared to uncovering and correcting these issues earlier in the software development life cycle.

- **Application developers generally write their own SQL DML**.
  - Structured Query Language Data Manipulation Language (SQL DML) statements are those statements that are located throughout the body of the application. They lay out (define, one time only) the database. But they run every day, thousands of times, throughout the life of the application.
  - With all of the difficulty of staying current on a given application language, and having skill to design and deliver capable user interfaces, and working with business goals and objectives, frankly we are typically satisfied if the application programmer is expert in their given application programming language. It is less likely that we will expect the application programmer to be an expert at reading 400 table SQL schemas than for the application programmer to know how to program highly performant (query optimizer expert) SQL DML statements or to develop multi-user SQL record concurrency models.
  - Creation of these SQL DML statements often uncovers errors in the data model. With thousands of columns and dependencies, there typically are errors in the data model, even if only a few. Discovering these errors sooner avoids such activities as unit redevelopment, unit retesting, and system retesting.
- **Final cut over to the new system is a risky procedure.**
  - At this point of the software development life cycle, we are moving as much data into the new system as it took the existing system to acquire over a number of years. And, we need to perform this cutover quickly and without error.
  - Cutover of the existing data will also typically uncover forgotten usages of existing fields and produce additional application requirements.

11.1.2 Process modeling within a life cycle

Figure 11-2 on page 604 depicts a modified Waterfall Method SDLC. The primary objective in this modified SDLC is to leverage the natural strengths of a

relational database server, and better manage the commonly observed application development life cycles issues.

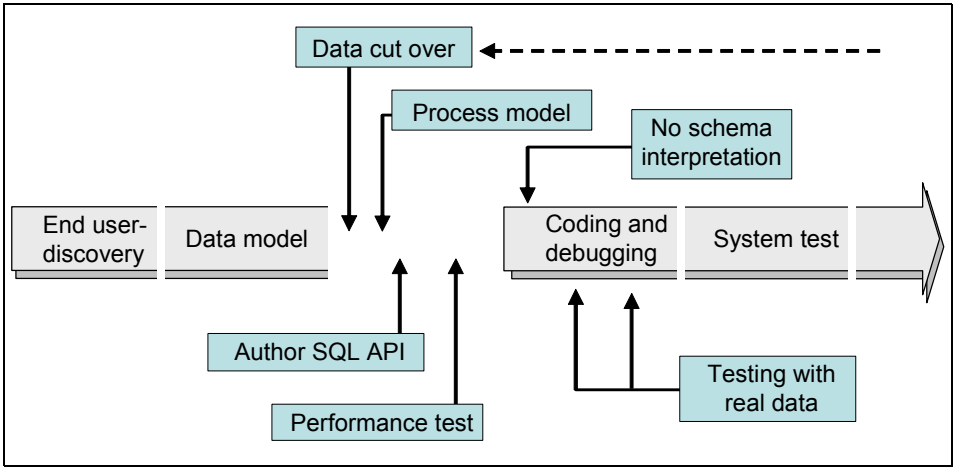


Figure 11-2 Waterfall Method SDLC, optimized for SQL development

Related to Figure 11-2, consider the following:

- **Immediate existing data cutover following data model.**
  - Immediately after the creation of the physical data model, a cutover of the existing data is loaded into the relational database server.
  - The existing data model was created at the start of this software development effort. The data migration scripts can be developed by engineers who are not on the critical path, since this work requires no expertise (skills) which will benefit the new system.
  - Loading existing data into the new system allows for discovery of new application requirements. For example, receiving character data to numeric conversion and existing fields which were supposed to be of one data type, are now discovered to have new meaning and use.
  - Successful loading of the existing data serves as one validation of the newly created data model.

    Note: Besides careful study and engineering, how does your current SDLC truly verify the accuracy and performant nature of the data model?

  - This procedure also allow for early measurement of the difficulty of performing the migration, an event most persons only become intimately familiar with when they actually perform the (final) cut over.

- All development is done on production-sized and actual data. This condition increases the accuracy and likelihood of performant unit and system deliverables.

- **Create a process model**.
  - The data modeling team gathers great expertise with the business rules and objectives as they design and deliver their data model. If the modelers truly understand the business, a requirement for accurate data modeling, then they understand every business process and every state that records occupy as data moves through the application (and data model).
  - What we are doing here is leveraging the already existing expertise the data modelers have, and producing a labor savings.
  - The final product of the process model is a set of application programming interfaces (API). In effect, every SQL data manipulation language (DML) statement that is expected to be run against (supported by) this data model is produced here.
  - If an 18 person development team has two data modelers, 10 application developers, and assorted other persons, we move one of the application developers over to the data modeling side. This resource will author every SQL DML that the application system is expected to run.
  - Figure 11-3 on page 606 displays the Model-View-Controller (MVC) design pattern. MVC came into public prominence in 1995.
  - Today, MVC is the industry de facto application development design pattern. In short, MVC calls for a separation of the application programming for the (data) model, view (the user interface), and controller (the business logic).
  - A programmer with special skills creates the process model (the SQL API). These skills include query optimizer and SQL statement tunings, SQL concurrency models and programming, enterprise-wide SQL error recovery, and others.

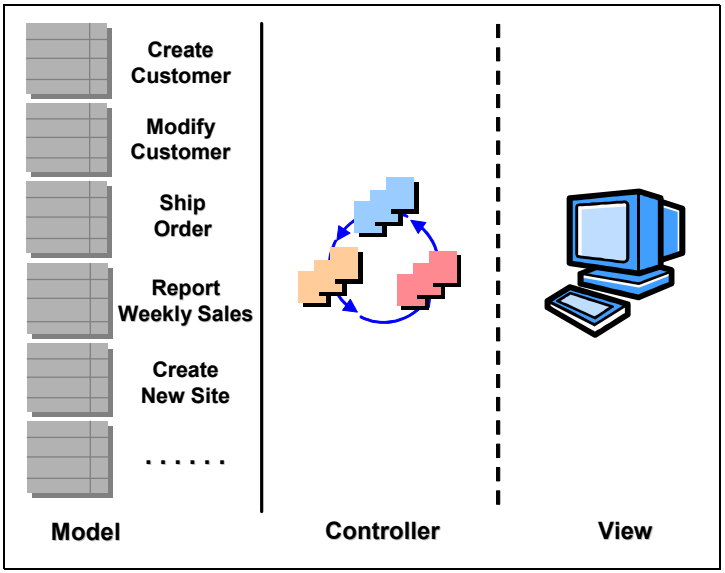


Figure 11-3 Model-View-Controller (MVC) with SQL API

- **Execute an early performance trial.**
  - SQL has a unique capability. Running the SQL SELECT statement for the previously mentioned month end summary report effectively models the performance load of that entire program. Most of that program load is the processing of the relational database SQL statements. Whether the output uses a given font or font size, paginates in some specific manner, or similar, does little to the performance load. It is the SQL data manipulation language statements that form 90% or more of the system load.
  - Execute the SQL API routines with the anticipated frequency, concurrency and spread. Model for the expected number of users on average, at peak load, and with consideration for future system growth.
  - This operation allows us to determine months before final delivery if this system will perform adequately when finished.
  - If the system does not perform adequately, take appropriate action now. This may entail adjusting the data model, changing the design of data loads and summations, individual SQL statement tuning and more.

    The point is, we can make these changes now with little or no cost, because few dependencies exist. No application program code has been authored, no user documentation has been written, and we are still weeks or months from delivery.

Note: It is at this point, months before delivery, that we know the system will perform in a given manner, either meeting expectations (needs) or not. We can then proceed to alter the data model or system design, at little or no additional cost. Successful execution of the process model is final validation of the accuracy and performant nature of the data model.

Following the process model, with its performance trial, all unit and system development and testing is done on real data and at the expected volume. The application programmers are given defined points to interact with the relational database server. The application programmers are not expected to know SQL, and certainly not how to create and model highly performant SQL. And finally, the data model can be code frozen, which is always important as changes to the data model during unit and system test is one of the most costly changes that can occur.

11.2 Artifacts created from a process model

To this point, all attention has been given to application development. The artifacts created as part of a process model benefit not only development, they also greatly aid the maintenance phase of an SDLC.

The SQL API is that set of routines that run inside a given application, and execute (generally, SQL DML) statements against the relational database server. The SQL API contains a number of function calls to perform tasks such as run month end summary report, and load new location dimension data. In a 400 table data model, you might have 2000 or more distinct operations that need to operate against that data model.

Note: The SQL API is being created. The question is, do we organize this work as a distinct delivery within the entire application development effort? And, do we use a specific (human) resource with specialized skills to create this strategic deliverable?

If creation of the SQL API is not done as part of a distinct effort, then the SQL API is created by every application programmer on the project, and the functions of the SQL API are scattered throughout the entire (tens of thousands of lines of) application program code.

11.2.1 Create the SQL API

After the creation of the physical data model, we recommend you load the existing system data into the database server. The SQL API is then created by the data modelers, who have knowledge of the requirements of the application, and one additional (human) resource. This new resource, and the data modelers have skills that include:

- Knowledge of the data model, and the design of those SQL DML statements that need to run against this data model. These SQL DML statements, along with the anticipated frequency, concurrency, and spread that these statements will run in production, form the *logical process model*. If the data modelers do not already know every entity attribute and every program state that each entity goes through in support of the application, then they could not have modeled that application accurately. Creating the logical process model is a record of information the data modelers already have.
- Ability to create the Model portion of Model-View-Controller, and the ability to create functions in the application programming language of choice, one for each single or set of SQL DML routines above. This is a skill set that is outside of the normal data modeler world, and why a new resource is included in this activity. These program code artifacts form the *physical process model*, or referred to more simply as the SQL API.
- Knowledge of that information provided in Chapter 10. This group of engineers know how to tune SQL statements and work with the query optimizer.

> **Note:** There is not an industry standard name for the (human) resource who can write SQL DML and SQL API in an application language, work with the query optimizer in-depth, and demonstrate *all of these skills.* However, it can be a team of persons. If we gave a name to this single resource, it would be a name such as an *Enterprise Data Architect.*

- Knowledge about executing a performance trial against the relational database server of choice. Here we run the functions within the SQL API. There is relational database server tuning, SQL statement tuning, and other work performed here.
- This team authors the SQL API, and executes a performance trial with the elements of the SQL API with the anticipated frequency, concurrency, and spread that those statements are expected to run once the application is put into production. Inadequacies in performance may affect the data model (including column definitions, table relationships, index plans, and others), the application program code syntax, and every other tuning topic covered thus far.

11.2.2 Record query plan documents

For each function in the SQL API the query optimizer query plan is recorded, as well as the various execution times for a given function such as peak performance time, best, second best, worst, and average.

This performance trial is automated via whatever technology is available, such as a load trial tool or shell(C) scripts. As an automated process, this test can be repeated and in a controlled environment. One of these automated test profiles should be non-intrusive from a load perspective, allowing it to run on a production system. We can then run these tests weeks or months after the system has gone into production as part of our available maintenance and test routines. If an user complains that a given routine now takes two hours to run when it used to take 15 minutes, we have greater options to consider. We have baselines on which to compare this claim, and tools to authenticate and correct the condition. Figure 11-4 depicts a coarse data model of the information managed in this activity.

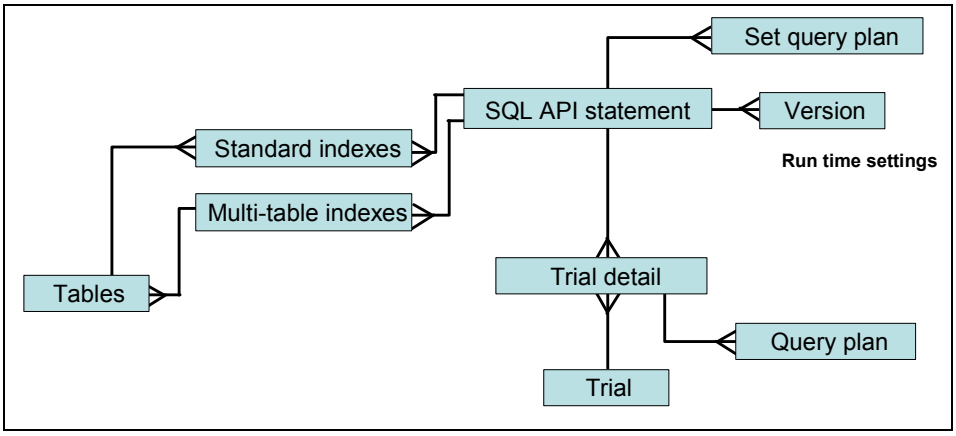


Figure 11-4 Coarse model of data managed in this activity

From Figure 11-4, consider the following:

- The data model proper produces SQL tables. Generally, an SQL index specifies a single or set of columns from one SQL table. Some relational database vendors support multi-table SQL indexes.
- Since the normal update and maintenance of indexes have costs, each index is validated (known to exist) for a given purpose. The purposes include support of primary key or unique data integrity constraint, foreign key constraints, and then support for a given SQL API statement (routine, function). A given SQL API function may require use of zero or more SQL indexes.

- Each SQL API function is associated with a single or set of query optimizer query plans. These various plans for each SQL API function may be recorded as best case or acceptable case, or with given run-time settings.
- Performance trials are also recorded, which have a grouping of SQL API functions which are run (with a frequency, concurrency, and spread). At this level too, we observe query statistics (times, resource consumed), and query plans, as these may vary under load and other variables.

11.3 Example of process modeling

Example 11-1 displays a sample data model. This sample SQL data definition language code is referenced throughout the remainder of this section.

Example 11-1 Data model example

```
create table cust_calls
   (
   customer_num            integer,
   call_dtime              datetime year to minute,
   user_id                 char(32) default user,
   call_code               char(1),
   call_descr              char(240),
   res_dtime               datetime year to minute,
   res_descr               char(240),
      --
   primary key (customer_num, call_dtime),
   foreign key (customer_num)
      references customer (customer_num),
   foreign key (call_code)
      references call_type (call_code)
   );

create table catalog
   (
   catalog_num              serial(10001),
   stock_num                smallint not null,
   manu_code                char(3) not null,
   cat_descr                text,
   cat_picture              byte,
   cat_advert               varchar(255, 65),
      --
   primary key (catalog_num),
   foreign key (stock_num, manu_code)
      references stock constraint aa
```

```
   );

create table customer
   (
   customer_num            serial(101),
   fname                   char(15),
   lname                   char(15),
   company                 char(20),
   address1                char(20),
   address2                char(20),
   city                    char(15),
   state                   char(2),
   zipcode                 char(5),
   phone                   char(18),
      --
   primary key (customer_num)
   );
create index zip_ix on customer (zipcode);

create table items
   (
   item_num                smallint,
   order_num               integer,
   stock_num               smallint not null,
   manu_code               char(3) not null,
   quantity                smallint check (quantity >= 1),
   total_price             money(8),
      --
   primary key (item_num, order_num),
   foreign key (order_num)
      references orders (order_num),
   foreign key (stock_num, manu_code)
      references stock (stock_num, manu_code)
   );

create table manufact
   (
   manu_code               char(3),
   manu_name               char(15),
   lead_time               interval day(3) to day,
      --
   primary key (manu_code)
   );

create table orders
   (
   order_num               serial(1001),
   order_date              date,
   customer_num            integer not null,
```

```
   ship_instruct           char(40),
   backlog                 char(1),
   po_num                  char(10),
   ship_date               date,
   ship_weight             decimal(8,2),
   ship_charge             money(6),
   paid_date               date,
      --
   primary key (order_num),
   foreign key (customer_num) references customer (customer_num)
   );

create procedure read_address  (lastname char(15))
   returning char(15) as pfname, char(15) as plname,
      char(20) as paddress1, char(15) as pcity,
      char(2)  as pstate, char(5) as pzipcode;

   define p_fname, p_city char(15);
   define p_add char(20);
   define p_state char(2);
   define p_zip char(5);

   select fname,  address1, city, state, zipcode
      into p_fname,  p_add, p_city, p_state, p_zip
      from customer
      where lname = lastname;

   return p_fname, lastname, p_add, p_city, p_state, p_zip;
end procedure;

create table state
   (
   code                    char(2),
   sname                   char(15),
      --
   primary key (code)
   );

create table stock
   (
   stock_num               smallint,
   manu_code               char(3),
   description             char(15),
   unit_price              money(6),
   unit                    char(4),
   unit_descr              char(15),
      --
   primary key (stock_num, manu_code),
   foreign key (manu_code) references manufact
```

```
   );

create table sports
   (
   catalog_no              serial unique,
   stock_no                smallint,
   mfg_code                char(5),
   mfg_name                char(20),
   phone                   char(18),
   descript                varchar(255)
   );

create table call_type
   (
   call_code               char(1),
   code_descr              char(30),
      --
   primary key (call_code)
   );

create table catalog
   (
   catalog_num             serial(10001),
   stock_num               smallint not null,
   manu_code               char(3) not null,
   cat_descr               text,
   cat_picture             byte,
   cat_advert              varchar(255, 65),
      --
   primary key (catalog_num),
   foreign key (stock_num, manu_code)
      references stock constraint aa
   );

CREATE TABLE msgs
   (
   lang                    char(32),
   number                  integer,
   message                 nchar(255)
   );

SET NO COLLATION;

CREATE INDEX idxmsgs_enus on msgs (message);

SET COLLATION 'fr_fr.8859-1';   -- Create an index using fr_fr.8859-1.

CREATE INDEX idxmsgs_frfr on msgs (message);
```

```
SET NO COLLATION;

create table log_record
   (
   item_num    smallint,
   ord_num      integer,
   username     character(32),
   update_time  datetime year to minute,
   old_qty      smallint,
   new_qty      smallint
   );

create trigger upqty_i
   update of quantity on items
      referencing old as pre_upd
      new as post_upd
   for each row
      (
      insert into log_record
         values (pre_upd.item_num, pre_upd.order_num, user, current,
         pre_upd.quantity, post_upd.quantity)
      );

create view custview (firstname, lastname, company, city)
    as
    select fname, lname, company, city
    from customer
    where city = 'Redwood City'
    with check option;

create view someorders (custnum,ocustnum,newprice)
   as
   select orders.order_num,items.order_num,items.total_price*1.5
   from orders, items
   where orders.order_num = items.order_num
   and items.total_price > 100.00;
```

11.3.1 Explanation of the process example

With regards to Example 11-1 on page 610, refer to the following:

- This data model contains:
  - Thirteen SQL tables.
  - Two SQL views. The SQL view Custview has a new integrity constraint built within it (a filter constraint) but references only one table. The SQL view Someorders joins two tables and has a single filter within it.

  - One SQL stored procedure, which does a single table read with a filter.
  - One SQL trigger. One SQL update of Items. Quantity, a new record, is SQL inserted into another SQL table.
  - Ten SQL primary keys, and eight SQL foreign keys.
  - There are three additional (non-primary key, non-foreign key) indexes which are made, and they are Customer.Zipcode, and two on Msgs.Message.
- The two indexes on Msgs.Message are present to demonstrate indexes on varying language sets. As indexes on a 255 multi-byte column in a 291 total byte record length table, these indexes will surely be removed (again, these are present for demonstration purposes only).
- Additionally, we will not bother to cost the need for the primary key and foreign key indexes since their use (presence) is more feature functionality than pure support for query processing.

Example 11-1 on page 610 provides us the product of our logical and/or physical data model. At some point before, during, or after the creation of the data model, we considered the concept of security and roles. For example:

- During user discovery (phase one of the Waterfall Method SDLC), we may have discovered that there were two categorizations of users accessing this system; operators and senior operators. Operators are given certain SQL permissions (such as SELECT and UPDATE) on given tables and columns, and so are senior operators (presumably with more permissions).
- From the perspective of the defined user roles, we also document the expected use of the system:
  - An operator is found to execute, for example, SQL-SELECT-001, then 002, then 003, in sequence, at least 22 times per hour, with a think time (pause) of five seconds per iteration.
  - The senior operator is also defined to follow given operations and sequences.
  - Acceptable service levels (timings) are determined for all of the above.
- The creation of this package of application definition, knowledge, and requirements forms the logical process model.

11.4 An SQL DML example

Also from the demonstration database (the source of the SQL DDL in Example 11-1 on page 610), we capture a number of SQL DMLs. These are shown in Example 11-2 on page 616.

Example 11-2 A contributor to the logical process model

```
0000
0001 --  This file has been edited for clarity.
0002
0003 delete from stock
0004 where stock_num = 102;
0005
0006 insert into sports
0007    values (0,18,'PARKR', 'Parker Products', '503-555-1212',
0008    'Heavy-weight cotton canvas gi, designed for aikido or
0009    judo but suitable for karate. Quilted top with side ties,
0010    drawstring waist on pants. White with white belt. Pre-washed
0011    for minimum shrinkage. Sizes 3-6.');
0012
0013 select
0014    max (ship_charge),
0015    min (ship_charge)
0016 from orders;
0017
0018 select
0019    o.order_num,
0020    sum (i.total_price) price,
0021    paid_date - order_date span
0022 from orders o, items i
0023 where o.order_date > '01/01/89'
0024    and o.customer_num > 110
0025    and o.order_num = i.order_num
0026 group by 1, 3
0027 having count (*) < 5
0028 order by 3
0029 into temp temptab1;
0030
0031 SELECT message
0032 FROM msgs
0033 WHERE lang = 'en_us'
0034 ORDER BY message;
0035
0036 select
0037    order_num,
0038    count(*) number,
0039    avg (total_price) average
0040 from items
0041 group by order_num
0042 having count(*) > 2;
0043
0044 select
0045    c.customer_num, c.lname, c.company,
0046    c.phone, u.call_dtime, u.call_descr
```

```
0047 from customer c, cust_calls u
0048 where c.customer_num = u.customer_num
0049 order by 1;
0050
0051 select
0052    c.customer_num, c.lname, c.company,
0053    c.phone, u.call_dtime, u.call_descr
0054 from customer c, outer cust_calls u
0055 where c.customer_num = u.customer_num
0056 order by 1;
0057
0058 select
0059    c.customer_num, c.lname, o.order_num,
0060    i.stock_num, i.manu_code, i.quantity
0061 from customer c, outer (orders o, items i)
0062 where c.customer_num = o.customer_num
0063    and o.order_num = i.order_num
0064    and manu_code IN ('KAR', 'SHM')
0065 order by lname;
0066
0067 select
0068    c.customer_num, lname, o.order_num,
0069    stock_num, manu_code, quantity
0070 from customer c, outer (orders o, outer items i)
0071 where c.customer_num = o.customer_num
0072    and o.order_num = i.order_num
0073    and manu_code IN ('KAR', 'SHM')
0074 order by lname;
0075
0076 select
0077    c.customer_num, lname, o.order_num,
0078    order_date, call_dtime
0079 from customer c, outer orders o, outer cust_calls x
0080 where c.customer_num = o.customer_num
0081    and c.customer_num = x.customer_num
0082 order by 1
0083 into temp service;
0084
0085 select *
0086    from stock
0087 where description like '%bicycle%'
0088    and manu_code not like 'PRC'
0089 order by description, manu_code;
0090
0091 select
0092    order_num, total_price
0093 from items a
0094 where 10 >
0095       (
```

```
0096       select count (*)
0097       from items b
0098       where b.total_price < a.total_price
0099       )
0100 order by total_price;
0101
0102 select distinct stock_num, manu_code
0103    from stock
0104    where unit_price < 25.00
0105 union
0106 select stock_num, manu_code
0107    from items
0108    where quantity > 3;
0109
0110 update sports
0111    set phone = '808-555-1212'
0112 where mfg_code = 'PARKR';
```

11.4.1 Explanation of DML example

With regards to Example 11-2 on page 616, refer to the following:

- Entries such as the SQL DELETE on lines 0003-0004 are less than ideal in a production applications, but serve our needs here.
- Each of the distinct SQL statements in Example 11-2 on page 616 will be found within function calls of the given application development language, such as Java or Visual Basic. Following the design pattern of Model-View-Controller, each of these (Model component) functions will offer little in the way of program logic. Each will perform their given SQL statement, and contain the minimum amount of error processing and recovery code.
- Each of the individual SQL DML statements found inside Example 11-2 on page 616 will be further discussed as follows:
  - Each single statement will be reviewed as though it were a member of a distinct SQL API function call.
  - Example 11-2 on page 616 contains one SQL DELETE, one SQL INSERT, one SQL UPDATE, and twelve SQL SELECTs.

Lines 003-0004, the SQL DELETE

The SQL DELETE on lines 003-0004 exist in the real world as something similar to:

```
DELETE FROM stock WHERE stock_num = ?;
```

And a host variable will be supplied on execution. See the following:

- From the SQL schema in Example 11-1 on page 610, the Stock table acts as our inventory table, where Manufact acts like the list of valid stock providers (Manufacturers). We can determine this from the relationships in these two SQL tables, as well as from the sample SQL SELECTs in Example 11-2 on page 616.
- The primary key on the Stock table is based on the composite key (Stock_num, Manu_code).
- Since the SQL DELETE in this example references a subset of the composite key (the leading columns) we will be concerned about to the cardinality of this key value if we were performing a two table join under this relationship (see example in “Query optimizer by example” on page 503).
- As it stands, the filter criteria in this SQL WHERE clause is supported via an index, the primary key to this table.
- The resultant query plan will be reviewed to confirm use of the index, and the observed service level (timing) of this statement will be checked against necessary targets.

Lines 0006-0011, the SQL INSERT

Again, this code is taken from a sample database. The real-world SQL INSERT will resemble SQL closer to:

```
INSERT INTO sports VALUES (?, ?, ?, ?, ?, ?);
```

Host variables will supply the column values on execution. We check for the presence of SQL TRIGGERS on the event of this SQL INSERT. If SQL TRIGGERS cause subsequent SQL SELECTS, UPDATES, or DELETES, then these statements are costed. In this our actual case, no SQL TRIGGERs are in effect for this SQL INSERT. The observed service level (timings) of this statement are checked against necessary targets.

Lines 0013-0016, SQL SELECT # 1

This SQL SELECT returns two aggregate columns from a single SQL table with no joins or filters. Since Orders.Ship_charge is not indexed, this SQL SELECT will cause a sequential scan of this entire table. If a higher observed service level (timings) was needed for this specific statement, tuning will have to be performed to this database server or data model. (Does the relational database server provide a technology to know the maximum and minimum of this column without requiring an index, other?)

If the required service level is met, then we are fine. If the service level is not met, then we need to measure the anticipated frequency of the execution of this statement, and the importance that it execute as requested. Solutions to improve the performance of this statement will include:

- Indexing this column.
- Creating a materialized view or materialized query table to support this query.
- Creating SQL TRIGGERS on the SQL INSERT, UPDATE, or DELETE of this table and keeping these statistics as calculated values in a second, supporting SQL table.
- Others, as allowed for by the given relational database vendor.

Lines 0018-0029, SQL SELECT #2

This SQL SELECT reads from two SQL tables where the tables are joined and the Orders table has two range operand filter criteria (lines 0023 and 0024). There is a SQL GROUP BY to support one of the aggregate expressions, (line 0020). Also, there is a filter on an additional aggregate expression, (line 0027).

Since Items.Order_num is indexed, as shown in Example 11-1 on page 610 (it is an anchored primary key contributor within Items), the table order is likely to be Orders then Items, with an indexed table access method on Items. The aggregate calculations will be expensive and timings could be reduced by maintaining these columns within Orders using SQL TRIGGERs. The redundant columns we will keep in Order include Price (the calculated order total price) and Item Count (the number of line items in a given order). Span will not need to be precalculated since its resultant value is the product of two columns on the same row (entity instance) from Orders.

Lines 0031-0034, SQL SELECT #3

This SQL SELECT reads from one SQL table, and has one filter criteria. From the SQL DDL in Example 11-1 on page 610, we see that Msgs.Lang is not indexed, but that the SQL ORDER BY Column Msgs.Message is. Earlier we commented that the index on Msgs.Message is not likely to be performant, since it occupies 255 of 291 bytes (column length versus total record length in the table).

This SQL SELECT will be processed via a sequential scan on this table to satisfy the filter criteria, and results will be sent to the relational database server sort package to satisfy the SQLORDER BY clause. If observed service levels are not being met, an index on Msgs.Lang will help. If Msgs.Message were a smaller field, then a composite index on Msgs (Lang, Message) would be ideal, since this single SQL index will service both the filter criteria and the SQL ORDER BY clause. See also pseudo by clause in 10.6.1, “Query rewrite” on page 568.

Lines 0036-0042, SQL SELECT #4

This is a very interesting SQL SELECT. In simple language, we explain these lines:

- In the data model, the Orders table is the order header. That is, the columns within a customer order that do not repeat (Order Date, Customer Number, and other). The Items table serves as the Order Detail table for the Order Line Items.
- The SQL SELECT essentially reports on Order header conditions; the count of line items in the order, and the average price.

Note: This is a very interesting SQL SELECT because it exemplifies the structure and cost of SQL SELECTs received when you do not process the model.

- As the SQL SELECT currently exists, there is a SQL GROUP BY on Items.Order_num, which is an anchored column in a foreign key. We could receive index support for the processing of this query, but that fact needs to be confirmed by reading the query plan. While there are no filters on this SQL SELECT, the presence of the SQL GROUP BY calls for records to be read in sorted fashion (therefore, desire for an index).
- If the current SQL SELECT does not meet requested service levels (timings), the ideal index in this table will be on Items (Order_num, Total_price), allowing for a key only indexed retrieval (also assuming we do not keep these aggregate columns in the Order header table).
- As we review this query, we can discover one other area for optimization:
  - The SQL table, Items, has three indexes, that include a primary key on (Item_num, Order_num), foreign key on (Order_num), and foreign key on (Stock_num, Manu_code).
  - As a column, the item number (that is the sequencing of line items within a customer order), is probably insignificant. “Show me all third line items on a multi-item customer order” is not a typical query, and probably not a query we need to support.
  - If the primary key were made to be on (Order_num, Item_num), then the foreign key for (Order_num) could also be served from this one index. The net result is one less index on this table; always a good thing.
- As implied above, it might be better to model the item count (per order) at the order header level, and average (Items.Total_price) at the Items level, as a calculated column. *All of this depends of the needs of the user application; the observed frequency, concurrency and spread of (SQL) statements run, and the requested service levels (timings).*

Lines 0044-0049, SQL SELECT #5

This SQL SELECT returns all records from two SQL tables where the tables are joined. From Example 11-1 on page 610, we can see that both Customer.Customer_num and Cust_calls.Customer_num are indexed.

Similar to SQL SELECT #4 above, and from Example 11-1 on page 610, we see that Cust_calls.Customer_num could lose one index. That column is anchored in both a primary key and foreign key in that table, which is largely redundant.

Lines 0051-0056, SQL SELECT #6

The only difference between SQL SELECT #5 and #6 is the presence of the SQL OUTER keyword on line 0054. Because SQL Table Cust_calls in subservient to Customer, the Customer table will be processed first in SQL SELECT #6. If this query criteria causes less than requested service levels (timings), we might seek to model around this condition.

Lines 0058-0065, SQL SELECT #7

This SQL SELECT reads from three SQL tables where all tables are joined, and the SQL table Items has a filter, (line 0064). Customer is the dominant SQL table, (line 0061), while SQL Tables Orders and Items are placed in a SQL OUTER clause.

From Example 11-1 on page 610, we can determine that all of the join columns are indexed. The filter on Items.Manu_code is not anchored via an index, and is not a member of the index that will be needed to perform the join into SQL table Items.

If requested service levels for this SQL SELECT were not being met, you might consider possibly removing the SQL OUTER clause, replacing it with a non-OUTER SQL SELECT, and then UNION ALL customers with no orders (the net result of the SQL OUTER clause). Also, reviewing the indexes in place on Items to allow for key only processing, or single index processing of both the join and the filter criteria.

Lines 0067-0074, SQL SELECT #8

The only difference between SQL SELECT #7 and #8 is the presence of another SQL OUTER clause, as shown on line 0070. The same tuning guidelines will be in effect as for SQL SELECT #7. That is, removal (engineering around) the SQL OUTER clause, and including the Items.Manu_code filter in any participating indexes.

Lines 0076-0083, SQL SELECT #9

While SQL SELECT #9 references different table and join columns than SQL SELECT #7 and #8, the same tuning guidelines apply here as well. All of the join columns are supported via indexes.

Lines 0085-0089, SQL SELECT #10

This SQL SELECT reads from one SQL table with two filter criteria. Both filter criteria offer poor selectivity and are not likely to use indexes (negation of index, non-initial substring, line 0087, and negation of index, poor selectivity filter, line 0088, the inequality).

Note: This is the only SQL SELECT in the grouping that makes use of a "SELECT *"; that is, the wildcard to select all columns in the list. We want to remove this syntax, since adding or dropping columns from the SQL table may break our application program code. In effect, we will later attempt to SELECT 10+1 columns into a list of 10 variables.

If this were a critical query, we might need to model a Keywords type table; as in, the description of a given Stock unit is recognized by the keywords as recorded in a primary key table of symbols.

Lines 0091-0100, SQL SELECT #11

This SQL SELECT provides a correlated subquery, as shown on line 0098 and the reference to an outer table column, A.Total_price, inside the inner SELECT.

Note: This is another interesting example. An experienced engineer would detect that this query could be rewritten, if the query optimizer query rewrite capability does not. This specific example supports the argument for a process model versus having the application programmers author their own SQL DML statements.

The query optimizer query rewrite capability could detect that the inner SQL SELECT can be executed once, to retrieve the effect of,

```
SELECT total_price, COUNT(*) FROM items GROUP BY 1;
```

And then join the outer SQL SELECT to that result. The correlated subquery can be rewritten as a join, since the cardinality of the outer to inner SQL SELECTs is many to one.

Lines 0102-0108, SQL SELECT #13

This is a unioned SQL SELECT. The presence of the UNION keyword, without a UNION ALL, will call to remove duplicates from the resultant data set. This condition makes the DISTINCT keyword in the first SQL SELECT redundant (not necessary).

Table access methods for both SQL SELECTs are sequential, since neither filter criteria is supported by an index.

In simple terms, this SQL SELECT seems to report the effect of; show me every item I have sold for fewer than $25 and of which I have more than three. If this is an important query, we could data model it as such.

Lines 0110-0112, the SQL UPDATE

The Sports table has no index, and the table access method for this query will be sequential. If a greater service level (timings) is requested, an index on Sports.Manu_code will work.

Results: SQL objects not used

By the review of the SQL API above, we see that the two SQL VIEWS are currently unused, and the only additional index, that on Customer.Zipcode, is also unused. The index, at least, could be removed since maintaining it adds cost to the application.

Based on the necessary timings of the above SQL API routines, there may have been a call to change the data model. This is primarily to maintain redundant or precalculated (expression) columns.

11.5 Conclusions

In this chapter we have discussed how you can apply the query optimization techniques presented in Chapter 10. The discussions are heavily oriented to application development, but that is the means by which we access and analyze the data stored in the data warehouse to feed the business intelligence environment.

Whether by stored application or by ad hoc query, the basic objective of business intelligence is to get the data out of the data warehouse, analyze it, and enable more informed business decisions to improve management and control of the business enterprise.

The base of your data structure is the data model. And to support the requirements of BI, we have focussed specifically on dimensional modeling.

However, having a data structure is just part of a solution. The key is to have a data structure that supports understanding and use of the data, ease of access for analysis, and high performance of the applications and queries.

Knowing that your data model is highly performant before ever writing any applications or queries is of significant value. It can enable improved client satisfaction by delivering highly performant queries from the initial introduction of the system. It can also save time and money by eliminating the need to rebuild poorly designed data structures and applications. In short, it can enable you to do it right the first time.

Glossary

Access Control List (ACL). The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

Additive Measure. Measure of a fact that can be added across all dimensions.

Aggregate. Pre-calculated and pre-stored summaries, kept in the data warehouse to improve query performance

Aggregation. An attribute level transformation that reduces the level of detail of available data. For example, having a Total Quantity by Category of Items rather than the individual quantity of each item in the category.

Analytic. An application or capability that performs some analysis on a set of data.

Application Programming Interface. An interface provided by a software product that enables programs to request services.

Associative entity. An entity created to resolve a many-to-many relationship into two one-to-many relationships.

Asynchronous Messaging. A method of communication between programs in which a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to its message.

Attribute. A characteristic of an entity, such as a field in a dimension table.

BLOB. Binary Large Object, a block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

Business subject area. A particular function or area within an enterprise whose processes and activities can be described by a defined set of data elements.

Candidate key. One of multiple possible keys for an entity.

Cardinality. The number of elements in a mathematical set or group.

Commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

Compensation. The ability of DB2 to process SQL that is not supported by a data source on the data from that data source.

Composite Key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

Computer. A device that accepts information (in the form of digitalized data) and manipulates it for some result based on a program or sequence of instructions on how the data is to be processed.

Configuration. The collection of brokers, their execution groups, the message flows and sets that are assigned to them, and the topics and associated access control specifications.

Connector. See Message processing node connector.

Cube. Another term for a fact table. It can represent “n” dimensions, rather than just three (as may be implied by the name).

DDL (Data Definition Language)**.** a SQL statement that creates or modifies the structure of a table or database. For example, CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE DATABASE.

DML (Data Manipulation Language)**.** An INSERT, UPDATE, DELETE, or SELECT SQL statement.

Data Append. A data loading technique where new data is added to the database leaving the existing data unaltered.

Data Append. A data loading technique where new data is added to the database leaving the existing data unaltered.

Data Cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content. This is typically to improve the quality, consistency, and usability of the data.

Data Federation. The process of enabling data from multiple heterogeneous data sources to appear as if it is contained in a single relational database. Can also be referred to “distributed access”.

Data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope - such as for a department, workgroup, or subject area. It could be independent, or derived from another data warehouse environment (dependent).

Data mart - Dependent. A data mart that is consistent with, and extracts its data from, a data warehouse.

Data mart - Independent. A data mart that is standalone, and does not conform with any other data mart or data warehouse.

Data Mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

Data Model. A representation of data, its definition, characteristics, and relationships.

Data Partition. A segment of a database that can be accessed and operated on independently even though it is part of a larger data structure.

Data Refresh. A data loading technique where all the data in a database is completely replaced with a new set of data.

Data silo. A standalone set of data in a particular department or organization used for analysis, but typically not shared with other departments or organizations in the enterprise.

Data Warehouse. A specialized data environment developed, structured, shared, and used specifically for decision support and informational (analytic) applications. It is subject oriented rather than application oriented, and is integrated, non-volatile, and time variant.

Database Instance. A specific independent implementation of a DBMS in a specific environment. For example, there might be an independent DB2 DBMS implementation on a Linux® server in Boston supporting the Eastern offices, and another separate and independent DB2 DBMS on the same Linux server supporting the western offices. They would represent two instances of DB2.

Database Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DataBlades. These are program modules that provide extended capabilities for Informix databases, and are tightly integrated with the DBMS.

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP® Business Information Warehouse.

Debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

Deploy. Make operational the configuration and topology of the broker domain.

Dimension. Data that further qualifies and/or describes a measure, such as amounts or durations.

Distributed Application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

Dynamic SQL. SQL that is interpreted during execution of the statement.

Engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

Enrichment. The creation of derived data. An attribute level transformation performed by some type of algorithm to create one or more new (derived) attributes.

Entity. A person, place, thing, or event of interest to the enterprise. Each entity in a data model is unique.

Extenders. These are program modules that provide extended capabilities for DB2, and are tightly integrated with DB2.

FACTS. A collection of measures, and the information to interpret those measures in a given context.

Federated data. A set of physically separate data structures that are logically linked together by some mechanism, for analysis, but which remain physically in place.

Federated Server. Any DB2 server where the WebSphere Information Integrator is installed.

Federation. Providing a unified interface to diverse data.

Foreign Key. An attribute or set of attributes that refer to the primary key of another entity.

Gateway. A means to access a heterogeneous data source. It can use native access or ODBC technology.

Grain. The fundamental atomic level of data represented in a fact table. As examples, typical grains that could be used, when considering time, would be day, week, month, year, and so forth.

Granularity. The level of summarization of the data elements. It refers to the level of detail available in the data elements. The more detail data that is available, the lower the level of granularity. Conversely, the less detail that is available, the higher the level of granularity (or level of summarization of the data elements).

Instance. A particular realization of a computer process. Relative to database, the realization of a complete database environment.

Java Database Connectivity. An application programming interface that has the same characteristics as ODBC but is specifically designed for use by Java database applications.

Java Development Kit. Software package used to write, compile, debug and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit that allows you to run Java applets and applications.

Key. An attribute of set of attributes that uniquely identifies an entity.

Materialized Query Table. A table where the results of a query are stored, for later reuse.

Measure. A data item that measures the performance or behavior of business processes.

Message domain. The value that determines how the message is interpreted (parsed).

Message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

Message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible to generate a bit stream for an outgoing message from the internal representation.

Meta Data. Typically called data (or information) about data. It describes or defines data elements.

MOLAP. Multidimensional OLAP. Can be called MD-OLAP. It is OLAP that uses a multidimensional database as the underlying data structure.

Multidimensional analysis. Analysis of data along several dimensions. For example, analyzing revenue by product, store, and date.

Multi-Tasking. Operating system capability which allows multiple tasks to run concurrently, taking turns using the resources of the computer.

Multi-Threading. Operating system capability that enables multiple concurrent users to use the same program. This saves the overhead of initiating the program multiple times.

Nickname. An identifier that is used to reference the object located at the data source that you want to access.

Node Group. Group of one or more database partitions.

Node. See Message processing node and Plug-in node.

Non-Additive Measure. Measure of a fact that cannot be added across any of its dimensions, such as a percentage.

ODS. (1) Operational data store: A relational table for holding clean data to load into InfoCubes, and can support some query activity. (2) Online Dynamic Server - an older name for IDS.

OLAP. OnLine Analytical Processing. Multidimensional data analysis, performed in real time. Not dependent on underlying data schema.

Open Database Connectivity. A standard application programming interface for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call level interface (CLI) specification of the X/Open SQL Access Group.

Optimization. The capability to enable a process to execute and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the user.

Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

Pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

Pivoting. Analysis operation where user takes a different viewpoint of the results. For example, by changing the way the dimensions are arranged.

Primary Key. Field in a table record that is uniquely different for each record in the table.

Process. An instance of a program running in a computer.

Program. A specific set of ordered operations for a computer to perform.

Pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be executed. More simply, a pushdown operation is one that is executed at a remote server.

Relationship. The business rule that associates entities.

ROLAP. Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

Roll-up. Iterative analysis, exploring facts at a higher level of summarization.

Semi-additive Measure. Measure of a fact that can be added across only some of its dimensions, such as a balance.

Server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

Spreadmart. A standalone, non-conforming, non-integrated set of data, such as a spreadsheet, used for analysis by a particular person, department, or organization.

Static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

Static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

Subject Area. A logical grouping of data by categories, such as customers or items.

Synchronous Messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

Task. The basic unit of programming that an operating system controls. Also see Multi-Tasking.

Thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. Also see Multi-Threading.

Type Mapping. The mapping of a specific data source type to a DB2 UDB data type

Unit of Work. A recoverable sequence of operations performed by an application between two points of consistency.

User Mapping. An association made between the federated server user ID and password and the data source (to be accessed) used ID and password.

Virtual Database. A federation of multiple heterogeneous relational databases.

Warehouse Catalog. A subsystem that stores and manages all the system meta data.

Wrapper. The means by which a data federation engine interacts with heterogeneous sources of data. Wrappers take the SQL that the federation engine uses and maps it to the API of the data source to be accessed. For example, they take DB2 SQL and transform it to the language understood by the data source to be accessed.

xtree. A query-tree tool that allows you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

ACS	access control system
ADK	Archive Development Kit
AIX®	Advanced Interactive eXecutive from IBM
API	Application Programming Interface
AQR	automatic query re-write
AR	access register
ARM	automatic restart manager
ART	access register translation
ASCII	American Standard Code for Information Interchange
AST	Application Summary Table
BLOB	Binary Large OBject
BW	Business Information Warehouse (SAP)
CCMS	Computing Center Management System
CFG	Configuration
CLI	Call Level Interface
CLOB	Character Large OBject
CLP	Command Line Processor
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CS	Cursor Stability
DAS	DB2 Administration Server
DB	Database
DB2	Database 2™
DB2 UDB	DB2 Universal DataBase
DBA	Database Administrator
DBM	DataBase Manager
DBMS	DataBase Management System
DCE	Distributed Computing Environment
DCM	Dynamic Coserver Management
DCOM	Distributed Component Object Model
DDL	Data Definition Language - a SQL statement that creates or modifies the structure of a table or database. For example, CREATE TABLE, DROP TABLE.
DES	Data Encryption Standard
DIMID	Dimension Identifier
DLL	Dynamically Linked Library
DMDL	Dimensional Modeling Design Life Cycle
DML	Data Manipulation Language - an INSERT, UPDATE, DELETE, or SELECT SQL statement.
DMS	Database Managed Space
DPF	Data Partitioning Facility
DRDA®	Distributed Relational Database Architecture™
DSA	Dynamic Scalable Architecture
DSN	Data Source Name
DSS	Decision Support System
EAI	Enterprise Application Integration
EAR	Entity, Attribute, Relationship data model. Also denoted by E/R.
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDA	Enterprise Data Architecture

EDU	Engine Dispatchable Unit
EDW	Enterprise Data Warehouse
EGM	Enterprise Gateway Manager
EJB™	Enterprise Java Beans
E/R	Enterprise Replication
E/R	Entity, Attribute, Relationship data model. Also denoted by EAR.
ERP	Enterprise Resource Planning
ESE	Enterprise Server Edition
ETL	Extract, Transform, and Load
ETTL	Extract, Transform/Transport, and Load
FJS	Filter, then Join, then Sort
FP	Fix Pack
FTP	File Transfer Protocol
Gb	Giga bits
GB	Giga Bytes
GUI	Graphical User Interface
HADR	High Availability Disaster Recovery
HDR	High availability Data Replication
HPL	High Performance Loader
I/O	Input/Output
IBM	International Business Machines Corporation
ID	Identifier
IDE	Integrated Development Environment
IDS	Informix Dynamic Server
II	Information Integrator
IMG	Integrated Implementation Guide (for SAP)
IMS™	Information Management System
ISAM	Indexed Sequential Access Method
ISM	Informix Storage Manager
ISV	Independent Software Vendor
IT	Information Technology
ITR	Internal Throughput Rate
ITSO	International Technical Support Organization
IX	Index
J2EE	Java 2 Platform Enterprise Edition
JAR	Java Archive
JDBC™	Java DataBase Connectivity
JDK™	Java Development Kit
JE	Java Edition
JMS	Java Message Service
JRE™	Java Runtime Environment
JVM™	Java Virtual Machine
KB	Kilobyte (1024 bytes)
LDAP	Lightweight Directory Access Protocol
LPAR	Logical Partition
LV	Logical Volume
Mb	Mega bits
MB	Mega Bytes
MDC	Multidimensional Clustering
MPP	Massively Parallel Processing
MQI	Message Queuing Interface
MQT	Materialized Query Table
MRM	Message Repository Manager
MTK	DB2 Migration ToolKit for Informix
NPI	Non-Partitioning Index
ODBC	Open DataBase Connectivity
ODS	Operational Data Store
OLAP	OnLine Analytical Processing
OLE	Object Linking and Embedding

OLTP	OnLine Transaction Processing
ORDBMS	Object Relational DataBase Management System
OS	Operating System
O/S	Operating System
PDS	Partitioned Data Set
PIB	Parallel Index Build
PSA	Persistent Staging Area
RBA	Relative Byte Address
RBW	Red Brick™ Warehouse
RDBMS	Relational DataBase Management System
RID	Record Identifier
RR	Repeatable Read
RS	Read Stability
SCB	Session Control Block
SDK	Software Developers Kit
SDLC	Software Development Life Cycle
SID	Surrogage Identifier
SMIT	Systems Management Interface Tool
SMP	Symmetric MultiProcessing
SMS	System Managed Space
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPL	Stored Procedure Language
SQL	Structured Query
TCB	Thread Control Block
TMU	Table Management Utility
TS	Tablespace
UDB	Universal DataBase
UDF	User Defined Function
UDR	User Defined Routine
URL	Uniform Resource Locator
VG	Volume Group (Raid disk terminology).
VLDB	Very Large DataBase
VP	Virtual Processor
VSAM	Virtual Sequential Access Method
VTI	Virtual Table Interface
WSDL	Web Services Definition Language
WWW	World Wide Web
XBSA	X-Open Backup and Restore APIs
XML	eXtensible Markup Language
XPS	Informix eXtended Parallel Server

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 638. Note that some of the documents referenced here may be available in softcopy only.

- *Data Modeling Techniques for Data Warehousing*, SG24-2238
- *Business Performance Management . . .Meets Business Intelligence*, SG24-6349
- *Preparing for DB2 Near-Realtime Business Intelligence*, SG24-6071
- *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653

Other publications

These publications are also relevant as further information sources:

- “IBM Informix Database Design and Implementation Guide,” G215-2271.
- Ralph Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*, John Wiley & Sons, 1996.
- Ralph Kimball, Laura Reeves, Margy Ross, and Warren Thornthwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*, John Wiley & Sons, 1998.
- Ralph Kimball and Margy Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, 2nd Ed., John Wiley & Sons, 2002.
- Flemming, C., *Handbook of Relational Database Design*, Addison-Wesley, 1989.
- Yourdon, E., *Modern Structured Analysis*, Yourdon Press, 1989.
- Federal Information Processing Standards Publication 184, “Integration Definition for Information Modeling (IDEF1X),” December 1993.

- Claudia Imhoff, Nicholas Galemmo, Jonathan G. Geiger, *Mastering Data Warehouse Design: Relational and Dimensional Techniques*, John Wiley & Sons, 2003,
- Toby J. Teorey, *Database Modeling & Design*, The Morgan Kaufmann Series in Data Management Systems, Academic Press, 1999.
- W. H. Inmon, *Building the Data Warehouse*, John Wiley & Sons, 2005.
- W. H. Inmon, Claudia Imhoff, Ryan Sousa, *Corporate Information Factory*, 2nd Edition, John Wiley & Sons, 2000.
- Melissa A. Cook, *Building Enterprise Information Architectures*, Prentice-Hall, Inc., 1996.
- Lou Agosta, *The Essential Guide to Data Warehousing*, Prentice-Hall, Inc., 1999.

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Dimensional Modeling: In a Business Intelligence Environment

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Dimensional Modeling: In a Business Intelligence Environment

Dimensional modeling for easier data access and analysis

Maintaining flexibility for growth and change

Optimizing for query performance

In this IBM Redbook we describe and demonstrate dimensional modeling techniques and technology, specifically focused on business intelligence and data warehousing. It is to help the reader understand how to design, maintain, and use a dimensional model for data warehousing that can provide the data access and performance required for business intelligence. Business intelligence is comprised of a data warehousing infrastructure, and a query, analysis, and reporting, environment. Here we focus on the data warehousing infrastructure. But only a specific element of it, the dimensional model, or, more precisely, the topic of dimensional modeling and its impact on the business and business applications. The objective is not to provide a treatise on dimensional modeling techniques, but to focus at a more practical level. There is technical content for designing and maintaining such an environment, but also business content. For example, we use case studies to demonstrate how dimensional modeling can impact the business intelligence requirements. In addition, we provide a detailed discussion on the query aspects of BI and data modeling. For example, we discuss query optimization and how to evalueate the performance of the data model prior to implementation. You need a solid base for your data warehousing infrastructure . . . . a solid dimensional model.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7138-00 ISBN 0738496448