Decision Support Databases Essentials

Antonio Albano, Salvatore Ruggieri
University of Pisa
Department of Computer Science
tonio.albano@gmail.com salvatore.ruggieri@unipi.it

Februray 13, 2015
Revision, December 20, 2023

CONTENTS

PREFACE

In our information-based society one of the most important applications for computers is *information storage and management* with a DBMS to support organizations both in *performing the business* and in *bringing the business* to the Web to allow new routes to market.

It is also well known that information overload is a huge challenge for businesses, but it is also an enormous opportunity in making smarter decisions based mainly on data analysis to improve productivity. Consequently, starting from the 1990s another important application of *information storage and management* to support organizations is *analyzing the business* with data-driven *Decision Support Systems* designed for summarizing large amounts of data into a form that is useful and easily interpretable to help managers to analyze the performance of *key business processes, worthy of improvement*.

Decision support applications involve quite complex analysis which cannot be efficiently executed against operational databases, optimized for online transaction processing. For this reason, organizations maintain a separate database, called a *data warehouse*, which is specifically organized for such complex analysis. The term *data warehouse* is a metaphor: a warehouse is a large structure where things are stored and organized for easy accessibility. However, a *data warehouse* is not only a large repository for historical data extracted from operational systems, but is organized to create the right models for measurable key business processes, to support informed decisions about how to improve them.

Organization

The text is organized in three major parts, plus an appendix with case studies.

- **Part I: Decision Support Systems and Multidimensional Modeling** (Chapters 1 through 4). **Chapter 1** provides a general overview of the purpose of *decision support systems*, and of the concepts of *data warehouse* and of *data warehousing process*. We also introduce the reason why data warehouses are used to analyze key business processes that are measurable and worthy of improvements, and consequently what is modeled in a data warehouse. **Chapter 2** presents the fundamental concepts about a conceptual model for designing data warehouses, and the logical data model to implement them. **Chapter 3** presents a data warehouse design process and a methodology for the implementation of a logical design schema. **Chapter 4** gradually presents the design of a data warehouse to support Customer Relationship Management (CRM).
- **Part II: Multidimensional Analysis** (Chapters 5). The chapter introduces the *business intelligence* tools for data analysis, and then focuses on the extensions of SQL for online analytic processing, called *Analytic SQL*, the fundamental user-oriented relational languages to analyze data for producing interesting reports to evaluate the performance of the modeled key processes in order to improve them.
- **Part III: Data Warehouse Systems: Storage, Indexing and Query Evaluation** (Chapters 6 through 9). These chapters present the relational DBMS technological extensions needed to support the analytic queries on data warehouses, because traditional DBMSs do not work well on data warehouses with huge volumes of data. Consequently, it is important not only to design a good data warehouse logical schema, but also to know how to use a specialized DBMS in the right way to develop a better performing business intelligence application. **Chapter 6** presents index types, different from the classical inverted indexes that should be available in a DBMS to optimize analytic star query plans. Then new data warehouse platforms are presented, such as *Column-based*, *In-memory* and *Data Warehouse Appliances*, which provide cost-effective scalability and simplify big data warehouse implementations. **Chapter 7** presents the importance of *materialized views*, i.e., pre-computed view query results

persistently stored, which, in addition to indexes, can significantly speed up complex analytic query processing. Examples of algorithms are presented to solve the problem of selecting an appropriate set of views to materialize. **Chapter 8** deals with analytic query optimization to exploit the technique of commuting group-by and join operators. **Chapter 9** deals with the query rewriting problem to exploit the use of materialized views. The existence of a materialized view is transparent to SQL queries, so a DBA can create or drop materialized views at any time without affecting the validity of SQL queries, as happens with indexes. A materialized view is useful if it is used by the DBMS to rewrite complex analytic queries to improve their performance, so it is important to know how a DBMS can rewrite an analytic query using materialized views in order to define those more useful.

– **Appendix**. A set of case studies is presented to apply the concepts presented in the chapters of the book.

Moreover, it has been decided to make this edition available for free on the web.

Acknowledgments

We would like to thank M. Mauro and the following students who provided useful feedback on draft versions of the book: F. Boscia, M. Borghi, N. Corti, F. Marchini, L. Milli, L. Morlino, S. Pietrosante, P. Serra and A. Tarquini.

A. A.
S. R.

Part I

Decision Support Systems and Multidimensional Modeling

Chapter 1

DECISION SUPPORT SYSTEMS

An overview of *decision support systems* is given below, particularly those data-driven, used to synthesize large amounts of data into a form that is useful to manage the business. The data are first organized in a special database called *data warehouse* and then analyzed with appropriate techniques, called *On Line Analytical Processing* (*OLAP*) or with semi-automatic and exploratory techniques, called *data mining*. Finally, the characteristics of systems for managing data warehouse are presented, which information should be represented using an appropriate *data model*, and how data can be used for decision-making.

1.1 Information Systems

Organizations have used information systems for centuries and they have used a variety of technologies to deal with information.

Definition 1.1

An **information system** is an organized collection of resources, people, and procedures finalized to collect, store, process and communicate the information needed to support the on-going activities.

Nowadays, information is considered to be a critical resource of any organization, as fundamental as capital or machinery, and, in fact, the majority of the labor force in the industrialized countries works in some way with information.

Information can be represented as data, images, text, and voice. Clearly, different types of organizations will have differing needs with respect to the kinds of information they use. However, the attention here will be on information represented as *structured data* shared by a variety of users within an organization, and managed using computers. Reductions in the costs of computer technology, improvements in performances, and new facilities to support the development of applications have created an increasing demand for data processing systems. We use the term *computerized information system* to refer to the hardware and software which is used for storing, retrieving, and processing the information which supports the functions of an organization.[1] In the following, in brief, we use the term *information system* for *computerized information system*.

Over time, there are continuous interactions between the two components of the information system and the rest of the organization that will change each other, and this requires attention of management to plan the evolution of both the organizational structure and the employee tasks (Figure 1.1)

1. Frequently in the literature, *information system* is used as synonym of *computerized information system*. Here, we prefer to make a distinction between the two terms to evidence the fact that a *computerized information system* will never completely substitute the global *information system* of an organization.

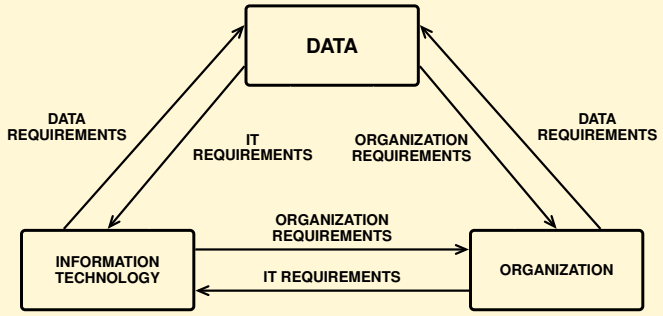


Figure 1.1: System Conception of an Information Systems

For example, new requirements of the organization may include the need of new categories of data being managed by the information system, and to adapt the information technology to provide new services (e.g, think of an organization that decides to offer web services). New categories of data to be managed can result in (a) a review of the organizational structure to review, for example, the tasks and professional employees, (b) an adjustment of the information technology used. The evolution of information technology can enable new opportunities for data management and set new requirements for employees and the organizational structure, and so on. The data, the organization, and the technology all interact and change each other.

1.2 Types of Information Systems

Information systems can be classified in several ways, but for our purposes it is useful to classify them in the following categories on the basis of the business activities that are required to support.

Definition 1.2 Taxonomy of Information Systems
Information systems can be classified into the following categories:

- **Operational**, to *perform* the business operational activities.
- **Decision support**, to provide the information that managers need for *analyzing the business*.
- **Web-based** for *E-commerce*, to *bring the business to the Web* to allow new routes to market.

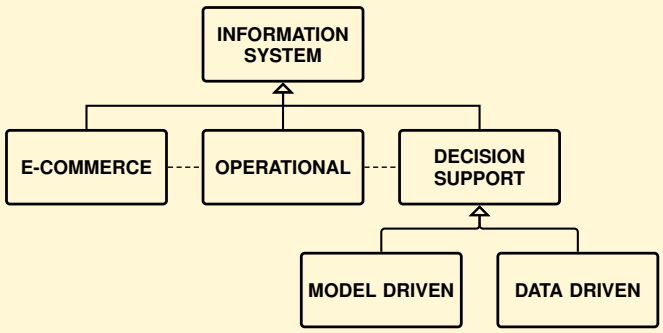


Figure 1.2: Types of Information Systems

These information system categories are all ongoing and in a constant state of improvement. They use different technology, have different objectives and require different skills to develop. In the following the

attention will be on the first two categories, and in particular on the *decision support* one, a key driver in the business world today (Figure 1.2).

1.2.1 Operational Systems

When an information system is implemented using the database technology, it will consist of an *operational database* and a collection of *application programs* (*transactions*) which are used to access and update the data quickly and efficiently (Figure 1.3). The main goal of such a transaction processing system is to maintain the correspondence between the database and the real-world situation it is modeling, as events occur in the real world.

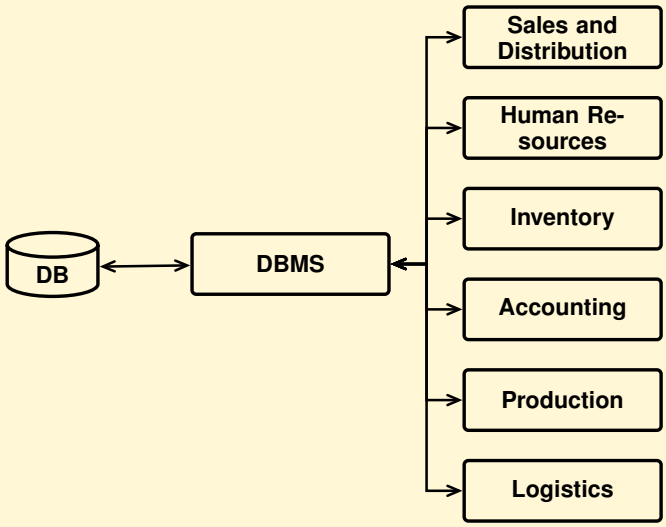


Figure 1.3: Transaction processing system

The data are under the control of a *Data Base Management System* (DBMS), a centralized or distributed software system, which provides the tools to define the database, to select the data structures needed to store and retrieve the data easily, and to access the data, interactively or by means of a programming language.

1.2.2 Decision Support Systems

Decision support information systems can be classified in *Management Information Systems* and *Decision Support Systems*.

The first one is used by middle tactical or administrative managers in monitoring and controlling their units to correct problems by making decisions based on comparing the actual performance and the planned performance (*variance report*). Decision support systems are used to make strategic decisions about the future directions of the business enterprise, using both historical internal data and external data.

For brevity, in the following we will use the term *Decision Support Systems* (DSS) for both types of decision support information systems.

DSS have been introduced in the organizations since the late '70s to help managers to make decisions of three types:

- **Structured**, when a well-defined decision-making procedure exists.
- **Unstructured**, when a well-defined decision-making procedure does not exists and the experience and creativity of the manager are required.

– **Semistructured**, when the decision-making procedure is partly defined and so it is also required the manager's creative intervention.

There is no strict correspondence between types of decision and levels of decision-making processes, however, at the operational level decisions tend to be more structured, at the tactical level decisions are mainly semistructured and at the strategic level decisions are typically unstructured.

The DSS have very different characteristics, but it is useful to classify them into two main types: *model-driven*, to take structured or semistructured decisions, or *data-driven* to take unstructured decisions.

The model-driven DSS are an evolution of the first proposals made at the end of the 70s for decision support systems and their value depends on the quality of the model used. The simplest solutions utilize spreadsheets for analysis of "what if", while more sophisticated models are used from operations research, simulation and artificial intelligence.

The data-driven DSS are designed to synthesize large amounts of data into a form that is useful and easily interpretable to help managers to assess the performance of business processes and make decisions to address and resolve any critical issues found. Their value depends on the type and quality of data generated using synthetic instruments called **Business Intelligence**. The term **intelligence** is used with the meaning of *investigating to find out something interesting*, like in *Intelligence Service*.

The operational data accumulated over time, integrated with those from external sources, are a potential source of information used by managers regardless of their decision-making level in the organization. The information is derived from the data summarized in an appropriate form and its relevance depends on the recipient. When experience, competence, and attitude are added to information, knowledge is created, and actions can be taken. To become actionable, knowledge should also be closely integrated with an organization's business processes.

In the following, data-driven DSS will be considered to see how they can be designed to support informed decisions.

Decision support applications involve quite complex analysis which cannot be efficiently executed against operational databases, optimized for *On Line Transaction Processing* (OLTP). For this reason, organizations maintain a separate database, called **data warehouse**, which is specifically organized for such complex analysis.

1.3 Data Warehouse: A Decision Support Database

The first and still now the most widely cited definition of data warehouse was provided by William Inmon in 1990:

Definition 1.3

A **data warehouse** is a subject-oriented, integrated, nonvolatile, and time-varying collection of data in support of management's decisions.

Let us examine each of these distinctive aspects of a data warehouse.

1. **Subject-oriented**. A data warehouse stores data by subject, not by applications, which is what distinguishes a data warehouse from an operational database, that stores information in order to optimize transaction processing. Business subjects differ from organization to organization. They are the critical subjects for an organization. For example, for a manufacturing company, these would include, sales, shipments, returns, and inventory.

   A *data mart* is database that has the same characteristics as a data warehouse, but is usually smaller and is focused on the data for one subject.
2. **Integrated**. Data are gathered into the data warehouse from a variety of sources and merged into a coherent whole. For example, a bank can collect different data on customers for the management of loans, current accounts, or stocks, but they must then be integrated for the purposes of the analysis of the services offered to customers.

3. **Time-variant**. For an operational system, the stored data contains the *current* values. On the other hand, the data in the data warehouse is meant for analysis and decision support, and is thus historical data identified with a particular time period.

   An operational system contains *current* data, while a data warehouse contains historical data over long time for analysis and decision support, therefore a time dimension is explicitly included in data so that trends and changes over time can be analyzed.
4. **Non-volatile**. The data in a data warehouse is primarily for query and analysis, and it is never changed interactively. This enables management to gain a consistent picture of the business. Periodically, new data may be added or those considered obsolete may be removed.
5. **Decision support**. The primary function of the data warehouse is for decision support, and so it must be specifically designed to answer business questions. Data is the reality that a computer records, stores, and processes. The lowest level in the perception of reality is sometimes referred to as "raw data". This data is of little benefit unless it can be turned into useful information and knowledge. Data must be *condensed into a more informative format* in such a way that managers (or more in general *knowledge workers* – executives, managers, and analysts) can get the essence of the underlying data.

   Three categories of decision support can be provided. Specifically:

   (a) **Reports**. Reporting is considered the lowest level of decision support. A reporting facility capable of generating informative reports for managers in time to be useful is of the utmost importance for the successful operation of any business.

   (b) **Multidimensional data analysis**, sometimes called *On Line Analytic Processing* (OLAP). Data analysis is usually accomplished interactively with some kind of data analysis tool. The goal of data analysis is to get useful information from the data.

   (c) **Exploratory data analysis**. This data analysis technique is very different from reports and multidimensional analysis: it uses what is called a *discovery technique of useful data models* with *data mining* algorithms. That is, the user does not ask a particular question about data, but rather he uses specific algorithms that analyze the data and report what they have discovered. Unlike reports and multidimensional analysis, where the user has to create and execute queries based on hypotheses, data mining algorithms search for answers. A comparison of the two approaches is shown in Table 1.1 with some example queries. Data mining algorithms are beyond the scope of this book.

Table 1.1: Comparison between OLAP and Exploratory data analysis

OLAP Query	Exploratory data analysis
Which customers spent most with us in the past year?	Which types of customer are likey to spend most with us in the coming year?
How much did the bank lose from loan defaulters in the past two years?	What are the characteristics of the customers most likely to default on their loans before the year is out?
What where the highest selling fashion items in our London stores?	What additional products are most likely to be sold to customers who buy sportswear?

A data warehouse is usually separated from an operational database for the following reasons:

– **Performance.** Special data organization, access and implementation methods are needed to support multidimensional views and data analysis which usually requires complex queries that would degrade the performance of operational transactions. Moreover, concurrency control and recovery DBMS modes are not compatible with data analysis.
– **Function.** Decision support requires (a) historical data, which operational databases do not typically

maintain, (b) consolidation (aggregation, summarization) of data from heterogeneous sources, such as operational databases, external sources, and (c) different sources typically use inconsistent data representations, codes and formats which have to be reconciled to enforce data quality.

Table 1.2 summarizes the differences between the traditional applications that use databases (*On Line Transaction Processing*, *OLTP*), and the decision support applications that use data warehouses (*On Line Analytical Processing*, *OLAP*).

Table 1.2: Comparison between OLTP and OLAP

	OLTP	**OLAP**
Function	Operational processing	Decision support
Users	Clerk, IT professional	Knowledge worker
DB design	Application-oriented	Subject-oriented
Usage	90% repetitive	90% ad-hoc
Data	Current, detailed, relational	Historical, summarized, multidimensional, integrated
Access	Read/write	Complex read query
No of users	A lot	Few
DB size	100 MB to GB	100 GB to TB
Orientation	Transactions	Analysis

OLAP is a term that was coined in an unpublished 1993 white paper, "Providing OLAP to User Analysts: An IT Mandate", by E. F. Codd. By introducing this new term as a play on the then-familiar term on-line transaction processing (OLTP), the paper signaled a shift in the paradigm for business analysis, in parallel with the shift that had already occurred for transaction processing. Instead of reviewing piles of static reports printed on green-bar paper, the OLAP analyst could explore business results interactively, dynamically adjusting the view of the data – asking questions and getting answers almost immediately. This freedom from static answers to fixed questions on a fixed schedule allows business analysts to operate more effectively and to effect improvements in business operations. In the white paper, the authors outlined 12 characteristics of an OLAP system. In a 1995 update to the white paper, six more characteristics were added.

In the 2004, Nigel Pendse, an analyst with Business Intelligence Ltd. who publishes The OLAP Report, provides another valuable point of view. In a Web page entitled "What Is OLAP", Pendse introduces a simpler model, FASMI (*Fast Analysis of Shared Multidimensional Information*), to characterize OLAP systems. Although no single definition is likely to receive universal support, Pendse's characterization is much simpler than the Codd rules. Briefly, the FASMI characteristics are:

Fast. In keeping with the spirit of the "O" in OLAP, such systems need to provide results very quickly – usually in just a few seconds, and seldom in more than 20 or 30 seconds. This level of performance is key in allowing analysts to work effectively without distraction.

Analytic. Considering the "A" in OLAP, such systems generally must provide rich analytic functions appropriate to a given application, with minimal programming.

Shared. An OLAP system is usually a shared resource. This means that there is a requirement for OLAP systems to provide appropriate security and integrity features. Ultimately, this can mean providing different access controls on each cell of a database.

Multidimensional. Multidimensionality is the primary requirement for an OLAP system, which must present the data in a multidimensional framework. The motivation for this requirement will be discussed later on.

Information. OLAP systems must allow the user to easily condense large amount of data into a form that is useful to business manager and decision makers.

1.4 Data Warehousing Architecture

The term *data warehousing* is used to refer to the process used to organize data in a data warehouse and then allow end users to analyze them with business intelligence applications. In practice three types of solutions are adopted, depending on the number of data layers employed.

One-Layer Architecture. This solution has only one layer of data handled by the operational system, and the data warehouse is virtual, i.e. it is defined as a view of operational data, possibly materialized), and it is used by the business intelligence applications (Figure 1.4). This solution does not require a specific system for managing the data warehouse and it is usually used as the first low-cost solution for small organizations, but it does not meet the requirement for separation between operational and analytical applications.

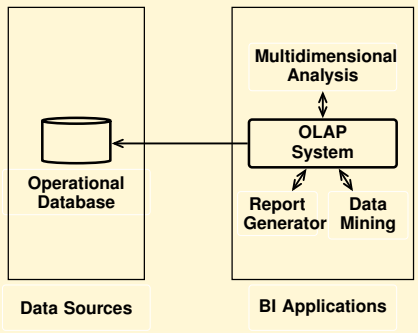


Figure 1.4: One-layer architecture

Two-Layer Architecture. This solution is more general than the previous one, because a data warehouse exists separated from the operational database and managed by a specific system. The data warehouse is loaded with data extracted with *Extract, Transform and Load (ETL)* applications from the operational database, and any other structured data sources, to bring them to a consistent form (Figure 1.5). While the data sources are updated continuously by operational applications, the data warehouse is updated periodically with the ETL applications. This situation typically arises when there are high quality operational databases with schemas sufficiently similar to that of the data warehouse.

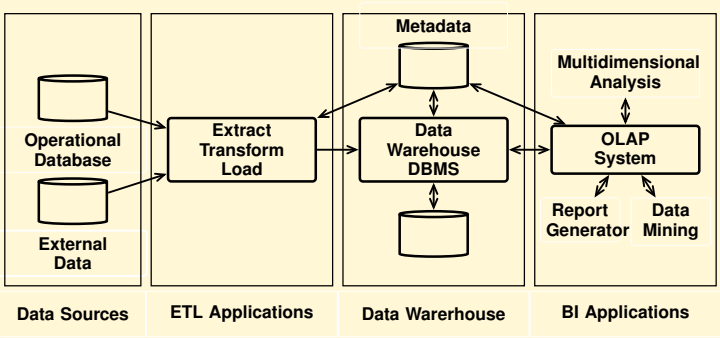


Figure 1.5: Two-layer architecture

This solution separates

- the system for operational database management from the system for data warehouse management and decision support,
- the operational applications from the business intelligence applications, so that business analysis would not interfere with and degrade the performance of operational applications.

Metadata is information about the structure, content and interdependencies of data warehouse components, to support developers, administrators responsible for the data warehouse and the business intelligence applications.

Three-Layer Architecture. This solution is the most general with three data layers: the *data sources*, the *data staging* and the *data warehouse*. The data staging contains data obtained from the integration of different data sources and prepared for loading into the data warehouse (Figure 1.6). The data staging may just be a set of files or, at other extreme, a fully developed relational database. The complexity of data staging layer depends on the quality of the data sources.

This solution separates the process of extraction and integration of data sources from the process of data reorganization and loading into the data warehouse.

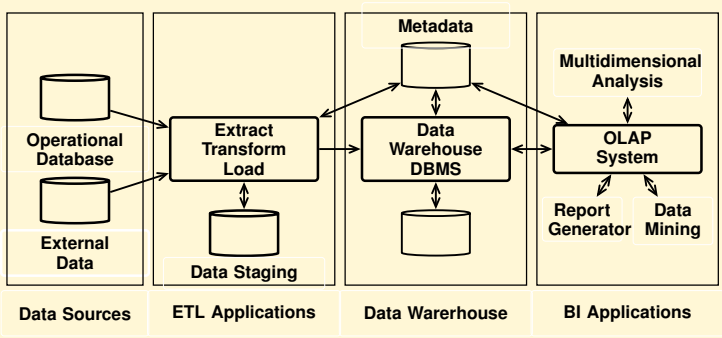


Figure 1.6: Three-layer architecture

1.5 What to Model

According to [Artz, 2005], to support managers in decision-making, data must be organized taking into consideration how they use such data to support their decisions about the performance of key business processes.

Definition 1.4

A **database** is designed to represent some aspects of a reality in terms of the information available about collections of entities with properties and relationship sets between them, while a **data warehouse** is a *specialized database* designed to represent some aspects of *key business processes* in terms of collections of facts about the interesting process measurements, that represent how the processes are being performed, and a set of dimensions, which provide the context of the facts, to be used for analyzing the process performances.

Let us describe more precisely what to model to help managers in analyzing a business process.

Managers are interested in analyzing collections of **facts** *about the performance of a key business process*, *measurable* and *worthy of improvement*.

A fact, in this context, is represented by a set of numerical attributes (hereafter *measure*) by which the process performance is tracked and measured in order to maintain or improve their efficiency. In data warehousing terminology, the interval at which we take measurements is called the *grain*.

Examples of measures for a sale of a product are Quantity, Price, and Revenue. However, without some context, the measures are useless.

> *Managers think in terms of* **business dimensions***, which give facts their context, and are used to analyze them to evaluate their effects.*

For sales data, the dimensions could include Product, Date, and Store. Dimensions contain the *descriptions* of the subject being measured. Examples of questions managers use to ask for decision-making are: "Show me the total sale revenue by product, year, and store", "Show me the current and previous year-to-date sales revenue, and percentage change, by product and by store".

> *Managers analyze measurable business process performance using summary data (called* **metrics***) obtained by grouping facts by different dimensions and combinations of dimensions, and then aggregating measures into useful forms.*

Commons metrics are about economic and financial indicators, but when they are about efficiency and quality of process, are called **Key Performance Indicators, KPI**, because they help understanding how a business process is doing against an objective.

> *Managers are interested in analyzing metrics in various levels of details, by exploiting the fact that some dimensions have a set of associated attributes that can be structured as a* **hierarchy**.

A date dimension, for example, with attributes Day, Month, Quarter and Year, could have a hierarchy Day $<$ Month $<$ Quarter $<$ Year, with the meaning that Year is the highest level of generality within the hierarchy, the second level Quarter tells us that more than one quarter is contained in an year, and so on. The combination of a multidimensional and a hierarchical view allows managers to get a good deal of information from data analysis. For example, managers first see the total sales revenue for the entire year by product, then they move down to quarters to look at the sales by quarter and product.

Example 1.1

Let us consider the sales data stored in the relational table Sales(Product, Store, Date, Quantity), where Quantity is the measure and the other attributes are the dimensions that describe a sale fact. A data analysis usually does a *dimensionality reduction* (*grouping*) to partition a set of rows whose membership is characterized by the fact that all of the rows in a single group agree on the values of the dimensions that are left out. Each group is then aggregated by a function to compute a metric from the measure values. By *aggregation* means to compute a single value from a list of values using an aggregate function such as SUM, COUNT, MIN, MAX, AVG.

Let us look at some examples of an interactive multidimensional data analysis concerning the total quantity of products sold (the metric) to be analyzed by a subset of the dimensions Product, Store, Date. The point is that the user begins with a business question to which wants to answer with the data, gets the results, analyzes the results, uses this new information to formulate another business question, and so on. Later on we will see how to express business questions in SQL to produce the results.

1. The total sales quantity by product, to determine which product is sold best.

Product	Total Sales Qty
P1	27 407
P2	5 179
P3	3 446

2. The total sales quantity by product and by store, to determine where it is best to sell certain products.

Product	Store	Total Sales Qty
P1	S1	13 945
	S2	9 875
	S3	3 587
P2	S1	1 950
	S2	2 500
	S3	729
P3	S1	1 000
	S2	1 200
	S3	1 246

3. A common type of analysis is a generalization of the former: we want to aggregate the measures on some dimensions and also provide the subtotals for each value of all dimensions. This analysis produces a report such as the one in which shows the total sales quantity by product and by store, extended with subtotals for products, for stores, and with the overall total.

Product	Store	Total Sales Qty
P1	S1	13 945
	S2	9 875
	S3	3 587
P1	**Total**	**27 407**
P2	S1	1 950
	S2	2 500
	S3	729
P2	**Total**	**5 179**
P3	S1	1 000
	S2	1 200
	S3	1 246
P3	**Total**	**3 446**
Total		**36 032**

4. Starting with the results of a previous analysis, we can proceed to a more detailed one. For example, after a look at the percentage change of annual quantity sales of products we can also do an analysis by store to understand the decrease in sales of the product ‘P2’.

Product	Total Sales Qty 2009	Total Sales Qty 2010	Change (%)
P1	12 845	14 562	13
P2	2 753	2 426	−12
P3	1 567	1 879	20
Total	**17 165**	**18 867**	**10**

Product	Store	Total Sales Qty 2009	Total Sales Qty 2010	Change (%)
P1	S1	6 445	7 500	16
	S2	4 225	5 650	34
	S3	2 175	1 412	−35
P1	**Total**	**12 845**	**14 562**	**13**
P2	S1	900	1 050	17
	S2	1 200	1 300	8
	S3	653	76	−88
P2	**Total**	**2 753**	**2 426**	**−12**
P3	S1	450	550	22
	S2	580	620	7
	S3	537	709	32
P3	**Total**	**1 567**	**1 879**	**20**
Total		**17 165**	**18 867**	**10**

The reports for decision support are usually represented in a very different form from the ones shown above. They are much more visually pleasing and intuitive, like the dashboard of a vehicle, using graphics and color-coded alarms to highlight trends, exceptions or values lower than predefined ones (Figure 1.7). Microstrategy, a very active company in the *Business Intelligence* arena, has some interesting examples on http://www.microstrategy8.com.

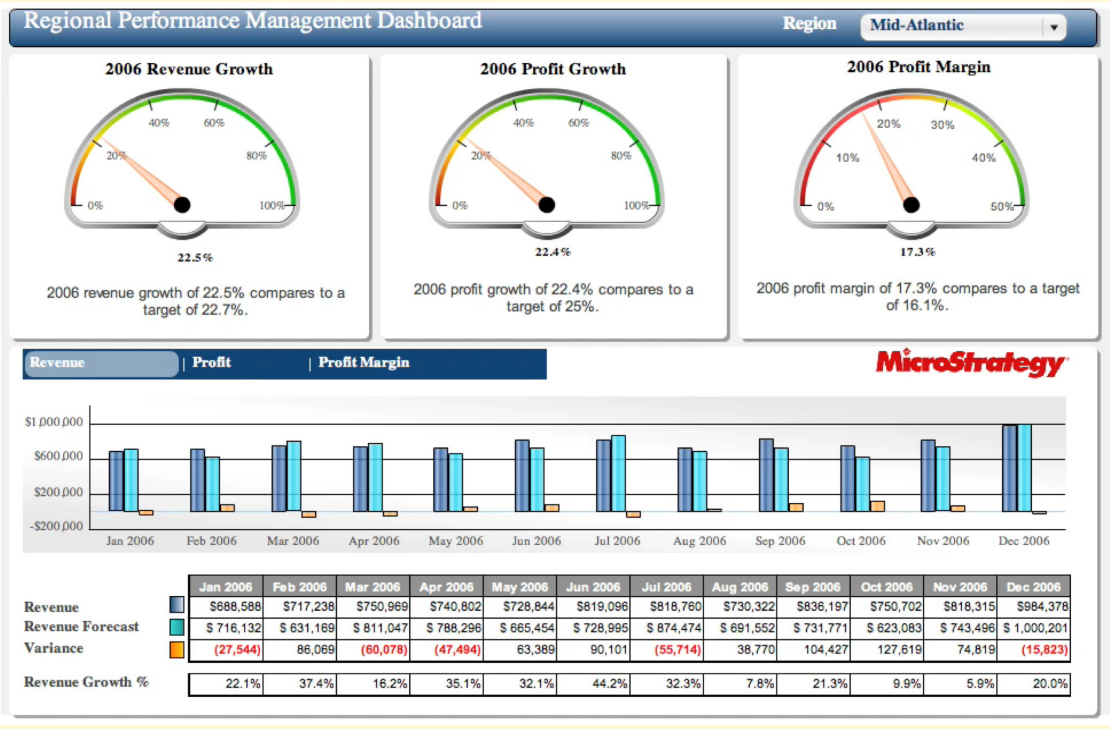


Figure 1.7: Example of Scorecard & Dashboard

1.6 Concluding Remarks

Decision support systems, designed to synthesize, with business intelligence tools, large amounts of data in ways useful to make more rapid and objective decision making, had a growing popularity in recent years for their value strategic and competitive. There has been three very interesting analysis of this trend:

- T. H. Davenport, G. C. Harris, *Competing on Analytics: The New Science of Winning*, Harvard Business School Press, Boston 2007, for the American context.
- Monitoring Business Intelligence, Report 2007-2008, SDA Bocconi, for the Italian context.
- T. Burelli, A. Marzona, M. Pighin, *From intuition to knowledge*, Arachne, Roma, 2007, for the Italian context of small and medium businesses.

It is also interesting to read an article that appeared in print on April 23, 2011 of the *The New York Times* edition with the heading *When There's No Such Thing as Too Much Information*, by Steve Lohr. Here is an excerpt of what he says:

> *Information overload is a headache for individuals and a huge challenge for businesses. Companies are swimming, if not drowning, in wave after wave of data — from increasingly sophisticated computer tracking of shipments, sales, suppliers and customers, as well as e-mail, Web traffic and social-network comments. These Internet-era technologies, by one estimate, are doubling the quantity of business data every 1.2 years.*
>
> *Yet the data explosion is also an enormous opportunity. In a modern economy, information should be the prime asset — the raw material of new products and services, smarter decisions, competitive advantage for companies, and greater growth and productivity.*
>
> *Is there any real evidence of a data payoff across the corporate world? It has taken a while, but new research led by Erik Brynjolfsson, an economist at the Sloan School of Management at the Massachusetts Institute of Technology, suggests that the beginnings are now visible.*
>
> *Mr. Brynjolfsson and his colleagues, Lorin Hitt, a professor at the Wharton School of the University of Pennsylvania, and Heekyung Kim, a graduate student at M.I.T., studied 179 large companies. Those that adopted data-driven decision making achieved productivity that was 5 to 6 percent higher than could be explained by other factors, including how much the companies invested in technology, the researchers said.*
>
> *In the study, based on a survey and follow-up interviews, data-driven decision making was defined not only by collecting data, but also by how it is used — or not — in making crucial decisions, like whether to create a new product or service. The central distinction, according to Mr. Brynjolfsson, is between decisions based mainly on data and analysis and on the traditional management arts of experience and intuition.*

After having presented **what is modeled** in a data warehouse, in the following the attention will be on:

- **How to model**: which data model is used to model a data warehouse.
- **How data warehouses are designed**: which methodology is used for the design of a data warehouse.
- **How data are analyzed**: which operators are available to analyze data.
- **How to implement a data warehouses system**: which relational DBMS technological extensions are needed to support operations on data warehouses.

1.7 Summary

- An *information system* is a system whose purpose is to store, process, and communicate information.
- The focus of an *operational information system* is the *execution* of business processes, the focus of a *decision support information system* is the *evaluation* of the processes, while the focus of a *web-based information system* is the *use of internet web* to allow new routes to market.

- A *data warehouse* is a decision support database with historical, nonvolatile data, to facilitate analysis of the performance of key business processes, worthy of improvement.
- *Data warehouses* and *operational databases* provide different functions and require different kinds of data, therefore they need to be maintained separately.

Chapter 2

DATA WAREHOUSE MODELING

The purpose of a data warehouse is not *just to store data* but rather to *facilitate decision making*. As such, the first step is to model a data warehouse on the basis of the relevant types of business analyses. Data warehouse modeling is a process that produces a well-organized abstract dimensional data model to understand the structure and contents of the data to best support the needs of the business users. In the following sections, three examples of data models are presented that are relevant in dimensional modeling, using the basic concepts of facts, measures, dimensions and hierarchies:

- A **conceptual multidimensional model**, useful to reason about the characteristics of data at a conceptual level, independent of implementation concerns, as it happens with the Entity-Relationship model for databases.
- A **multidimensional relational model**, the traditional logical model to represent data in data warehouse systems.
- A **multidimensional cube model**, useful to show the basic operators for data analysis.

2.1 Conceptual Multidimensional Model

While it is universally recognized that a data warehouse is based on a multidimensional model, there is no agreement on the approach to the conceptual modeling. In what follows, we will present a simplified version of the *Dimensional Fact Model* (DFM), proposed in [Golfarelli et al., 1998], a graphical conceptual model for data warehouses, aimed at

- effectively supporting conceptual design,
- enabling communication between the designer and the final user in order to refine requirements specification,
- supplying a stable platform for logical design, and
- providing an expressive and non-ambiguous design documentation.

The formalism enables the representation of the following basic information.

Facts

The most important abstraction mechanism of the conceptual model is the collections of facts, i.e., the collection of observations of the performance of a business process. Facts are modeled by a rectangle divided in two parts, which contain the facts name and the set of *measures*. A measure is a numerical property of a fact that describes one of its quantitative aspects of interests for analysis.

Sometimes, facts are without measures, and are usually called *factless facts*, but in accordance with our terminology we call them *measureless facts*. This happens when facts represent events that only need to be counted.

Dimensions

Dimensions give facts their context, and are used to analyze them. Dimensions are represented by lines emanating from the rectangle of facts and ending with a circle (Figure 2.1). In general a dimension is described by a set of attributes used to qualify, categorize, or summarize facts in reports. For example,

the dimension Date has the attributes Day, Week, Month, Quarter, and Year, while the dimension Store has the attributes City, State and Country. Dimensional attributes are represented as shown in Figure 2.2a, and the same names should not be used for attributes of different dimensions.

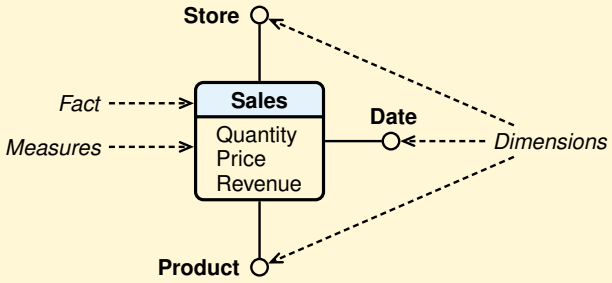
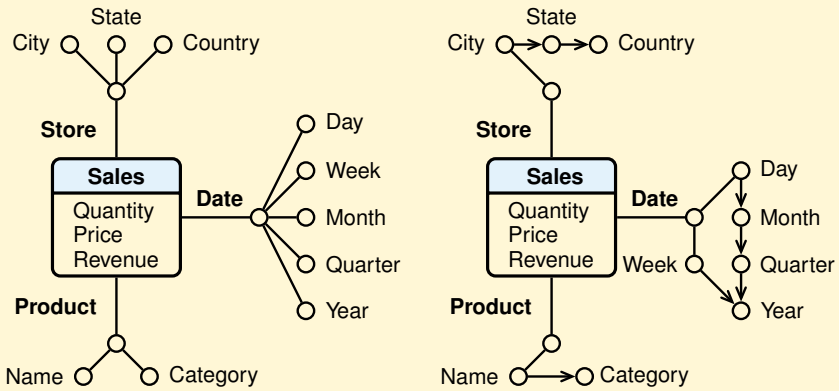


Figure 2.1: A conceptual design without dimensional attributes

Dimensional Hierarchies

In the presence of dimensional attributes, an interesting aspect to model, for the purposes of the data analysis, is a particular hierarchical relationship between their values, i.e., a many-to-one association between pairs of dimensional attributes. For example, the values of Month are in the hierarchy with those of Quarter and Year (Month $\rightarrow$ Quarter $\rightarrow$ Year), in the sense that a year is made up of more quarters, and a quarter is made up of more months, and, viceversa, a month corresponds to a single quarter, and a quarter corresponds to a single year. For this reason it is said that Year is more general than Quarter, and Quarter is more general than Month. In the terminology of the relational data model, each arc of the hierarchy models a *functional dependency* between two attributes.

Dimensional hierarchies are represented as shown in Figure 2.2b, with a directed tree, rooted in a dimension, and leaves representing the most general attributes.



(a) *Without dimensional hierarchies* **(b)** *With dimensional hierarchies*

Figure 2.2: A conceptual design with dimensional attributes

Since a week usually crosses the boundary of two consecutive months, it is usually not treated as a lower abstraction of month. Instead, it is treated as a lower abstraction of year, since a year contains approximately 52 weeks.

The presence of a hierarchy between the dimensional attributes increases the possibilities of data analysis from different perspectives (*Multidimensional Analysis*). For example, once the sales of products have been analyzed by year, we can have a deeper analysis at a different level of detail to analyze product sales by quarter.

The formalism enables the representation of other information. Let us see some examples (Figure 2.3).

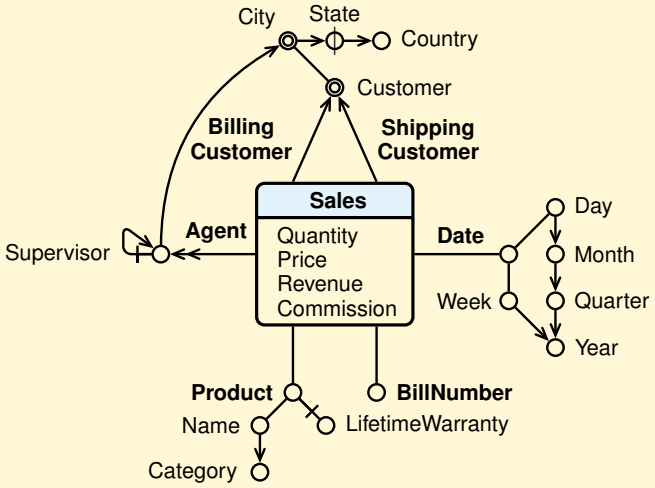


Figure 2.3: A conceptual design with other Dimensional Fact Model features

1. **Descriptive attributes**. Dimensions and dimensional attributes are usually represented with arcs ending with a circle to model that their values may be used in data analysis for selecting or grouping facts data. However there are cases in which dimensions and dimensional attributes are to be considered *descriptive* in the sense that in the data analysis is used only for selecting data, or to show their values in the report result, but not for grouping or aggregating data. A descriptive attribute is represented with an arc without a circle.
2. **Degenerate dimensions**. Dimensions without any attributes are called *degenerate dimensions*. Usually these are transaction-based numbers which describe the fact, but are not measures because it is meaningless to aggregate them. A typical example is a *Bill number* (Figure 2.3).
3. **Optional attributes or dimensions**. When the value of an attribute or a dimension may be undefined, the corresponding arcs are "cut".
4. **Types of hierarchy**. A hierarchy among dimensional attributes can be of the following types:
   - **Balanced**, when the possible levels are a predefined number and the attribute values are always defined. For example, the Date attributes Month, Quarter and Year belong to a balanced hierarchy with three levels.
   - **Ragged**, when the values of one or more attributes may be undefined. A ragged hierarchy is graphically denoted by marking with a dash the attributes whose values may be undefined. For example, a location dimension with attributes Country, State and City, is balanced in the US, but it is ragged for most European countries where State is non used.
   - **Recursive** (*unbalanced*), when the possible levels are a variable number. For example, in the dimension Agent there is the attribute Supervisor representing a recursive hierarchy among agents.

   In the conceptual schema, a ragged hierarchy is represented by cutting the circle of the interested attribute, and a recursive hierarchy is represented with a loop.
5. **Shared hierarchy**. The dimensions can share some hierarchy attributes, such as City and Customer. To avoid ambiguity the circle is doubled and the arcs are oriented. Another typical example is the

date hierarchy: a fact may have more than one Date dimension, with different semantics, and it may be useful to share among them the hierarchy month-quarter-year.

6. **Multivalued dimension or attribute**. A fact may be associated with more than one value of a dimension. For example, the fact sale is associated with several salespeople who have promoted it. In this case, the outgoing arc from the fact ends with a double arrow. Besides dimensions, dimensional attributes also may be multivalued, and represented in the same way.

2.1.1 Considerations on the Conceptual Modeling of a Data Mart

While in a database project, the focus is on collections of entities, their properties, associations and hierarchies between collections, in a data mart project, the focus is on the collection of facts, their measures, their dimensions, attributes and hierarchies [Kimball and Ross, 2002a], [Adamson and Venerable, 1998].[1]

Let us present the key steps in conceptual design of a data mart, assuming that the business process of interest has already been identified together with the key analysis to be performed on the data to get the necessary information to make better business decisions. The example is about the business process of registration of customer orders. The objective is the analysis of the portfolio to adapt marketing strategies, promotion and inventory management.

Step 1: Identify the Granularity of the Fact

When modeling a data mart, the first fundamental decision to be taken is the *meaning of the fact*, because from this choice derive *the measures that characterize the fact and the dimensions for its analysis*. This is a classic problem in the design of data marts: you must carefully choose the right *grain* of the fact, i.e., *the precision with which the measurements are taken*.

In the case of customer orders, it could be said that the interesting thing is the Order, but, thinking about its meaning, we find that there is a problem, because an order is composed of a header and one or more lines, and we should decide if the fact is the header, about all products ordered, or the line for each product ordered.

As a general rule, it is best to choose a fine grain, even if it increases the number of facts to be treated, and so to choose a order line as a fact, because later in the analysis with aggregation functions, you can always go from the measures about the lines to the measures about the orders. If, instead, we focus on orders, there is no way to do the analysis in reverse order to move from measures about the orders to measures of individual lines.

Another consideration to keep in mind for the choice of the granularity of the fact is its nature, which may be of the following types (Figure 2.4).

Feature	Transaction	Periodic	Accumulating
Time period represented	Instant of time	Regular interval	Indeterminate period of time, usually of short duration
Grain	One fact per transaction	One fact per time period	One fact for the entire lifetime of an event
Update	No	No	For each state change
Measures	Related to transaction activities.	Related to periodic activities.	Related to activities which have a definite lifetime lifetime.
Dimension *Date*	Event Date	Date at the end-of-period	Multiple date dimensions to show the achievement of different milestones

Figure 2.4: Comparison of fact types

1. We are grateful to Nicola Ciaramella for his contribution to the preparation of this section.

Definition 2.1

A **Transaction Fact** represents the information on a specific event that occurred at a specific point in time during the execution of a business process.

For example, a fact is a transaction (withdrawal or deposit) on a bank account.

Definition 2.2

A **Periodic Snapshot Fact** represents the information on a series of events that have occurred over a period of time.

For example, a fact is the monthly summary of all transactions on a bank account.

Definition 2.3

An **Accumulating Snapshot Fact** represents the information on the lifetime of an evolving event that has a duration and change over time.

For example, the fact is about a mortgage application which is processed with the following phases: 1) presentation of the documents by the applicant as requested by the bank, 2) the bank's assessment of the documentation made available by the applicant, 3) approval of the practice and the initiation of investigative procedures; 4) mortgage completion. At the end of each phase, the fact about a mortgage application changes with the specification of the relevant information about its state. A solution for this case is presented in the appendix on case studies.

Cardinality of the facts

The grain of the fact determines the size of the set of facts that can be estimated using the estimates of the number of possible values for each dimension.

For example, let us consider the monthly Sales facts of the last five years, with the following total number of dimension values: Date ($12 \times 5 = 60$), Product (5 000), and Store (200).

If all the products are sold by all stores during a given month, there is a fact for each combination of dimension values, and so the number of sales facts is obtained by multiplying the dimension sizes ($60 \times 5000 \times 200 = 60\,000\,000$). Thus, the number of facts is many times larger than the dimension sizes.

In general, the number of combinations that actually appear in the set of facts is much less than this maximum number, because only some products are likely to be sold by each store during a given month. This property is referred to as *facts sparsity*.

The cardinality of the facts depends on both the number of dimensions and the grain of the fact. Suppose that the marketing department requests that *daily* sales must be considered as facts. With the grain of sales changed to daily, the number of fact sales becomes $1\,825\,000\,000$. In this way, a fine granularity could result in a huge cardinality of the facts. Conversely, a too coarse granularity could result in facts that are not detailed enough for users to perform meaningful analysis.

Step 2: Identify the Fact Measures

Once the fact to represent has been chosen, the numerical measurements of interest are defined. A measure describes one of the fact's quantitative aspects of interest for analysis.[2] *Facts may be also without measures, when used only to represent the occurrence of an event*, such as the attendance of a student in a course.

In choosing a measure we need to ask whether it makes sense to aggregate them with the function SUM, for analysis of the type "total value of the measure M, grouping data by dimension D", which

2. The measurements are referred to as *measures* or *facts*, but we prefer the term *measures* because it is more descriptive.

is usually expressed in the abbreviated form "total of M, by D". In general, the aggregations with the function SUM are the most used in the analysis, but do not fall into the trap of believing that everything that can add up is an interesting measure, or that the sum is always meaningful. In general, the following measure types are considered.

Definition 2.4

An **additive measure** (also called a *flow* or *rate* measure) *can be meaningfully aggregated with the function SUM by any dimension.*

An additive measure is the most common type of measures. It refers to a time period and it is evaluated at the end of the period to record the cumulative effect over the period. For example, the number of products sold in a day or the monthly income.

Definition 2.5

A **semi-additive measure** (also called a *stock* or *level* measure) *can be meaningfully aggregated with the function SUM by certain dimensions, but not all.*

A semi-additive measure refers to a particular point in time and it is evaluated to record the state of an event. For example, the monthly account balance or the monthly inventory quantity-on-hand.

Definition 2.6

A measure M is *semi-additive with respect to a dimension* D_1 when it can not be aggregated with the function SUM for groups of data with different values of D_1.

Therefore, it makes sense to perform an analysis of the type "total of M, by D_1" — but not by a different dimension D_2 — or to perform analyses of the type "total of M of data with a certain value of D_1, by D_2". However, M may be aggregated with other functions such as AVG, MIN, MAX, for groups of data with different values of D_1.

For example, the bank measure *Account balance* is *semi-additive* with respect to a dimension Date, but adding the *Account balance* for a particular day by the dimension Customer, or Branch, or Account can provide a meaningful information for the total amount of money the bank is holding at a given point in time.

Example 2.1

Let us consider the monthly *Quantity-on-hand* measure for different products and store at the end of every month. *Quantity-on-hand* is *semi-additive* with respect to both the dimension Month, and the dimension Product. In fact, it is not meaningful to total the *Quantity-on-hand* by Product because we would total values of different months, but it also not meaningful to total the *Quantity-on-hand* by Month because we would total values of different products. It is correct to total the *Quantity-on-hand* by Month and by Product, or to total the *Quantity-on-hand* of the product P1 by Month, or to total the *Quantity-on-hand* of the month M1 by Product.

Inventory			
Product	**Store**	**Month**	**Quantity-on-hand**
P1	D1	M1	300
P1	D2	M1	100
P2	D1	M1	500
...	...	...	...
P1	D1	M12	100
P1	D2	M12	0
P2	D1	M12	900

Definition 2.7

A **non-additive measures** (also called *value-per-unit* measures) *cannot be aggregated with the function* SUM *by any dimension.*

Non-additive measures are usually the result of ratios. The only calculation that can be made for such a measure is counting the number of facts with such measures. Examples of non-additive measures include:

- *Per-unit price* cannot be added by any dimensions, while an extended price, such as *Per-unit price* $\times$ *Quantity purchased*, it is correctly additive by all dimensions.
- *Percentages* and *ratios*. A ratio, such as Gross Margin = Margin / Revenue, is non-additive. Whenever possible, such measures should be replaced with the underlying calculation measures (numerator and denominator) so that the calculation is made in the analysis as a metric. It is also very important to understand that when adding a ratio, it is necessary to take the sum of numerator and denominator separately and these totals should be divided.
- *Measure of intensity* such as the room temperature.
- *Averages* such as average sales price.

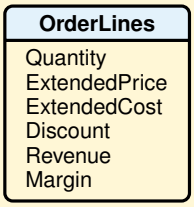
Definition 2.8

A **calculated measure** *is a measure calculated on the basis of other measures.*

It is strongly suggested that standard calculated measures are defined to avoid having users perform these calculations, because often they do not agree on their semantics and may perform wrong calculations. Moreover, having users doing standard calculations runs the risk of making the data warehouse seem unfriendly and complex, and, much worse, if the answers are wrong or inconsistent, the data warehouse will be viewed as wrong.

Example 2.2

Let us consider the following interesting measures for the fact OrderLines



- The Quantity is the total number of products ordered.
- The ExtendedPrice and the ExtendedCost are calculated as follows

$$\text{ExtendedPrice} = \text{Unit Price} \times \text{Quantity}$$
$$\text{ExtendedCost} = \text{Unit Cost} \times \text{Quantity}$$

- The Discount is the value to be subtracted from the extended price.
- The Revenue and the Margin are calculated as follows

$$\text{Revenue} = \text{ExtendedPrice} - \text{Discount}$$
$$\text{Margin} = \text{Revenue} - \text{ExtendedCost}$$

Step 3: Identify the Fact Dimensions

The dimensions are chosen to provide context for facts. Without context, facts are impossible to analyze.

To choose the dimensions, it is useful to consider the classic suggested questions to analyze the facts of everyday life (the **5W-1H rule**: *who, what, when, where, why, how*).

Who is the fact about?

With reference to the orders, they are generated by customers and, for example, it is interesting to analyze the order lines by customers to compute the total revenue. Thus, we select a Customer dimension and then later on we will define its attributes of interest. The question about *who* has another interesting answer: an order involves both the customer and the sales person that promotes the order on behalf of the company, and therefore SalesPerson is another relevant dimension.

The choice of a dimension is not always clear. However, it is useful to ask a question like this to find the right answers in the specific case under consideration.

What is a fact about?

As regards the order lines, a fact is about a product. Therefore, there is a Product dimension, and the choice is justified by the fact that it is meaningful and interesting to analyze order lines by products involved. This dimension is used to analyze the total revenue and cost of order lines by products of a certain category.

We wonder now if there are other interesting answers to the question of *what is an order line about*. No other relevant answers immediately come to mind.

When did a fact take place?

For *when* the answer is that we identify an instant in time or a time period. The two choices are not equivalent. In the case of customer orders, if we consider the order as an instant in time, then we can always perform an analysis by a time period, the converse is not true. For an analyst of business trends the time period is more interesting: to know the hour and minute of an order has its operational importance, but how orders are going is unlikely to be relevant; the preferred analysis will be by day or even better by month.

For our example we decided to choose the Date dimension for order lines, as it usually happens in any multi-dimensional model used in companies.

Where did a fact take place?

The question involves the definition of a Location dimension, another dimension that appears very often in real multidimensional models. For our example, this dimension is not considered because it is assumed that the location information is the customer city.

Similarly we could proceed further by asking questions such as *Why did a fact happen?*, *How did a fact happen?* to discover other dimensions, but the example does not suggest interesting answers.

We have a multidimensional model with the fact OrderLines and the dimensions Customer, SalesPerson, Product, Date, and now it is necessary to establish the attributes of dimensions and any hierarchy among them.

Before proceeding to the definition of the dimension structure it is useful to ask whether it is appropriate to associate to the facts *descriptive attributes* or *degenerate dimensions*. For the fact OrderLines, in addition to the measures discussed above, we consider useful also the degenerate dimension OrderNumber and the descriptive attribute OrderLineNo (Figure 2.5): analysis by OrderNumber is useful for finding the average revenue by order, and (OrderLineNo, OrderNumber) is useful for identifying each line on an order.

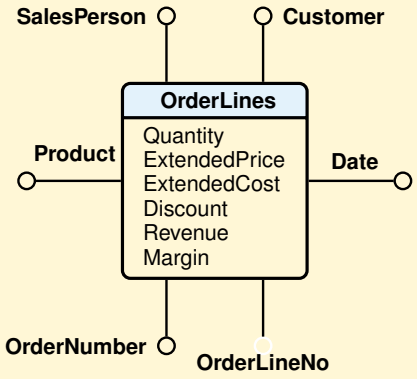


Figure 2.5: The data mart OrderLines conceptual design: the dimensions

Step 4: Identify Dimensional Attributes

Dimensions are the qualifiers that make the measures of the facts meaningful, because they answer the 5W-1H aspects of a question. To perform a more interesting analysis, it is generally necessary to describe each dimension with attributes relevant to the analysis that must be performed, and thus such that for each their value a subset of the facts on which a measures aggregation is somewhat interesting can be identified.

Let us consider the Date dimension. It is easy to imagine that among the requirements there may be both analysis of order lines by the Day attribute to compute the sum of revenues, and analysis by other date attributes, such as Month, Quarter and Year.

Let us consider then the Customer dimension and ask if it makes sense and is relevant to group customers by city of residence. The answer is yes because the information that the customers of a city have issued orders for a total amount higher than those of other cities helps to make a decision of whether to intervene on customers of different cities in a different way with different promotions.

This reasoning draws similar concepts that underlie the *segmentation of customers*. A customer segmentation is useful if

- segments behave differently with respect to their buying behavior;
- segments have a certain homogeneity behavior;
- it is possible to operate on segments with differentiated promotion actions.

Theories of customer segmentation also require other properties of a good segmentation, such that the segments are large enough to warrant different marketing actions, but the three properties listed capture the essence of the idea of segmentation: identifying customer groups that have a common behavior, which is very different from that of other groups, and so different marketing effort must be studied.

This statement is simply the principle underlying the clustering, one of the most important and interesting strategies for data mining. If we apply these principles to the structuring of dimensions, it turns out that the grouping of data may also be done following other criteria. For example, it is usually not necessary that the Date dimension is structured into periods, but if the sales are about products with strong seasonality, then it will be interesting to divide time into seasons defined according to the logic inherent to the phenomenon to be analyzed. Suppose that some products are sold almost exclusively in the pre-Christmas period, in this case the year can be divided into two periods, one from the beginning of December until Christmas, the other covering the rest of the year. Another example is that it may be useful to make a distinction between sales on the weekend and those on other days.

If we think of a dimensional model of the data of an urban public transport company, we discover that we need to move from the day to time periods, such as entry and exit from offices or schools. The definition of time periods is not standard: it is a decision that must be taken according to the logic inherent

to the movements of travelers, but also according to the logic of company operations. For example, 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.), and so on.

If the company cannot change its way of operating at night, then breaking the night time in time periods serves only to satisfy curiosity, but ends up complicating the report without any real added value for decision-making.

In general, the structure of a dimension should therefore reflect two logics:

- *The logic of the event to be analyzed*: the values of dimensional attributes at every level of the hierarchy, are used to group fact data so that groups are internally homogeneous and different between themselves with respect to the values of the measures, to help the analyst to understand what the factors that influence the event are.
- *The logic of company operations*: the values of dimensional attributes at every level of the hierarchy are used to group fact data so that groups are internally homogeneous and different between themselves with respect to their reaction to the actions of the company, to help the decision maker to revise their actions in order to influence the event.

Step 5: Identify the Dimensional Attribute Hierarchies

Dimensional attributes are useful for generating readable reports, but the most interesting attributes for interactive multidimensional analysis are organized into hierarchies to allow groupings of facts data and aggregations of the measures at different levels of generality, as usually required in practice. For example, in the case of the Date dimension, the hierarchy Day → Month → Quarter → Year is relevant. The hierarchies of interest for the other dimensions are shown in Figure 2.6.

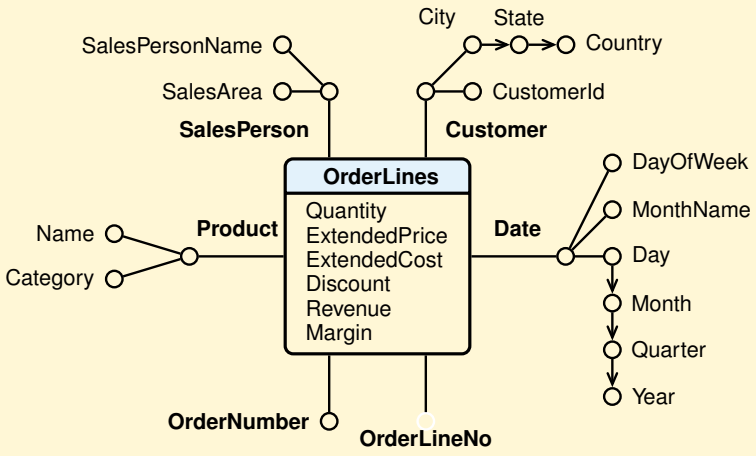


Figure 2.6: The data mart OrderLines conceptual design with dimensional hierarchies

2.2 Multidimensional Relational Model

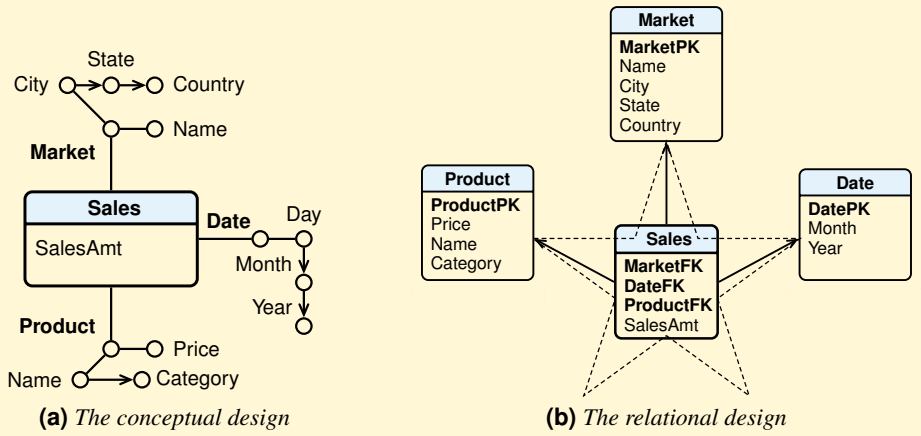
A conceptual multidimensional schema is transformed into a relational logical schema by applying a set of mapping rules, as will be described in the following chapter. The result depend on the complexity of the conceptual schema, and in this section, we show only the basic idea of the structures of specialized schemas usually used, called *star schema*, *snowflake schema* and *constellation schema*.

Definition 2.9

A **star schema** consists of a **fact table**, which contains the data about the facts to be analyzed, and a set of **dimension tables**, one for each dimension. Each of the dimension tables has a single attribute primary key which has a one-to-many relationship with a foreign key in the fact table. Usually a dimensional primary key is a simple integer *surrogate key* that is numbered sequentially from 1 to the number of records in the dimension table. Usually a meaningful integer surrogate key of the form YYYYMMDD is used for a date with the granularity of a day (e.g. 20140926 for September 26, 2014).

The fact table is at the center of the "star", whose tips are the dimension tables (Figure 2.7). A *star schema* is an intentional simplification of the database design that would be achieved by following the standard rules of normalization.

Note that using the surrogate key YYYYMMDD for the dimension Date, the Day attribute is useless and a value of the Month attribute is an integer of the form YYYYMM to represent correctly the dimensional hierarchy Month → Year.



(a) *The conceptual design* **(b)** *The relational design*

Figure 2.7: Example of a star schema

Definition 2.10

A **snowflake schema** is a variant of the star schema, where some dimension tables are *normalized*, thereby further splitting the data into additional tables.

The saving of space is usually negligible in comparison to the typical magnitude of the fact table (Figure 2.8). Furthermore, the snowflake structure can increase the time to execute queries that require hierarchies to be traversed, since more joins will be performed to execute them. Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.

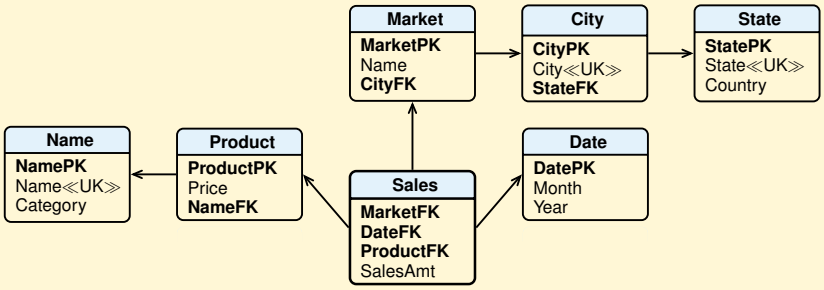


Figure 2.8: Example of a snowflake schema

Definition 2.11

A **constellation schema** has multiple fact tables that share dimension tables.

The example given in Figure 2.9 has two fact tables Sales and Returns sharing the Date and Product dimensions.

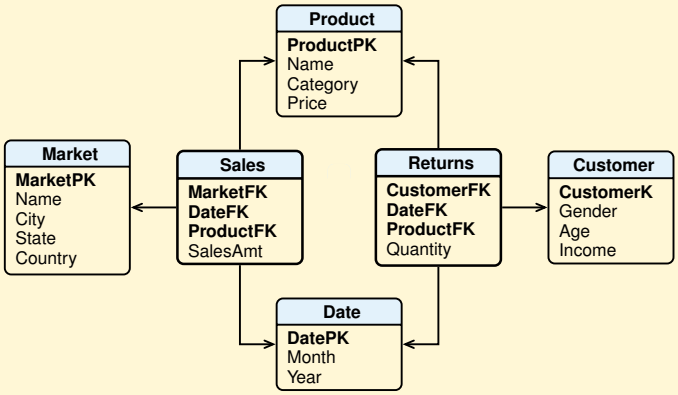


Figure 2.9: Example of a constellation schema

The main relational DBMS vendors provide OLAP servers that map operations on multidimensional data to standard relational operations on specialized relational DBMS to store and manage data warehouses. Such servers are referred to as **ROLAP** (*Relational OLAP*).

2.3 Multidimensional Cube Model

Definition 2.12

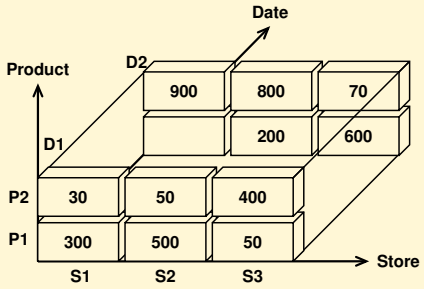
A multidimensional **cube model** (**data cube**) represents facts with n dimensions by points in an n-dimensional space. A point (a fact) is identified by the values of dimensions and has an associated set of measures.

Such a multidimensional view is an intuitive way to think about OLAP queries and their results. For the sake of simplicity, we will consider a cube with at most three dimensions and one measure.

Example 2.3
Let us consider the analysis of the daily sales of different products in different stores over different days. Let us assume that data are stored into a fact table such as that shown in the figure (a). Store identifies a store, Product identifies a product, Date identifies a day, and Qty identifies the quantity sold of that product at that store in that time period.

Store	Product	Date	Qty
S1	P1	D1	300
S2	P1	D1	500
S3	P1	D1	50
S1	P2	D1	30
S2	P2	D1	50
S3	P2	D1	400
S2	P1	D2	200
S3	P1	D2	600
S1	P2	D2	900
S2	P2	D2	800
S3	P2	D2	70

(a) *Fact Table*



(b) *Data Cube*

We can view this sales data as *3-dimensional*, because the value of the *measure* Qty is a function of the Store, Product, and Date attributes, which form the so-called *dimensions*. Consequently, we can also think of the data in a fact table as being arranged in a 3-dimensional cube shown in the figure (b). For example, the cell ('S1', 'P1', 'D1') contains the sales for the product P1 on date D1 by the store S1.

The 3-dimensional cube is a generalization of a 2-dimensional cross-tabulation commonly used to give a basic picture of how two attributes inter-relate because it helps to search for patterns of interaction.

	Store		
Product	**S1**	**S2**	**S3**
P1	300	500	50
P2	30	50	400

When dimensions have attributes and hierarchies, the multi-dimensional cube is more complex. We will assume that additional information about the dimensions are stored in tables, which describe dimensions' attributes.

Some vendors provide OLAP servers that implement the fact table as a data cube using a specialized data structure. Such implementations are referred to as **MOLAP** (*Multidimensional OLAP*).

2.3.1 OLAP Operations in the Multidimensional Data Model

Let us show some typical OLAP operations for multidimensional data. Each of the operations described below is illustrated in Figure 2.10.

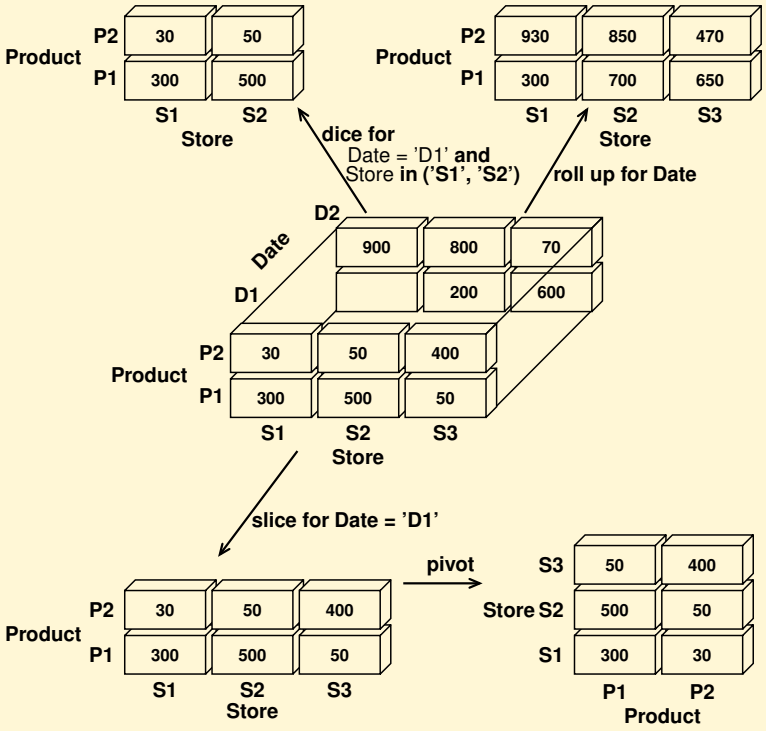


Figure 2.10: Examples of typical OLAP operations on multidimensional data

Slice and *dice*

The operators *slice* and *dice* generate sub-cubes by selections, but they do not change the measures values, that is they do not make summarizations:

- The *slice* operator selects a cross section that cuts across a cube with a selection on one dimension (Figure 2.10).
- The *dice* operator selects a sub-cube with a selection on two or more dimensions (Figure 2.10).

Roll-up and *Drill-down*

The *roll-up* operator, also called *drill-up*, performs summarizations at different levels of details either by dimension reduction or by climbing up dimension hierarchy.
Figure 2.10 shows the result of a roll-up operation by removing the date dimension, summarizing the the quantity sold by product and by store.

The *drill-down* operator is the reverse of roll-up. It produces more detailed data from less detailed data. Drill-down can be used by either stepping down a hierarchy for a dimension or introducing additional dimensions.

Pivot

The *pivot* operator (also called *rotate*) performs a rotation of the data axes to provide an alternative presentation of data (Figure 2.10).

2.3.2 The Extended Cube

Let us assume that each dimension is extended with an additional value "∗". This value has the intuitive meaning "all", and it represents summarization along the dimension in which it appears. A cube can be extended with new "borders" made of cells containing the value of aggregate functions (we consider here only the SUM) as shown in Figure 2.11.

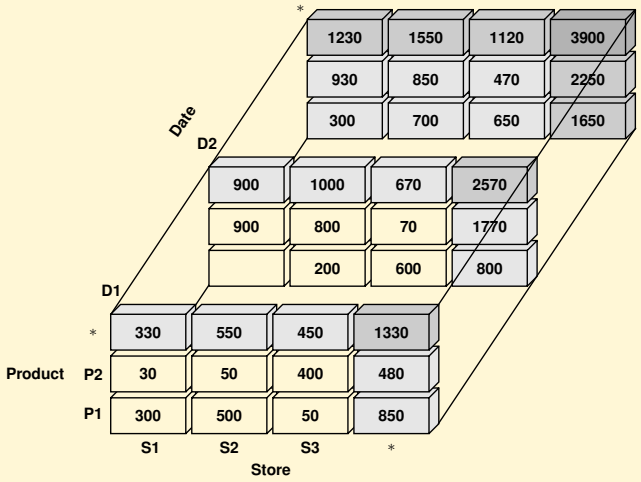


Figure 2.11: Three-dimensional cube extended with cuboids

For example, using the notation Sales(Store, Product, Date, Qty) for a cube with dimensions Store, Product, Date and a measure Qty, we can denote subcubes as follows:

- ('S1', 'P1', 'D1') is the cell that contains 300, the sales for the product P1 on date D1 by the store S1;
- ('S1', ∗, 'D1') is the cell that contains 330, the sum of sales for all products on date D1 by the store S1;
- ('S1', ∗, ∗) is the cell that contains 1 230, the sum of sales for all products over all time by the store S1;
- When a dimension is used as a coordinate instead of one of its values, the notation denotes a so called *cuboid*. For example (Store, Product, ∗) is the cuboid "roll up for Date" in Figure 2.10, with two dimensions with the cells that contain the sum of sales over all time by the dimensions Store and Product (in SQL terms, the sales data are grouped by Store and Product, and the aggregate function SUM(Qty) is computed).

In Figure 2.11, the border with the lightest shading represents aggregates in one dimension, darker shading for aggregates over two dimensions, and the darkest cuboid in the corner for summarization over all three dimensions. In general the border represent only a small addition to the volume of the data cube (the white cuboid).

The 3-dimensional extended cube is a generalization of a 2-dimensional extended cross-tabulation (Table 2.1).

Table 2.1: Sales extended cross-tabulation

	Store			
Product	**S1**	**S2**	**S3**	**Total**
P1	300	500	50	**850**
P2	30	50	400	**480**
Total	**330**	**550**	**450**	**1330**

To speed up data analysis, commercial data cube systems precompute all or some of the cuboids and store them as *materialized views* of the data cube. The problem of selecting the cuboids to precompute will be studied in a later chapter.

The total number of cuboids for a data cube with three dimensions is $2^3 = 8$. The possible cuboids can also be denoted without using the "$*$" as follows: (Store, Product, Date), (Store, Product), (Store, Date), (Product, Date), (Product), (Date), (Store), ( ). (Store, Product, Date) denotes the data cube, while ( ) denotes the total sum of all sales.

These cuboids can be represented as a lattice, also called the *data warehouse lattice*, as shown in Figure 2.12. We say that the cuboid C_1 is below the cuboid C_2, written $C_1 \preceq C_2$, if and only if C_1 can be computed from C_2. The cuboids are named using the abbreviations P for Product, S for Store, D for Date.

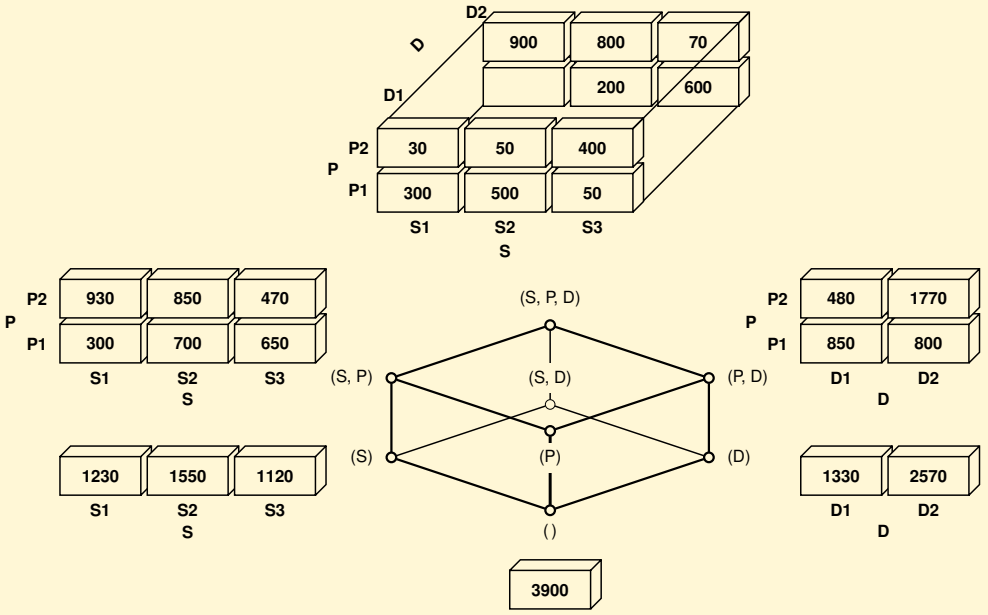


Figure 2.12: Lattice of cuboids

In general the computation of the cuboid C_1 from C_2 depends on the aggregate function used, which can be of one of the following types.

Definition 2.13 Distributive Aggregate Functions

An aggregate function f on a multiset of values V is *distributive* if there is a *local* aggregate function f_l and a *global* aggregate function f_g, such that for any k-partition $\{V_1, \ldots, V_k\}$ of V we have

$$f(V) = f_g(\{f_l(V_1), \ldots, f_l(V_k)\})$$

For example, the SQL functions SUM, MIN, MAX and COUNT are distributive aggregate functions:

- $\mathsf{SUM}(V) = \mathsf{SUM}(\{\mathsf{SUM}(V_1), \ldots, \mathsf{SUM}(V_k)\})$
- $\mathsf{MIN}(V) = \mathsf{MIN}(\{\mathsf{MIN}(V_1), \ldots, \mathsf{MIN}(V_k)\})$
- $\mathsf{MAX}(V) = \mathsf{MAX}(\{\mathsf{MAX}(V_1), \ldots, \mathsf{MAX}(V_k)\})$
- $\mathsf{COUNT}(V) = \mathsf{SUM}(\{\mathsf{COUNT}(V_1), \ldots, \mathsf{COUNT}(V_k)\})$

Definition 2.14 Algebraic Aggregate Functions
An aggregate function is *algebraic* if it can be computed from a finite algebraic expression defined over distributive functions.

For example, the functions average (AVG), variance (VAR), and standard deviation (STDEV) are algebraic aggregate functions, which can be computed on V using the following distributive aggregate functions on the multiset of a k-partition $\{V_1, \ldots, V_k\}$

- $n_i = \mathsf{COUNT}(V_i)$
- $s_i = \mathsf{SUM}(V_i)$
- $s_i^2 = \mathsf{SUM}(V_i^2)$, where V_i^2 is the set of the squares of the various elements of V_i.

Let $n = \mathsf{COUNT}(V) = \mathsf{SUM}(\{n_1, \ldots, n_k\})$.
The functions $\mathsf{AVG}(V)$, $\mathsf{VAR}(V)$ and $\mathsf{STDEV}(V)$ are computed as follows:

- $\mathsf{AVG}(V) = \mathsf{SUM}(\{s_1, \ldots, s_k\})/n$
- $\mathsf{VAR}(V) = \dfrac{\mathsf{SUM}(\{s_1^2, \ldots, s_k^2\}) - (\mathsf{SUM}(\{s_1, \ldots, s_k\}))^2/n}{n-1}$
- $\mathsf{STDEV}(V) = \sqrt{\mathsf{VAR}(V)}$

Definition 2.15 Holistic Aggregate Functions
An aggregate function is *holistic* if it can not be computed from other aggregate functions.

For example MEDIAN, MODE, RANK.

2.4 Summary

- A data warehouse conceptual model is the best support for discussing, verifying, and refining user specifications since it achieves the optimal trade-off between expressivity and clarity.
- A multidimensional relational model is used to implement data warehouses. This model can adopt a *star schema*, *snowflake schema* or a *constellation schema*. The core of a multidimensional model is a *fact table* and a set of *dimensional tables*.
- The core of a *multidimensional model* is the *data cube*. An extended cube consists of a lattice of cuboids, each corresponding to a different degree of summarization of data. *Full* or *partial materialization* refers to the pre-computation of all or some of the cuboids in the lattice. Commercial systems use different strategies both about which cuboids to materialize, and how to store them.

Chapter 3

DATA WAREHOUSE DESIGN

The purpose of a data warehouse (DW) is not just to store data but rather to facilitate decision making. Therefore, a data warehouse must be designed taking into account the different types of analyses that are needed by the business users to make better decisions about key business processes worth of improvements. Since a data warehouse design process is complex, a methodology organized in phases is presented, like the one that is used to design operational databases, to highlight the importance of conceptual design and shows how to transform the conceptual design into the logical one using the relational model.

3.1 Introduction

A data warehouse must be designed to provide the information needed to solve a business problem. If the problem is solved there should be some economic gain in order to allow a cost benefit analysis for the data warehousing project.

As happens for databases, it has become fairly standard to divide the DW design process into the following four phases:

1. **Requirements Analysis**. The goal is to produce a description of the business processes, the typical information analysis activities with which users are involved, and the measures and dimensions of interest. Typically, requirements at this stage are documented rather informally.
2. **Conceptual Design**. The goal is to produce a formal description of the data to be analyzed in high-level-term using a conceptual data model. We will use the *Dimensional Fact Model, DFM*, to describe facts, dimensions, dimensional attributes and attribute hierarchies.
3. **Logical Design**. The goal is to transform the conceptual design into the logical structures used for storing the DW in a relational DBMS.
4. **Physical Design**. The goal is to define the data structures needed for storing the database tables created by the logical design. The main issues are what indexes and materialized views to define to optimize the overall performance of the system.

Once the DW has been implemented, data must be extracted from the operational and external systems, transformed into a usable format for the DW, and finally loaded into the DW in order to be usable for query processing and analysis. These *Extract, Load, Transform* (*ETL*) processes have historically been batch-oriented.

In general the data to load in the DW are processed with two important and complex kinds of operations:

1. *Transform*. When the data come from different sources, their *formats* are revised to align them by eliminating syntactic and semantic differences.

   (a) *Syntactic transformation*. The same data can have both attributes with different names, and the names are not those to be used in the DW, and different types. For example, a code is defined in some cases of a type string, and in others a type integer; a gender is defined as (M, F), (m, f), (0, 1) or (male, female); a value is defined with different units of measurement, and so on.

(b) *Semantic transformation*. The data in source databases may have been used with a different meaning. For example, sales can be daily or weekly.

2. *Cleaning*. The data are analyzed in order to eliminate errors of representation or to complete missing information. For example, in the case of addresses the zip code can be wrong or the name of the town can be written in different ways (Busto Arsizio also written as BustoArsizio or BArsizio)

The information generated during the design and implementation of a DW is organized and stored as metadata using appropriate specialized tools or taking advantage of the capabilities of DW systems that provide, as any DBMS, a *catalog* that contains information about the logical and physical organization of the data managed. In the case of DW metadata are about other aspects of the data and, in short, can be classified into the following main categories:

– *Business metadata*. Concern the meaning of the terms used to define the logical structure of data in corporate terminology. This type of metadata is usually used by users to understand the nature of the data available.
– *Structural metadata*. Concern the logical structure of facts and dimensions, types of attribute values, hierarchies, dimensions and meaningful aggregations. This type of metadata is usually used by users to understand what types of analysis can be performed on the data.
– *Technical metadata*. Are concerned with the physical data property, such as storage structures, data sources, date of loading, transformations applied etc. This type of metadata is usually used by technicians for the maintenance and development of a DW.
– *Operational metadata*. Concerns the types of predefined analysis reports and what parameters should be used.
– *Design metadata*. Concern the results of the DW design phases.

Loading metadata is only partially automated and has a cost: the time that the personnel involved in the design and implementation of the DW must dedicate to the problem. Finally, note that metadata is useful not only to gather information on the data, but also to be exchanged between different OLAP tools. For this reason, proposals have been made to define standards like *Common Warehouse Metamodel* (*CWM*).

In the following, the focus will be on how to proceed in the conceptual and logical design phases of a DW.

3.2 Data Warehouse Design Approaches

According to [Artz, 2005] and [Ballard et al., 2006], the approaches to DW design can be of the following types (Figure 3.1):

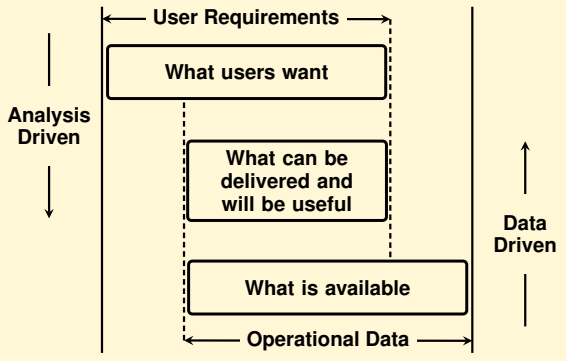


Figure 3.1: Data warehouse design approaches

1. *Data-driven.* This approach was originally proposed by Inmon, one of the first authors on the subject of data warehousing, which he describes as *top-down*, and whose supporters are referred to as "Inmonites". The goal is to design first an enterprise DW based on the data available in the operational information system, and then the data marts are created from the DW. This is done by analyzing a conceptual model of data, if one is available, or the actual logical record layouts and selecting data elements deemed to be of interest. This approach is the only possible when the demand for information from a DW does not exist until the DW is actually available. An initial DW design on the basis of the data available can help both users to discover new ways in which to use the available data, and the designer to identify areas on which to focus data warehouse development efforts. The disadvantage of this approach is that without user involvement there is the risk of producing a non interesting result.
2. *Analysis-driven.* This approach was originally proposed by Kimball, a well-known author on data warehousing, which he describes as *bottom-up*, and whose supporters are referred to as "Kimballites". The goal is to design first the data marts based on the data analysis that the users want to perform, and then the data marts are integrated to build the DW. 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 data-driven approach. Therefore, it generally produces useful data marts in a shorter time span. The disadvantage of this approach is the risk that some of the data that the analysis needs are not available. Moreover, if a user is too tightly focused, it is possible to miss useful data that is available in the operational information system.

Both the approaches can be useful in certain cases. In the following we will use a combination of the two using the following design phases for each data mart of interest:

1. *Requirements analysis*
2. *Initial analysis-driven data mart conceptual design*
3. *Candidate data-driven data mart conceptual design*
4. *Final data mart conceptual design*
5. *Data mart and DW logical design*

3.2.1 Requirements Analysis

The requirements analysis phase is divided into two main sub-phases, characterized by the different language used for the preparation of the documents they produce. The first sub-phase, *Requirements gathering*, produces a natural language specification of requirements. The second sub-phase, *Requirements specification*, produces a description of the requirements for data analysis outlining the salient features to be modeled then with the conceptual design.

1. *Requirements gathering*
   (a) Analysis of the problem domain for which the modeling will be done.
   (b) Analysis of the business processes to select, with end-user interviews, those more interesting to consider for designing the data warehouse.
   (c) Business questions that end-users issue and try to answer in the course of their information analysis activities.
   Examples of frequently encountered categories of business questions are:
   – *Existence checking analysis*, such as "A given product has been sold to a particular customer."
   – *Item comparison analysis*, such as "Compare the value of purchases of two customers over the last six months," or "Compare the number of items sold for a given product category, by store, and by week."
   – *Trend analysis*, such as "The growth in item sales for a given set of products, over the last 12 months."
   – *Queries to analyze ratios, rankings, and clusters*, such as "Rank our best customers in terms of dollar sales over the last year."

– *Statistical analysis*, such as "The average item sales by product category, by sales region."

Note that the business questions must usually be "interpreted" in order to express them in a form more useful for designing the data warehouse. For example, a business question a manager of a company wish to ask of their data might be: "Why are our sales not meeting our targets". The business question might be interpreted in a more useful form as "For the current year, the cumulative quantity sold and targets, by product". That is, an interpreted business question should be expressed with the types of reports to be produced, or phrases that reveal the following information:

– The constraints on data to analyze.
– The requested dimensional attributes and the metrics (*aggregation operation*) to compute.
– The coordinates (dimensions) against which the fact must be analyzed.
– The result sorting criteria and if metrics' partial value are needed.

When business questions are expressed by means of phrases, the use of the following form is suggested: "For a data *subset* to use, the *metrics* to compute, *by* dimension, ..., *by* dimension. How the result should be presented". For example, "For the year 2010 in Italy, the total of sale revenue, by region, by month, by customer name. The result must be sorted by region, month, and customer name, and must include partial totals for all regions."

It is important also to check that the information analysis requirements need data that are currently available or can be obtained as external data that exist outside the enterprise. If there are multiple data sources, the analysis is complicated by the need to reconcile the likely differences in representation of information. In the following we will not consider this aspect.

2. *Requirements Specification*. The business questions are specified using a set of worksheets with the following structure:

Business Process Requirements

				Process
N	**Business questions**	**Dimensions**	**Measures**	**Metrics**

Each business question is analyzed to identify the fact measures, the preliminary dimensions, and the metrics to be computed.

Fact Description

		Fact
Description	**Preliminary dimensions**	**Preliminary measures**

The meaning of the fact is described, together with its grain and type (transaction, periodic, or accumulating), and the preliminary measures and dimensions.

Dimensions

		Dimensions
Name	**Description**	**Granularity**

The meaning of each dimension is described, together with its name, a description, and the grain.

Dimensional Attributes

Dimension	
Attribute	**Description**

Of all dimensions the attributes and their description are listed. Dimensional attributes must be chosen carefully to express the analysis in a natural way and display results in a comprehensible way. If there are attributes with values that are codes, providing the description of the code is also suggested. It is also advisable not to use the same names for attributes of different dimensions that have different meaning.

Dimensional Hierarchies

Dimensional Hierarchies		
Dimension	**Hierarchy decription**	**Hierarchy Type**

It describes the dimensional hierarchies for each dimension, and their type (balanced, ragged, recursive).

Dimensional Attributes Changes

Changing Dimensions		
Name	**Changing Attribute**	**Treatment of changes**

It is important to understand how the business wants to deal with the dimension attributes that can change over time. Consequently, for each of them, besides the name, the type of strategy is specified to deal with them in the logical design phase and data loading.

For example, suppose that the dimension Customer of facts Order contains the attribute Residence, with a value that can change over time, and that there are several sales involving a customer from Lucca, which to a certain date changes residence to Pisa. How can we carry out sales analysis to account for this change? What should the result be of analysis such as " How many sales are made in Pisa?".

The strategy to be specified depends on the objectives of the analysis, and for this reason it should be documented in the requirements specification. We consider four options, of which the first three are considered for *slowly changing dimensions*:

Type 1 (overwriting the history) If a dimensional attribute changes its value, only the latest value is required to be held in the data mart. This means that there is no need to preserve the previous value.
For example, if a customer changes address, the new one replaces the present value of the dimension Customer. It is the easiest solution, but the history of customer addresses is lost. For example, if a customer at a certain date changes their address from Pisa to Lucca, all sales concerning him before and after this date are considered made in Lucca, and this changes the outcome of analyses such as "How many sales are made in Pisa?"

Type 2 (preserving the history) If a dimensional attribute changes its value, both the old and the new value are required to be held in the data mart, but the structure of the dimension must not be changed. It is the solution commonly used.

Type 3 (preserving one or more versions of history) If a dimensional attribute changes its value, the structure of the dimension is extended with additional attributes to keep the tracking history with both old and new values. Moreover, we also add another attribute EffectiveDate for the date of the change. This solution is rather quirky and it is rarely used. We will not consider it further.

Type 4 (fast changing) The dimensional attributes change frequently, and must not be treated with one of the previous solutions.

Measures

Fact measures

Measure	Description	Aggregability	Calculated

It describes each measure of the fact identified from the requirements, how it is calculated from other measures, and the aggregate functions that are applicable to the measure when the data are grouped by dimensions.

Descriptive attributes of the facts

Descriptive attributes

Attribute	Description

It contains each descriptive attribute of the facts, with a description of what they represent.

Summary of Dimensions and Measures

Facts Dimensions

Dimension	Fact_1	...	Fact_n

Facts Measures

Measure	Fact_1	...	Fact_n

If the requirements concern different facts, the worksheets specify in which facts the selected dimensions and measures are used. The worksheet about dimensions, called the *data warehouse bus architecture*, is useful to identify which dimensions are used by multiple data marts, and therefore they must have a unique interpretation and representation (*conformed dimensions*) to be then shared in the DW. If the dimensions must be kept different, they must be renamed.

3.2.2 Initial Analysis-Driven Data Mart Conceptual Design

An *initial* data mart conceptual design is defined from the analysis that the users perform (*the design from what the users want*), without any claim to completeness but useful as a formal description of requirements. In the conceptual design the dimensional hierarchies are modelled together with their type (balanced, incomplete, recursive), degenerate dimensions and descriptive attributes of the facts.

3.2.3 Candidate Data-Driven Data Mart Conceptual Design

A method is described for developing a *candidate* data mart conceptual design from the operational database relational schema (*the design from what is available*). This approach to data mart design ensures that its schema reflects the underlying structure of the data available. The following steps are based on the proposal presented in [Moody and Kortink, 2000]:

1. **Operational data analysis**. In this step, the relational database schema is analyzed to perform two actions:

   (a) Standardize terminology and units of numerical quantities that have an identical time reference.
   (b) Delete tables, and attributes, considered not relevant to the analysis of the data. For example, information such as the tax code and telephone number are usually not relevant for the analysis of the data.

2. **Tables classification**. In this step, the relational database tables are classified in three categories to identify the possible facts, measures, dimensions and hierarchies between dimensional attributes.

   (a) **Transaction Entities**. These are tables with records that represent events of interest for the business process to be analyzed (orders, insurance claims, salary payments, sales, hotel booking, etc.). Transaction entities have two fundamental characteristics:
      – Describe events that occur at a point in time.
      – Contains measurements or quantities that may be summarized (e.g. monetary value, quantity, weight, volume).

      It is very important to correctly identify the pertinent transaction entities because they are the natural candidates to be considered later for the definition of facts that decision makers want to understand and analyze. However, it must be kept in mind that not all transaction entities will be of interest for decision support, so they must be analyzed carefully with users to identify which of them are important.
   (b) **Component Entities**. These are tables directly related to a transaction entity via a *one-to-many relationship* (Figure 3.2). These entities define the details or components for each transaction event, and so are useful to answer the *who*, *what*, *when*, *where*, *why* and *how* of a business event. Component entities are the basis for defining dimensions in the data mart conceptual design.

      An important component entity of any transaction entity should be the one that represents the time: the historical analysis, in fact, play a key role in all the DW, but usually in the operational database time is not represented with a table but with an attribute of type *Date* and this must be taken into account when defining the data mart design.

      Note that the definition of component entity does not allow us to isolate multivalued dimensions, resulting instead from tables directly related to a transaction entity via a *many-to-one relationship*. In general, this type of table should be considered in the choice of a component entity, as will be shown in a following example.
   (c) **Classification Entities**. These are tables related to a component entity by a chain of *one-to-many relationships* (Figure 3.2). These entities usually represent hierarchies embedded in the data schema. Their interesting attributes are collapsed into component entities to define then in the data mart design the dimensional attributes and hierarchies.

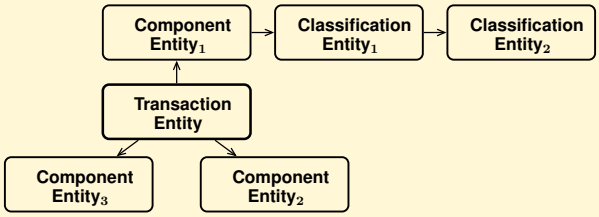


Figure 3.2: Tables classification

In some cases, entities may fit into multiple categories. We therefore define a precedence hierarchy for resolving such ambiguities:

(a) Transaction entity (highest precedence).
(b) Classification entity.
(c) Component entity (lowest precedence).

For example, if an entity can be classified as either a classification entity or a component entity, it should be classified as a classification entity. In practice, some entities will not fit into any of these categories. Such entities do not fit the hierarchical structure of a dimensional model, and cannot be included in the conceptual design.

Example 3.1
Figure 3.3 shows an operational database schema for an orders sales application, assuming that a row of an order may have dealt with more than one shipment.

- Transaction entity: it is interesting to collapse Order into OrderLine, with the OrderLine granularity to define the data mart facts. Other possibles transaction entities might be Invoice, Product or Shipment, but they are not considered of interest for decision support.
- Component entity: Customer, Invoice and Product.
- Classification entity: ProductCategory is collapsed into Product.

The table Shipment does not satisfy the condition to be considered as a component entity of OrderLine because of the *many-to-one relationship*, but it might be considered to define a multivalued dimension.

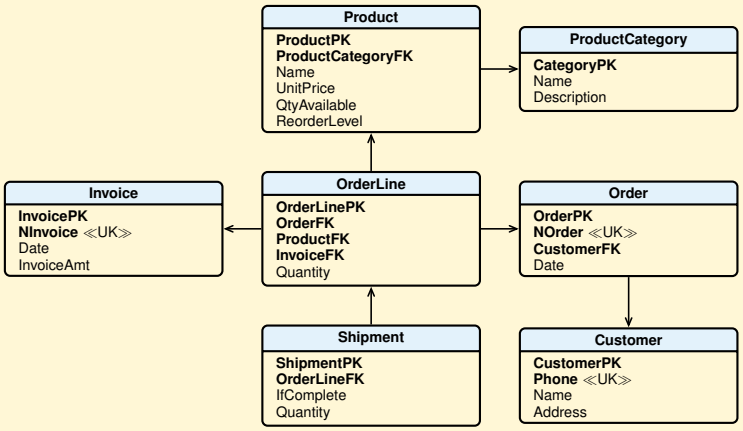


Figure 3.3: A database for order management

3. ***Candidate* data mart conceptual design.**
   The *Candidate* data mart conceptual designs are defined in the following way:
   - For each transaction entity, a data mart fact is defined.
   - A dimension is formed for each component entity, by collapsing hierarchically related classification entities into it. The dimensional attributes are analyzed to decide possible hierarchies. For example, an attribute Date of a transaction entity is usually substituted with attributes Day, Month, Year, and a hierarchy is defined among them; an attribute Address may be replaced by City, and Region, and a hierarchy is defined among them.
   - Where hierarchical relationships exist between transaction entities, the child entity inherits all dimensions (and key attributes) from the parent entity. This provides the ability to "drill down" between transaction levels.

3.2.4 Final Data Mart Conceptual Design

From a comparison of the initial and candidate conceptual designs the *final* data marts are defined (*the design of what can be delivered and will be useful*), which in general will be an extension of the common parts.

3.2.5 Data Mart and Data Warehouse Logical Design

Assuming that the multi-dimensional model is implemented with a ROLAP system, firstly each final conceptual data mart design is translated in a relational schema, deciding whether to make a star or snowflake schema, and then integrating the various data marts schemas in a single DW schema, considering the following possibilities:

- Standardize and share the fact tables with the same dimensions.
- Standardize and share common dimension tables.

In the definition of the relational tables of the data mart logical schema, the following problems will be considered, among others that may arise [Kimball and Ross, 2002b].

Primary keys of dimension tables

The primary key of each dimension table should be an attribute with numerical values automatically generated (*artificial* or *surrogate* key) in addition to any primary key used in the original data, if it is considered relevant to also keep this information in order to determine from which original data it comes from, but which is not necessarily a key for the dimension table.

For the Date dimension table with the granularity of the day, usually present in every data mart, it is useful not to use a surrogate key for the primary key, but an integer representing a day in the form YYYYMMDD. With a similar format it is useful to represent attribute values in the dimensional hierarchy Month $\rightarrow$ Quarter. Usually there are also other attributes useful to show in reports, such as DayName, MonthName, Week Number, etc.

Foreign keys in the fact table

When modeling the facts, foreign keys for dimension tables have the values of surrogate primary keys, and it is useful to assume that foreign keys are always defined, or that their values are not equal to Null. To deal with cases in which for a fact record the dimension value may be unknown, a common solution is to add into the dimension table a special record with an attribute, different from the primary key, with a default value such as "Not Found", and then the foreign key of fact record without the dimension data points to the row "Not Found".

As in the case of dimension tables, the fact table too may have descriptive attributes, such as the primary key used in the operational database to know the source of the fact.

Changing dimensions

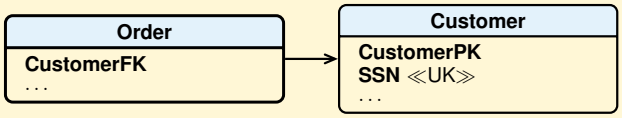
For a slowly changing dimension, we adopt the following solutions on the basis of the strategy specified in the requirements:

Type 1 (overwriting the history) The new attribute value replaces the old value in the record of the dimension table.

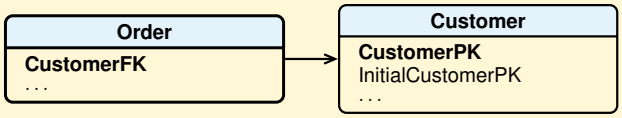
Type 2 (preserving the history) A new record is inserted in the dimension table, with a different surrogate key. For example, if a customer changes residence, a new record is inserted in the dimension Customer, as if there were two customers with different surrogate keys. The orders of the past relate to the customer with the old residence, the orders of the future will refer to the customer with the new residence.

This solution is an example that motivates the use of surrogate primary keys, but creates a problem for the analysis that requires counting the number of different customers who have made a certain order: if the count in the analysis is done with a COUNT(DISTINCT CustomerFK), customers who have changed address would be counted several times and, therefore, the result would not be correct.

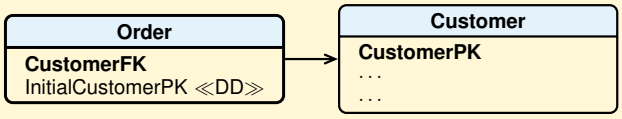
The problem is solved by adding an attribute to the dimension table with a value appropriate to establish that records with different surrogate keys relate to the same customer who changed residence. Possible solutions are (Figure 3.4): (a) use a customer "natural" key different from the surrogate, like the *Social Security* number (SSN), (b) use the first surrogate key that was assigned to the customer and, (c) to avoid having to do some frequent analysis with junctions, this information is stored in the fact table as a degenerate dimension.



(a) *A natural key in the dimension table*



(b) *First surrogate key in the dimension table*



(c) *First surrogate key in the fact table*

Figure 3.4: Slowly changing dimension

Type 3 (preserving one or more versions of history) In the dimension two attributes are added, one for the new value and the other for the modification date. For example, if a customer changes residence, the Customer dimension structure three attributes are used for the residence: Residence, NewResidence, ChangeDate. If the residence changes again, a new record may be inserted as the solution of the Type 2. Other solutions are possible on the basis of expected analysis, but they all make the solution and the queries for the analysis more complex, and their use should be considered carefully.[1]

1. For examples see http://en.wikipedia.org/wiki/Slowly_changing_dimension

If a dimension changes frequently due to numerical attributes, an alternative to consider to the previous ones is the following:

Type 4 (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, such as age or income. 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*.

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*. For example, the dimension is stored in two tables, one with the attributes that do not change (or change slowly) and the other with only those attributes that change frequently, and defined by range of values (e.g. with strings like "From-To"), agreed with users based on the type of analysis to be done. The fact table would then have two foreign keys: one for the primary dimension table and another for the fast changing attributes.

Shared Hierarchies

If there is a shared hierarchy, its attributes are stored in a separate table.

Recursive Hierarchies

An example of a recursive hierarchy is presented in Figure 3.5: in the dimension Agent of Order there is an attribute Supervisor to represent a recursive relationship among agents (Figure 3.5).

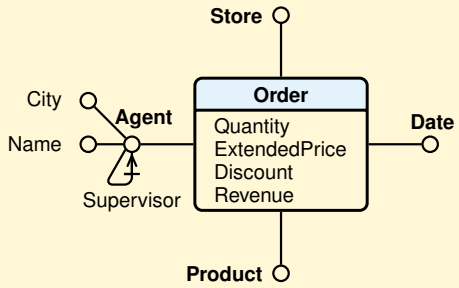


Figure 3.5: A dimension with a recursive hierarchy

Let us consider two examples of relational schemas to represent this conceptual model (without considering the dimensions Store, Date and Product).

Type 1 A relational schema with a recursive hierarchy.

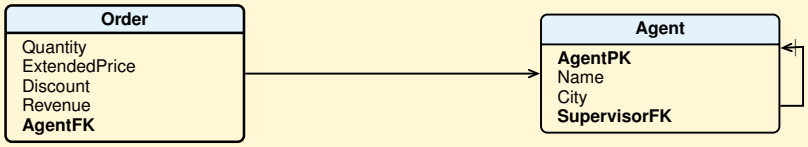


Figure 3.6: A relational schema with a recursive hierarchy

For example, suppose that we have the following agents hierarchy:

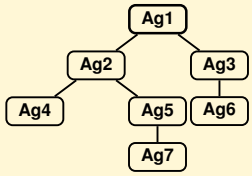


Figure 3.7: A hierarchy for an organization chart

Then a set of records for the table Agent could be:

Agent

AgentPK	Name	City	SupervisorFK
1	Ag1	Pisa	NULL
2	Ag2	Pisa	1
3	Ag3	Firenze	1
4	Ag4	Pisa	2
5	Ag5	Pisa	2
6	Ag6	Firenze	3
7	Ag7	Pisa	5

Let's see some examples of queries, assuming that the table Order has the following data:

Order

Quantity	Extended Price	Discount	Revenue	AgentFK
10	10	0	100	1
10	20	0	200	2
10	30	0	300	3
10	40	0	400	4
10	50	0	500	5
10	60	0	600	6
10	70	0	700	7

The queries that exploit the hierarchy can be defined as follows by using the so-called *Common Table Expression* (CTE), using the **WITH RECURSIVE** syntax.

Q1. This query finds the total order revenue placed by the agent Ag2, including subordinates (Ag4, Ag5, Ag7), for which he is responsible at every level:

```
WITH RECURSIVE Ag2andSubordinates AS
   ( SELECT  AgentPK
     FROM    Agent
     WHERE   Name = 'Ag2'
       UNION
     SELECT  A.AgentPK
     FROM    Agent  A JOIN Ag2andSubordinates  S ON A.SupervisorFK = S.AgentPK
   )
SELECT 'Ag2' AS Name, SUM(Revenue)
FROM    Ag2andSubordinates  S JOIN Order  O ON S.AgentPK = O.AgentFK;
```

Result

Name	SUM(Revenue)
Ag2	1800

Note that the first part of the query, with **UNION**, returns the primary key of the agent Ag2 and all its subordinates, while the second part sum the revenue for all the four agents.

Q2. This query finds the total order revenue for Ag6 and all its supervisors (Ag3 and Ag1):

```
WITH RECURSIVE Ag6andSupervisors AS
   ( SELECT  AgentPK, SupervisorFK
     FROM    Agent
     WHERE   Name = 'Ag6'
       UNION
     SELECT  A.AgentPK, A.SupervisorFK
     FROM    Agent  A JOIN Ag6andSupervisors  S ON A.AgentPK = S.SupervisorFK
   )
SELECT 'Ag6' AS Name, SUM(Revenue)
FROM    Ag6andSupervisors  S JOIN Order  O ON S.AgentPK = O.AgentFK;
```

Result

Name	SUM(Revenue)
Ag6	1000

Q3. This query finds the names of the agents that do not have subordinates:

```
SELECT      A1.Name AS Name
FROM        Agent  A1
WHERE       NOT EXISTS
            (SELECT     *
            FROM        Agent A2
            WHERE       A2.Supervisor = A1.AgentPK);
```

Result

Name
Ag4
Ag6
Ag7

Q4. This query finds the names of the agents that do not have supervisors:

```
SELECT      Name
FROM        Agent
WHERE       Supervisor IS NULL;
```

Result

Name
Ag1

Non recursive solution

It is interesting to note that the same problem, instead of using a recursive table, can be resolved by eliminating the SupervisorFK in the Agent table and introducing an auxiliary table that explicitly lists all the hierarchy links between the agents. In this case, the relational schema is shown in Fig. 3.8.

The auxiliary table ForTheHierarchy contains one row for each pair of (Agent, Subordinate), as well as a row for each agent with itself, and has the following structure:

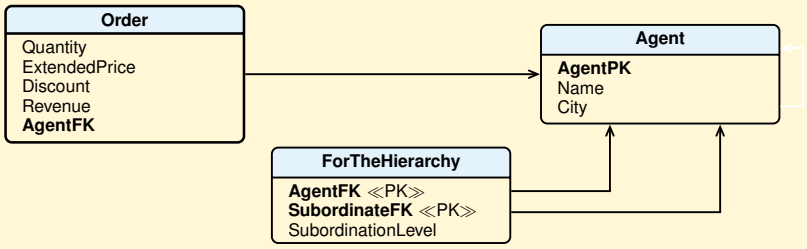


Figure 3.8: A non recursive solution to represent a recursive hierarchy

– AgentFK, a foreign key for the table Agent that represents the supervisor agent.
– SubordinateFK, a foreign key for the table Agent that represents a subordinate agent (at any level) or itself (AgentFK = SubordinateFK).
– SubordinationLevel with a value of the number of arcs of the path from the agent to the subordinate.
– (AgentFK, SubordinateFK) is the primary key.

For example, the records of the table ForTheHierarchy, for the agents hierarchy of Figure 3.7, are the following:

ForTheHierarchy

AgentFK	SubordinateFK	SubordinationLevel
1	1	0
1	2	1
1	3	1
1	4	2
1	5	2
1	6	2
1	7	3
2	2	0
2	4	1
2	5	1
2	7	2
3	3	0
3	6	1
4	4	0
5	5	0
5	7	1
6	6	0
7	7	0

We show now how to perform the previous four queries, producing exactly the same results.

Q1. This query finds the total order revenue placed by the agent Ag2, including subordinates (Ag4, Ag5, Ag7), for which he is responsible at every level:

```
SELECT    A.Name, SUM(Revenue)
FROM      Order O JOIN ForTheHierarchy H ON O.AgentFK = H.SubordinateFK
          JOIN Agent A ON H.AgentFK = A.AgentPK
          WHERE A.Name = 'Ag2'
GROUP BY  A.Name;
```

Q2. This query finds the total order revenue for Ag6 and all its supervisors (Ag3 and Ag1):

```
SELECT    A.Name, SUM(Revenue)
FROM      Order O JOIN ForTheHierarchy H ON O.AgentFK = H.AgentFK
          JOIN Agent A ON H.SubordinateFK = A.AgentPK
          WHERE A.Name = 'Ag6'
GROUP BY  A.Name;
```

Q3. This query finds the names of the agents that do not have subordinates:

```
SELECT    A.Name
FROM      Agents A
WHERE     A.AgentPK IN
          (SELECT    AgentFK
          FROM       ForTheHierarchy H1
          GROUP BY   AgentFK
          HAVING     COUNT(*) =1);
```

Q4. This query finds the names of the agents that do not have supervisors:

```
SELECT    A.Name
FROM      Agents A
WHERE     A.AgentPK IN
          (SELECT    SubordinateFK
          FROM       ForTheHierarchy H1
          GROUP BY   SubordinateFK
          HAVING     COUNT(*) =1);
```

With the *non recursive solution*, it is easy to perform a query with a data hierarchy restriction to a certain level (SubordinationLevel < 2), but it has the following disadvantages: the auxiliary table ForTheHierarchy data is complex to generate and update.

For instance, if in the *recursive solution* we must insert two other agents (Ag8 supervisioned by Ag7, and Ag9 supervisioned by Ag8), this would require only the insertion of two rows in the Agent table for the *recursive solution*, while in the *non recursive solution* we have to insert, in addition to the two new rows in the table Agent, *other* **11** *rows* in the table ForTheHierarchy, producing the following table (the updates are in strong black).

ForTheHierarchy

AgentFK	SubordinateFK	SubordinationLevel
1	1	0
1	2	1
1	3	1
1	4	2
1	5	2
1	6	2
1	7	3
1	**8**	**4**
1	**9**	**5**
2	2	0
2	4	1
2	5	1
2	7	2
2	**8**	**3**
2	**9**	**4**
3	3	0
3	6	1
4	4	0
5	5	0
5	7	1
5	**8**	**2**
5	**9**	**3**
6	6	0
7	7	0
7	**8**	**1**
7	**9**	**2**
8	**8**	**0**
8	**9**	**1**
9	**9**	**0**

Multivalued Dimensions

If there is a multivalued dimension, for example, an order item has been promoted by several agents (Figure 3.9), one of the following relational representations might be used (other solutions are presented in [Song et al., 2001]):

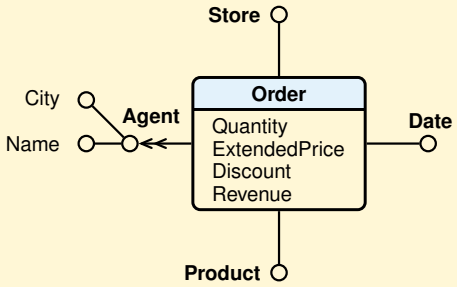
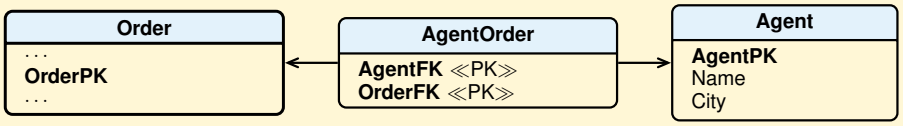
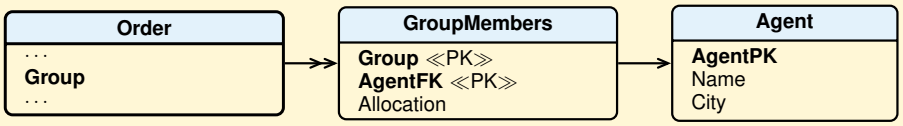


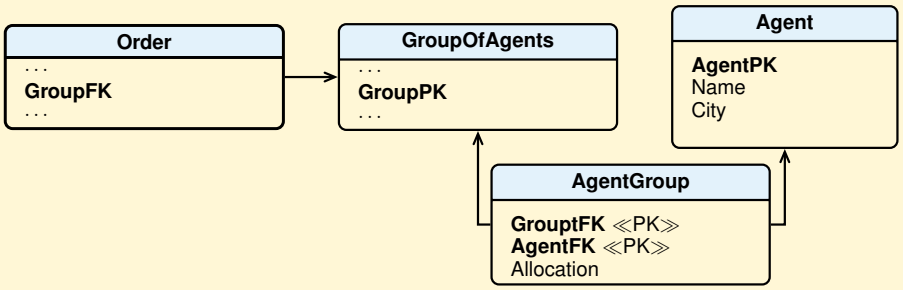
Figure 3.9: A multivalued dimension



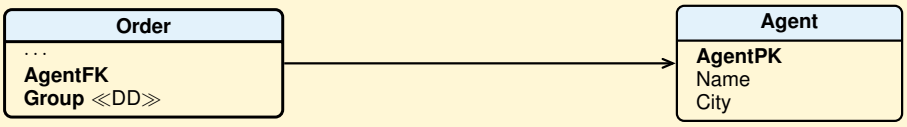
(a) *A traditional representation of a many-to-many relationship*



(b) *Another type of an auxiliary table*



(c) *A bridge table*



(d) *New fact granularity*

Figure 3.10: An auxiliary table to associate any number of agents with an order

1. The *many-to-many relationship* between the fact table and the dimension table is represented with a traditional auxiliary table (Figure 3.10a), called even in this case *bridge table*. This solution, however, violates the properties of a scheme to be a star and may not be accepted by some BI systems (e.g. SQL Server Analysis Services).
2. The *many-to-many relationship* between the fact table and the dimension table is represented with another type of an auxiliary table GroupMembers (Figure 3.10b), with attributes

- Group, the code of a group of agents,
- AgentFK, the foreign key for the table Agent, and
- Allocation with a value between 0 and 1, which represents the contribution to the order promotion credited to each group member, such that the sum of all the allocation factors belonging to a single group is exactly 1.

The table GroupMembers has, for each group, as many elements as are the agents of the group. An agent may be present in several groups. This solution may not be accepted by some BI systems (e.g. SQL Server Analysis Services).

In the fact table Order there is the attribute Group which indicates the group of agents of the table GroupMembers involved in a particular order. The relationship between the fact table and the GroupMembers table in Figure 3.10b is *many-to-many*. This is not a mistake: if the same group of agents collaborate again for another order, the same group number will be used.

To generate the report to find out the *total order revenue by agent name*, to avoid a wrong SQL query, we must distinguish whether we are looking for the *total order revenue contribution of a group member* (*weighted analysis*), or if we are looking for the *total order revenue of the groups to which an agent belongs* (*impact analysis*). In the first case the value of the attribute Allocation must be used as follows:

```
SELECT    A.AgentPK, A.Name, SUM(Revenue * GM.Allocation)
FROM      Order O, GroupMembers GM, Agent A
WHERE     O.Group = GM.Group AND GM.AgentFK = A.AgentPK
GROUP BY  A.AgentPK, A.Name;
```

while in the second case the attribute Allocation is not used, and in general a different result is found.

3. Two auxiliary tables are used (Figure 3.10c): GroupOfAgents, which contains one row for each group of agents, with the primary key Group, and AgentGroup to model the *many-to-many* relationship between the tables Agent and GroupsOfAgents.

   In the fact table Order there is the foreign key GroupFK for the table GroupOfAgents which indicates the group of agents involved in a particular order. This solution is usually accepted by BI systems (e.g. SQL Server Analysis Services).
4. Another solution is to change the fact granularity: instead of using a record for each order item, a record is used for each agent who has promoted the order item, with the weighted values of the measures, and the attribute Group as a degenerate dimension to recognize groups of records that relate to the same order item (Figure 3.10d). This solution increases the memory occupied by the fact table by a factor equal to the average number of agents that promote an order, while the one with a *bridge table* in general uses less memory.

Multivalued Dimensional Attributes

If a dimension has multivalued attributes (Figure 3.11), the problem is solved as in the case of the multivalued dimension, by treating the dimensional table as the fact table in the previous case, and preserving the relationship between the fact table and dimension table. Figure 3.12 shows only the first solution.

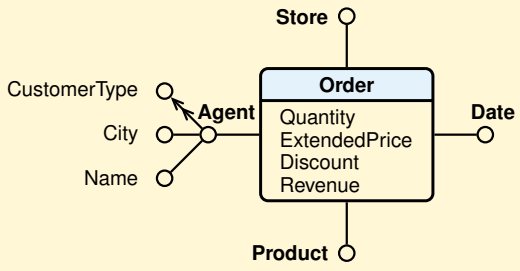


Figure 3.11: A dimension with a multivalued attribute

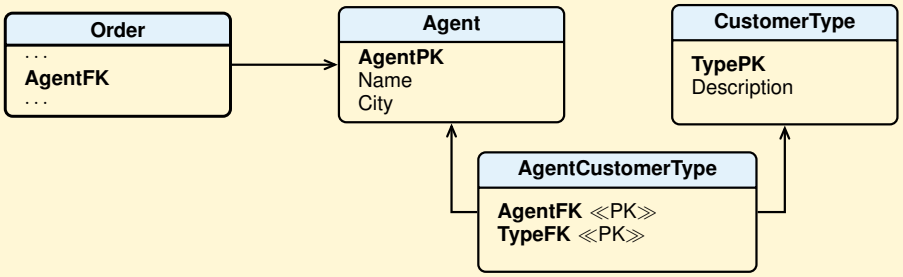


Figure 3.12: An auxiliary table to deal with a multivalued dimensional attribute

3.3 A Case Study

A case study is presented to show how to apply the methodology to design a DW. Do not be misled by the simplicity of the example. In practice the procedure is complicated by the difficulties of the requirements analysis phase for the quantities of the details and the many exceptions that usually occur. Aspects that are neglected in the example.

3.3.1 Requirements Analysis

1. **Requirements gathering**

   (a) *Analysis of the nature and purpose of the company*. CelPhone is a company that deals with the production and sale of cellular phones with its own sales outlets.

   To meet growing market demand the company has expanded by opening new plants and sales outlets. The company growth has started to level off, and management is refocusing on the performance of the organization using a DW to facilitate the analysis of the inventory and revenue from product sales. It has created a team consisting of one data analyst, one process analyst, one manufacturing plant manager, and a sales manager for the project.

   (b) *Business processes analysis.* The products are available in different models and are constructed from a set of common components. Each model may be eligible for discounting, and in this case the salesperson may discount the price if the customer buys a large quantity of the model or a combination of models. The discount must be approved by the manager of the sales outlet.

   The plant keeps an inventory of the 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.

   A customer places orders from a sales outlet. Unless a discount is negotiated, the suggested retail price is charged. Each sales outlet, on average, creates 500 orders per day, seven days per week. Each order consists of an average of 10 product models.

   (c) *Collection of business requirements for data analysis (business questions) and verification that the data needed are available.* Let us assume that the expert in DW, after an analysis of the life cycle of a product, inventory and sales processes, organization structure, the meaning of cost, discount and revenue, has interviewed executives interested in the data analysis, and has collected the following set of typical online data analysis of users interest:

	Inventory process
1	Average quantity on hand and reorder level for each model by month, by model identifier and description, by manufacturing plant, name and region.
2	Models that have reached the reorder level at least once in all manufacturing plants of a certain region.

	Sales process
3	The total cost and revenue by model sold, by month, by manufacturing plant, name and region.
4	Percentage of models eligible for discounting, and of those, what percentage are actually discounted when sold, by sales outlet, for all sales this week (or this month, or this quarter).
5	The top five models sold last month by total revenue, or by quantity sold, or by total cost.
6	Total cost and revenue by model Id and description, by month of the last year, by sales outlet and region.
7	Number of customers who last month bought the 5 models that have produced the highest margin, by the region of the sales outlet.

The operational database, which contains all the necessary information for the analysis, is shown in Figure 3.13.

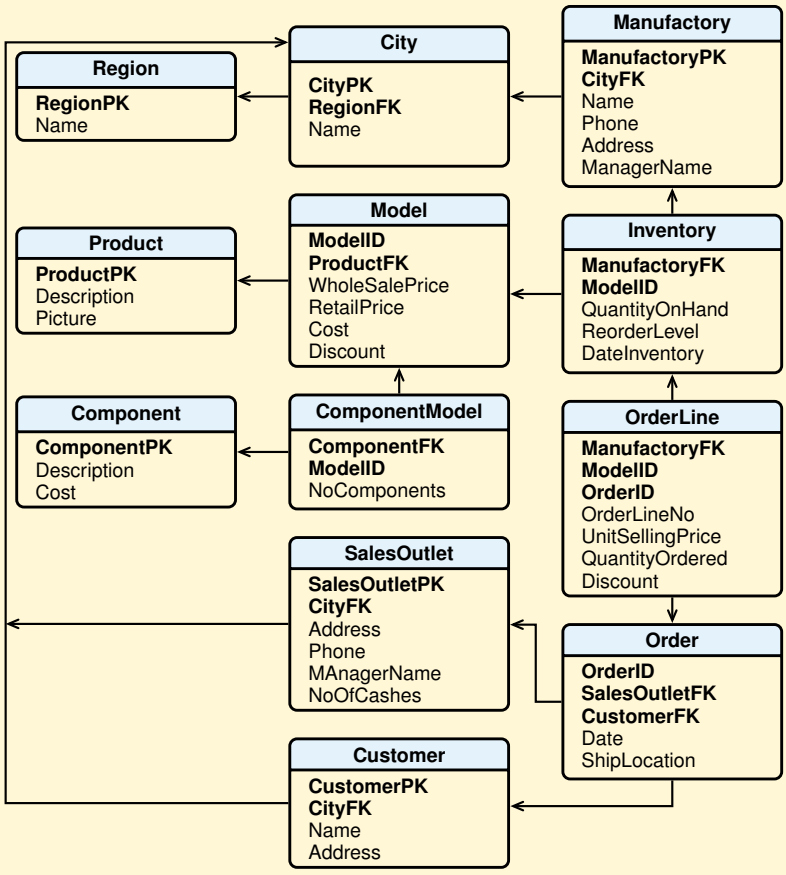


Figure 3.13: The operational database

2. **Requirements specification**
   Each business question is analyzed to identify the preliminary dimensions (in parentheses, the attributes) and measures of interest (for brevity, the metrics are not described), and then the grain of the fact.

Inventory process

N	Business questions	Dimensions	Measures
1	Average quantity on hand and reorder level for each model by month, by model identifier and description, by manufacturing plant, name and region.	Model (ModelID Description), Manufactory (Name, Region), Date(Month)	QuantityOnHand, ReorderLevel
2	Models that have reached the reorder level at least once in all manufacturing plants of a certain region.	Model, Date(Week), Manufacturing(Region)	ReorderLevel

With regard to the granularity of the facts, for the Inventory the data of interest are those about each product at the end of the month.

Inventory fact

Description	Preliminary Dimensions	Preliminary measures
A fact is about each product state at the end of the month.	Model, Manufactory, Date	QuantityOnHand, ReorderLevel

The description of dimensions, attributes, and fact *Inventory* measures follow.

Dimensions

Name	Description	Granularity
Date	...	A month
Model	...	A model
Manufactory	...	A manufacturing plant

Date

Attribute	Description
Month	...
Year	...

Model

Attribute	Description
ModelID	...
Description	...

Manufactory

Attribute	Description
Name	...
Region	...

Measures

Measure	Description	Aggregability	Calculated
QuantityOnHand	...	Semi additive across Date	No
ReorderLevel	...	Non additive	No

Sales process

N	Business questions	Dimensions	Measures
3	The total cost and revenue by model sold, by month, by manufacturing plant, name and region	Model (ModelID, Description), Date(Month), Manufactory (Name, Region)	ExtendedCost, Revenue
4	Percentage of models eligible for discounting, and of those, what percentage are actually discounted when sold, by sales outlet, for all sales this week (or this month, or this quarter)	Model(Discount), SalesOutlet, Date (Week, Month, Quarter)	ExtendedPrice, Discount
5	The top five models sold last month by total revenue, or by quantity sold, or by total cost.	Model, Date(Month)	ExtendedCost, QuantityOrderd, Revenue
6	Total cost and revenue by model Id and description, by month of the last year, by sales outlet and region	Model (ModelID, Description), SalesOutlet(Region), Date(Month, Year)	ExtendedCost, Revenue
7	Number of customers who last month bought the 5 models that have produced the highest margin, by the region of the sales outlet.	Customer, Model, SalesOutlet(Region), Date(Month)	Margin

With regard to the granularity of the facts, for the Sales the data of interest are those about each single *line item* of an order.

Sales fact

Description	Preliminary Dimensions	Preliminary Measures
A fact is about a product sold	Model, Manufactory, Customer, SalesOutlet, Date	QuantityOrdered, ExtendedPrice, Revenue ExtendedCost, Discount

The description of dimensions, attributes, and fact *Sales* measures follow.

Dimensions

Name	Description	Grain
Model	...	A model
Date	...	A day
Manufactory	...	A plant
Customer	...	A customer
SalesOutlet	...	A sales outlet

Model

Attribute	Description
ModelID	...
Description	...
Discount	...

Date

Attribute	Description
Day	...
Month	...
Quarter	...
Year	...
Week	...

Manufactory

Attribute	Description
Name	...
Region	...

Customer

Attribute	Description

SalesOutlet

Attribute	Description
Region	...

Dimensional Hierarchies

Dimension	Description	Hierarchy type
Date	Day $\rightarrow$ Month $\rightarrow$ Quarter $\rightarrow$ Year	Balanced

Measures

Measure	Description	Aggregability	Calculated
QuantityOrdered (Q)	...	Additive	No
ExtendedPrice (P)	UnitPrice $\times Q$	Additive	Yes
ExtendedCost (C)	UnitCost $\times Q$	Additive	Yes
Discount (D)	ExtendedPrice reduction	Additive	No
Revenue (R)	$P - D$	Additive	Yes
Margin	$R - C$	Additive	Yes

Before moving on to other phases of design, dimensions and measures of the facts are represented in the following tabular form highlighting what measures and dimensions are common to different facts and therefore need to be conformed or renamed. The dimensions Date and Model have different attributes in the two facts, and it is assumed that those of the fact Sales are used.

Fact dimensions

Dimension	Inventory	Sales
SalesOutlet		X
Model	X	X
Manufactory	X	X
Customer		X
Date	X	X

Fact measures

Measure	Inventory	Sales
QuantityOnHand	X	
ReorderLevel	X	
ExtendedPrice		X
ExtendedCost		X
Revenue		X
Margin		X
QuantityOrderd		X
Discount		X

3.3.2 Initial Analysis-driven Data Mart Conceptual Design

The data analysis requirements show that the facts are about Inventory and Sales. The attributes used in the data analysis suggest that the two possible initial conceptual data marts are those shown in Figure 3.14.

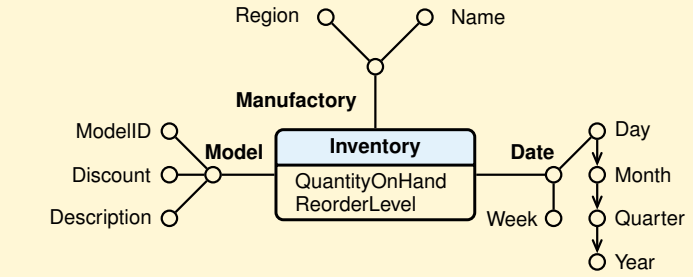
3.3.3 Candidate Source-driven Data Mart Conceptual Design

1. **Operational data analysis**. Based on the data analysis requirements, the relational schema is examined to decide which tables and attributes are interesting, other than primary and foreign keys.

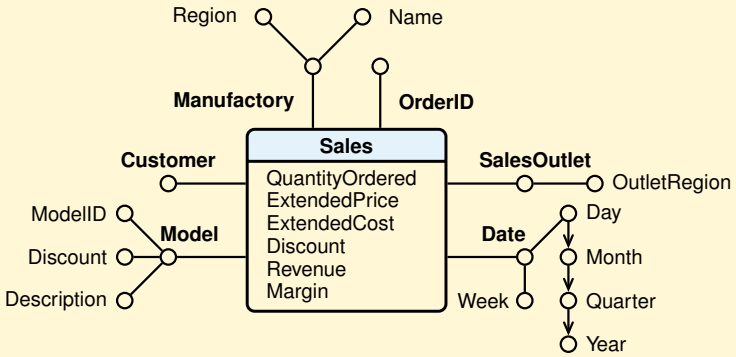
   As for the tables, we observe that ComponentModel and Component contain information that is not relevant for the purposes of data analysis.

   As for the attributes of other tables, the following considerations apply:

   – Region: the attribute Name is of interest.



(a) *Inventory*



(b) *Sales*

Figure 3.14: Initial data mart conceptual designs

- City: although not explicitly required, it is good to retain the information about the city, because, as we shall see later, the table City is related to the table Region via a *many-to-one relationship*, and because, in principle, it is always better to consider some more information, potentially relevant, than what is strictly necessary.
- Manufactory: the attributes Phone, Address and ManagerName are not relevant for data analysis; the pertinent attribute is the geographic location (City and Region).
- Inventory: all attributes are of interest for the data analysis of process Inventory.
- Model: the relevant attributes are ModelID, Discount and Description.
- Product: the attributes of this table are not of interest for data analysis.
- OrderLine: the attributes of this table are important for data analysis; Discount is a percentage.
- Order: ShipLocation is not useful for our purposes, while Date is of interest.
- SalesOutlet: the relevant attribute is the geographic location (City and Region).
- Customer: we are interested in Name and the geographic location (City and Region).

In this first analysis the keys to the tables were deliberately neglected, because they will be considered later. Once the relevant information has been chosen, we proceed to the next phase of design, namely the classification of entities.

2. **Entity classifications**. We classify the tables in the relational schema based on their content and the relationships between them:

   - **Transaction entity**: Recalling the definition of transaction entity, it is quite easy to fit into this category the tables Inventory and the merging of OrderLine and Order, with the granularity of OrderLine. They, in fact, (a) describe events that occur frequently at certain dates and (b) contain numerical attributes that represent possible measures of interest for the analysis of data. Note that the table

Model contains other numerical attributes, but they are not relevant for data analysis.

– **Component entity**: They are the tables related to a transaction entity via a *one-to-many relationship*. Analyzing the relational schema it is discovered that
  – for the transaction entity Inventory, the component entities are Manufactory and Model,
  – for the transaction entity OrderLine, merged into Order, the entities component are Customer, SalesOutlet and Inventory.

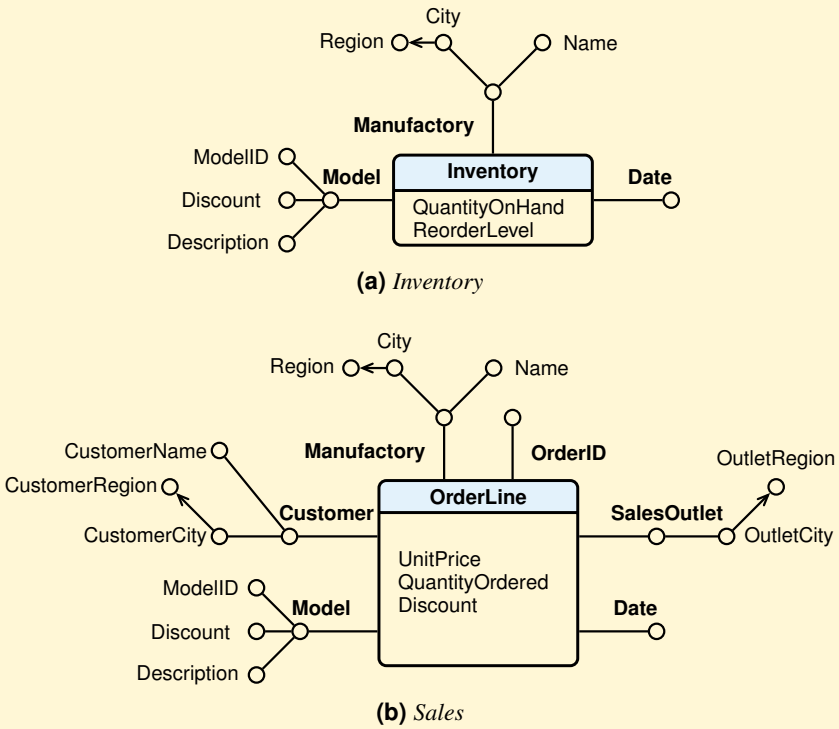
  Finally, as mentioned earlier, among the entities of each entity component, we add the time entity (present in the relational schema with attributes of type Date).
– **Classification entities**: These are the tables related to a component entity by a chain of *one-to-many relationships*. Their interesting attributes are added to those of minimal component entity.

  For the entity Manufactory, component of Inventory, the classification entities of interest are City and Region.

  For the entity SalesOutlet and Customer, component of OrderLine, the relevant classification entities are City and Region.

  For the entity Inventory, component of OrderLine, the relevant classification entities are Models and Manufactory, with City and Region. Since in the requirements analysis of Sales process there is no interest in Inventory attributes, the classification entities Models and Manufactory are treated as components of the entity event OrderLine.

3. **Definition of the candidate data mart conceptual designs**. Having identified two interesting event entities, we proceed with the definition of two conceptual designs for the data marts with the relative dimensions (Figure 3.15).



(a) *Inventory*

(b) *Sales*

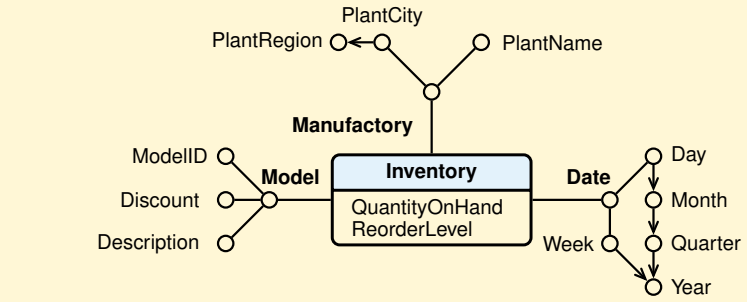
Figure 3.15: The candidate data mart conceptual designs

As far as the dimensional hierarchies are concerned, the case contains only one which is explicit, that between City and Region.

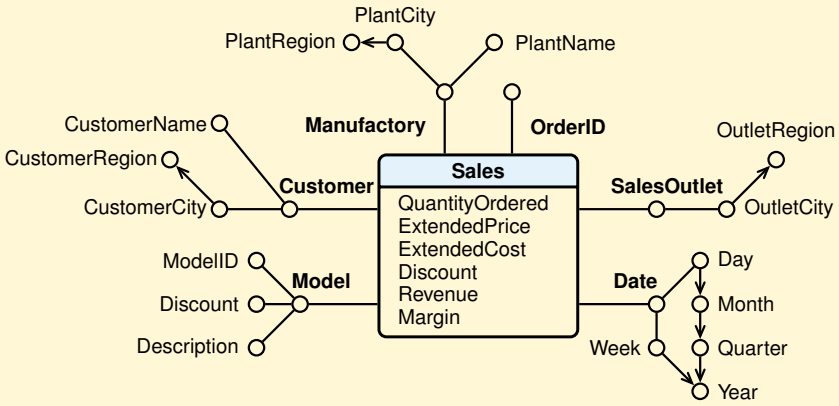
4. **Analysis of data marts fact granularity and measures additivity**. This step produces no information other than that already known with regard to the fact Inventory. For the fact OrderLines, however, we note that the measures UnitPrice and Discount% are not additive, and so in the final step of the conceptual design, the solution used to the fact Sales is preferred.

3.3.4 Final Data Mart Conceptual Design

From a comparison of candidate designs and the initial ones, the terminology is unified and the final designs are those of Figure 3.16.



(a) *Inventory*



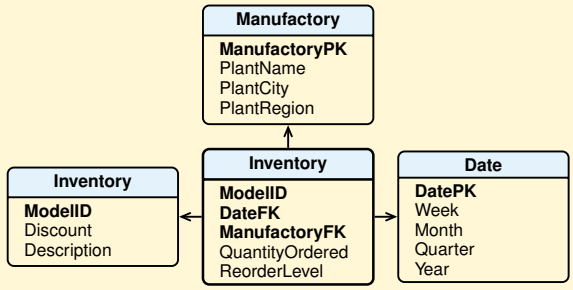
(b) *Sales*

Figure 3.16: The final data mart conceptual designs

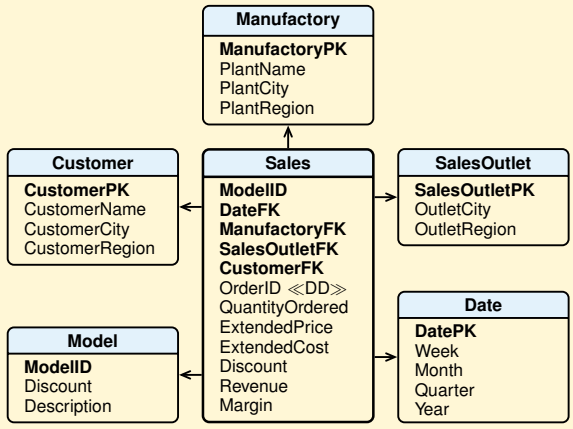
3.3.5 Data Mart and Data Warehouse Logical Design

Two data mart star schemas are defined with a different fact table for each conceptual design, while for each dimension a table is defined in association with the fact table, by defining appropriate surrogate primary keys and foreign keys (Figure 3.17a,b).

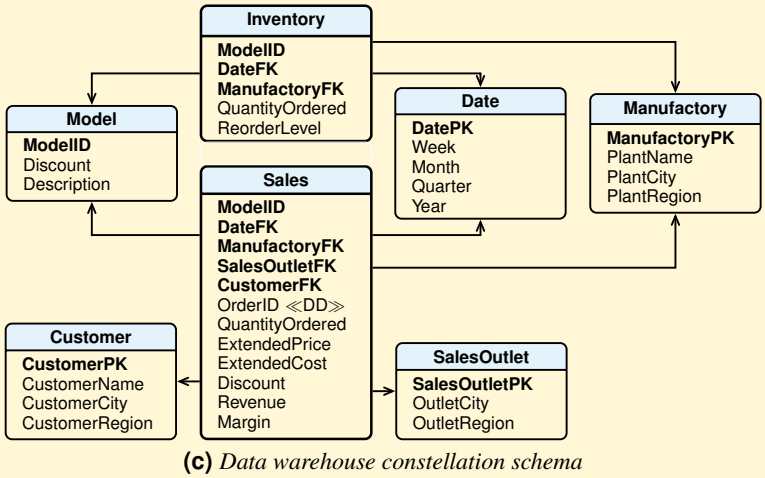
The data marts relational schemas are then integrated to define the DW schema. Note that the two star schemas share the dimensions Date, Model and Manufactory. The structure of the DW is shown in Figure 3.17c.



(a) *Inventory data mart star schema*



(b) *Sales data mart star schema*



(c) *Data warehouse constellation schema*

Figure 3.17: The data mart star schemas and the data warehouse constellation schema

3.4 Project Quality Control

Let us consider some checks for the final review of a project to improve its quality.

Conceptual Design

1. **Granularity of the facts**
   The determination of the grain of the facts is the first key step in the design of a data mart. Choosing the grain means deciding the meaning of a fact, and so the pertinent measures and dimensions.

   For example, for orders with multiple lines, if the granularity is the order, it makes sense to consider a measure that concerns the total order value, but not the amount of product ordered on each order line, and so the Product dimension can not be used, but if the granularity is the order line, then it is meaningful to consider measures about the quantity of product ordered, the price and the revenue (but not the total order value), and the Product dimension.

   In the data mart conceptual design, the measures have numerical values that can be added across dimensions. Descriptive attributes must be modeled as *degenerate dimensions*.

   The grain decision for the facts also determines the grain of each dimensions. For example, if the grain for the PropertySales is an individual property sale, the grain of the Client dimension is the detail of the client who bought a particular property.
2. **Measures**
   Measures are numerical quantities useful for evaluating the performance of the processes to be analyzed. It is also useful to define measures that can be calculated from others, at the time of loading the data. This is particularly true for values fundamental in the analysis, such as revenue and margin, to avoid their being calculated by users incorrectly, or in different ways, within reports at the time of the analysis. If the answers are wrong or inconsistent, the data warehouse will be viewed as wrong.

   It is also important to document, as part of the conceptual design, whether they are *additive*, *semi-additive* or *non-additive*, to avoid very common mistakes when they are summed up.

   Another common error to avoid is modeling unit amounts (e.g. unit price) as measures rather than extended amounts (e.g. extended price). This does not mean that unit amounts must be excluded from conceptual design, because they may be valuable information for analysis. If there is not a dimension where they can be stored, they may be placed in the data mart conceptual design as *degenerate dimensions*.

   The most useful measures are those additives that can be aggregated with any type of function and by combining facts with various dimensions to answer common business questions.

   The most critical measures, often to be avoided, are the *non-additive* because they can not be aggregated with the sum. Typical examples are measures defined as rates or percentages. These measures must be broken down into underlying components that are additive, to calculate the *ratio of the sums*, not the sum of the ratio. For example, the *margin rate* is the ratio of the margin to revenue. These components are fully additive, and they are usually defined as measures that can be safely aggregated to any level of detail. The non additive margin rate is computed in a query, or by additional processing logic in the reporting tool, as the ratio of the sums of margin and of revenue.
3. **Date and Time**
   They should always be modeled separately as dimensions. They are modeled as facts descriptive attributes only when are not used for analysis.
4. **Dimensions quality**
   The dimensions should be chosen considering the user's need for examining facts, and the future development of the DW. If the same dimension appears in multiple data marts they must defined in the same way to be shared (*conformed dimensions*). Examples of these dimensions are time, date, customer and product.

(a) The dimensional attributes must be useful (a) to analyze facts (restrictions, groupings, and aggregations) and (b) to produce summary reports with the headers using the users vocabulary to facilitate understanding.

(b) The names of the attributes of different dimensions should be different: a way to disambiguate them is to prefix the attribute with the dimension name. Attribute names should not be those used in the operational databases, but those that are used in the analysis and that appear in reports.

(c) Dimensional attributes already represented as numerical measures of the facts should not be repeated in dimensions.

(d) The values of dimensional attributes, usually strings of characters, should facilitate the interpretation of the reports: avoiding codes (0/1, F/M, etc.) or adding attributes that describe them.

(e) Represent as a string data type attributes such as Date and Address only if there is no interest in exploiting in the analysis the implicit hierarchies among their attribute values.

(f) If a fact is associated with more elements of a dimension, it must be modeled as multi-valued.

(g) The dimensions can be *degenerate*, that is they are without attributes because their values are numbers, such as the order number, invoice number etc., or strings of characters.

(h) The dimensional hierarchies should always be present to make the analysis more useful at several levels of detail.

Logical Design

1. **Surrogate keys**
   Surrogate keys must be used for dimension tables, which may also include the primary key of the source data.
2. **No attributes with null values**
   Default attribute values must be set in the database schema to avoid nulls in the database. The default value for all the fact measures must be zero in the schema. In SQL, NULL plus a number equals a NULL, and the aggregate functions perform a NULL-elimination step, so that NULL values are not included in the final result of the calculation. The only aggregate function that does not implicitly eliminate NULL is the COUNT(∗) function. However, an aggregate function AGG(A), with A a set of NULL, returns NULL, while COUNT(A) returns zero.

   To deal with cases in which for a fact record the dimension value may be unknown, the dimension table must have a special record with an attribute value "Not Found", and when a fact record is missing a dimension data, the foreign key value is the surrogate key of the record "Not Found".
3. **Degenerate dimensions**
   In the logical schema degenerate dimensions become attributes of the fact table as the foreign keys to other dimensions, if the attributes take up little storage space, otherwise they are stored in separate dimension tables.

   Another type of degenerate dimension arises when there are a few attributes that take different values (status indicators), e.g. order line with status (closed, open, canceled), customer satisfaction, type of delivery, payment terms etc. These attributes could be added (a) in the fact table, increasing its memory size, (b) in the relevant dimension tables, duplicating the records (if a customer pays in cash, by bank transfer, credit card, three records are needed), (c) in different dimension tables of small cardinality, by increasing the number of foreign keys in the fact table, (d) collect them all into a separate dimension table with as many attributes as there are fields with discrete values, and as many elements as are the possible combination of values (*junk dimension*). The latter solution is preferable in the presence of several attributes of state indicators used in the fact table or in different dimension tables.
4. **Shared data**
   If the shared data are dimensional hierarchies, such as geographic hierarchies, in the logical design a way to treat them is to deconstuct the dimension table into a tree structure. So a snowflake dimension

is defined, and its advantage must be evaluated considering the savings in space, the greater complexity of the scheme, the execution time analysis, and any ambiguity in analysis, such as "Analysis of the total revenue for the city": What city does it refers to? Customer, agent, or the warehouse city?

To facilitate understanding of the data mart schema, and to avoid ambiguous analysis, if the tables are small, it is usually preferable to duplicate shared hierarchies in the dimension tables using different attribute names.

If the shared data are dimensions, or there are more dimensions with different attributes that have the same values (e.g. two dates with the same attributes day, month, year, or with different attributes to highlight the role of different dates), such as OrderDate and ReceivedDate, another solution may be used. Instead of using two separate date tables, with the same granularity, two views are created, with different attribute names, from a single Date table.

5. **Dimensional hierarchies**.
   Check that the hierarchies type (balanced, incomplete or recursive) is correctly represented. Moreover, verify that functional dependencies hold over a loaded dimension table with dimensional hierarchies. For example, let the dimension be Date(PkDate, Month, Quarter, Year). If the dimensional hierarchy Month $\rightarrow$ Quarter is valid, then following query returns an empty result set.

```
SELECT      Month
FROM        Date
GROUP BY    Month
HAVING      COUNT(DISTINCT Quarter) > 1;
```

6. **Snowflake dimensions**
   Do not normalize (snowflake) dimension tables, since it will be harder for the users to analyze data. Moreover, in general, there is not a very significant memory saving because of the relatively small cardinality of the dimension tables. Snowflakes are meaningful only when it is necessary to define interesting dimension tables shared among several data marts.
7. **Changing dimensions**
   Recognize the dimensions with attributes that change over time and treat them appropriately.

3.5 Summary

- The design of a data warehouse to support business decisions is a complex task that requires a methodology organized into phases, like that used to design operational databases, but the phases objectives must be revised properly to adapt them to multidimensional modeling.
- A possible design methodology has been presented, with the documentation to be produced at the end of each phase, to proceed by considering both the requirements analysis and the operational database available.
- The logical design phase has been presented to highlight some critical aspects of the transition from the conceptual design to the relational one, especially for treating dimensions that change over time, multivalued dimensions and multivalued dimensional attributes.
- Finally, some controls have been listed for the final review of a project to improve its quality.

Chapter 4

A DATA WAREHOUSE TO SUPPORT ANALYTICAL CRM ANALYSIS

Customer relationship management (CRM) comprises a set of processes and enabling systems supporting a business strategy to build long term, profitable relationships with specific customers. Customer data and Business Intelligence applications form the foundation upon which any successful CRM strategy is built. From the architecture point of view, the CRM framework can be classified into *operational* and *analytical*. *Operational CRM* refers to the automation of business processes, whereas *Analytical CRM* refers to the analysis of customer characteristics and behaviors so as to support the organization's customer management strategies. A taxonomy of typical *Analytical CRM* decision support analyses is presented first, and then the design of a starter data warehouse for a case study to support them with analytic SQL queries.[1]

4.1 Introduction

A business enterprise (company) is an organization that transforms a set of resources into products (goods or services) that can profitably be sold in the market. While the guiding principle of the production (efficiency and innovation) has not changed over time, the company strategy of approaching the market, and so the concept of *marketing*, has changed profoundly because of technological, production and market events which have characterized the industrialized countries evolution in the past.

In a nutshell, Philip Kotler summarizes the evolution of strategies to approach the market in the recent economic history as follows:

- *Production orientation*. The goal is to reduce production costs, because the market is characterized by a shortage of manufactured goods relative to demand. To put it another way, *if somebody makes a product, somebody else will want to buy it*. This orientation dominated business from the time of the *industrial revolution* until the early 1920's, beginning of capitalism to the mid 1950s, and it still exists in some developing countries industries, where the market wants cheap commodities. In this context, the main problem is how to reach the market; the marketing function corresponded to what is now called physical distribution and sale of the product, and the American Marketing Association defined marketing as "the set of activities to direct the flow of goods and services from producers to consumers and commercial users."
- *Product orientation*. The goal is the continuous product improvement, in terms of quality, performance and innovation, because customers prefer products that offer a high level of quality and performance, and they are willing to pay a higher price for the differential characteristics. It is the belief that good products require a modest marketing effort (they sell themselves). This orientation is typical around the 30s of the twentieth century.
- *Sales orientation*. The goal is to sell what is produced with an activity of advertising and selling on a large scale, because the market is characterized by a balance of supply and demand. With this

1. The material is based on some ideas and examples presented in [Adamson and Venerable, 1998], [**?**].

orientation, the selling philosophy gave way to the concept of marketing. This orientation is typical of the 50s and 60s of the twentieth century.

– *Market orientation*. The goal is to produce what is desired by customers, because the market is characterized by an excess of supply over demand. It is therefore necessary first to determine what products are desired by customers and then produce them, instead of first creating products and then trying to persuade customers that they need them (from the strategy *produce-and-sell* to the strategy *listen-and-make*). This orientation is typical from the later years of the twentieth century and is always in continuous development nowadays. Business scholars began to actively discuss the concept of market orientation in 1990 with a paper by Ajay K. Kohli and Bernard J. Jaworski for the "Journal of Marketing" that defined market orientation as organizational business intelligence focusing on the needs of the customer and how to apply that intelligence to the operations of the organization.

> *Marketing is the activity, set of institutions, and processes for creating, communicating, delivering, and exchanging offerings that have value for customers, clients, partners, and society at large. (American Marketing Association, 2007)*

By understanding the market environment, customer needs, and by defining a customer-oriented marketing strategy, the company is able to reach the goal of a greater personalization of the offer based on the wishes of its customers and, therefore, to create profitable relationships with them. In this context the *Customer Relationship Management* (CRM) comes into play, defined by Kotler as follows:

> *CRM is a business strategy that aims at the creation and consolidation of profitable relationships with specific customers by offering superior value and satisfaction.*

More profit from each customer relationship can be obtained (1) acquiring new customers, (2) increasing the profitability of existing customers and (3) extending the duration of customer relationships.

The CRM therefore requires, more and more, companies looking for marketing and the use of business intelligence techniques for analyzing and understanding customer purchase behavior and characteristics, and to use this information to focus on the most promising and profitable customers.

In the next section the focus will be on the main components of CRM and on the most common data analyses to support it.

4.2 Operational and Analytical CRM

The customer relationships are considered by modern companies a key strategic resource to be managed appropriately to increase the business value and gain competitive advantage.

The CRM is the answer to this need that requires

1. A *strategic approach* based on the integration of following concepts and tools
   – *organization* (review of processes, incentive systems, skills, etc.),
   – *marketing* (relational and one-to-one, *Customer Value Management*, etc.),
   – *information and contact systems* with customers, potential or current.
2. A *relationship management* along the *customer lifecycle*.
3. An *in-depth and integrated customer knowledge* built using a variety of data sources *organized* and *analyzed* consistently with the analytical, decision-making, and operational capabilities present in the company.

From the architecture point of view, the main components of the CRM framework are the *Operational CRM* and the *Analytical CRM* (Figure 4.1).

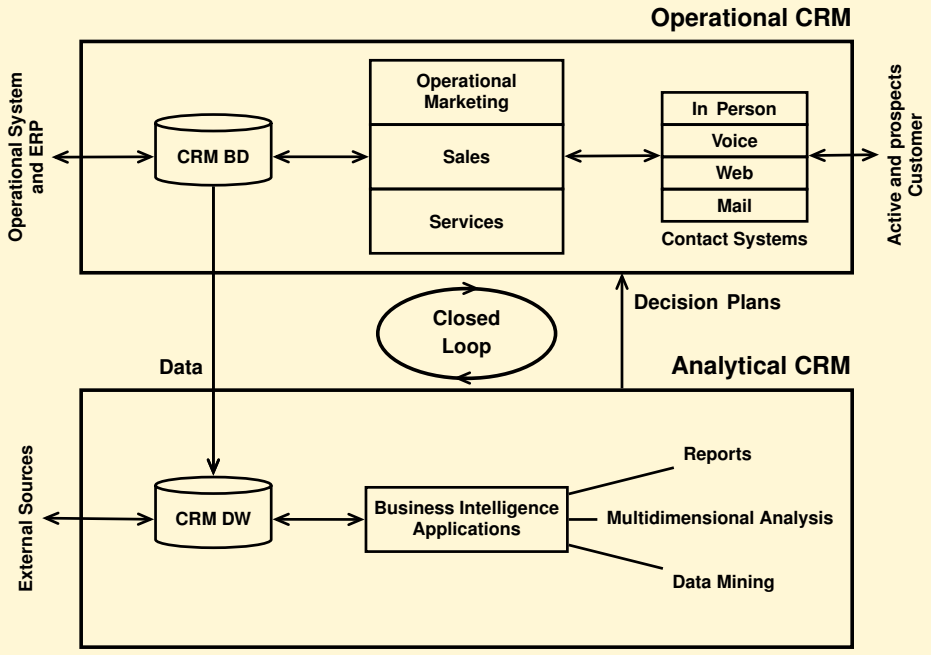


Figure 4.1: The components of the CRM framework

4.2.1 Operational CRM

The *Operational CRM* system manages the execution of operational activities and processes of interaction with customers using the most suitable channels for the content of the activity or service to be provided. There are several channels of interaction that differ in the quality of information that can be exchanged (in person, voice, mail, fax, SMS, email, websites, etc.).

Computer applications relate to three main processes of company-customer interactions:

- *Operational marketing* to create new opportunities for contact with customers and prospects, organizing the activities of advertising, communication and promotion. Some examples of applications are:
  - Management of new contacts.
  - Management of advertising campaigns.
  - Telemarketing.
  - Management of promotional campaigns.
- *Sales* for the activities after the first contact that can lead to the formulation of a business proposal, to the drafting of a contract, to the formulation and process of orders. Some examples of applications are:
  - Contact management.
  - Management of sales opportunities that could arise.
  - Management of the business estimates;
  - Management of customer information relevant to the sale.
  - Configuration of the products or services.
  - Management of products catalog and related price lists;

  - Management of the sales force.

- *Customer service* to respond to customer requests for information or assistance, consistent with the company strategies and the characteristics of the customers. Some examples of applications are:

  - Management and analysis of requests for assistance.
  - Management of the operations in the territory;
  - Assistance "self-service" on the web (FAQs, technical documentation, etc.).
  - Management of service level for customers;
  - Management of remote assistance centers (help desk, call center etc.).

An important element of the *Operational CRM* is its database that extend the operational one with all other available data on customers, sales, contact, motivation, satisfaction, and purchase behavior:

- *Customer data*: Socio-demographic, on purchased products or services, on customer satisfaction, status of the customer life cycle, on the origin of the contact etc.
- *Products and services data*: Data on commodities (raw materials, delivery, price lists, discounts, warranties, etc.) and data on "variant" of goods or services (customization, configuration options, etc.).
- *Marketing activities data*: Types (e.g, institutional advertising, "targeted" print campaign, brochures, web marketing initiatives), duration (start and end times), correlation with the results (contacts activated, contacts coming to fruition) etc.
- *Data on interaction systems used*: Types, performance data, etc.

4.2.2 Analytical CRM

The *Analytical CRM* system supports decision making by providing useful information extracted by the analysis of data collected by *Operational CRM* and properly organized in a data warehouse based on the needs of decision makers. The objective is to identify both the best customers for more profitable business relationships, and the customers having high defection risk, based on their profiles and purchasing behavior.

The two main components of the CRM must operate in a highly integrated and synergistic manner to create a closed loop between them: with the activities of operational marketing, sales and customer service, the *Operational CRM* produces data on the dynamics of relationships with customers (*customer life cycle*) which are then analyzed by the *Analytical CRM* to support the organizations decision-making process. These decisions lead to new operational actions for customer contact and so on. The interaction between the two components are critical to the achievement of the objectives and advantages of CRM.

Typical examples of *Analytical CRM* analyses categories are: *Sales and Marketing Analysis*, *Profitability Analysis*, *Service Quality Analysis*, and *Customer Analysis* (Figure 4.2).[2]

2. For the sake of simplicity, we will not consider (a) *Contact Center Analysis* to analyze the effectiveness of the company contact center used to interact with customers, i.e. the activities within a call center, the problems reported to and resolved via the help desk, and (b) *eCRM or Web-based CRM Analysis* to determine the effectiveness of the site as a channel-to-market by quantifying the user's behavior while on the Web site.

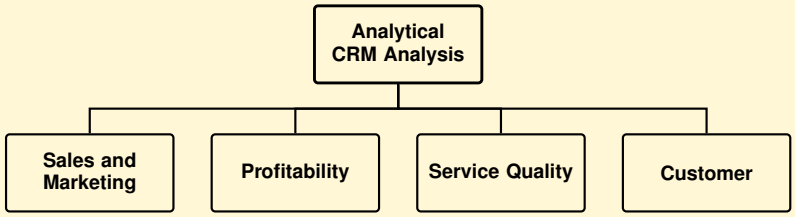


Figure 4.2: Types of Analytical CRM analyses

An analysis category is a logical grouping of related *business questions* with the KPIs to evaluate. A business question is an inquiry that the end user want answered, and the KPIs reflect the organization's goals and success factors.

Sales and Marketing Analysis

The goal of each company is to profitably sell what it produces. Profit is important, but there is no profit unless there are sales. For this reason, each company initially focuses on the analysis of product sales (revenues or volumes) during its strategic planning and resource allocation. Among the possible types of analysis there are those by product, sales channel, geographic area, and customer. The sales analysis allows then to plan and evaluate the effects of promotional campaigns on current and potential customers.

Profitability Analysis

Revenues from sales must be analyzed taking into account the cost of resources used to produce them, because the profitability of sales (revenue minus costs) is an indicator of the firm's ability to maintain, over time, an advantage over its competitors. As in the case of sales, profitability is analyzed by product, sales channel, geographic area, customer, and promotional campaigns.

Service Quality Analysis

The success of a company depends not only on the ability to profitably sell its products, but also on ability to meet the demands and expectations of customers in a more efficient and effective manner than competitors. This is increasingly seen as critical to the long-term success. For this reason, specific analyses are needed both to understand the most common reasons for return of products not conforming to what was ordered, and the impact that the returns have on the revenues of the company, and the ability to deliver orders in full and timely.

Customer Analysis

The analysis of data on customer purchases are of great importance in the *Analytical CRM* to consolidate the *orientation to relationships* with customers. The purpose of a company is to create and serve a customer, to meet his needs, and therefore the value of a company depends primarily on the value of its relationships with customers, and so they focus on business strategies issues such as: acquiring a thorough knowledge of customers, figure out who they are, what they buy and what they might buy, as their purchasing behavior changes. The better one knows one's customers characteristics and behavior, the better one can maintain long-lasting, valuable relationships with them.

Typical examples of analyses focuses on finding the best customers (those that produce larger profit margins for the company), the most loyal customers, the customer defected, the customer groups (segments) with similar purchase behavior, or characterized by combinations of very specific attributes (geographic, demographic, psychographic). Each type of customer segment is then considered to implement specific marketing strategies or to redefine the production.

In the following, the focus will be on the the conceptual design of a DW to support *Analytical CRM multidimensional analysis* in the context of a company that sells its products under orders issued by customers. The project will be done gradually starting with the sales and marketing requirements, and then the DW design will be extended to support the other types of analyses.

Although a DW allows several interesting analyses, there are many others, especially those that involve predictive analytics, behavior recognition, cluster analysis and patterns finding, that require an exploratory approach to discovery useful models of data with *Data Mining* algorithms on data subsets extracted with queries from the DW. Data mining techniques, used to gain additional insights into customer behaviors and other characteristics that are of interest to the organization, are beyond the scope of this book.

4.3 Sales and Marketing Analysis

Sales and marketing analysis allows a company to analyze its sales of products as well as the effectiveness of its marketing and campaign efforts.

A common use for sales data is to identify the *best* members of business dimensions in terms of the total revenue value that exceeds some threshold. Other example of questions are: Which products are increasing in popularity and which are decreasing? Which products are seasonal? Which customers place the same orders on a regular basis? Are some products more popular in different parts of the country? Do customers tend to purchase a particular type of product?

The answers to these questions motivate the design of the first data mart for sales.

4.3.1 Sales Analysis

Sales drive business.

The case study is about a company interested in analyzing confirmed purchase orders. The following terminology will be used:

- Each order line is about the request for a product in a certain amount (*Quantity ordered*), which can be confirmed in a smaller amount (*Quantity sold*).
- A product has a *unit price*, a *unit cost* and a *unit discount*. When a product is sold in a certain amount, the following quantities are of interest:

  $$\textit{extended price} = \textit{unit price} \times \textit{quantity sold}$$
  $$\textit{extended cost} = \textit{unit cost} \times \textit{quantity sold}$$
  $$\textit{extended discount} = \textit{unit discount} \times \textit{quantity sold}$$
- The *revenue* from a product sales is:

  $$\textit{revenue} = \textit{extended price} - \textit{extended discount}$$
- The *margin* from a product sale and the margin as percentage of the revenue are:

  $$\textit{margin} = \textit{revenue} - \textit{extended cost}$$
  $$\textit{margin\%} = 100 \times \textit{margin} \;/\; \textit{revenue}$$

Requirements Analysis

Let us assume that from the business requirements analysis the following metrics have been identified for the following types of sales analysis of the products ordered by customers:

- Number of products ordered.
- Number of products sold.
- Number of different customers.

– Number of orders.
– Total revenue.
– Average revenue.
– Average order revenue.

The metrics are analyzed by the following dimensions:

– The Date, with attributes Month, Quarter, Year, to understand how the customers' purchase preference changes over time, especially that one of the customers who purchase regularly or more than others.
– The Product, with attributes Name, Type, Category, Year, to understand which products are preferred by customers, which are seasonal, and what promotional activities take.
– The Channel, with values "Direct", "Telemarketing", "Web"', to understand the different means in which a customer interacts with the company.
– The Customer, with attributes Name, BirthDate, Gender, City, Region, to understand customer behaviors in specific markets, profitability in those markets, as well as campaign effectiveness in those markets. For simplicity, we assume that the Market, where the company sell the products, is the city of residence of the customers.

Conceptual Design

The fact to analyze has the following grain:

Fact granularity

Description	A fact is the sale of a product
Preliminary dimensions	Product, Date, Order, Channel, Customer
Preliminary measures	Quantity Ordered, Quantity Sold, Product Extended Price, Product Extended Discount, Revenue

All the measures are additive.

Figure 4.3 shows the data mart conceptual design for sales analysis.

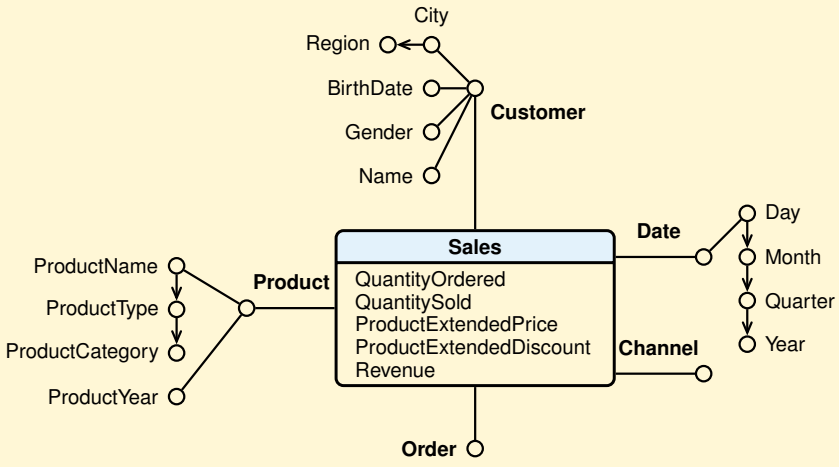


Figure 4.3: The sales data mart conceptual design

4.3.2 Marketing Analysis

Sales drives the business. Marketing, in turn, drives sales.

The marketing department, based on appropriate analysis of sales, identifies segments of customers with similar purchasing behavior to which address a direct marketing campaign, with certain types of promotion, to motivate customers to place orders through various channels. At the end of the campaign, it will be evaluated its impact on sales, and which customers respond to which campaigns.

Requirements Analysis

The main requirements expressed by the managers are the following:

- The goal of a promotion campaign is to stimulate the purchase of certain products by a customer segment identified based on their purchasing behavior and their demographic characteristics, considering both the active ones and those recently inactive.
- Customers are contacted directly and informed of the promotion type.
- The information of interest of a promotion is the start and end date, the promotion cost, the channel used to communicate the offer to customers (email, phone), the type (coupon, discount, gift, etc.), and the value.
- The company's interest is to analyze the sales generated by a promotion and the percentage of inactive customers who have been reactivated with the promotion, and to compare sales where the product was on promotion with sales where the product was not on promotion.

Conceptual Design

A new data mart Contacts is defined with the dimensions shown in Figure 4.4, which shares the Sales dimensions Date, Customer and Product.

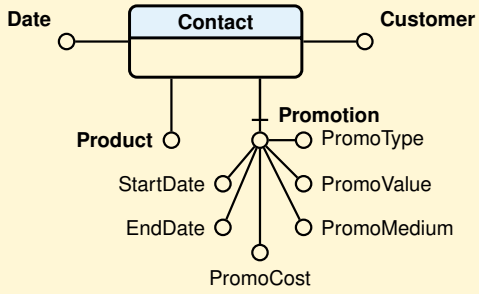


Figure 4.4: The data mart conceptual design for promotions contacts

To simplify the analysis to evaluate the success of marketing promotions, the Sales data mart has been extended with the Promotion dimension of the Contacts data mart (Figure 4.5). The promotions goal is to promote sales, so usually there is some kind of discount on the price of the products. If in the facts we want distinguish the types of discounts, the one for promotion will be considered too.

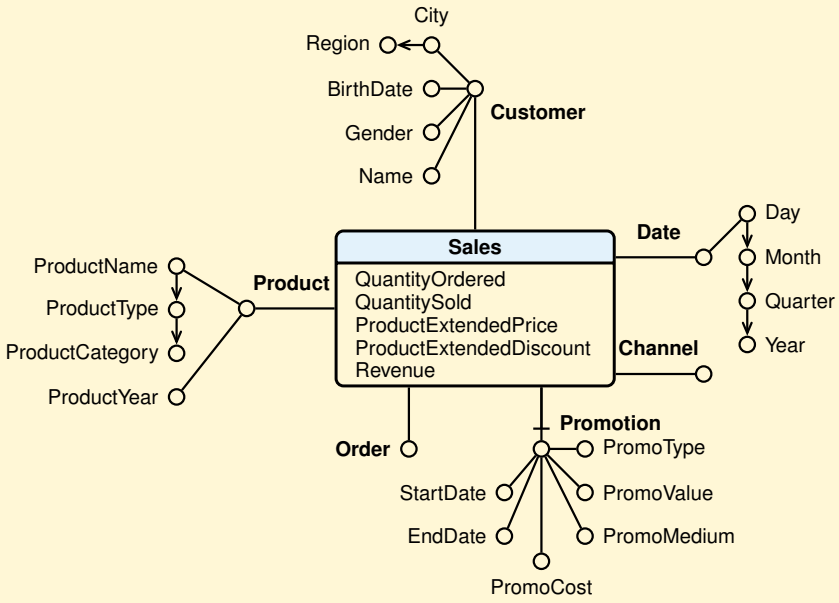


Figure 4.5: The Sales data mart conceptual design extended to deal with promotions

The data mart allows different analyses, but not those relating to products on promotion that have not been sold, because these events are not represented in the Sales data mart. To perform these types of analyses the Contact data mart must be used too to find out first the products that have promotions, then the Sales is used to find out products that have promotion that sell, and finally the difference is the products that have promotion but did not sell. According to Kimball, Contact is called a *coverage fact* that contains the information about which product where on promotion at which times.

4.4 Profitability Analysis

Whatever the value proposition of a business's product is, the business must realize enough revenues in delivering it to cover the associated costs.

The profitability analyses deal with identifying the *best* members of business dimensions in terms of the total margin that exceeds some threshold. For example, the *best customers*, the *best products*, the *best markets*, the *best channels* or *best campaign* that generates the greatest margin.

In the following, for brevity, we only consider the profitability of products, but the others are analyzed in a similar manner.

4.4.1 Products Profitability

The product profitability analysis allows a company to determine the profitability of each product or service that the business provides.

Products profitability is measured by sales margins. Revenues are easily evaluated, but the cost of the products is not easy to identify. Since we are interested in product profitability, we will focus on the costs that the business associates with products.

– *Product cost* is about all the costs that are involved in acquiring or making product.
– *Marketing cost* is about the costs of all activities performed to generate interest in the consumer.

– *Nonconformance cost* is about the costs of non-conforming products replacement or refund, assistance and technical support during the warranty period.

If, for simplicity, in the data mart conceptual design of daily sales the measure "product cost" only has been considered, for the process of sales (Figure 4.6), then we can only perform a partial analysis of the profitability of the type shown in Figure 4.7.

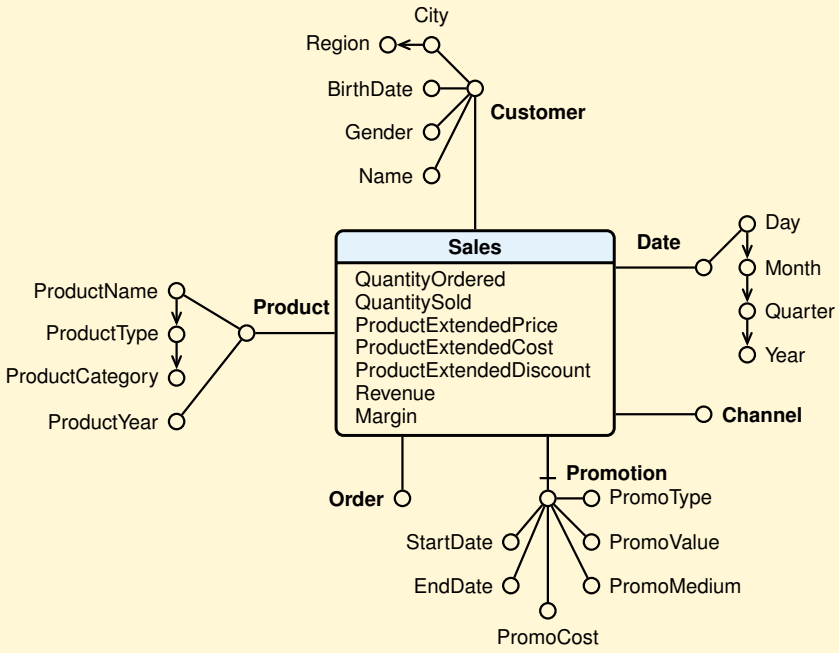


Figure 4.6: The Sales data mart conceptual design for products profitability analysis

Products Profitability
Year 2010

Product	Revenue (€)	Product Cost (€)	Margin (€)	Margin% (%)
P1	41 093	39 000	2 093	5
P2	4 674	3 830	844	18
P3	240 125	181 542	58 583	24
...	...	...	...	...

Figure 4.7: A limited products profitability analysis

In general, however, the products profitability must be analyzed considering the costs due to other processes that affect sales. Two typical examples are:

– The cost for product promotion.
– The cost for the management of product returns.

Considering the costs of these processes, a full analysis can be made of the profitability of the type shown in Figure 4.8, with different results.

Products Profitability Year 2010								
Product	**Revenue (€)**	**Product Cost (€)**	**Returns Value (€)**	**Returns Cost (€)**	**Promotion Cost (€)**	**Total Cost (€)**	**Margin (€)**	**Margin% (%)**
P1	41 093	39 000	1 650	248	4 800	44 048	−4 605	−12
P2	4 674	3 830	84	14	0	3 844	746	16
P3	240 125	181 542	4 367	888	6 200	188 630	47 128	20
...	...	...	...	...	...	...	...	...

Figure 4.8: A more general Product Profitability analysis

Figure 4.9 shows how to review the project by adding the cost measure ProductPromotionCost to the Sales data mart.

The value of this measure is the promotion cost allocated to a product, and it is calculated when the promotion has expired, by dividing the total cost of the promotion by the number of products sold during the promotion.

The nonconformance cost ProductReturnsCost is discussed in the next section.

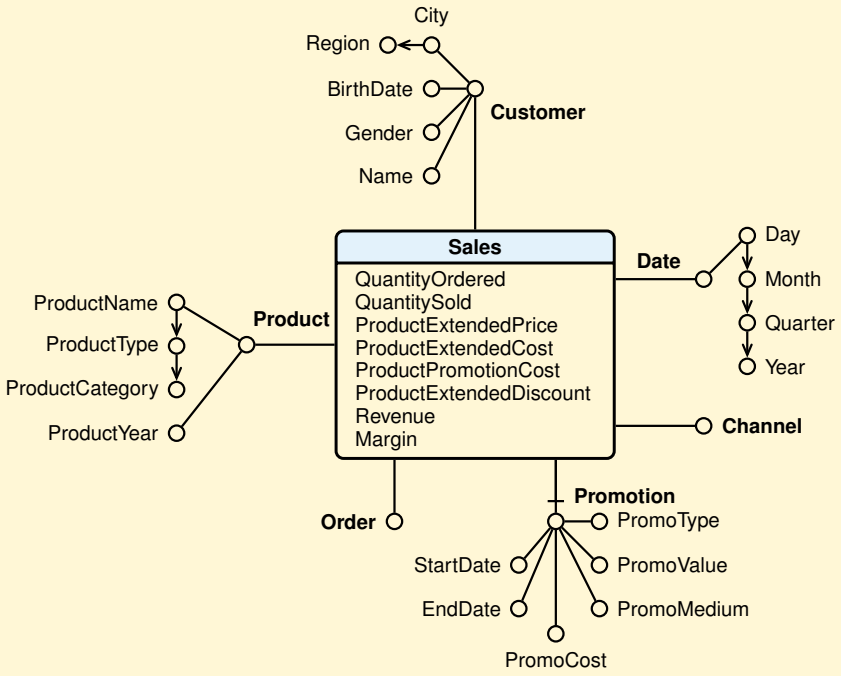


Figure 4.9: The Sales data mart conceptual design for products profitability analysis with the promotion cost

4.5 Service Quality Analysis

Customers do business with companies that continually meet the expectations created by the marketing process.

The customer service analysis allows an organization to analyze the conformance of a product or service to customers expectation. In particular, we will consider *product return analysis* to investigate the number of product returned as well as to provide insight as to why products are being returned, and *product delivery performance analysis*, also known as *order fulfillment analysis*, to investigate the company's ability to deliver product and services on time.

4.5.1 Returns Analysis

A common use for sales data is to identify the *best* members of business dimensions in terms of the total profit value that exceeds some threshold. But returns might have an impact on the analysis of some of the major business dimensions. To avoid the risk of overestimating the products profitability, the effects of the returns and the reasons must be considered: the best product that is returned because defective it can become the worst in terms of margins and may impair the image of company.

For the returns analysis the following measures are considered:

- *Returns value*. It is the amount refunded or credited to customers returning products. This amount is an important part of product profitability. It must be deducted from revenue so that we do not credit products with revenue that was subsequently lost. When a customer exchanges a product for the same product, the returns value is zero.
- *Returns cost*. It is the costs due to the transport of returns, for the repair of damaged products, and any other costs that includes the process of handling returns. The company's accounting system provides the procedures for determining the values of these cost items.

With the returns analysis the company, beside evaluating the overall impact on profitability, it can monitor the main causes of returns and determine interventions to address the main issues. Finally, the analysis of returns by customer can show potential dissatisfied customers that may leave.

Requirements Analysis

The main requirements expressed by the managers are the following:

- Of a return product is of interest the amount returned, the order number, the return value, the return cost, the reason for return, the date, the customer and the order in which it was requested.
- The returns are described by a *Disposition* (replacement, credit, refund, repair), and by a *Reason* (late delivery, product other, product damaged, etc.)

The following are some examples of business questions to be answered by the data marts:

- Total number and percentage of returns on the amount of sales, by market and year.
- Total amount of returns and costs, by product, reason and month.
- Number of orders with returns, its percentage of the total quantity of orders, and number of returns, by market and years.
- Total amount of returns, by customer, product and return disposition.
- Total revenues, margins, returns value, returns costs, residual margins and residual margins of revenues, by product and year.
- Total revenues, product costs, promotion costs, returns costs, residual margins and residual margins of revenues, by product and year.
- The top 10 reasons for the returns of products.

Conceptual Design

An approach to the analysis of returns is to provide a fact data mart *Returns* with the following granularity:

	Fact granularity
Description	A fact is the quantity of a returned product
Preliminary dimensions	Product, Date, Order, Customer, Disposition, Reason
Preliminary measures	Returns quantity, Returns value, Returns cost

All measures are additive.

In Figure 4.10 it is shown the data mart conceptual design for Returns, which shares the Sales dimensions Date, Customer, Orders and Product.

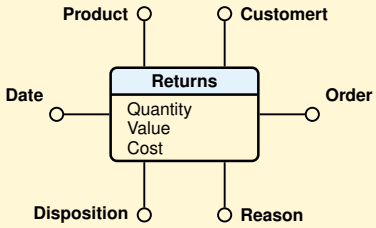


Figure 4.10: The data mart conceptual design for Returns

To consider in the sales analysis the nonconformance cost, the measure ReturnCost and the dimension Return have been added to the Sales data mart.

Since in the returns analysis it is interesting to know who are the suppliers of the products with quality problems, the Supplier dimension is also added to the data mart (Figure 4.11).

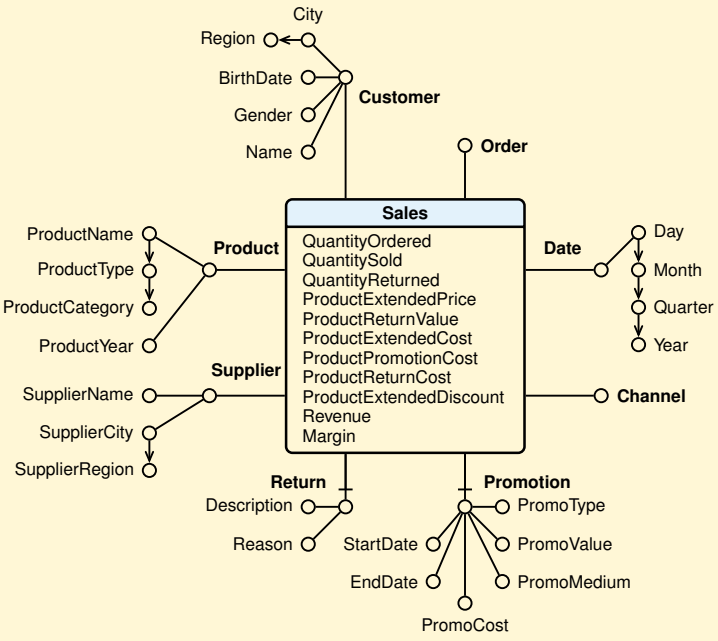


Figure 4.11: The data mart conceptual design for sales with returns

4.5.2 Order Fulfillment Analysis

The full and timely orders fulfillment are other important aspects to improve the competitiveness of companies. The analysis of the order fulfillment process, as the analysis of returned products, can highlight any customer dissatisfaction which can be the basis for their abandonment.

Requirements Analysis

The managers are interested in the following examples of business questions:

- Number and percentage of orders processed on time and late, by months.
- Number and percentage of orders processed on time and late, by distribution channel and by year.
- Number and percentage of orders processed on time and late, by customer and by month.
- Comparison of the amount of products ordered and sold (e.g., by differences or ratios).

Conceptual Design

To analyze order fulfillment, the Sales conceptual design is modified by adding to the dimension Order the attributes OnTime and OrderNo (Figure 4.12).

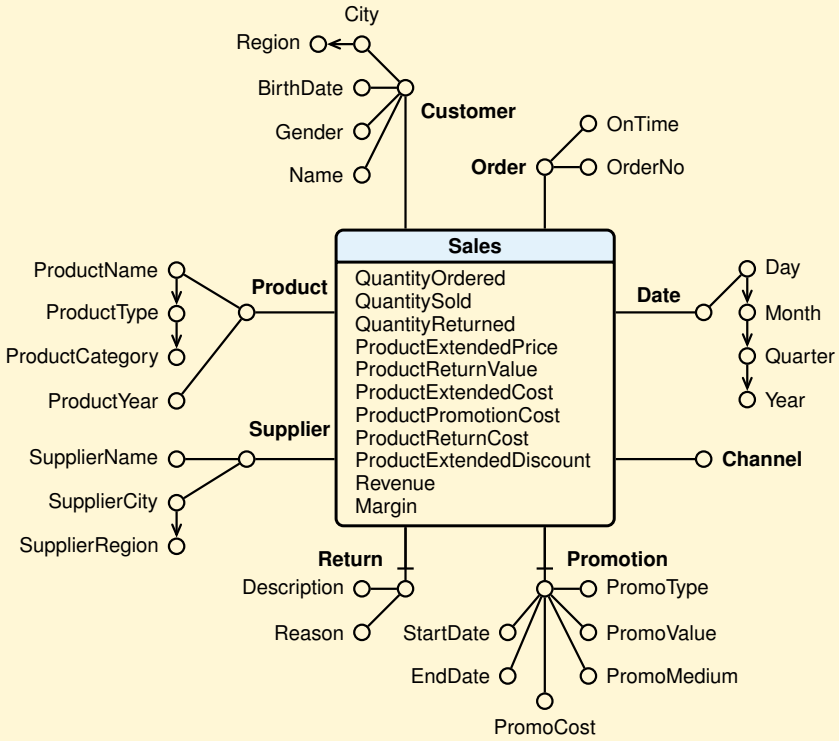


Figure 4.12: The data mart conceptual design for sales and order fulfillment analysis

4.6 Customer Analysis

The demographic attributes and the purchases behavior of the most profitable customers can be used to focus sales efforts.

The customer analysis is a very important part of *Analytical CRM* because it provides insight into an organization's customer base. That insight can be achieved by the following fundamental analysis:

1. *Customer segmentation analysis* allows a company to group similar customers into groups based upon characteristics that are common to the members of that group. There are different types of customer segmentation analysis:
   - *Demographic analysis* pertains to analyzing inherent characteristics of a customer such as age, gender, income, and geography.
   - *Customer behavioral analysis* enables a company to study a customer's buying propensities. In other words, customer behavioral analysis is used to determine which products and/or services a customer buys or is likely to buy.
   - *Customer lifetime value analysis* is used to analyze a customer's historical value and future value to the company.
2. *Customer retention analysis* tracks and analyses the number of customers that a company is able to keep from one time frame to another (i.e. customer loyalty).
3. *Customer attrition (or churn) analysis* allows a company to determine the number of customers lost over a period of time and provides insight into why customers leave. Its goal is to reduce the number of turnover of profitable customers by allowing a company to identify and take appropriate actions to retain customers that are likely to leave.
4. *Customer satisfaction analysis* allows a company to gain insights into how a customer perceives the organization by analyzing customer survey responses about the company and its products and services (i.e. measures how satisfied the customer is). The results allow a company to understand the key drivers for customer satisfaction and loyalty.

Requirements Analysis

For simplicity, we will assume that the conceptual design of the sales data mart must be enriched with other customer information to do analysis not only on products purchased, but also to identify customer segments to which address different marketing efforts, according to *demographic variables* (date of birth, sex, age, occupation, education level), the *geographic location* (city and region) and the *types of customer behavior* (for brevity, *typology*). The age of a customer is that when the order is made, while the typology is determined as follows, based on their purchases behavior in the last month and previous months:

New	With at least an order last month and no order in the past.
Constant	With at least two orders per month for three months in the last four months.
Occasional	With at least one order in the last four months, but not as for typology Constant or New.
Churn risk	With no order in the last four months after being Constant at least once in the last 12 months.
Inactive	With no order in the last four months, and not Constant in the last 12 months.

The following examples of business questions have been collected during the user interviews:

- Number and percentage of customers, by type, by market and year.
- Number and percentage of customers by income.
- Number and percentage of customers, by age range.
- Number and percentage of customers, by occupation and sex.
- Number and percentage of customers by education level and by sex.

– Number of distinct customers who responded to a promotion, percentage of total responses and percentage of total contacts, by customer typology.

Users are also interested in the following monthly analysis of the customers typology:

– Number of customers in one year, by typology and by month.
– Comparison of the number of customers in one year that have changed typology, by month.

The metrics will be analyzed by the following dimensions:

– Date, with attributes Month, Quarter, Year
– Product, with attributes Name, Type, Category, Production Year
– Customer, with attributes Name, Gender, Birth Date, Age, Qualification, Profession, Typology, City, Region.

It is assumed that (a) sales data are collected every day and (b) some customer data may change over time and must be treated as follows:

Type 1 (overwriting the history): Qualification, Profession.
Type 2 (preserving the history): City, Region.
Type 4 (fast changing): Age, Typology

Conceptual Design

To support customers analysis, the Customer dimension is extended with demographic and typology attributes, and a new data mart CustomerTypology has been defined for the monthly analysis of the customers typology evolution.

All the data mart conceptual designs to support the *Analytical CRM* analysis are shown in Figure 4.13. The data marts share dimensions with the same name.

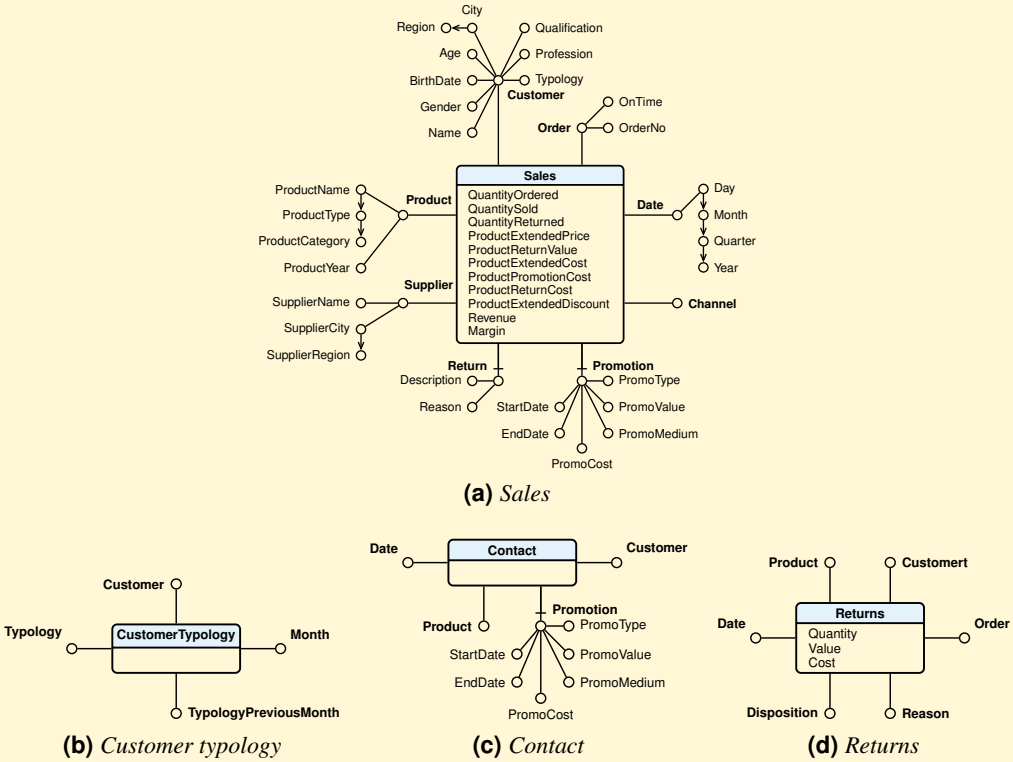


Figure 4.13: The data mart conceptual designs for Analytical CRM analysis

4.7 Data Warehouse Logical Design

For simplicity we consider only the data mart Sales, and we define a relational table for the facts with the following attributes:

- A foreign key for each dimension, with its own surrogate primary key.
- The measures.
- The degenerate dimensions Order and Channel.
- The simple measures OnTimeCount, OrderCompleteCount, DamageFreeCount, with value 1, if respectively the order delivery was on time, the line order is complete and it didn't entail any return, otherwise 0. It is useful to include these measures about order fulfillment to simplify some analysis interested in counting events.

The Product table contains all possible products on sale, the Customer table contains all the customers who have made at least one order and the Date table contains all the dates of the period of interest.

For the treatment of the customers attributes that change slowly, City and Region, with mode **Type 2**, a surrogate primary key is used in the Customer table, and for each attribute change a new record is created with a different primary key surrogate. To find out which of Sales data refer to the same customer, regardless of residence (for example, to know the number of different customers who have made orders), in the fact table the attribute InitialCustomerKey is added as a degenerate dimension, with the first surrogate key value assigned to a customer.

Finally, for the treatment of customer attributes that change frequently, Age and Typology, with mode **Type 4**, one of the following solutions can be used: To store possible attribute values in two dimensional tables or to store different possible combinations of the two attribute values in a single dimensional table. Since every attribute has few values (for Age we consider the following ranges: (up to 25), (26-34) (35-60) and (61 and over)), the second solution is preferred (Figure 4.14).

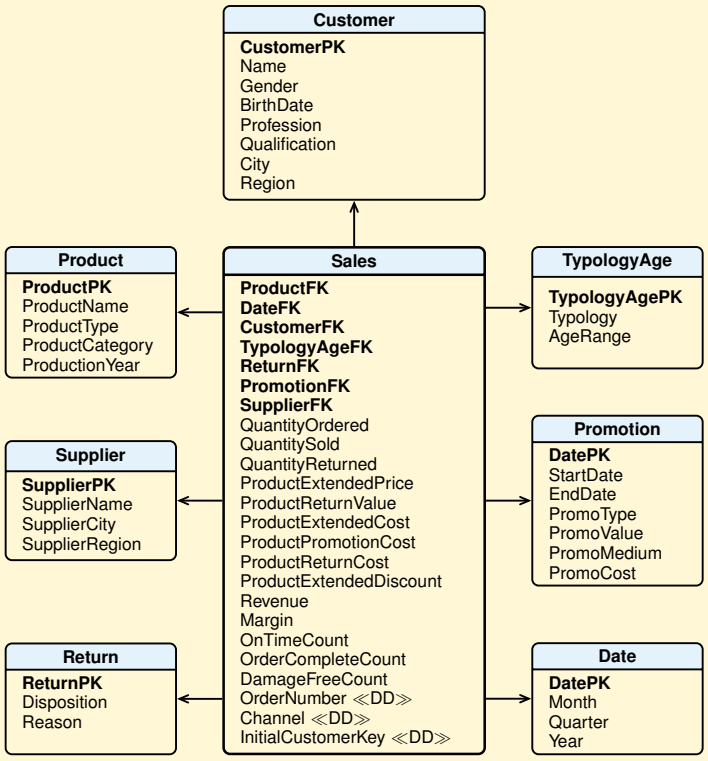


Figure 4.14: Initial logical design of a data mart for sales

4.7.1 Data loading

The operation is not trivial, as usually happens in the process of data warehousing, and it requires an appropriate program that processes the results of SQL queries. The operation may also require a review of logical design, as shown in this example.

Suppose that the data on sales are loaded into the data mart every day, from the data in the following tables created from the operational database:

- Products(PKProductBD, Name, Type, Category, ProductionYear, UnitCost), with data on products sold.
- DailySales(FKProductBD, FKCustomerBD, Date, QuantityOrderder, QuantitySold, ExtendedPrice, Discount, OrderNumber), with data on sales.
- CustomerData(PKCustomerBD, Name, Gender, BirthDate, Qualification, Profession, City, Region), with data on customer who have made orders.
- Suppliers(PKSupplierBD, Name, City, Region), with data on suppliers.
- Returns(PKReturnsBD, FKProductBD, FKCustomerBD, OrderNumber, Date, Quantity, Value, Cost, Disposition, Reason), with data on returns.
- Promotions(PKPromoBD, FKProductBD, FKCustomerBD, StartDate, EndDate, Name, Type, Value, Medium, Cost), with data on promotions.

There is also the table Classification(FKInitialCustomerKey, FKDate, FKTypologyAge) which contains, for each month, the information on the value of the customer typology established at the end of each month based on their buying behavior calculated from the fact Sales data.

The table Classification is another fact that shares with the fact Sales the dimensional tables Customer, TypologyAge and Date (Figure 4.15).

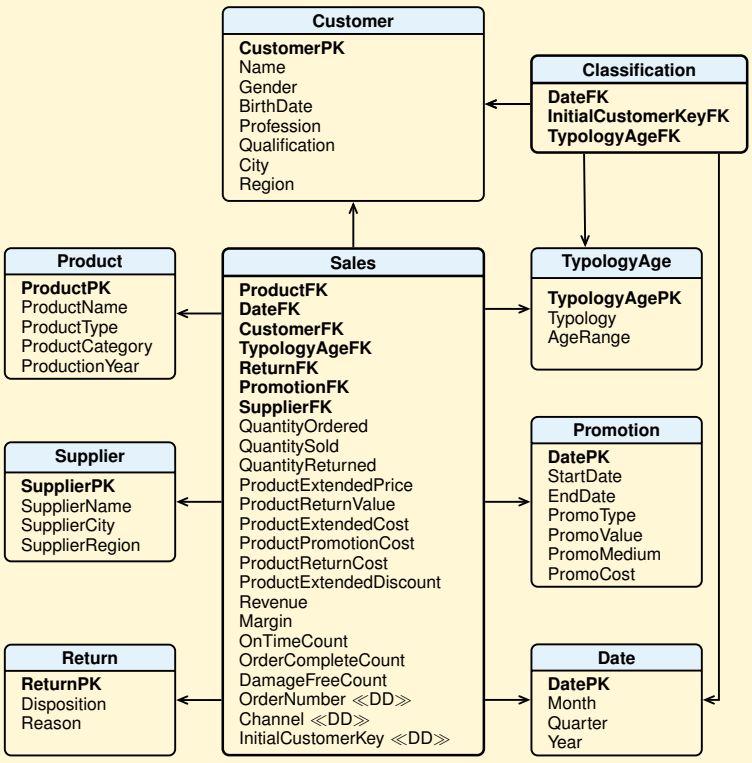


Figure 4.15: New version of the logical design of a data mart for sales

During the loading of data on sales at the end of the day, the age of a customer is that at purchase date, while his typology is determined by a process more complex because the requirements rule is that (a) for orders made during the current month, the customer typology is that of the previous month (which is derived from the Classification table), if a customer is known, otherwise the value is New, (b) then at the end of current month M the value is determined as follows:

1. Based on the purchasing behavior of customers over the last four months in Sales, a record is added to Classification, for each value of InitialCustomerKey, with the value of customer typology, determined as follows:
   - Constant, if the customer has made at least two orders per month for three months in the last four months. The number of orders in a month is determined by counting the number of distinct values of OrderNumber of the customer purchases. Then examining the number of customer orders for each of the four months, it is checked if in three months of the four, there are at least two orders.
   - New, if the customer has done at least an order last month and none in the past. Customers who have not made orders in the past can be identified in two ways: customers are not present in the table Classification in the months prior to the current; or adding to the table Customer an attribute with the *initial validity date* of the surrogate key (assigned at the date of the first order) and checking that this falls within the current month M. The second method is preferred because it is more efficient, since the table Customer is of much smaller size of the table Classification (Figure 4.16).
   - Occasional, if the customer is not New, but he has done at least an order over the past four months, but not with the typology Constant. To determine whether the typology is Occasional, we proceed as to Constant, by considering the number of orders in each of the last four months, and checking that the threshold Constant is not reached, and that the customer is not new (i.e. if there are only purchases in the last month, then the customer initial key has a validity date before the current month M).
   - Churn risk, if the customer has no order in the last four months after being Constant at least once in the last 12 months. To determine whether the typology is Churn risk, two sets of values for the InitialCustomerKey are computed: the first is the customers of the table Classification that have been Constant at least once in the last 12 months; the second is the customers who have made orders in the past four months. A customer in the first group but not in the second is classified as Churn risk.
   - Inactive, if the customer has no order in the last four months and has not been Constant in the last 12 months. To determine whether the typology is Inactive, two sets of values for the InitialCustomerKey are computed: the first is the customers of the table Classification that have been Constant zero times in the last 12 months; the second is the customers who have made orders in the past four months. A customer in both groups is classified as Inactive.
2. The value of the FKTipologyAge in the fact table Sales is updated in all the customer purchases made from the beginning of the month.

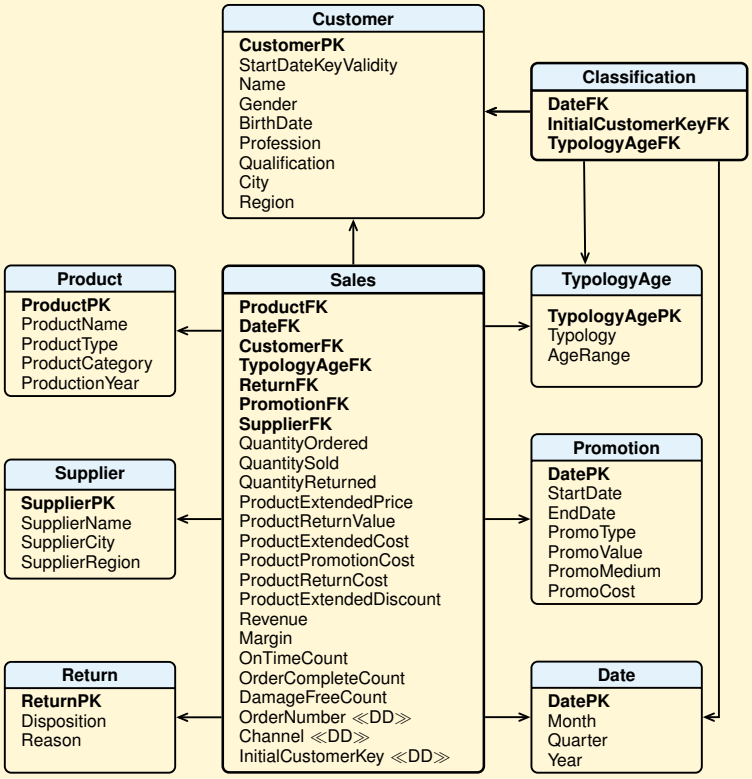


Figure 4.16: Final version of the logical design

4.8 Summary

- *Customer Relationship Management* (CRM) is a business strategy that aims at the creation and consolidation of profitable relationships with specific customers by offering superior value and satisfaction.
- CRM is grounded on high quality customer-related data, and is enabled by *information technology*, in particular *Business Intelligence* techniques to get information about customer purchase behavior and characteristics, customer buying preferences, and customer profitability, in order to develop strategies to retain the top high-value customers, and to expand relationships with all customers.
- *Analytical CRM* is considered an essential component of the CRM framework, and the design of a data warehouse is one of the key factors in successfully implementing *Analytical CRM*.
- Examples of typical *Analytical CRM* business questions have been considered to show their impact on the design of a data warehouse starter model to support them (customer profitability, customer retention, customer attrition, product profitability, returns analysis, order delivery performance). Data analysis allows to retrieve data that fulfill certain criteria to use then to discovery useful models of data with *Data Mining* algorithms. The models are high-level, actionable summaries of data (decision rules, clustering, association rules, etc.) that describes large set of data in understandable way, and it would be difficult to discover the same information off a data analysis result.
- Finally, as usually happens, it has been shown how the process of data warehousing is not a trivial task.

Part II

Multidimensional Analysis

Chapter 5

DATA ANALYSIS

Once the data warehouse has been implemented, the final step is *data analysis*, that is to identify and to develop a suite of reports showing how the information provided in these reports can be used by decision makers to improve the business process modeled. Data analysis is usually done interactively with tools that provide graphical interfaces to make the requests, which are then translated automatically into SQL queries on the data warehouse. To facilitate the implementation of complex analysis, the SQL language has been extended with new operators to group and aggregate data using several analytic functions. Some of them will be presented with examples to show how to express in SQL basic OLAP operations on multidimensional data.

5.1 OLAP Systems Solutions

When talking about systems for data analysis, terms are used such as OLAP, ROLAP, MOLAP, HOLAP, DOLAP, OLAP Server, OLAP Services, etc., which can create problems of interpretation for the way they are used by vendors of these types of products.

The term OLAP is used to refer to the activity of multidimensional analysis of large amounts of data, with interactive and intuitive ways of changing the perspectives of analysis and moving to different levels of synthesis of the detailed data.

An *OLAP client* provides graphical environments where the business users can click on actions and perform drag-and-drop operations to provide input to summarize data. More experienced users can also create complex queries with languages such SQL or MDX. An *OLAP client* interacts with the data manager using one of the following solutions (Figure 5.1):

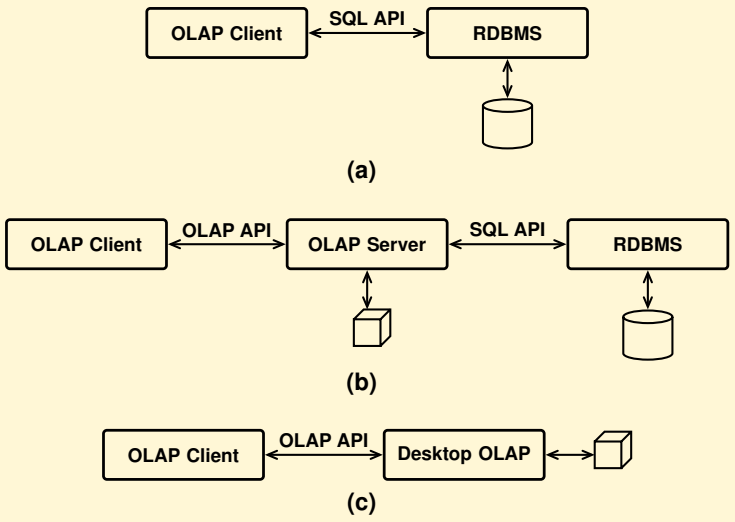


Figure 5.1: OLAP Systems Solutions

(a) The data warehouse is stored in a relational database system RDBMS (*Data server*) and the interactions with the *OLAP client* occur in SQL. The benefit of this solution is that it uses a standard technology usually already available. In the past the approach was not considered satisfactory for the performance of RDBMS as systems for data warehouses and limitations of the SQL as an OLAP language. But now the main producers of RDBMS systems have made them more and more specialized for OLAP applications (*OLAP-Aware RDBMS*), aware of managing data warehouses with special relational schemas (star, snowflake or constellation), dimensional hierarchies, specific storage structures, and materialized views (also called *aggregates*, *MQT* (*materialized query tables*), or *summary data*).

(b) An *OLAP client* interacts with an *OLAP server*, a system that provides a multidimensional cube vision of a data mart, which can be analyzed with the typical operations slice, dice, drill down, roll up, pivot, etc. An *OLAP server* can be one of the following types:

- **MOLAP** (Multidimensional-OLAP), which stores in the local memory both the data cube, taken from a *Data server*, and the aggregates of the extended cube (materialized views), using a specialized multidimensional arrays structure. A MOLAP server does not support SQL, but proprietary languages not for business users, the most popular being MDX from Microsoft. The solution provides excellent performance, but is not suitable for large amounts of data. Examples of products are Hyperion Essbase, Microsoft Analysis Services, Cognos PowerPlay and DB2 OLAP, using Hyperion Essbase technology.
- **ROLAP** (Relational-OLAP), which stores both the data and the materialized views in the relational *Data server*. ROLAP servers may also need to implement functionality not supported in the SQL of the *Data server*, for example, analytic functions. Examples of products are Informatica, MicroStrategy, Microsoft Analysis Services and SAP BW.
- **HOLAP** (Hybrid-OLAP), which combines ROLAP and MOLAP by splitting storage data in a MOLAP and a relational *Data server*. Splitting the data can be done in different ways. One method is to store the detailed data in the *Data server*, and precomputing aggregated data in MOLAP. Another method is to store more recent aggregate data in MOLAP to provide faster access, and older aggregates in the *Data server*. Microsoft Analysis Services is an example of product that can operate as MOLAP, ROLAP or HOLAP.

Data update is usually done by batch, at predetermined time intervals. There are also systems capable of doing *proactive caching*, updating MOLAP data incrementally at time intervals or after each transaction on an operational database. This permits the use of OLAP in real-time, or near real-time, useful in certain contexts such as, for example, the stock market.

The requests of the *OLAP client* to the *OLAP server* are formulated in SQL or proprietary languages such as MDX of SQL Server Analysis Services and OLAP DML of Oracle. The results are communicated to the client in proprietary formats or in XML, for example using to the standard XMLA.

(c) The *OLAP client* interacts with a local **DOLAP** system (*Desktop OLAP*), which manages small amounts of data extracted from the *OLAP server*, the *Data server* or an operational DBMS. The fact that a subset of a data cube is transferred on a user's machine makes it a good choice for those who travel and move extensively, such as sales people, by using portable computers, or who do not regularly perform such complex queries that a faster server is preferred to the speed of the client. The main product of this type is Business Objects.

Among the DOLAP systems there are those specialized for interactive multidimensional analysis, with some limitations as regards the functionalities of the *OLAP server*. For example, Business Objects and MicroStrategy allows the definition of interactive reports with operations such as drill down and roll up. The system does not maintain aggregates in the local memory, but only the results of recent operations. The aggregates are calculated by the *OLAP server*, by the *Data server* or by the operational DBMS.

In all the solutions, the *metadata*, with information on the structure of the fact table, dimensions and hierarchies, are created and maintained by the *OLAP or Data server* and are imported from the *OLAP*

client using the standard *CWM* (*Common Warehouse Metamodel*) or proprietary formats.

Finally, in all the various solutions, the systems are supported by ETL (*Extract, Transform, Load*) tools to load data from operational databases and other external sources.

5.2 Data Analysis Using SQL

In the following, several examples are presented to show how to write SQL queries to produce reports for commonly asked business questions. The examples are based on the following table with attributes without null values.

Sales(Customer, Product, Brand, Date, City, Region, Area, Quantity, Revenue, Margin)

Only in some cases will a graphical representation of the result also be shown, but this is, in general, *essential* to make the results understandable and useful to those who need information for decision support, the main motivation of the multidimensional analysis. Sometimes patterns can be seen in visual data that cannot be seen in numerical data. All reporting tools allow us to perform both an analysis of data without writing the query in SQL, and to produce a graphical representation of the result. Some DBMS, such as Oracle, can produce a graphical representation of the result with analysis expressed in SQL too.

Some of the more commonly used business reports follow.

Simple Reports.

Many kinds of commonly requested business reports can be readily expressed as SQL queries.

- What were the total revenue and margin (in value terms and as a percentage of the revenue) of sales for the month of January 2009, by brand and by product?
- What were the total revenue and margin (in value terms and as a percentage of the revenue) of sales for the month of January 2009, by brand and by product, and the brand subtotals for all products?

Moderately Difficult Reports.

However, many other commonly requested reports cannot be expressed so easily. Reports that require comparisons often challenge both the query writers and SQL itself.

- Revenues for 2009 by brand and product, with the percentage change from the previous year?
- How did product revenues in 2009 compare by geographic area, in a readable spreadsheet, or "cross-tab", format?
- Which suppliers charge the most for bulk tea products?
- What was the most successful promotion last December in Rome?

Very Difficult Reports Without Analytic SQL.

Reports that require sequential processing are very difficult to express as SQL queries, for example deriving a simple running total. Data analysts typically run several queries with a program, then paste the results together. This approach is awkward because it requires a sophisticated user.

The standard SQL analytic functions provide a better solution because they are easy to use, and perform a broad range of calculations that execute quickly on the server.

- What were the cumulative totals (*running totals*) for Best coffee sales during each month of last year?
- What were the ratios of monthly sales to total sales (expressed as percentages) for Best coffee during the same period?
- Which ten cities had the worst coffee sales in 2010 with regard to dollar sales and quantities sold?
- Which supermarket falls into the top 25% in terms of sales revenue for the first quarter of 2010?
- What products fell into the top 20%, middle 60%, and bottom 20% of sales margins totals for the second week of 2011, at stores in the Center area?

5.3 Simple Reports with SQL

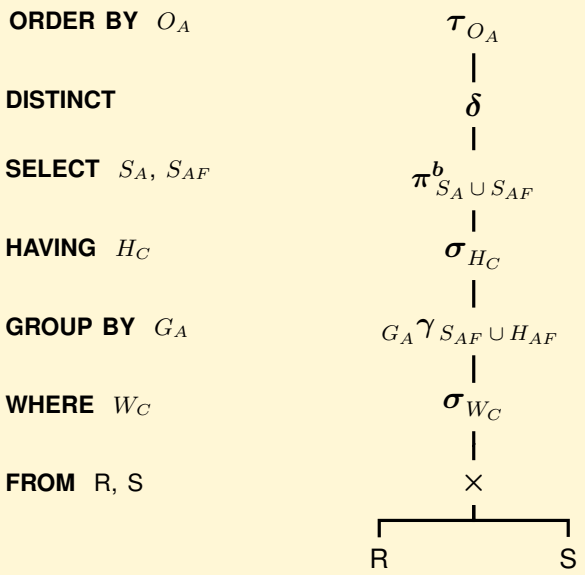
A simple kind of query involves *grouping* and *aggregation* of the data.

To write such kind of **SELECT** the **GROUP BY** clause must be used with the following version of the command syntax.

SELECT **DISTINCT** S_A, S_{AF}
FROM T
WHERE W_C
GROUP BY G_A
HAVING H_C
ORDER BY O_A;

where (a) S_A are the **SELECT** attributes and S_{AF} are the **SELECT** aggregation functions; (b) T are the **FROM** tables; (c) W_C is the **WHERE** condition; (d) G_A are the grouping attributes, with $S_A \subseteq G_A$; (e) H_C is the **HAVING** condition with aggregation functions H_{AF}; (f) O_A are the **ORDER BY** attributes; (g) the **DISTINCT**, **WHERE**, **HAVING** and **ORDER BY** clauses are optional.

The command semantics with tables R and S, and all the optional clauses specified, in terms of the extended relational algebra is shown in Figure 5.2.



(a) *SQL query* **(b)** *Logical query plan*

Figure 5.2: SQL query with **GROUP BY** semantics

Figure 5.3 shows an example of the analysis report for "total revenue and margin (in value terms and as a percentage of the revenue) of sales for the year 2009, by brand and by product."

Margin by Brand and by Product Year 2009				
Brand	**Product**	**Revenue (€)**	**Margin (€)**	**Margin% (%)**
B1	P1	2 100	273	13
	P2	3 720	624	17
	P3	15 300	1 803	12
B2	P4	12 600	756	6
	P5	22 500	2 196	10
	P6	48 300	4 496	9

Figure 5.3: A Simple Report

The following query with a **GROUP BY** produces the desired result:[1]

```
SELECT    Brand, Product, SUM(Revenue) AS Revenue,
          SUM(Margin) AS Margin,
          ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Brand, Product
ORDER BY  Brand, Product;
```

Note that the operations *Slice* and *Dice* are expressed by a selection and projection, while *Roll-up* and *Drill-down* require a **GROUP BY**. For example, a *roll-up* on Brand, to find, for the year 2009, the total revenue, total margin and margin percentage by Brand, is expressed with the following query:

```
SELECT    Brand, SUM(Revenue) AS Revenue,
          SUM(Margin) AS Margin,
          ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Brand
ORDER BY  Brand;
```

while a *drill-down* on Customer, on the previous result, to find, for the year 2009, the total revenue, total margin and margin percentage by Brand and Customer, is expressed with the following query:

```
SELECT    Brand, Customer, SUM(Revenue) AS Revenue,
          SUM(Margin) AS Margin,
          ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Brand, Customer
ORDER BY  Brand, Customer;
```

In general, adding an attribute to the **GROUP BY** and **SELECT**, a *drill-down* is made, while dropping an attribute, a *roll-up* is made.

1. In all the examples in this chapter, the SQL queries produce the data needed for reports, but not their graphic representation. The YEAR, QUARTER, MONTH functions retrieve subfields from DATE values.

5.3.1 The Operator *ROLLUP*

Many OLAP queries use a **GROUP BY** to partition data into groups that are reduced to a single row of aggregates and grouping columns. However, standard SQL limit the types of OLAP queries that can be easily expressed. One extension is the **ROLLUP** clause. Suppose that we want to obtain the report of Figure 5.4. Any report that contains a metric is likely to contain a "total" at the end. If the report has more than one dimensional attribute, the metric may also be subtotaled.

Margin by Brand and by Product
Year 2009

Brand	Product	Revenue (€)	Margin (€)	Margin% (%)
B1	P1	2 100	273	13
	P2	3 720	624	17
	P3	15 300	1 803	12
B1	**Total**	**21 120**	**2 700**	**13**
B2	P4	12 600	756	6
	P5	22 500	2 196	10
	P6	48 300	4 496	9
B2	**Total**	**83 400**	**7 448**	**9**
Total		**104 520**	**10 148**	**10**

Figure 5.4: A report with some subtotals

A possible solution with standard SQL would be:

```
SELECT      Brand, Product, SUM(Revenue) AS Revenue,
            SUM(Margin) AS Margin,
            ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM        Sales
WHERE       YEAR(Date) = 2009
GROUP BY    Brand, Product

UNION ALL

SELECT      Brand, NULL AS Product, SUM(Revenue) AS Revenue,
            SUM(Margin) AS Margin,
            ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM        Sales
WHERE       YEAR(Date) = 2009
GROUP BY    Brand

UNION ALL

SELECT      NULL AS Brand, NULL AS Product, SUM(Revenue) AS Revenue,
            SUM(Margin) AS Margin,
            ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM        Sales
WHERE       YEAR(Date) = 2009
ORDER BY    Brand, Product;
```

Three statements are required because the report requires three aggregations applied to groups of values produced by a different **GROUP BY** clause. Computing all of these queries independently is time consuming, and this is the main motivation for the **ROLLUP** clause which is included in the SQL of several

DBMSs:

```
SELECT    Brand, Product, SUM(Revenue) AS Revenue,
          SUM(Margin) AS Margin,
          ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  ROLLUP (Brand, Product)
ORDER BY  Brand, Product;
```

Definition 5.1 *ROLLUP in SQL:1999*

A **ROLLUP** group is an extension of the **GROUP BY** clause that produces a result that contains subtotal records in addition to the regular grouped records, whose aggregate values are derived by applying the same functions. A **ROLLUP**$(A_1, A_2, \ldots, A_{n-1}, A_n)$ group is equivalent to the union of the $n+1$ grouping results on the attributes $(A_1, A_2, \ldots, A_{n-1}, A_n)$, $(A_1, A_2, \ldots, A_{n-1})$, $\ldots$, (A_1, A_2), (A_1), and $()$. Notice that each grouping result is created by eliminating an attribute from the list specified in the **ROLLUP** clause, by moving from right to left. Therefore, the order in which the attributes are specified is significant for the **ROLLUP** result. The operator produces its results with just one table access.

For example, the rows of the table in Figure 5.4 are calculated first grouping on Brand, Product, and then the subtotals are calculated progressively moving from right to left through the list of grouping columns: first grouping on Brand, and then on () (super-aggregate rows).

5.3.2 The Operator *CUBE*

Suppose now that we want to obtain a table such as that in Figure 5.5, similar to the table in Figure 5.4 except that, in addition, it has totals for each row and each column.

Margin by Brand and by Product
Year 2009

Brand	Product	Revenue (€)	Margin (€)	Margin% (%)
B1	P1	2 100	273	13
	P2	3 720	624	17
	P3	15 300	1 803	12
Total B1		**21 120**	**2 700**	**13**
B2	P4	12 600	756	6
	P5	22 500	2 196	10
	P6	48 300	4 496	9
Total B2		**83 400**	**7 448**	**9**
	Total P1	**2 100**	**273**	**13**
	Total P2	**3 720**	**624**	**17**
	Total P3	**15 300**	**1 803**	**12**
	Total P4	**12 600**	**756**	**6**
	Total P5	**22 500**	**2 196**	**10**
	Total P6	**48 300**	**4 496**	**9**
Total		**104 520**	**10 148**	**10**

Figure 5.5: Report with subtotals

Again, computing all of these queries independently is time consuming, and this is the main motivation for the **CUBE** clause which is included in the SQL of several DBMSs:

```
SELECT    Brand, Product, SUM(Revenue) AS Revenue,
          SUM(Margin) AS Margin,
          ROUND(100*SUM(Margin)/SUM(Revenue)) AS Margin%
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  CUBE (Brand, Product)
ORDER BY  Brand, Product;
```

Definition 5.2 *CUBE in SQL:1999*

A CUBE group is an extension of the **GROUP BY** clause that produces a result that contains subtotal records in addition to the regular grouped records, whose aggregate values are derived by applying the same functions. A **CUBE** $(A_1, A_2, \ldots, A_{n-1}, A_n)$ group is equivalent to the union of the 2^n grouping results on the attributes of all possible subsets of the attributes specified in the **CUBE** clause. Unlike **ROLLUP**, the order in which the attributes are specified doesn't matter for **CUBE**. The operator produces its results with just one table access.

Some systems also provide the operator **GROUPING SETS** to group only for certain combinations of attributes. For example, replacing in the previous query **GROUP BY CUBE**(Brand, Product) with **GROUP BY GROUPING SETS**((Brand, Product), (Brand)) data are grouped only for the two combinations listed.

5.3.3 Observations

In general, in the **GROUP BY** clause both attributes and different **ROLLUP** and **CUBE** can be used. For example, the following query

```
SELECT    Date, Brand, Product, SUM(Revenue) AS Revenue
FROM      Sales
GROUP BY  Date, ROLLUP(Brand, Product);
```

generates the following groupings: (Date, Brand, Product), (Date, Brand) and (Date), but not (). The result is justified by recalling that Date generates the set of groupings $\{$(Date)$\}$, ROLLUP generates the set of groupings $\{$(Brand, Product), (Brand), ()$\}$ and their combination generates the cartesian product of two sets. The following query

```
SELECT    Date, Brand, Product, SUM(Revenue) AS Revenue
FROM      Sales
GROUP BY  CUBE(Date), ROLLUP(Brand, Product);
```

generates the following groupings instead: (Date, Brand, Product), (Date, Brand), (Date), (Brand, Product), (Brand) and ().

Usually, in relational systems when using operators **ROLLUP** and **CUBE**, the result shows the value NULL to indicate a running total, creating ambiguity because the value might be present in the data and not a result of the operators **ROLLUP** and **CUBE**. To correctly interpret the meaning of a record the function **GROUPING** is used with an attribute parameter in the **GROUP BY**: the function returns 1 if the value NULL has been created by **ROLLUP** or **CUBE**, and returns 0 otherwise. For example, the result of the query

```
SELECT    Brand, Product, SUM(Revenue) AS Revenue,
          GROUPING(Brand), GROUPING(Product)
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  ROLLUP(Brand, Product);
```

produces the result of the query without **GROUPING**, extended with two more columns that have the value 1 when the record has a field NULL, which corresponds to a total, as shown in Figure 5.6.

Brand	Product	Revenue	GROUPING (Brand)	GROUPING (Product)
B1	P1	2 100	0	0
B1	P2	3 720	0	0
B1	P3	15 300	0	0
B1		21 120	0	1
B2	P4	12 600	0	0
B2	P5	22 500	0	0
B2	P6	48 300	0	0
B2		83 400	0	1
		104 520	1	1

Figure 5.6: Report with **ROLLUP** and **GROUPING**

To get the result without additional columns, but with the value Total when necessary, we write:

```
SELECT    CASE WHEN GROUPING(Brand) = 1 THEN 'Total' ELSE Brand
          END AS Brand,
          CASE WHEN GROUPING(Product) = 1 THEN 'Total' ELSE Product
          END AS Product,
          SUM(Revenue) AS Revenue
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  ROLLUP(Brand, Product);
```

where, with the first **CASE**, if the value of Brand is a NULL, then the string Total will appear (any string can be chosen). Otherwise, its actual value will appear, as shown in Figure 5.7.

Brand	Product	Revenue
B1	P1	2 100
B1	P2	3 720
B1	P3	15 300
B1	**Total**	**21 120**
B2	P4	12 600
B2	P5	22 500
B2	P6	48 300
B2	**Total**	**83 400**
Total	**Total**	**104 520**

Figure 5.7: Displaying the ALL values with Total

The function **GROUPING**, like any other aggregate function, can be used in **HAVING** to select only some of the records produced by **ROLLUP** or **CUBE**. For example, the following query finds only the record with totals:

```
SELECT    CASE WHEN GROUPING(Brand) = 1 THEN 'Total' ELSE Brand
          END AS Brand,
          CASE WHEN GROUPING(Product) = 1 THEN 'Total' ELSE Product
          END AS Product,
          SUM(Revenue) AS Revenue
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  ROLLUP (Brand, Product)
HAVING    GROUPING(Brand) = 1 OR GROUPING(Product) = 1;
```

5.4 Moderately Difficult Reports with SQL

Let us show examples of queries to present results in a spreadsheet-type cross-tab format, rather than the form of lists of values, or to produce reports with metrics to be calculated by comparison with others.

Example 5.1
Let us produce a report with the total revenue in 2009 by product and by geographical areas. The following simple SQL query

```
SELECT    Product, Area, SUM(Revenue) AS TotalRevenue
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Product, Area
ORDER BY  Product, Area;
```

produces a vertically ordered result set that makes it difficult to compare the revenues by product and by geographical area.

Total Revenue by Product and by Geographical area Year 2009

Product	Area	Revenue
P1	Center	600
P1	Islands	300
P1	North	900
P1	South	300
P2	Center	1 200
P2	Islands	360
P2	North	1 800
P2	South	360
P3	Center	4 680
P3	Islands	1 980
P3	North	7 020
P3	South	1 620
P4	Center	3 600
P4	Islands	1 800
P4	North	5 400
P4	South	1 800
P5	Center	6 300
P5	Islands	3 150
P5	North	9 450
P5	South	3 600
P6	Center	15 000
P6	Islands	5 100
P6	North	22 500
P6	South	5 700

This kind of data is much easier to compare when it is formatted like a spreadsheet-type cross-tab or *pivot table*:

Comparison between Revenues by Product and by Area Year 2009				
Product	**North**	**Center**	**South**	**Islands**
P1	900	600	300	300
P2	1 800	1 200	360	360
P3	7 020	4 680	1 620	1 980
P4	5 400	3 600	1 800	1 800
P5	9 450	6 300	3 600	3 150
P6	22 500	15 000	5 700	5 100

The result is obtained by grouping the data by Product and by using in the **SELECT** the aggregate function SUM with argument a **CASE** expression:

```
SELECT    Product,
          SUM(CASE
            WHEN Area = 'North' THEN Revenue ELSE 0 END) AS North,
          SUM(CASE
            WHEN Area = 'Center' THEN Revenue ELSE 0 END) AS Center,
          SUM(CASE
            WHEN Area = 'South' THEN Revenue ELSE 0 END) AS South,
          SUM(CASE
            WHEN Area = 'Islands' THEN Revenue ELSE 0 END) AS Islands
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY Product
ORDER BY Product;
```

DBMSs such as Oracle 11g and SQL Server 2005 provide an extension to SQL to create the cross-tab with a **PIVOT** clause.

Figure 5.8 shows a graphical representation of product revenues by geographic area with multiple groups of stacked bars, while Figure 5.9 shows another graphical representation often used to show the percentage revenue mix of product by geographic area, but the SQL query to produce the revenue percentage is more complex, and we will see later how to write it.

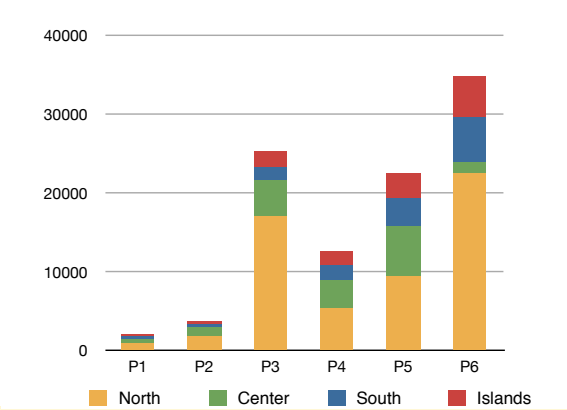


Figure 5.8: An example of a stacked bar report

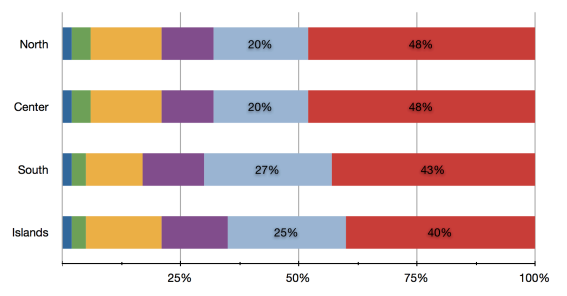


Figure 5.9: Another example of a stacked bar report

Example 5.2
Another very common type of analysis requires reports that compare data columns that refer to different periods (*variance report*). For example, "Revenues for 2009 by brand and by product, with the percentage change from the previous year ($Delta = 100 \times ((\text{Revenue}_{2009} - \text{Revenue}_{2008})/\text{Revenue}_{2009})$".

Comparison between Revenue by Brand and by Product 2009 – 2008

Brand	Product	Revenue (€) 2009	Revenue (€) 2008	Delta (%)
B1	P1	2 100	13 560	−546
	P2	3 720	23 640	−535
	P3	15 300	20 340	−33
B2	P4	12 600	1 440	89
	P5	22 500	2 100	91
	P6	48 300		100

The annual revenues for 2009 and for 2008, by brand and by product, are obtained with the following SQL queries:

```
Revenue09 = SELECT   Brand, Product, SUM(Revenue) AS Revenue2009
            FROM     Sales
            WHERE    YEAR(Date) = 2009
            GROUP BY Brand, Product ;

Revenue08 = SELECT   Brand, Product, SUM(Revenue) AS Revenue2008
            FROM     Sales
            WHERE    YEAR(Date) = 2008
            GROUP BY Brand, Product ;
```

If the same products were sold in both 2009 and 2008, the final result would be obtained with a *natural join* of Revenue09 and Revenue08.

Instead, to take into account that not necessarily the same products were sold in both 2009 and 2008, the analysis in SQL requires a *full join* of Revenue09 and Revenue08.

```
SELECT    Revenue09.Brand AS Brand, Revenue09.Product AS Product
          , Revenue2009
          , Revenue2008
          , CASE
            WHEN Revenue2009 IS NULL THEN −100
            WHEN Revenue2008 IS NULL THEN 100
            ELSE ROUND(100∗(Revenue2009 − Revenue2008) / Revenue2009)
          END AS Delta
FROM      ( SELECT    Brand, Product, SUM(Revenue) AS Revenue2009
            FROM      Sales
            WHERE     YEAR(Date) = 2009
            GROUP BY  Brand, Product
          ) AS Revenue09

          FULL JOIN

          ( SELECT    Brand, Product, SUM(Revenue) AS Revenue2008
            FROM      Sales
            WHERE     YEAR(Date) = 2008
            GROUP BY  Brand, Product
          ) AS Revenue08

          USING (Brand, Product)
ORDER BY  Brand, Product;
```

In Figure 5.10 there is shown a graphical representation of the result with a histogram, useful for comparing metrics.

Another very useful graph is the comparison of revenues for the months of different years (Figure 5.11), calculated with an SQL query similar to the previous one.

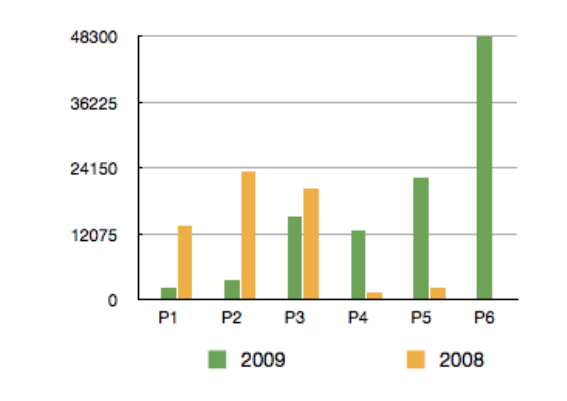


Figure 5.10: The histogram of the revenues variation by product

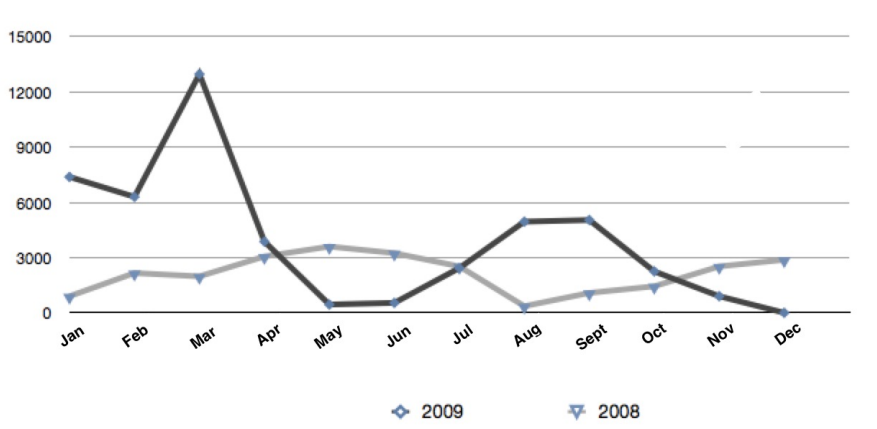


Figure 5.11: The trend in revenues per month of different years

5.4.1 The WITH Clause in SQL

The above example shows how SQL queries for complex analysis generally require the use of subqueries in the **FROM** clause, usually handled by relational systems as a temporary view definition. To make it easier to understand these types of queries, it is useful to write them using the **WITH** clause to break down complex queries with subqueries into simpler parts.

For example, the previous query is written as follows.

```
WITH    Revenue09 AS
        ( SELECT    Brand, Product, SUM(Revenue) AS Revenue2009
          FROM      Sales
          WHERE     YEAR(Date) = 2009
          GROUP BY  Brand, Product
        )
        , Revenue08 AS
        ( SELECT    Brand, Product, SUM(Revenue) AS Revenue2008
          FROM      Sales
          WHERE     YEAR(Date) = 2008
          GROUP BY  Brand, Product
        )

SELECT  Revenue09.Brand AS Brand, Revenue09.Product AS Product
        , Revenue2009
        , Revenue2008
        , CASE
          WHEN Revenue2009 IS NULL THEN −100
          WHEN Revenue2008 IS NULL THEN 100
          ELSE ROUND(100∗(Revenue2009 − Revenue2008) / Revenue2009)
        END AS Delta
FROM    Revenue09 FULL JOIN Revenue08 USING (Brand, Product)
ORDER BY Brand, Product;
```

A temporary view defined with **WITH** can use one of the previous defined views or be recursive, if defined with **WITH RECURSIVE**. Recursive queries are typically used to deal with hierarchical or tree-structured fact tables. For example, suppose we have the relation

Flights

Code	From	To
10	MI	PI
11	MI	TO
12	MI	RM
13	PI	FI
14	TO	RM
15	RM	VE
16	NA	BA
17	TO	PA

The result of the query

```
WITH RECURSIVE
        CitiesReachableFrom AS
        ( SELECT    From, To
          FROM      Flights
                UNION
          SELECT    Flights.From AS From, CitiesReachableFrom.To AS To
          FROM      Flights, CitiesReachableFrom
          WHERE     Flights.To = CitiesReachableFrom.From
        )

SELECT  *
FROM    CitiesReachableFrom
WHERE   From = 'MI';
```

is a relation with pairs of cities reachable from Milano by taking one or more flights.

From	To
MI	PI
MI	TO
MI	RM
MI	FI
MI	VE
MI	PA

The definition of the SQL temporary recursive view CitiesReachableFrom has the structure *BasisSelect* **UNION** *RecursiveSelect*, with a *linear recursion*, that is the **FROM** of the *RecursiveSelect* contains one occurrence only of the temporary recursive view. The *FinalSelect* of the **WITH** clause is executed with the CitiesReachableFrom relation value that might be computed as follows:

```
1)  CitiesReachableFrom := BasisSelect;
2)  WHILE ( changes to CitiesReachableFrom) DO
3)      CitiesReachableFrom := CitiesReachableFrom  UNION RecursiveSelect;
```

CitiesReachableFrom is initially empty and with rule (1) its records become those of the *BasisSelect*, the cities reachable with direct flights, because the *RecursiveSelect* produces an empty relation.

On the next iteration, with rule (3) possible, other cities reachable with more flights are added to the relation *RecursiveSelect*. If the new value is equal to the previous one, the final result has been obtained and the loop ends. Otherwise, the value obtained is used to find other cities reachable.

The *FinalSelect* is executed with the final value of CitiesReachableFrom and the result is the subset of records with From = 'MI'.

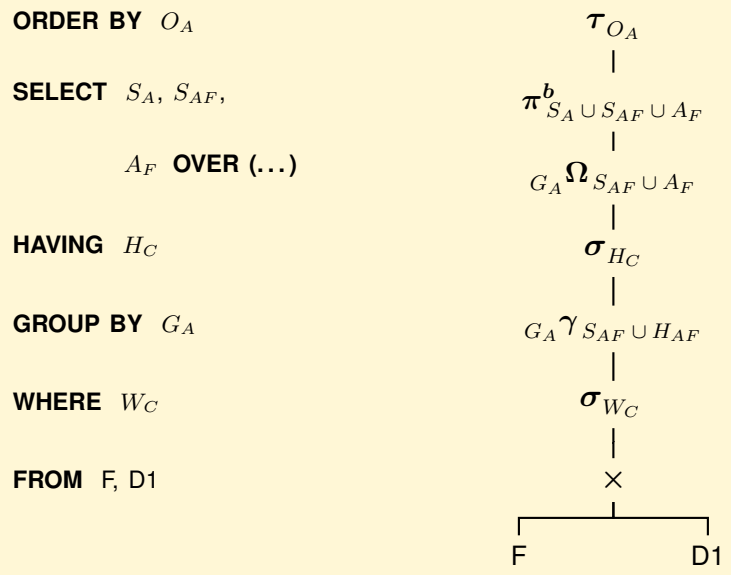
5.5 Very Difficult Reports Without Analytic SQL

The SQL has been extended to allow the use of several analytic functions, also known as *Windows Functions*, to use them in new ways in order to facilitate the development of complex data analysis.

A **SELECT**, for simplicity with a single analytic function, without **DISTINCT**, with a fact table and one dimensional table only, has the following structure:

SELECT Select Attributes (S_A), Select Aggregation Functions (S_{AF}),
Analytic Function (A_F) **OVER(**
[**PARTITION BY** <attribute list>]
[**ORDER BY** <sort attribute list>
[<window clause>]]**)**
FROM Fact table (F) and a dimension table (D1)
WHERE Where condition (W_C)
GROUP BY Grouping Attributes (G_A)
HAVING Having condition (H_C) with aggregation functions (H_{AF})
ORDER BY Sorting attributes (O_A);

The command semantics in terms of the extended relational algebra is shown in Figure 5.12.



(a) *SQL query* **(b)** *Logical query plan*

Figure 5.12: Analytic SQL query semantics

The query is processed in the following steps:

1. Perform the operations specified in **FROM**, **WHERE**, **GROUP BY** and **HAVING** clauses to compute the set of records resulting from the subtree rooted at σ_{H_C}.
2. Apply the specified analytic functions to the result of the subtree rooted at σ_{H_C}, to produce a new set of record that differs from the previous ones only for new attributes calculated using the analytic functions.
3. Apply any projection, and then the **ORDER BY** to produce the query result.

The analytic functions can be used only in **SELECT**, with the **OVER** clause, and usually they are applied to the entire set of records produced in the first phase, but may also be applied separately to disjoint subsets obtained by partitioning the records by the value of an expression defined on the attributes of the record

(option **PARTITION BY**). The aggregate functions can also be applied to non-disjoint subsets of records in a partition defined by the notion of *moving window*: for each record r of a partition, aggregate functions apply to the data identified by a "window" placed on r. They are useful for analysis of data, such as: "What is the moving average of weekly sales?".

The result of the traditional aggregate functions SUM, COUNT, AVG, MIN, MAX does not depend on the order of records in the collection on which they operate. Instead, the result of the new analytic functions, which we will see that later on, may depend on the order of the data specified with the **ORDER BY** clause in the **OVER** of the **SELECT**.

The partitioning operation is like that for the calculation of a **GROUP BY**, but **PARTITION BY** does not produce a record for each group as with the **GROUP BY**, but rather produces as many records as there are elements of the group, which will then be extended with new attributes calculated using the analytic functions. When the **PARTITION BY** clause is not present, the set of records behaves as a single group.

Figure 5.13 shows the main analytic functions available in some DBMS.

Function	Oracle	DB2	SQL Server	PostgreSQL	MySQL
COVAR_POP	x	x	x	x	
CUBE	x	x	x		
CUME_DIST	x		x	x	
DENSE_RANK	x	x	x	x	
LAG, LEAD	x	x	x	x	
NTILE	x	x	x	x	x
PERCENT_RANK	x		x		
RANK	x	x	x	x	
RATIO_TO_REPORT	x	x			
REGR_Functions	x	x		x	
ROLLUP	x	x	x		x
ROW_NUMBER	x	x	x	x	
STDDEV_POP	x	x	x	x	x
VAR_POP	x	x	x	x	x
Window_Clause	x	x	x		

Figure 5.13: Analytic SQL in some DBMS

5.5.1 Premise

Note the difference between the result of a query with an aggregate function, and the traditional **GROUP BY**, and an analytic function.

Let us consider the relation

R

P	...
P1	...
P1	...
P2	...
P2	...
P2	...
P2	...
P2	...

The query

```
SELECT    P, COUNT(*) AS No
FROM      R
GROUP BY  P;
```

returns the relation

P	No
P1	2
P2	5

while the query

```
SELECT   P,
         COUNT(*) OVER (PARTITION BY P) AS No
FROM     R
ORDER BY P;
```

returns the relation

P	No
P1	2
P1	2
P2	5
P2	5
P2	5
P2	5
P2	5

While the **GROUP BY** groups a set of records and a record for each group with two attributes is obtained – with the value of the grouping attribute and the value of the aggregate function COUNT – with the analytic function, for each record of the set the value of the aggregate function COUNT is computed when applied to subsets obtained with the **PARTITION**.

If the query was without **PARTITION**, the analytic function would be applied to the whole record set:

```
SELECT   P, COUNT(*) OVER( ) AS No
FROM     R
ORDER BY P;
```

to get the table

P	No
P1	7
P1	7
P2	7
P2	7
P2	7
P2	7
P2	7

Compare this result with that of

```
SELECT    COUNT(*) AS No
FROM      R
```

No
7

The next example shows the usefulness of the **OVER** clause to solve a nontrivial problem left unresolved in a previous example.

Example 5.3
Let us reconsider the example of the report with the total sales revenue in 2009, by product and by geographical area:

Total revenue by Product and by Geographical area Year 2009

Product	North	Center	South	Islands
P1	900	600	300	300
P2	1 800	1 200	360	360
P3	7 020	4 680	1 620	1 980
P4	5 400	3 600	1 800	1 800
P5	9 450	6 300	3 600	3 150
P6	22 500	15 000	5 700	5 100

to change it by replacing the total revenue by product and by geographical area, with the revenue percentage of total revenue for the area.

Percentage of total revenue by area by Product and by Geographical area Year 2009

Product	PctNorth	PctCenter	PctSouth	PctIslands
P1	2	2	2	2
P2	4	4	3	3
P3	15	15	12	16
P4	11	11	13	14
P5	20	20	27	25
P6	48	48	43	40

The result is obtained with the query

```
SELECT    Product
          , ROUND(100*SUM(CASE
                          WHEN Area = 'North'
                          THEN Revenue ELSE 0 END)
                  / SUM(SUM(CASE
                          WHEN Area = 'North'
                          THEN Revenue ELSE 0 END))
                  OVER ( )
            ) AS PctNorth
          , ROUND(100*SUM(CASE
                          WHEN Area = 'Center'
                          THEN Revenue ELSE 0 END)
                  / SUM(SUM(CASE
                          WHEN Area = 'Center'
                          THEN Revenue ELSE 0 END))
                  OVER ( )
            ) AS PctCenter,
          , ROUND(100*SUM(CASE
                          WHEN Area = 'South'
                          THEN Revenue ELSE 0 END)
                  / SUM(SUM(CASE
                          WHEN Area = 'South'
                          THEN Revenue ELSE 0 END))
                  OVER ( )
            ) AS PctSouth
          , ROUND(100*SUM(CASE
                          WHEN Area = 'Islands'
                          THEN Revenue ELSE 0 END)
                  / SUM(SUM(CASE
                          WHEN Area = 'Islands'
                          THEN Revenue ELSE 0 END))
                  OVER ( )
            ) AS PctIslands
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Product
ORDER BY  Product;
```

GROUP BY with CASE

We have seen how the construct **CASE** is useful in the **SELECT** for certain analyses, and other examples will be seen later. Let us see how it can also be useful in the **GROUP BY**, for grouping the data of a report not on the values of certain attributes, but on values calculated from those of other attributes.

Let us consider the relation

S

P	Prc	...
P1	10	...
P2	20	...
P3	30	...
P4	40	...
P5	50	...
P6	60	...
P7	70	...

Suppose we want a report to display the products classified in 3 categories: *Cheap* ($\text{Prc} \leq 20$), *Medium* ($20 < \text{Prc} \leq 50$) e *Expensive* ($50 < \text{Prc} \leq 100$).

P	Prc	Category
P1	10	Cheap
P2	20	Cheap
P3	30	Medium
P4	40	Medium
P5	50	Medium
P6	60	Expensive
P7	70	Expensive

The result is obtained with the query

```
SELECT    P, Prc
          , CASE
              WHEN Prc <= 20 THEN 'Cheap'
              WHEN Prc > 20 AND Pz <= 50 THEN 'Medium'
              ELSE 'Expensive'
            END AS Category
FROM      S;
```

Now suppose we want a report showing the average price of products by category.

Category	AvgPrice
Cheap	15
Medium	40
Expensive	65

```
WITH      CategoryAndPrice AS
          ( SELECT  CASE
                      WHEN Prc <= 20 THEN 'Cheap'
                      WHEN Prc > 20 AND Prc <= 50 THEN 'Medium'
                      ELSE 'Expensive'
                    END AS Category
                    , Price
            FROM    S )
SELECT    Category, AVG(Price) AS AvgPrice
FROM      CategoryAndPrice
GROUP BY  Category
ORDER BY  AvgPrice;
```

Without the **WITH** the query becomes

```
SELECT    CASE
            WHEN Prc <= 20 THEN 'Cheap'
            WHEN Prc > 20 AND Prc <= 50 THEN 'Medium'
            ELSE 'Expensive'
          END AS Category
          , AVG(Prc) AS AvgPrice
FROM      S
GROUP BY  CASE
            WHEN Prc <= 20 THEN 'Cheap'
            WHEN Prc > 20 AND Prc <= 50 THEN 'Medium'
            ELSE 'Expensive'
          END   ORDER BY AvgPrice;
```

The query must be written with two *syntactically equal* **CASE** expressions, one in the **GROUP BY** and another in the **SELECT**.

Some relational systems, such as PostgreSql, allow the use in the **GROUP BY** of the **CASE** expression label in the **SELECT** to avoid rewriting the expression in the **GROUP BY** and, therefore, the query can be written as

```
SELECT    CASE
             WHEN Prc <= 20 THEN 'Cheap'
             WHEN Prc > 20 AND Prc <= 50 THEN 'Medium'
             ELSE 'Expensive'
          END AS Category
          , AVG(Prc) AS AvgPrice
FROM      S
GROUP BY  Category
ORDER BY  AvgPrice;
```

We shall see later the use of **GROUP BY** with **CASE** in a case of more complex analysis.

5.5.2 Analytic Functions with the Use of Partitions

RANK and DENSE_RANK

These functions are used to sort the records out in a set based on the value of an attribute or of an expression (aggregate function), and to assign to each record its position (*rank*) in the set. The standard record order is ascending, and so the records with rank 1 have the minimum value of the attribute, but the descending order can be specified. The result is sorted by the rank value, unless otherwise specified. A ranking function is specified in the **SELECT** clause with the following syntax:

```
<RankFunction>()
OVER(
      [PARTITION BY <attribute list>]
       ORDER BY <sort attribute list>
     ) [ AS Ide ]
```

RANK and **DENSE_RANK** require an **ORDER BY** clause, because to determine the values rank the data must be ordered. If no partitioning is specified, the entire set of records composes a single partition.

The functions **RANK** and **DENSE_RANK** produce different results when the values to be ranked are not different. The rank of a value a_i is defined as 1 plus the number of values that *strictly* precede a_i. If $k > 1$ values are equal, they are assigned the same value rank p, and the next value has the rank $p+k$. Therefore there will be a gap in the sequential rank numbering. Instead, with **DENSE_RANK** there will be no gaps in the sequential rank numbering, with ties being assigned the same rank. The rank of a value a_i is defined as 1 plus the number of *distinct* values that *strictly* precede a_i. For example, the values in the ascending order $(10, 20, 20, 30, 30, 40)$ have the ranks $(1, 2, 2, 4, 4, 6)$ and the dense ranks $(1, 2, 2, 3, 3, 4)$.

Example 5.4
"Show for the year 2009, and the regions of Tuscany and Lazio, the total revenue by region and product, the rank of the products for total revenue in each region and for total revenue."

Revenues and Ranks in the 2009 by Region and by Product				
Region	**Product**	**Total Revenue**	**Product Rank by Region**	**Product Rank Global**
Lazio	P3	2880	3	4
	P2	960	5	8
	P4	2 700	4	5
	P1	480	6	10
	P5	4 800	2	2
	P6	11 400	1	1
Toscana	P1	120	6	12
	P6	3 600	1	3
	P3	1 800	2	6
	P5	1 500	3	7
	P4	900	4	9
	P2	240	5	11

```
SELECT    Region, Product
          , SUM(Revenue) AS TotalRevenue
          , RANK() OVER (PARTITION BY Region ORDER BY SUM(Revenue) DESC)
                    AS ProductRankByRegion
          , RANK() OVER (ORDER BY SUM(Revenue) DESC)
                    AS ProductRankGlobal
FROM      Sales
WHERE     YEAR(Date) = 2009
          AND Region IN ('Toscana', 'Lazio')
GROUP BY  Region, Product
ORDER BY  Region;
```

Example 5.5

"Show for the year 2009 total revenue, the rank and dense rank of total revenue for clients."

Revenues, Rank and Dense Rank in the 2009 by Customer			
Customer	**Total Revenue**	**Customer Rank**	**Customer Dense Rank**
C15	21 120	1	1
C03	13 650	2	2
C08	11 400	3	3
C09	9 240	4	4
C04	8 640	5	5
C11	7 560	6	6
C02	7 560	6	6
C12	7 200	8	7
C16	4 680	9	8
C14	4 200	10	9
C06	3 150	11	10
C10	2 520	12	11
C01	1 920	13	12
C13	1 680	14	13

```
SELECT    Customer
          , SUM(Revenue) AS TotalRevenue
          , RANK() OVER (ORDER BY SUM(Revenue) DESC)
                AS CustomerRank
          , DENSE_RANK() OVER (ORDER BY SUM(Revenue) DESC)
                AS CustomerDenseRank
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  Customer
ORDER BY  TotalRevenue DESC;
```

Example 5.6

The ranking functions can be used to find only the first N records with the value of higher or lower ranks (**TOP**_N, **BOTTOM**_N): (a) there is the SELECT with the function of rank in the **FROM** clause and then (b) an appropriate selection of the result of the external **SELECT** on the rank values. Let us see an example.

"Show for the year 2009, and the Islands geographic area, the region and the customers with the two highest revenues (TOP 2 customers)."

Top 2 customers in the islands Year 2009

Region	Customer	Total Revenue	Top 2
Sardegna	C15	2 640	1
Sicilia	C11	1 080	1
Sicilia	C04	1 080	1
Sardegna	C03	1 560	2

```
WITH      SalesWithRank AS
          ( SELECT    Region, Customer
                      , SUM(Revenue) AS TotalRevenue
                      , RANK( ) OVER (PARTITION BY Region
                                      ORDER BY SUM(Revenue) DESC )
                          AS Rank
            FROM      Sales
            WHERE     YEAR(Date) = 2009 AND Area = 'Islands'
            GROUP BY  Region, Customer
          )

SELECT    Region, Customer, TotalRevenue, Rank AS Top2
FROM      SalesWithRank
WHERE     Rank <= 2
ORDER BY  Top2;
```

NTILE(n)

A set of sorted records is partitioned into n groups with the same number of records (plus or minus 1) and the group number to which it belongs is assigned to each record.

Example 5.7
Customers are divided into 4 groups on the basis of total revenue, and their rank is calculated in each quartile.

Revenue and rank in quartiles by 4 customer groups

Customer	Total Revenue	Quartile	Rank by Quartile
C03	22 890	1	1
C15	21 120	1	2
C08	16 440	1	3
C04	14 400	1	4
C02	12 600	2	1
C12	11 760	2	2
C09	9 240	2	3
C05	9 240	2	3
C16	9 240	3	1
C14	8 820	3	2
C11	7 560	3	3
C07	7 200	3	4
C01	4 800	4	1
C06	4 410	4	2
C13	3 360	4	3
C10	2 520	4	4

```
WITH    CustomersQuartiles AS
        ( SELECT    Customer, SUM(Revenue) AS TotalRevenue
                  , NTILE(4) OVER (ORDER BY SUM(Revenue) DESC)
                      AS Quartile
          FROM      Sales
          GROUP BY  Customer
        )
SELECT  Customer,TotalRevenue, Quartile
        , RANK( ) OVER (PARTITION BY Quartile ORDER BY TotalRevenue DESC)
            AS RankByQuartile
FROM    CustomersQuartiles;
```

ROW_NUMBER() and CUME_DIST()
On a set of sorted records

- **ROW_NUMBER()** assigns a sequence number to each record and
- **CUME_DIST()** assigns a value between 0 and 1 to each record of a sorted set according to the number of records that precede it. For a record r in a set with n elements sorted in increasing order, if k is the number of records that precede it, the **CUME_DIST()** of r is $0 < (k+1)/n \leq 1$. To equal values **CUME_DIST()** assigns equal values.

Example 5.8
Consider the customers ordered by the sum of all their purchases (sales revenue) in descending order.

1. We want to see if the Pareto rule holds: 80 percent of revenue comes from 20 percent of customers. These customers are important because the business depends on their loyalty. In particular, of the Top20% of customers, that is the 20% of customers with the highest sales, we want to know their name and the sales revenue, total revenue of all sales, their position n in the Top20% and the percentage of the sum of their revenue compared to total revenue of all sales.

Customers Top20% by Revenue					
Customer	**Revenue by Customer**	**Total Revenue**	**n**	**Percent of Total Revenue**	**Percent of Running Totals**
C03	22 890	165 600	1	14	6
C15	21 120	165 600	2	13	12
C08	16 440	165 600	3	10	19

```
WITH    RowNumberCustomer AS
        ( SELECT    Customer, SUM(Revenue) AS RevenueByCustomer
                  , SUM(SUM(Revenue)) OVER( ) AS TotalRevenue
                  , ROW_NUMBER( )
                      OVER(ORDER BY SUM(Revenue) DESC) AS n
          FROM      Sales
          GROUP BY  Customer ORDER BY RevenueByCustomer
        )
        , RowNumberCustomerExtended AS
        ( SELECT    Customer, RevenueByCustomer, TotalRevenue, n
                  , ROUND(100*RevenueByCustomer / TotalRevenue)
                      AS PercentOfTotalRevenue
                  , ROUND(100*CUME_DIST( ) OVER(ORDER BY n))
                      AS PercentOfRunningTotals
          FROM      RowNumberCustomer
        )
SELECT  *
FROM    RowNumberCustomerExtended
WHERE   PercentOfRunningTotals <= 20;
```

   In this case the Pareto rule does not hold: only 37% of sales are related to 19% of the customers.
2. We want to partition the customers into four groups:
   - Top5%, with 5% of customers with the highest amount of revenues.
   - Next15%, with 15% of other customers with the highest amount of revenues.
   - Middle30%, with 30% of other customers with the highest amount of revenues.
   - Bottom50%, with 50 % of the other customers with the lowest amount of revenues.

   For each customer group we want to know their number, and the percentage of the sum of their revenues compared to total revenue of all sales.

Customers by Revenue Top5%, Next15%, Middle30% and Bottom50%		
Group	**Number of Customers**	**Percent of TotalRevenue**
Next15%	2	27
Middle30%	2	19
Bottom50%	12	55

```
WITH      RowNumberCustomers AS
          ( SELECT    Customer, SUM(Revenue) AS RevenueByCustomer
                      , SUM(SUM(Revenue)) OVER( ) AS TotalRevenue
                      , ROW_NUMBER( )
                          OVER (ORDER BY SUM(Revenue) DESC) AS n
            FROM      Sales
            GROUP BY Customer
          )
          , RowNumberCustomersExtended AS
          ( SELECT    Customer, RevenueByCustomer, TotalRevenue
                      , ROUND(100*CUME_DIST( ) OVER (ORDER BY n) )
                          AS PercentOfCustomers
            FROM      RowNumberCustomers
          )
          , RowNumberCustomersExtendedWithGroup AS
          ( SELECT    Customer, RevenueByCustomer, TotalRevenue
                      , (CASE
                          WHEN PercentOfCustomers <= 6
                              THEN 'Top6%'
                          WHEN PercentOfCustomers > 6 AND
                               PercentOfCustomers <= 14
                              THEN 'Next14%'
                          WHEN PercentOfCustomers > 14 AND
                               PercentOfCustomers <= 30
                              THEN 'Middle30%'
                        ELSE 'Bottom50%' END
                        ) AS Group
            FROM      RowNumberCustomersExtended
          )

SELECT    Group
          , COUNT(Customer) AS NumberOfCustomers
          , ROUND(100*SUM(RevenueByCustomer) / TotalRevenue)
              AS PercentOfTotalRevenue
FROM      RowNumberCustomersExtendedWithGroup
GROUP BY Group, TotalRevenue
ORDER BY Group DESC;
```

5.5.3 Analytic Functions with the Use of Windows

For each record in a set, called the *current record*, a *window* can be defined on the data to determine the record set 'nearby' to be taken into account for the calculation of the new fields to be added to the record. The window size can be determined in a physical way (option **ROWS**), based on the number of records, or in a logical way (option **RANGE**), using a condition usually based on an attribute of type DATE.

The current record of a set (or a partition) is both the one of reference for the calculation of an aggregate function, and that for which the window size is defined by specifying the start and the end record, which

can then change when the next current record is selected.

A window can include all records of the set on which it is defined, or include only the current record. For example, to calculate a cumulative sum function, the first record is fixed and the end of the set moves from first to last record, while to calculate a moving average both the first and last record move.

For each current record, the records of the specified window are considered, and with them the value of an aggregate function is computed. The general format of the window clause is:

```
<AggregateFunction>(<expr>)
OVER(
      [PARTITION BY <attribute list>]
      [ORDER BY <sort attribute list>
       [<ROWS or RANGE> <window size specification>]]
     ) [ AS Ide ]
```

Example 5.9
Consider the following table with the transactions data of bank accounts:

BankAccount(AccountNumber, TransactionDate, TransactionType)

We want to find the balance of the accounts sorted by date of transactions.

```
SELECT      AccountNumber, TransactionDate, TransactionType
            , SUM(TransactionType) OVER
              ( PARTITION BY AccountNumber ORDER BY TransactionDate
                             ROWS UNBOUNDED PRECEDING) AS Balance
FROM        BankAccount
ORDER BY    AccountNumber, TransactionDate;
```

where **ROWS UNBOUNDED PRECEDING** specifies that the window begins with the first record of the partition and ends with the current record.

Account Number	Transaction Date	Transaction Type	Balance
1234	2009-11-01	113.00	113.00
1234	2009-11-05	−52.00	61.00
1234	2009-11-13	36.00	97.00
4321	2009-11-01	10.00	10.00
4321	2009-11-21	32.00	42.00
4321	2009-11-29	−5.00	37.00

Example 5.10
The cumulative total (running totals) is another piece of useful information to highlight the features of the report data and to facilitate analysis. In the figure an example is shown to know about the "cumulative monthly revenue by quarter (*Quarter-to-Date*) and year (*Year-to-Date*) for the product P1 in 2009".[2]

Product P1 Revenue by Quarter and Month
Year 2009

Quarter	Month	Revenue (€)	Revenue QtoD (€)	Revenue YtoD (€)
Q1	January	16 500	16 500	16 500
Q1	February	14 220	30 720	30 720
Q1	March	27 480	58 200	58 200
Q2	April	7 920	7 920	66 120
Q2	May	1 200	9 120	67 320
Q2	June	1 260	10 380	68 580
Q3	July	5 400	5 400	73 980
Q3	August	11 730	17 130	85 710
Q3	September	10 860	27 990	96 570
Q4	October	5 850	5 850	102 420
Q4	November	2 100	7 950	104 520
Q4	December			

```
SELECT    Quarter_Name(QUARTER(Date)) AS Quarter
          , Month_Name(MONTH(Date)) AS Month
          , SUM(Revenue) AS Revenue
          , SUM(SUM(Revenue)) OVER
              (PARTITION BY QUARTER(Date)
                  ORDER BY MONTH(Date)
                      ROWS UNBOUNDED PRECEDING) AS RevenueQToD
          , SUM(SUM(Revenue)) OVER
                  (ORDER BY MONTH(Date)
                      ROWS UNBOUNDED PRECEDING) AS RevenueYToD
FROM      Sales
WHERE     YEAR(Date) = 2009
GROUP BY  QUARTER(Date), MONTH(Date)
ORDER BY  Quarter, Month;
```

Finally, an example is given of a moving window with a size defined in a physical way. Assuming that the total revenues from product sales vary greatly during the year, their values do not make a clear global trend, but it might be useful to know how to predict future revenues. For this reason, an interesting report is the one that shows a moving average of total revenues over three months – the current one, the one that precedes it and the other that follows it.

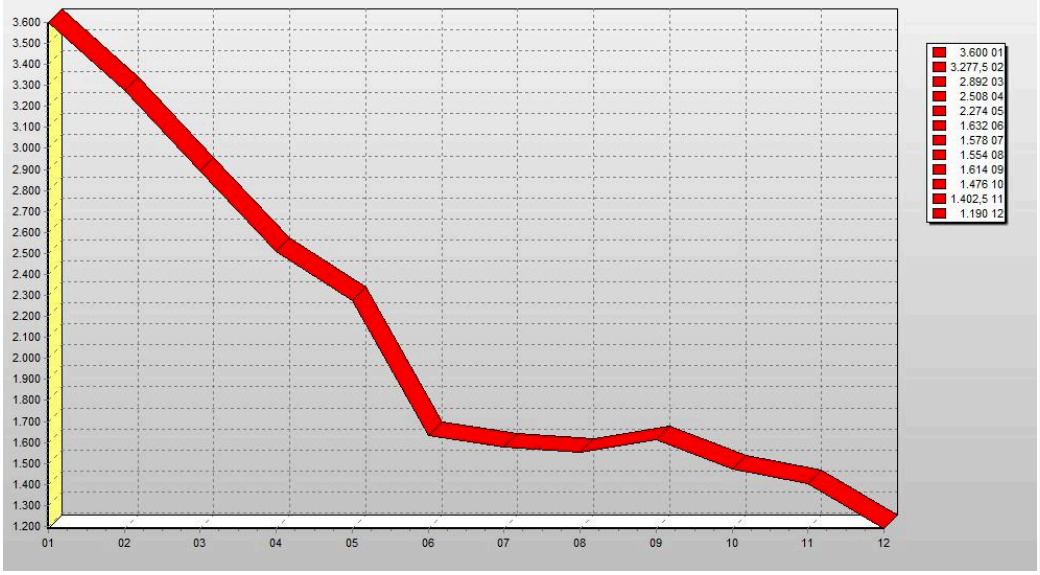
```
SELECT    MONTH(Date) AS Month
          , ROUND(AVG(SUM(Revenue))
              OVER (ORDER BY MONTH(Date)
                  ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING), 2)
              AS MovingAverageRevenue
FROM      Sales
GROUP BY  MONTH(Date)
ORDER BY  Month;
```

In Oracle, when you define a query in analytic SQL you can choose to display a graphical representation of the result. For example, Figure 5.14 shows the trends of the moving average of total revenue, with a

2. To obtain a result with the names of the months and quarters, instead of their numbers, the following functions have been defined in SQL: Quarter_Name and Month_Name.

moving window of 3 or 5 months.

Moving average of total revenue with a moving window of 3 months



Moving average of total revenue with a moving window of 5 months

Figure 5.14: Two graphical representations of the moving average of total revenue by product and month

5.6 Summary

– The main commercial solutions for OLAP systems have been presented. OLAP servers are implemented using either a multidimensional storage engine (MOLAP), a relational DBMS engine (ROLAP) as the backend, or a hybrid combination called HOLAP. Changes in the hardware technology will change how the backend of large data warehouses are organized, and as cloud data services take root, more changes are expected.
– The standard SQL language supports relatively simple data analysis, and so it has been extended with new operators and analytic functions to allow complex data analysis.
– Data analysis in SQL is useful for understanding both the functionality of tools that provide graphical interfaces to formulate the queries, and their limitations in expressive power.

Part III

Data Warehouse Systems: Storage, Indexing and Query Evaluation

Chapter 6

STORAGE STRUCTURES AND STAR QUERY PLANS

When there are a lot of data to analyze, the traditional DBMSs used for operational databases do not provide the desired performance, but they must be extended to support data warehouses by providing appropriate storage structures, and different techniques for star query optimization. Typical solutions are presented using both traditional relational DBMS extended with new types of indexes, such as IBM's DB2 and Oracle, and to create new types of systems for data warehouses that store data in a different way, by *columns* instead of by *records*, such as Sybase-IQ. In the next chapter it will be shown how to revise the query optimizer to deal with star queries.

6.1 Indexes Overview

An index is a data structure that allows the DBMS to locate quickly particular records of a table. An index in this context has a role similar to that of a book. The pages of a book are ordered, and to find information about a particular subject, we use the index in the back of the book, in which we look up a keyword to get the list of one or more pages on which the keyword appears. In a similar way, in the case of the set of records of a table, to find a record with a given attribute value, we first look at the index defined on the attribute to get the location of the records in the table, and then the records are retrieved. As in the book index analogy, the index is ordered on the attribute values.

To simplify the presentation we will assume that records in a table are fixed in size, allocated on consecutive pages of a file, and are numbered sequentially, starting from 1. To retrieve the record numbered n, it is easy to translate n into a file page number and a number that identifies the record within the page. A record number will be called the *Record Identifier, RID*.

Definition 6.1

Let R a table of records with an attribute A. An *index* I on A is a sorted table $I(A, \text{RID})$ on A, with $(N_{\text{rec}}(I) = N_{\text{rec}}(R))$. An element of the index is a tuple $(A := a_i, \text{RID} := r_i)$, where a_i is an attribute A value for a record, and r_i is a reference (RID) to the corresponding record in R.

An index can be defined on a *key attribute*, on a *non-key attribute* or on a *set of attributes*. In the last case, the index, called *composite*, contains an element for each combination of values of the attributes in the table, and can be used to execute efficiently queries that specify a value for each of these attributes or for a prefix of them.

Figure 6.1 shows two example of indexes on two attributes of the table R, the key K and a non-key attribute A, assuming for simplicity that the RIDs are integers that represent the position of the record in the table.

A typical implementation of an index is the *inverted index* organization defined as follows.

Definition 6.2

An *inverted index* I on a non-key attribute A of a table R is a sorted collection of entries of the form $(a_i, n, p_1, p_2, \ldots, p_n)$, where each values a_i of A is followed by the number of records n containing that value and the *sorted* RID list of these records (*rid-list*).

R

RID	K	A	...
1	k5	d	...
2	k3	b	...
3	k7	a	...
4	k6	c	...
5	k2	b	...
6	k4	g	...
7	k1	c	...
...	...	...	...

IdxK

K	RID
k1	7
k2	5
k3	2
k4	6
k5	1
k6	4
k7	7
...	...

IdxA

A	RID
a	3
b	2
b	5
c	4
c	7
d	1
g	6
...	...

Figure 6.1: Example of a table with two indexes

An inverted index on the attribute Quantity of the Sales table in Figure 6.2 is shown in Figure 6.3. An inverted index is usually organized as a B^+-tree, with the leaves containing the index entries.

Sales

RID	Date	Product	City	Quantity
1	20090102	P1	Lucca	2
2	20090102	P2	Carrara	8
3	20090103	P3	Firenze	5
4	20090103	P1	Arezzo	10
5	20090103	P1	Pisa	1
6	20090103	P4	Pisa	8
7	20090103	P2	Massa	5
8	20090104	P2	Massa	2
9	20090105	P4	Massa	2
10	20090103	P4	Livorno	5
11	20090103	P4	Lucca	8
12	20090106	P3	Lucca	5
13	20090106	P1	Pisa	8
14	20090106	P3	Pisa	8
15	20090106	P2	Firenze	1
16	20090106	P1	Firenze	8

Figure 6.2: Sales relation

Quantity	n	RID list
1	2	5, 15
2	3	1, 8, 9
5	4	3, 7, 10, 12
8	6	2, 6, 11, 13, 14, 16
10	1	4

Figure 6.3: An *inverted index* on Quantity

When the number of distinct values of an indexed attribute is small (i.e. the attribute is not selective), the inverted lists are long and the indexes are not useful to retrieve data.

6.2 Special-Purpose Indexes

Besides traditional inverted indexes, let us see how the DBMS technology has been extended to support data warehouses with two special kinds of indexes for drastically improving the analytic query performances: *bitmap indexes* and *join indexes*.

6.2.1 Bitmap indexes

A bitmap is an alternative method of representing a rid-list of an index. Each index element has a bit vector instead of a rid-list.

Definition 6.3
A *bitmap index* I on a non-key attribute A of a table R, with N records, is a sorted collection of entries of the form (a_i, B), where each values a_i of A is followed by a sequence of N bits, where the jth bit is set to 1 if the record jth has the value a_i for the attribute A. All other bits of the bitmap B are set to 0.

Figure 6.4 shows the *bitmap index* for the example of Figure 6.3.

Sales			Bitmap index				
RID	...	Quantity	1	2	5	8	10
1	...	2	0	1	0	0	0
2	...	8	0	0	0	1	0
3	...	5	0	0	1	0	0
4	...	10	0	0	0	0	1
5	...	1	1	0	0	0	0
6	...	8	0	0	0	1	0
7	...	5	0	0	1	0	0
8	...	2	0	1	0	0	0
9	...	2	0	1	0	0	0
10	...	5	0	0	1	0	0
11	...	8	0	0	0	1	0
12	...	5	0	0	1	0	0
13	...	8	0	0	0	1	0
14	...	8	0	0	0	1	0
15	...	1	1	0	0	0	0
16	...	8	0	0	0	1	0

Figure 6.4: A *bitmap index* on Quantity

Using bitmaps might seem a huge waste of space, however bitmaps are easily compressed, so this is not a major issue. With efficient hardware support for bitmap operations (AND, OR, XOR, NOT), a bitmap index offers better access to answer queries such as "how many sales of product P1 have been made in Pisa", since the answer is found by counting the 1's of one bit-wise AND of two bitmaps.

This kind of index can be declared with an SQL command such as:

CREATE BITMAP INDEX Name **ON** Table(Attribute);

Example 6.1
It is interesting to compare the memory occupied by an inverted index and a bitmap index defined on the same attribute, stored with B^+–tree, considering only the memory for the leaves, supposed completely full.

The number of leaves of an inverted index is:

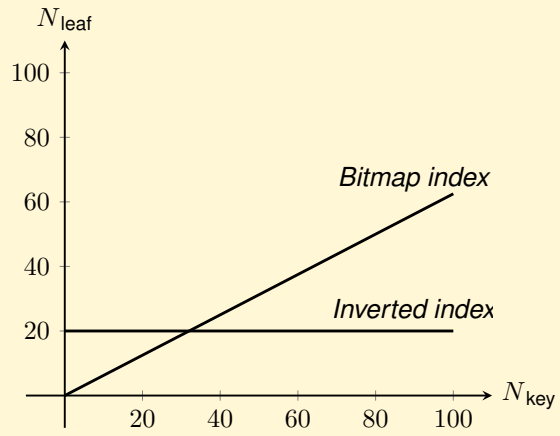
$$N_{\text{leaf}} = (N_{\text{key}} \times L_{\text{k}} + N_{\text{rec}} \times L_{\text{RID}})/D_{\text{pag}} \approx (N_{\text{rec}} \times L_{\text{RID}})/D_{\text{pag}}$$

where N_{key} is the number of distinct values of the attribute, D_{pag} is the leaves page size in byte, L_{k} is a value attribute size in byte and L_{RID} is the RID size in byte.

The number of leaves of a bitmap index is:

$$N_{\text{leaf}} = (N_{\text{key}} \times L_{\text{k}} + N_{\text{key}} \times N_{\text{rec}}/8)/D_{\text{pag}} \approx N_{\text{key}} \times N_{\text{rec}}/(D_{\text{pag}} \times 8)$$

For the approximations made, the number of leaves of a B^+–tree index does not depend on N_{key}, while the number of leaves of a bitmap index increases linearly with N_{key}.



The two values are equal for $N_{\text{key}} = 8 \times L_{\text{RID}}$ (for $L_{\text{RID}} = 4$ bytes, $N_{\text{key}} = 32$), for lower values the B^+–tree index takes more memory, while for higher values a bitmap index takes more memory.

An interesting aspect is that although the binary vectors are generally very long, and with very selective attributes (high values of N_{key}) become *scattered*, i.e. a large number of bits will be zero, they can be easily stored in a compressed form to reduce the memory requirements. For example, the Oracle system, which uses such techniques, suggests using them if $N_{\text{key}} < N_{\text{rec}}/2$.

6.2.2 Join indexes

Suppose we want to speed-up an equi-join between the primary key attribute of the dimension table D and the foreign key attribute in the fact table F ($D \underset{Pk=Fk}{\bowtie} F$). A *join index* is a pre-computed join that is stored in a compact form, defined as follows [Valduriez, 1987].

Definition 6.4

A *join index* is a set of ordered pairs of the form (d, f), where d is a RID of a record r_d in D and f is a RID of a record r_f in F such that $r_d.Pk = r_f.Fk$ (i.e. the two records join).

The join indexes are not usually used in relational DBMSs because of the cost of their updating for insertion or deletion of records that join, while they are useful for data warehouses, where data are static, to perform the typical operation of star join. Several solutions have been proposed, and let us see some of them, using the following example of star schema:

1. F is the fact table with a measure M,
2. D_1 and D_2 are dimensional tables with keys PkD1 and PkD2; the attributes of D_1 start with the letter A and those of D_2 with the letter B,
3. FkD1 and FkD2 are the foreign keys in F for D_1 and D_2,

with the data in Figure 6.5, and the query

```
Q:  SELECT   M, A2, B3
    FROM     F, D1, D2
    WHERE    FkD1 = PkD1 AND FkD2 = PkD2
    AND      A1 = 10 AND B2 = 20;
```

F

RID	FkD1	FkD2	M
1	v1	z1	150
2	v2	z1	300
3	v3	z2	400
4	v2	z2	200
5	v1	z2	90
...	...	...	...

D1

RID	PkD1	A1	...
1	v1	20	...
2	v2	10	...
3	v3	10	...
...	...	...	...

D2

RID	PkD2	B1	...
1	z1	10	...
2	z2	20	...
3	z3	20	...
...	...	...	...

Figure 6.5: Star schema data mart

Star Join Index

A *star join index* on a star schema is a multi-attribute join index between the dimension tables and the fact table. Figure 6.6 shows the star join index on D_1 and D_2, and the fact table F.

StarJoinIndex

D1	D2	F
1	1	1
1	2	5
2	1	2
2	2	4
3	2	3
...	...	...

Figure 6.6: A *star join index* example

A star join index is the result of the following query (where the notation R.RID stands for the RID of a record of R):

```
SELECT   D1.RID, D2.RID, F.RID
FROM     F, D1, D2
WHERE    FkD1 = PkD1 AND FkD2 = PkD2;
```

This kind of index can be declared with an SQL command such as:

```
CREATE STAR INDEX JIonF ON F(FkD1, FkD2);
```

A star join index with k dimensional tables is not useful if the join with the fact table only uses a subset of the dimensional tables that are not a prefix of $(D_1, \ldots, D_k)$. For this reason it is usually better to define only binary join indexes between the fact and dimensional tables.

Two common variations of star join indexes are used in data warehouse systems: *bitmapped join index* and *bitmapped foreign column join index*.

Bitmapped Join Index

[O'Neil and Graefe, 1995] suggest (a) to use a binary join index for each binary join ($D_i \bowtie F$), and (b)

for each RID d in D_i, all records of the form (d, f) in the join index are replaced with a single record of the form $(d$, *bitmap for matching records in* $F)$. Here the bitmap has a 1 in the jth position if and only if the jth record in F joins with the record in D_i with the RID d (Figure 6.7).

JoinIndex

D1	F
1	1
1	5
2	2
2	4
3	3
...	...

BitmappedJoinIndex

D1	bitmap for F
1	10001...
2	01010...
3	00100...
...	...

Figure 6.7: Join indexes between D1 and F

Foreign Column Join Index

While a traditional index maps an attribute value of a table to the records with that value, a *Foreign Column Join Index, FCJI*, maps an attribute value from a dimension table to the fact table records joining with the dimension table records with that value (Figure 6.8). The net effect is to have a bitmap index on a fact table F based on a dimensional attribute A, even if A is not an attribute of F. Using this index can eliminate the need to join the tables.

A1-FCJI

A1	F
10	2
10	3
10	4
20	1
20	5
...	...

Figure 6.8: *Foreign Column Join Index* on A_1

Combining this solution with that of *bitmapped join index* we have a bitmap index called a *Bitmapped Foreign Column Join Index, BMFCJI* (Figure 6.9).

A1-BMFCJI

A1	F
10	01110...
20	10001...
...	...

Figure 6.9: A *bitmapped foreign column join index* on A_1

Bitmapped Foreign Column Join Indexes were introduced in Oracle 9i, and called *Bitmap Join Indexes*. The index is declared with the command:

```
CREATE    BITMAP INDEX  Name ON F(A1)
FROM      F, D1
WHERE     FkD1 = PkD1;
```

Then the following query can be executed using the index without performing the join of F with D_1:

```
Q:  SELECT    SUM(M)
    FROM      F, D1   WHERE   FkD1 = PkD1 AND A1 = 10;
```

6.3 Physical Operators

Physical Operators for Inverted and Bitmap Indexes

To show possible uses of these indexes, will assume that the following physical operators are available for the physical query plans:

1. Operators on relations:

   TableAccess(O, R) where O is a relation with a column of RID. The operator result is a relation with records of R with the RIDs in O.

2. Operators on bitmaps:

   (a) **BMAnd**(O_E, O_I) returns the bitmap result of the AND of the O_E and O_I bitmaps;
   (b) **BMOr**(O_E, O_I) returns the bitmap result of the OR of the O_E and O_I bitmaps;
   (c) **BMMinus**(O_E, O_I) returns the bitmap result of the difference of the O_E and O_I bitmaps;
   (d) **BMNot**(O) returns the bitmap result of the NOT of the O bitmap;
   (e) **BMCount**(O, Ide) returns a relation with the attribute Ide and the value the number n of 1 in the O bitmap;
   (f) **RIDFromBM**(O) returns a relation with the attribute RID and the records RID with 1 in the O bitmap;
   (g) **BMFromRID**(O) returns a bitmap of the records with RID in O, a relation with one column of RID;
   (h) **BMMerge**(O) returns the bitmap result of the OR of the bitmap set in O.

3. Operators on inverted and bitmap indexes:

   (a) **RIDIndexFilter**(Idx, ψ) returns a relation of records with the attribute RID. The values of the attribute are the RIDs of the record in a relation with the values of the indexed attributes that satisfy the condition ψ.
   (b) **BMIndexFilter**(Idx, ψ) returns the bitmap result of the OR of the bitmaps of the records in a relation with the values of the indexed attributes that satisfy the condition ψ.
   (c) **BMKeyIteration**(O_E, O_I) with O_I a bitmap index on an atttribute A_i and O_E is a set of the A_i values. The operator returns the set of the index bitmaps associated with the A_i values in O_E.

Example 6.2
Let $R(A, B, C)$ be a relation with two bitmap indexes on B and C. Figure 6.10a shows a physical query plan for the query

```
SELECT    A
FROM      R
WHERE     B = 10 AND C = 5;
```

and Figure 6.10b shows a physical query plan for the query

```
SELECT    COUNT(*) AS N
FROM      R
WHERE     B = 10 AND C = 5;
```

If the index on B is an inverted index, the physical plan for the previous query is shown in Figure 6.10c.

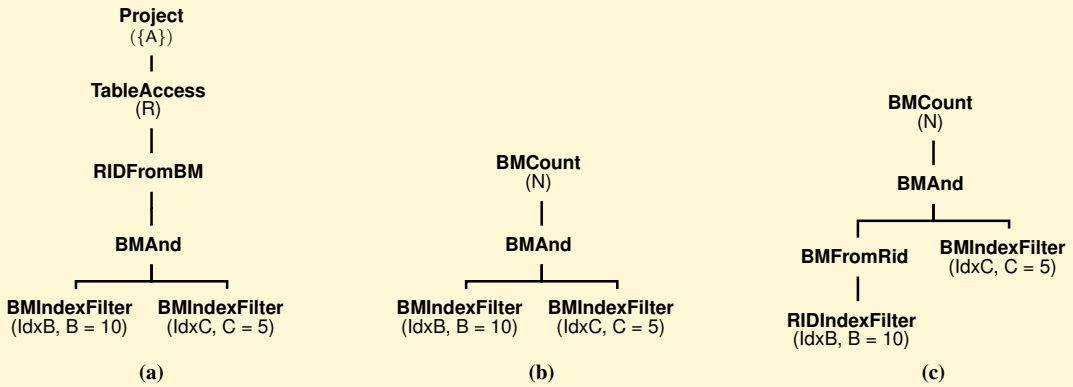


Figure 6.10: Physical query plans with bitmap indexes

Physical Operators for Join Indexes

To show possible uses of the join indexes, we assume that the following physical operators are available for the physical query plans, even if it is not required that a specific system has to support all of them.

1. **BMJIndexFilter**(JIdx, ψ) returns the bitmap of the records in a fact table F using the *Bitmapped Join Index* JIdx(D_i, F) that satisfy the condition $\psi = (D_i = \text{RID})$ on the attribute D_i.
2. **BMFCJIndexFilter**(FCJIdx, ψ) returns the bitmap result of the OR of the bitmaps of the records in a fact table F using the *Foreign Column Join Index* FCJIdx(A_i, F) that satisfy the condition $\psi = (A_i \ \theta \ c)$ on the dimensional attribute A_i.

6.4 Star Query Plans

Star queries are the most common kind of queries in data warehousing, OLAP and business intelligence applications. Thus, there is an imperative need for efficiently processing this kind of queries.

Definition 6.5

A *star query* is an equi-join between a fact table and some of dimension tables of a star schema, usually subject to selection conditions, and the result is further grouped and aggregated. Each dimension tables is joined with the fact table using primary key-foreign key equality conditions.

The major bottleneck in evaluating star queries is the join of the fact table, usually very large, with the dimensional tables.

For the sake of simplicity, we will assume that the star query is a single **SELECT** with **GROUP BY**, **HAVING** and aggregate functions, but without **DISTINCT**, subqueries and **ORDER BY**.

A star query is executed with a different algorithm by a standard relational DBMS and by a DBMS specialized for data warehouses. In the following, we present first the basic phases for evaluating star queries in the two DBMS types, and then examples of processing algorithms expressed as physical query plan will be shown.

Standard physical query plan

A query plan is produced using the query optimization techniques typically applied in a relational DBMS which computes the query result with the following basic phases:

1. *Selection*. The local selections are applied to the fact table and to the dimensional tables.

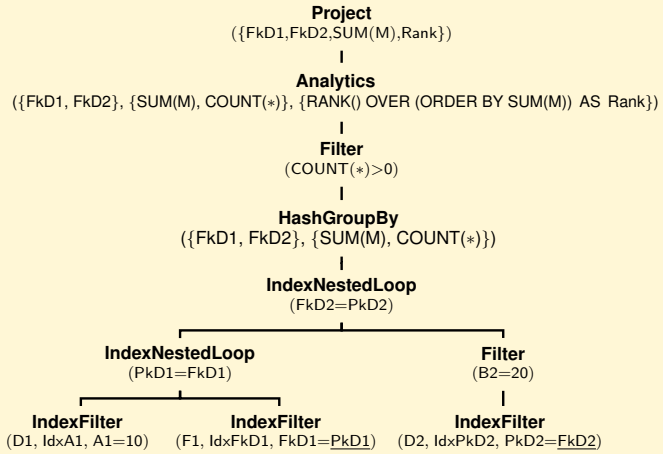
2. *Join*. Using the best join physical operators available (e.g., **IndexNestedLoop**, **MergeJoin**, **HashJoin**), the fact table, or the intermediate results table, is joined with the dimensional tables.
3. *Result*. The result of the joins is grouped and aggregated according to the attributes of the **GROUP BY** clause, the groups are then further filtered and projected to produce the query result.

Example 6.3
Let us consider the SQL query:

```
SELECT    FkD1, FkD2, SUM(M), RANK() OVER (ORDER BY SUM(M)) AS Rank
FROM      F, D1, D2
WHERE     FkD1 = PkD1 AND FkD2 = PkD2 AND A1 = 10 AND B2 = 20
GROUP BY  FkD1, FkD2
HAVING    COUNT(*) > 0;
```

Let us assume that inverted indexes exist on the foreign keys of the fact table, on the primary keys of the dimensional tables, and on the dimensional attributes.

A possible physical query plan produced by a traditional relational system has the structure shown in the figure. The records of a selection on a table are retrieved with the operator **IndexFilter**, the records of a join are computed with the operator **IndexNestedLoop**, the records of the grouping are computed with the operator **HashGroupBy**, and then the groups are filtered and processed to produce the query result.



Star query plan

When bitmap indexes are available, a physical star query plan is produced using specialized optimization techniques for data warehouses which compute the query result with the following basic phases significantly different from the ones in the previous approach to execute a star query more efficiently (Figure 6.11):

1. *Selection*. The local conditions are applied to the fact table and to each dimensional table that participates in the join to compute the *local rowsets*. A local rowset is a bitmap representing the selected records from a table: the n-th bit is 1 if the n-th record of the corresponding table satisfies the local condition. The result of the phase is a set of rowsets that represent which records from each table are candidates for inclusion in the join result.
2. *Semi-join*. Using a join index and the local rowsets, the *global rowset* is computed, which is a bitmap representing the records from the fact table that belong to the star join. The algorithm to execute this

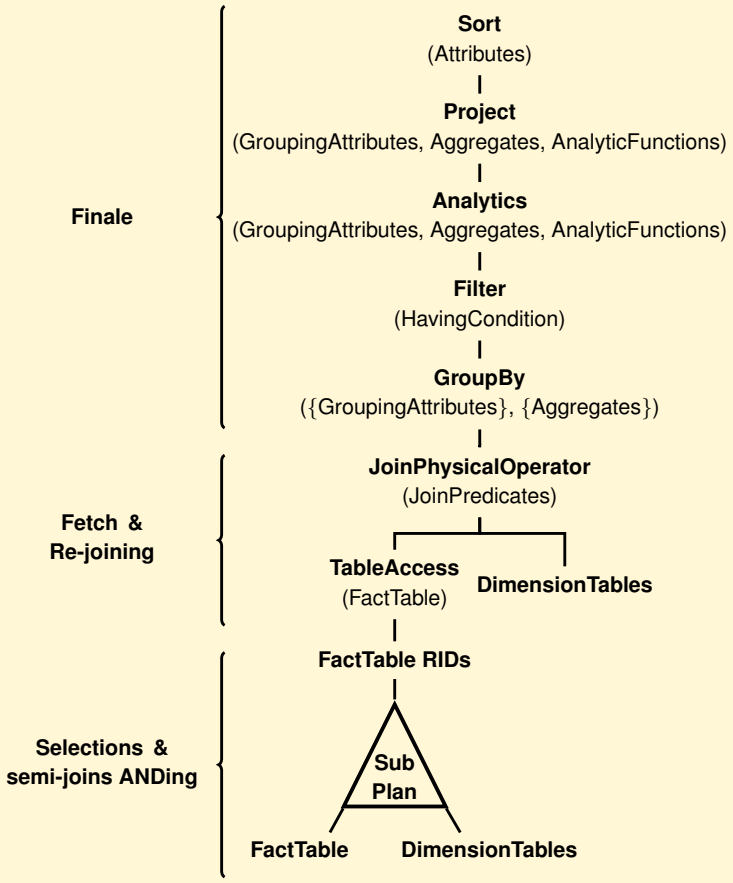


Figure 6.11: A star query evaluation strategy

phase depends on the kind of join index available.

3. *Fetch&Re-joining*. The qualifying fact table RIDs are computed from the *global rowset* and the corresponding records are retrieved with the operator **TableAccess** and joined with the dimension tables records only if some dimensional attributes are needed by the grouping operator, or the aggregate functions, to produce the final result.
4. *Finale*. The re-join result is grouped according to the attributes of the **GROUP BY** clause, and the aggregate functions in the **SELECT** and **HAVING** clauses are computed. Finally, the groups are filtered and processed to produce the query result.

Let us see some examples of physical query plans for a query without the **GROUP BY**. In the next chapter we will see how to treat the general case and, in particular, the optimization of the grouping operator, always present in the data analysis. The plans exploit the benefits of different types of bitmap indexes to evaluate star query joins with significant performance gains.

Example 6.4
Let us consider the star join query

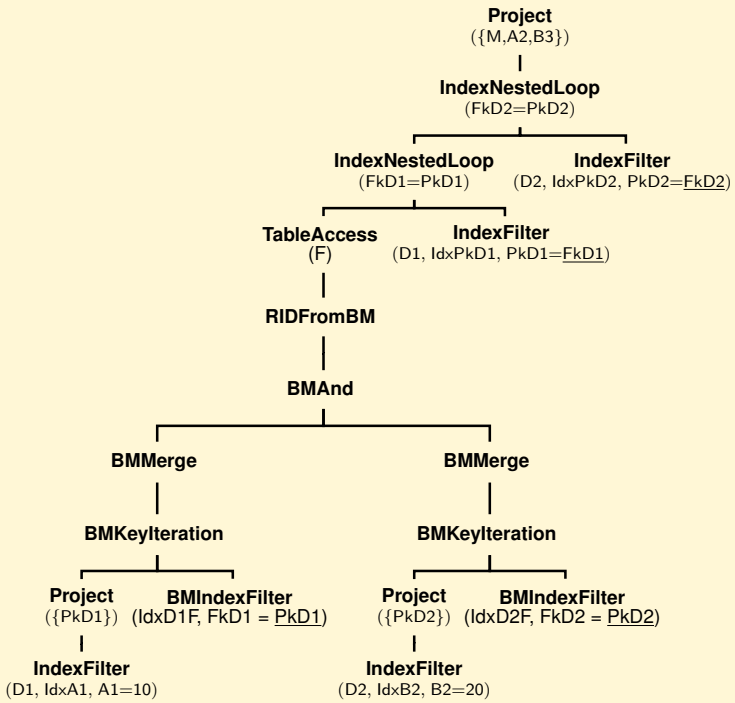
```
Q:  SELECT    M, A2, B3
    FROM      F, D1, D2
    WHERE     FkD1 = PkD1 AND FkD2 = PkD2
    AND       A1 = 10 AND B2 = 20;
```

and the physical query plans generated depending on the following cases of indexes available:

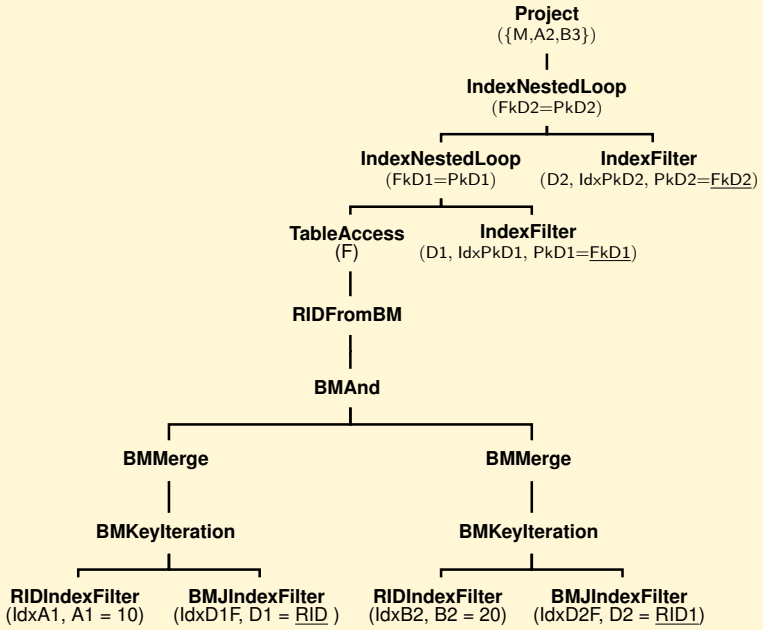
1. *Bitmap indexes* on the foreign keys of the fact table, inverted indexes on the primary keys of D_1 and D_2, and inverted indexes on the attributes D1.A1 and D2.B2

   In this plan, the fact table records are retrieved with the operator **TableAccess** using the record identifiers computed from the global rowset, obtained from a bitmap AND of two bitmaps generated by the **BMMerge** operators from the results of the physical trees underneath it. Each such physical query plan consists of a **BMKeyIteration** operator which retrieves the dimensional primary key values from the external operand, which in this example is an indexed table access, and for each such value, the **BMKeyIteration** operator retrieves the bitmap from the internal operand bitmap index on a foreign key of the fact table.

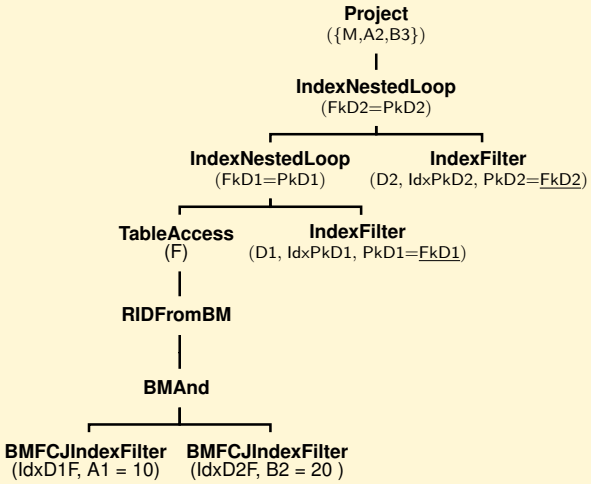
   After the relevant fact table records have been retrieved using the **TableAccess** operator, they are joined with the dimension tables, to produce the query result with the following physical star query plan.



2. *Bitmapped Join Index* between D_1, D_2 and the fact table F, inverted indexes on the primary keys of D_1 and D_2, and inverted indexes on the attributes D1.A1 and D2.B2 produce the query result with the following physical star query plan.



3. *Bitmapped Foreign Column Join Index* on the attributes D1.A1 and D2.B2, and inverted indexes on the primary keys of D_1 and D_2 produce the query result with the following physical star query plan.

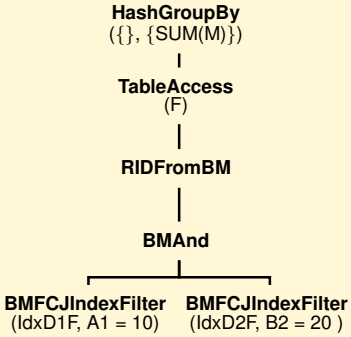


Note that if in the **SELECT** of the query Q there are only attributes of F, the physical query plans will not use the two physical operators **IndexNestedLoop** for the joins of the fact table F with the dimensions D_1 and D_2. The re-join operators are used only when it is necessary to enrich the fact table records with dimensional attributes required in the **SELECT**, **GROUP BY**, **HAVING** and **ORDER BY** clauses.

For example, the query

```
SELECT    SUM(M)
FROM      F, D1, D2
WHERE     FkD1 = PkD1  AND  FkD2 = PkD2
AND       A1 = 10  AND  B2 = 20;
```

is executed with the following plan, without physical joins operators, if there are the *Bitmapped Foreign Column Join Indexes* on A_1 and B_2.



6.5 Column-Oriented Data Warehouse Systems

The main commercial DBMSs are based on a *row-oriented* relational technology: a database table is stored row-by-row in a set of physical pages. This solution is optimized for OLTP applications which commonly access and update data on the granularity of a record. Relational DBMSs have subsequently been extended with new kinds of indexes and new optimization techniques to improve query performance of typical OLAP applications on data warehouses. The major players in the data warehouse commercial arena (Oracle, DB2, SQL Server, and Teradata) adopt this approach because their systems are historically focused on the large transactional database market and they prefer to maintain a single kind of system for all the possible applications.

The phrase "One size fits all" has been used to refer to the fact that the traditional relational technology has been used to support many data-centric applications with different features. However this strategy is no longer applicable to OLAP applications that have the following main features:

- Queries are complex, interactive, unpredictable, and concern only a few attributes.
- Queries are read-only and usually require groupings of large data sets on few attributes to compute several aggregation functions.

To improve the performance of a such kinds of queries over large, ever-increasing data sets, several authors have demonstrated that an implementation based on a *column-oriented* storage system (also called *transposed file, projection indexes*) can achieve substantial improvements in OLAP query performance. The technique of transposed files was first proposed in the 1970s [Batory, 1979]. The first interesting paper that motivated and presented the implementation of a DBMS based on transposed files was [Turner et al., 1979]. The basic ideas of an approach adopted in the Sybase IQ product are discussed in [French, 1995], [French, 1997]. Another interesting project is Monet [Boncz and Kersten, 1999], [Boncz et al., 2005].

A better performance for such kinds of DBMSs has been achieved by rethinking the way in which relational DBMSs deal with the following aspects, which will be discussed in the next subsections: *data storage*, *database page size* and *data compression* [French, 1995].

6.5.1 Data Storage

The idea of picking a simple but fundamental assumption underlying traditional relational DBMSs, to change it and then reconsider all aspects of data management and query processing can lead to interesting results. As far as data storage, the fundamental assumption to change has been the way tables are stored: by columns rather than by rows (Figure 6.12).

Customers

Name	Address	Pk
N1	A1	CK1
N2	A2	CK2
N3	A3	CK3

Sales

Fk	Qty	Price
CK1	Q1	P1
CK2	Q2	P2
CK3	Q3	P3
CK3	Q4	P4

(a) *Tables by rows*

Customers Columns

Name	Address	Pk
N1	A1 1	CK1
N2	A2	CK2
N3	A3	CK3

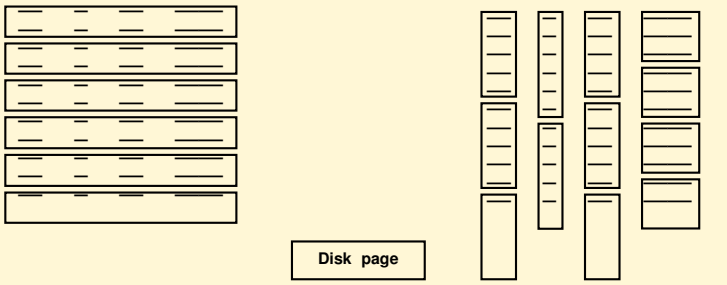
Sales Columns

Fk	Qty	Price
CK1	Q1	P1
CK2	Q2	P2
CK3	Q3	P3
CK3	Q4	P4

(b) *Tables by columns*

Figure 6.12: Storing tables by column rather than by rows

Relational DBMSs store tables by row because typical OLTP applications consist of queries that read and update typically only a few records at time and so high performance is achieved by storing contiguous records in disk pages. In contrast, typical OLAP applications consist of queries that read typically only a few attributes of large amounts of historical data in order to partition them into several groups and compute some aggregation functions. So high performance is achieved by storing contiguous values of a single attribute in disk pages (Figure 6.13).



(a) *Table stored by rows* **(b)** *Table stored by columns*

Figure 6.13: Row store versus column store

As in traditional DBMSs there are a variety of different indexing methods that may be more or less appropriate, depending on the circumstances, so in column-oriented systems the idea of changing the way data are stored is not in itself enough to achieve the expected queries performances. A data column (thinking of this as essentially an index) can be implemented in various ways, depending on the type of data that it contains. In addition, other storage structures must be considered to speed up star queries. The typical solutions are bitmap indexes on columns and a kind of join index for each foreign key column. Figure 6.14 shows an example of a join index, a column which contains only the RID (position) of the dimensional table. Assuming that each column has fixed length values different from null, a column element is identified by its position, and a RID is an element position value.
In the literature the solutions of the following column-based data warehouse systems are presented:

– **SDM**. The system adopts the two kinds of storage structures shown in Figure 6.14, but each column

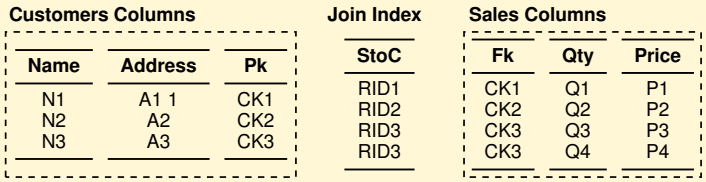


Figure 6.14: A join index example

has a pair of attributes (RID, Value) and there is an index on each column. Each column is stored twice: one ordered on RID, the other ordered on the value. The RID data is necessary to reconstruct a complete table record.

The join index is a table containing pairs of RIDs: the first is the RID of a fact table record, and the second is the RID of a record of the dimensional table that joins with the fact table record [Copeland and Khoshafian, 1985].

- **Sybase IQ**. The system adopts the two kinds of storage structures shown in Figure 6.14, and there is a bitmapped index on each column.
- **Curio**. The system adopts the two kinds of storage structures shown in Figure 6.14. Indexes are not used since it is assumed that an index is exactly what each column is. Thus the entire database is, in effect, indexed simply by virtue of the fact that the storage is column-based [Datta et al., 1998], [Datta et al., 1999].
- **C-Store**. The system does not store columns for the tables attributes, but rather a set of projections of the facts table, of the dimensional tables or of the joins of the facts with dimensional tables. The projections are like materialized views which are chosen to support a set of predefined queries. Each projection generally contains more than one attribute and the same attribute can belong to more than one projection. A projection can be ordered and in this case it also contains a field for the RID of the record of the table covered. Projections are horizontally partitioned into one or more segments identified by an identifier (SID). Join indexes are defined between partitions and they are a set of pairs (SID, RID) [Stonebraker et al., 2005]. The Vertica (Hewlett-Parckard) system is the commercial version of C-Store.
- **SADAS**. The system is the result of an industrial research project which began in March 2003 and was partially sponsored by the Italian Ministry of Education, University and Research (MIUR) to support the cooperation of universities and industries in prototyping innovative systems. The main contractor was the Italian software company Advanced Systems, based in Naples. Already in the early 1980s, Advanced Systems had been a precursor in column oriented technology. SADAS is the first European commercial DW system to adopt a column-oriented approach [Albano et al., 2006].

6.5.2 Database Page Size

The database page size determines the amount of data that is written to and read from the disk, and so it is a fundamental parameter that affects DBMS performance.

While OLTP applications work better with relatively small pages of some Kbyte, OLAP applications work better with relatively large pages of tens of Kbyte. This is because to answer a typical OLTP application query only a few records need to be read from the disk, while answering a typical OLAP applications query involves a high percentage of data. Results from experiments suggest a page size of 64K.

6.5.3 Data Compression

It is well known that data compression in traditional DBMSs improves performance significantly: it reduces disk space usage, seek times (the data are stored nearer to each other), transfer time (there is less

data to transfer), and it increases the buffer hit rate (a larger fraction of data fits in the buffer pool).

It is intuitive that data stored by columns are more compressible than data stored by rows, which results in faster query response times. In fact data stored by rows generally do not compress very well because the fields within a row are of different data types (5-10% compression is typical). Instead data stored by columns offer higher compression ratios because all the values within a column are of the same data type, especially when columns are sorted or their data have low information entropy (high data value locality). Moreover, it is possible to use a variety of algorithms, as is most appropriate for the data type of the column in question. Thus a column-based approach optimizes data compression as compared to traditional relational databases (50% compression is typical).

6.6 New DW Platforms

Choosing the right DW technology is not an easy task. Operational expenditure (OPEX) costs increase with traditional data warehouse systems for defining the right physical design and for tuning big DW to get the solution to perform well. The following solutions can reduce or eliminate the need for data indexing and storing pre-aggregated data.

– **In-memory appliances**.
  SAP HANA has introduced the term: the system, also referred to as the **SAP in-memory database**, uses a column-based storage in main memory.
  Three developments in recent years have made in-memory appliances increasingly feasible: 64-bit computing, multi-core servers and lower RAM prices.
– **DW Appliances**.
  Essentially, it is a fully integrated stack of CPU, memory, storage, operating system, and RDBMS software that is purpose-built and optimized for data warehousing and business intelligence workloads. It uses massive parallelism to optimize query processing. Ultimately, the vision of a DW appliance is to provide a self-managing, self-tuning, plug-and-play DB system that can be scaled out in a modular, cost-effective manner.

  According to Steve Norall[1], architectural approaches to data warehouse appliances vary widely, and the following four main points must be considered for assessing different vendors' approaches.

  First, most of the DW appliances replace traditional database kernels (e.g., Oracle, IBM DB2, and Microsoft SQL Server) with their own optimized database kernel.

  Second, the total cost and overall price-performance of the solution can be directly affected by the underlying components.

  Third, all vendors leverage some degree of parallelism to deliver the requisite performance and scalability. End users understand whether the trade-offs are well-suited to their database workload.

  Fourth, some appliances may be cost-prohibitive for smaller data warehousing deployments. Moreover, some solutions require users to purchase additional storage capacity in relatively large chunks (sometimes greater than 10 TB). As a result, some appliances may be cost-prohibitive for smaller data warehousing deployments.

6.7 Commercial Systems for Data Warehouses

The Gartner company in March 2014 published the report *Magic Quadrant for Data Warehouse Database Management Systems*, which graphically shows a matrix that classifies 16 vendors of systems for data

1. A senior analyst with the Taneja Group research and consulting firm (www.taneja.com).

warehouses on the basis of two evaluation criteria (Figure 6.15).[2]

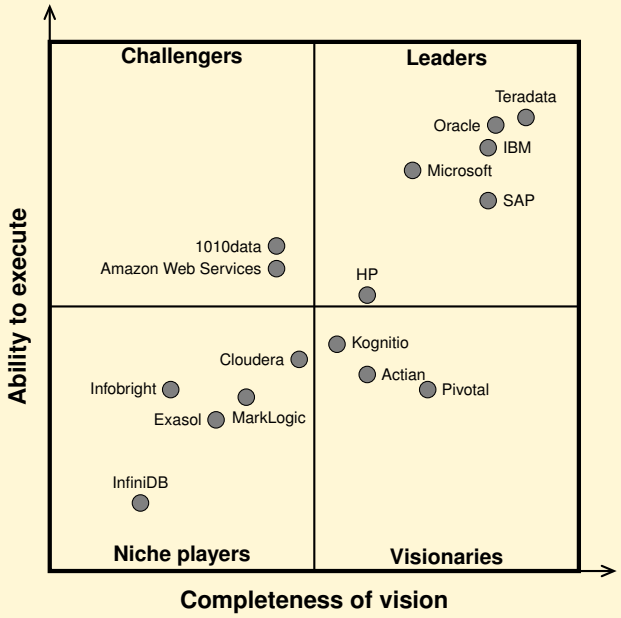


Figure 6.15: Commercial Systems for Data Warehouses

– *Ability to Execute*. This is a measure of the product and vendor quality, of the sales volume, and of the customer's level of satisfaction.
– *Completeness of Vision*. This is a measure of the ability of the vendor to satisfy the needs of current and future customers.

The company specifies that the *Magic Quadrant* is a graphical representation of a marketplace of some vendors in a given period according to Gartner's analysis of the market with its own criteria. The company does not visit any vendor, does not test any product or service included in the Magic Quadrant, and does not suggest that customers select a vendor that will appear in the Leaders quadrant. The Magic Quadrant is intended solely as a research tool and not as a specific guide to choosing a vendor. Gartner disclaims all warranties, express or implied, with respect to this research, including any warranties of merchantability and fitness for a particular purpose.

The following considerations, in a nutshell, were made about the analyzed products of the Leaders quadrant.

– Teradata has been in the data warehouse business for more than 30 years, delivering only data warehouse solutions that have always been a *data warehouse appliance*.
– Oracle, which prevails in the DBMS market, offers a solution for data warehousing, which, for quantity of data over 5 TB, requires a highly skilled DBA for tuning of the system. Customers can choose to build a custom warehouse, a certified configuration of Oracle products on Oracle-recommended hardware or on a database appliance with support for both OLTP and OLAP database systems. Oracle has announced in-memory columnar capability.

2. For the first time the Magic Quadrant consider also non-relational data management systems, such as Amazon Redshift, Cloudera, MarkLogic and Kognitio.

- IBM offers stand-alone DBMS solutions, as well as the data warehouse appliance Netezza, now known as the PureData System for Analytics, to support the combination of high speed multidimensional analysis and data mining.
- Microsoft offers SQL Server 2012, a reference architecture and the parallel data warehouse appliance with storage options to support 5 TB or up to 12 TB of user data and includes tools to enable easy integration of data from any source.
- SAP offers both SAP Sybase IQ and SAP Hana in-memory. Sybase IQ was the first *column-oriented* DBMS for data warehouses, i.e. it store database tables by columns rather than by rows, as happens in relational systems and other systems analyzed. SAP Sybase IQ is available as a stand-alone DBMS. SAP Hana in-memory uses a column-based storage in main memory.
- HP offers Vertica, another example of a *column-oriented* DBMS for data warehouses.

6.8 Summary

- Analytical queries can be made more efficient with the use of special-purpose indexes such as bitmap index, join index, bitmapped join index, and bitmapped foreign column join index.
- Optimizers for data warehouse systems generate particular physical plans for star queries, with a structure different from that generated by conventional relational systems. Examples of different solutions provided by two typical commercial systems have been shown: DB2 and Oracle.
- Several authors have demonstrated that, since a data warehouse system is query-intensive, an implementation based on a *column-oriented* storage system can achieve substantial improvements in OLAP query performance.

  To improve query performance of OLAP applications, besides the idea of changing the way data are stored – using large page sizes, compression techniques and specialized storage structures – new algorithms for generating physical query plans have been designed.
- New DW platforms, such as *In-memory* or *DW Appliances* provide cost-effective scalability and simplify big data warehouse implementations. Other modern technologies, such as Hadoop clusters and NoSQL databases, are also adopted for very big DW applications.
- The Gartner company periodically publishes a report, *Magic Quadrant for Data Warehouse Database Management Systems*, through which it proposes a classification of the best vendors of systems for data warehousing.

Chapter 7

MATERIALIZED VIEWS SELECTION

OLAP analysis requires the execution of statistical operations on large quantities of data, grouped by different criteria, to be performed very quickly to avoid jeopardizing the interest of users in interactive analysis of data from different perspectives. For this reason the systems for data warehouses include the use of materialized views with which the results of some queries are stored that are then automatically used to facilitate the execution of other more complex ones. After a presentation of issues to be resolved using this technique, we present some algorithms for selecting views to materialize.

7.1 Introduction

In traditional DBMS views are used frequently for various reasons and in particular to simplify the writing of complex SQL queries or to enable the writing of queries that are not expressible without the use of views. The use of views is explicit in the queries that then, when it is possible, are automatically rewritten by replacing the expression that defines the view. This approach increases the possibility of optimizing the query and reducing the query execution time.

A commonly used technique to improve query execution time on large data warehouses is to materialize (precompute) the result of some queries, in particular those that require grouping and calculation of aggregates. Also in this case a user query is rewritten to use materialized views, but, unlike the previous case, this is done automatically by the optimizer, without the user knowing of the materialized views' existence.

The use of materialized views requires the solution of three major problems:

1. The choice of views to materialize, and of the possible indexes on them. This is one of the most important decisions in designing a data warehouse.
2. The choice of the materialized views to use by *query rewrite*, an optimization technique that transforms a user query written in terms of basic tables into a semantically equivalent query that uses one or more materialized views.
3. The choice of when to update the views to align them with changes made on the database tables used.

In the following we will focus on some algorithms for solving the first problem, then in another chapter we shall see approaches for solving the second problem, while for the third we simply remember that two types of solutions have been studied, such as those for replicated data management in distributed databases:

- *Immediate Update*: When the reference tables are updated, the interested materialized views are updated too.
- *Deferred Update*: When the reference tables are updated, the events are stored in a log file, and then the views are updated in one of the following ways:
  - *Lazy Update*: When the view is used.
  - The views are updated periodically
  - The views are updated after a fixed number of updates of the reference tables.

The deferred update temporarily loses the alignment of views with the reference tables, but the problem is not critical because of the nature of the data warehouse applications.

7.2 The Lattice of Views

Let us consider a fact table with three dimensions and one measure m, without dimensional attributes. Let us assume that the business analyses are of the following types: Find the sum of the measure grouping data by some dimensions. The possible queries differ only for the *grouping attributes* and have the following structure in SQL:

```
SELECT    <Grouping attributes>, SUM(m) AS m
FROM      <Fact Table>
GROUP BY  <Grouping attributes>;
```

With three dimensions the possible views can be partially ordered in the *data warehouse lattice*, as is shown in Figure 7.1. The views are named using the abbreviations P for Product, S for Store, D for Date.

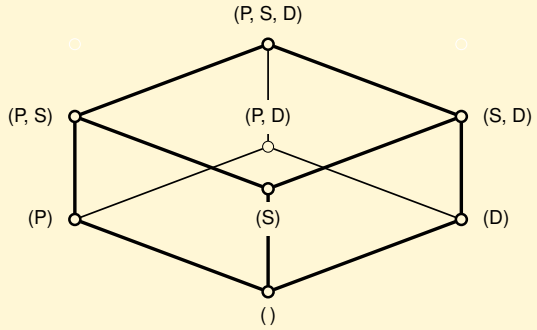


Figure 7.1: Lattice of possible views to materialize

The root node of the lattice represents the fact table, i.e. the grouping attributes are all the dimensions. The bottom view has the grouping attributes set empty. Views in higher lattice levels have more grouping attributes and are thus more *detailed* (hold data of finer granularity) than views in lower levels, which are more *specialized* (hold data of coarser granularity).

The lattice of possible views has an interesting property: if a node view has been materialized, then it can be used to compute the result of any query that groups only by a subset of the attributes view.[1]

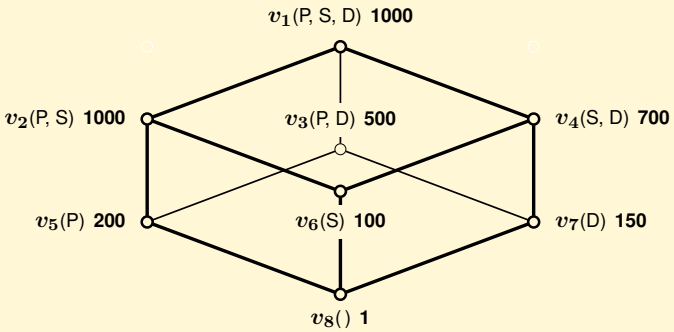
Suppose now that we are interested in the execution of a set of queries (the workload) which, for simplicity, coincides with the set of those defining the views of the lattice. To choose a set of views to materialize in order to facilitate the execution of the workload there are three possibilities:

1. *Materialize nothing*: the only data stored in the data warehouse are those of the fact table, i.e. the data associated with the lattice root, and therefore the result of each query can be computed from these data with a cost that depends on the facts table cardinality.
2. *Materialize all the views*: the result of any possible query has been already computed and stored as a materialized view, and so the best query response times are achieved. Obviously, the memory space occupied by all materialized views becomes very high in the presence of many data and many dimensions.

1. This is possible because of the assumption that the only aggregation function is the sum, which is distributive.

3. *Materialize only some of the views*: an appropriate subset of views to materialize is chosen to facilitate the execution of all queries.

Example 7.1
Let us see by an example why a full materialization of views is generally not convenient. Let us consider the lattice in the following figure, where the root is the fact table and the other nodes possible candidate views to materialize. Each node is labeled with the view grouping attributes. The numbers associated with the nodes represent the *view size*, measured in terms of the number of tuples in the view. These numbers are normally derived from a view size estimation algorithm. A given view can be calculated from any materialized ancestor view.



A full materialization of views will have a cost of 3 651. The only view that must necessarily be materialized is the fact table v_1. Suppose that a query q groups data by P, S. If the view v_2 has been materialized, the q execution requires a view scanning with a cost of 1 000. But v_2 has a cost equal to that of v_1, so therefore the query can be executed at the same cost using v_1 without materializing v_2.

The approach of partial materialization of views, which will be considered in the following, raises three interesting issues: (a) which views materialize, (b) which of them to use to execute a query and (c) how to update a view when the fact table is updated. For simplicity in the following the focus will be on the first two problems, because we assume that the fact table is static.

The problem will be studied following the approach presented in [Harinarayan et al., 1996] with the following assumptions:

– The views are defined in SQL with a GROUP BY, without restrictions and using only the aggregation function SUM on the measure m: ${}_X\gamma_{\mathsf{SUM(m)}}$, with X a set of dimensions.

  For simplicity, a view v is represented as ${}_X\gamma$, without the aggregation function, and $g(v)$ is the set of grouping attributes X. The order of attributes in X is irrelevant.

– The queries in the workload are equiprobable and are defined in SQL with a GROUP BY, a possible restriction on the dimensions (logical product of predicates Attr = constant), and the aggregation function SUM(m).

```
SELECT    <Grouping attributes>, SUM(m) AS m
FROM      <Fact Table>
WHERE     <Condition on some attributes>
GROUP BY  <Grouping attributes>;
```

  For brevity, a query q is represented as ${}_X\gamma\,\sigma_Y$, with X and Y disjoint set of attributes, and let $\mathcal{A}(q)$ be the set of attributes $X \cup Y$.

Each query q is associated with the minimum view of the lattice from which it can be computed. More precisely, between the queries and views is defined a *relationship computability* as follows:

Definition 7.1 *Relationship Computability*
Let q be a query and v a view; we say $q \ll v$ if q can be answered using the result of v. If $\mathcal{A}(q) = X \cup Y$ and $g(v) = Z$, $q \ll v$ if and only if $X \cup Y \subseteq Z$.

Given two views v_1 and v_2, and a query q, if $v_1 \preceq v_2$ and $q \ll v_1$, then $q \ll v_2$. Given a query q and a view v, if $q \ll v$, then v is called a *candidate view* to materialize.

7.3 View Sizes Estimation

The algorithms for the selection of the views to materialize assume that the size of the views in the lattice is known. Let us see some of the methods proposed in the literature to solve the problem [Shukla et al., 1996]:

1. **Analytic Approach.** Let us assume that the attributes values are uniformly distributed in the fact table F, and statistically independent, the view size $|v|$ of the view v defined as a grouping on attributes X of F, is estimated with Cardenas' formula:

$$|v| = n - n(1 - 1/n)^{|F|}$$

   where n is the number of possible values of X and $|F|$ is the fact table size.
   For example, let $R(A, B, C)$, $N_{\mathsf{rec}}(R)$ its size and $N_{\mathsf{key}}(A)$, $N_{\mathsf{key}}(B)$ the number of the attributes A and B distinct values. To estimate the view size of v on R, which groups on (A, B), we assume $|F| = N_{\mathsf{rec}}(R)$ and $n = N_{\mathsf{key}}(A) \times N_{\mathsf{key}}(B)$.

2. **A Sampling Approach.** The idea is to take a sample S of the fact table F, to estimate the view size v_s, and to estimate the size of the view v as:

$$|v| = |v_s| \times |F|/|S|$$

   This method is less accurate than the previous and tends to overestimate the size of a view as the number of duplicates in the data increases.

3. **Pareto Model Approach.** A recent proposal, more accurate than the previous approaches [Nadeau and Teorey, 2001].

7.4 A Greedy Algorithm for the Selection of Materialized Views

Following the approach presented in the interesting article [Harinarayan et al., 1996], we study the problem under the following assumptions (Figure 7.2):

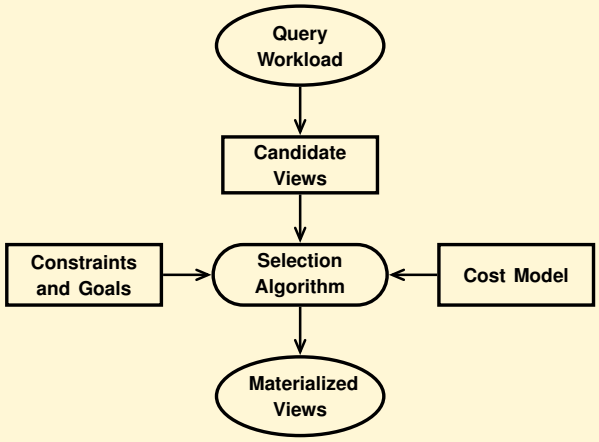


Figure 7.2: The Selection Process of the Views to Materialize

1. The set of candidate views to materialize are the lattice nodes, and the space cost $C(v_i)$ of each view (the number of rows in the view) v_i is known.
2. There are only k candidate views to materialize, different from the top view (the fact table) always materialized.
3. The workload is the set of queries that define the lattice views.
4. The dimensions are without attributes.

Let $C(q_i, M)$ be the cost of executing the query q_i using the set of materialized views M. The goal is to select the set of views which minimizes the overall execution cost of the queries Q, that is to say the quantity:

$$\tau(V, M) = \sum_{i=1}^{|Q|} C(q_i, M)$$

respecting the constraint that only k candidate views V can be materialized, along with the root of the lattice.

The optimization problem has been proved to be NP-complete. Therefore an approximate *greedy* solution to avoid an exhaustive search in the space of all possible solutions has been proposed that in each iteration calculates the *benefit* of the remaining nodes and selects for materialization the one with the maximum benefit.

The algorithm is iterative and terminates after k iterations. In each iteration, one more node is added to M, building the final result in steps. The choice of a node v in the i-th iteration depends on the quantity $B(v, M)$, which is called the benefit of v relative to M and is calculated as follows:

1. For each view $w \preceq v$ (i.e. w is v or one of its descendants), let v_m be the view of least cost in M such that $w \preceq v_m$. Note that since the top view is in M, there must be at least one such view in M. Then define B_w as $B_w = max\{C(v_m) - C(v), 0\}$.
2. $B(v, M)$ is defined as $B(v, M) = \sum_{w \preceq v} B_w$

In other words, the benefit of a view v is evaluated by considering how it can improve the cost of computing itself and all of its descendants $w \preceq v$.

For each descendant w of v, the cost of computing w using v is compared with the cheapest cost of computing w using some node v_m that already belongs to M. If v helps, which means that $C(v) < C(v_m)$, then the difference $C(v_m) - C(v)$ contributes to the total benefit, which is the sum of all such differences.

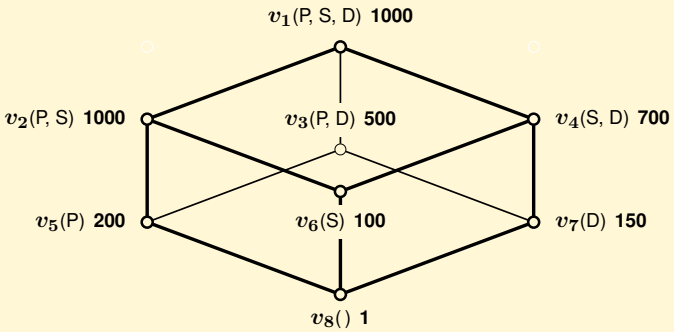
The *greedy* algorithm in Figure 7.3 is a solution of the problem (we refer to the algorithm described by Harinarayan, Rajaraman, and Ullman henceforth as HRU).

Algorithm HRU(k)

```
% Let v1 be the lattice root
M = {v1};
N = V − M;
for i = 1 to k
{ v = the view in N, such that B(v, M) the maximum;
  M = M ∪ {v};
  N = N − {v} };
return M ;
```

Figure 7.3: The *greedy* algorithm HRU to select k views to materialize

Example 7.2
Let us consider the lattice



The initial state for HRU has only the fact table v_1 materialized, with cost 1 000. HRU calculates the benefits of each possible view during each iteration, and selects the most beneficial view for materialization. Processing continues until k materialized views have been chosen.

	First Choice	Second Choice	Third Choice
$v_2(P,S)$	0	0	0
$v_3(P,D)$	$\mathbf{500 \times 4 = 2\,000}$		
$v_4(S,D)$	$300 \times 4 = 1\,200$	$300 \times 2 = 600$	$300 \times 1 = 300$
$v_5(P)$	$800 \times 2 = 1\,600$	$300 \times 2 = 600$	$300 \times 1 = 300$
$v_6(S)$	$900 \times 2 = 1\,800$	$\mathbf{900 + 400 = 1\,300}$	
$v_7(D)$	$850 \times 2 = 1\,700$	$350 \times 2 = 700$	$\mathbf{350 \times 1 = 350}$
$v_8()$	$999 \times 1 = 999$	$499 \times 1 = 499$	$99 \times 1 = 99$

When considering the view of v_2, since its cost is equal to that of v_1, there will be no benefit to using it. Considering instead v_3, if materialized, it reduces the associated query cost of $(1\,000 - 500)$ and also the three descendants benefit, which will be calculated by v_3 and not by v_1. Therefore, its benefit with respect to M is 500×4. When several views have the same benefit, the one that occupies less space is chosen. At the end of the first choice v_3 is also the most useful view to materialize.

In the second iteration, the calculation of the benefit of a view must consider whether its descendants can be calculated from v_1 or from v_3 already materialized. For example,

- In the calculation of the benefit of v_4, the descendant v_6 only is considered because v_7 and v_8 are computed from v_3, being $C(v_4) > C(v_3)$.
- In the calculation of the benefit of v_6, the descendant v_8 can be computed using v_3, but since $C(v_6) < C(v_3)$, the materialization of v_6 allows us to compute v_8 with the benefit $B_{v_8} = 500 - 100$, and so $B_{v_6} = 900 + 400$.

The algorithm ends with the solution $M = \{v_1, v_3, v_6, v_7\}$.

In general, the algorithm does not find the optimal solution, but the authors have shown that it provides good results and the following interesting properties hold [Harinarayan et al., 1996]:

Theorem 7.1
For each lattice, let B_{greedy} be the benefit of k views selected by the algorithm *greedy* and B_{opt} be the benefit of the optimum choice of k views, then B_{greedy} can never be less than $0,63 \times B_{\text{opt}}$.

7.5 Other Algorithms for the Choice of the Views to Materialize

By modifying the assumptions of the algorithm HRU other algorithms have been proposed for the selection of views to materialize [Morfonios et al., 2007]. Let us see some of them.

7.5.1 Algorithm PGA

The algorithm HRU has a time complexity $O(k \times n^2)$, with k the number of views to be selected and n the number of nodes of the lattice. Since the number of nodes depends exponentially on the number of dimensions d, the complexity of the algorithm is $O(k \times 2^{2d})$. An algorithm with polynomial time complexity on the number of dimensions is described in [Nadeau and Teorey, 2002] (*Polynomial Greedy Algorithm*, PGA).

The exponential complexity of HRU depends on two choices made for the calculation of a view benefit:

1. It considers *all* remaining views on the entire lattice that have not yet materialized, which are in the order of 2^d.
2. It considers for each view v *all* its descendants that benefit from the v materialization, which are again in the order of 2^d.

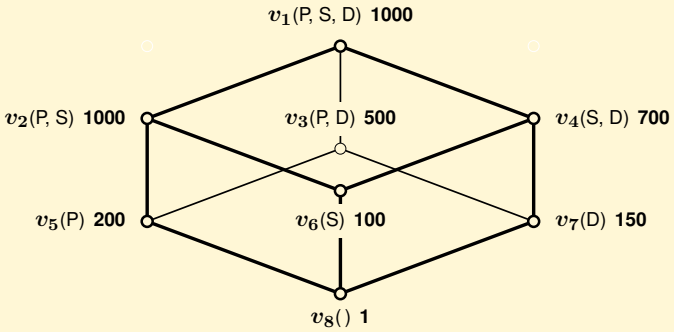
The basic idea of the algorithm PGA is to reduce the number of views to be considered in both cases proceeding during each iteration in two phases, called *nomination* and *selection*. In the first phase PGA nominates promising views into a candidate set, initially empty. The nomination phase begins at the root of the lattice and nominates the smallest view v from the children; then nominates the smallest child of v, and so on until the the bottom of the lattice is reached. Once a path of promising views has been nominated, the candidate set is considered for materialization in the second phase.

In the selection phase PGA estimates the benefits of materializing each candidate view, and selects the most beneficial one for materialization.

To compute the benefit of a candidate view v, the algorithm proceeds as follows:

- It is considered the smallest ancestor a_m of v in M.
- It considers whether there is a materialized view v_m in M such that v_m is smaller than v, v_m is not an ancestor of v and with the maximum number of descendants $d(v_m)$ in common in the lattice. Let $d = d(v) + 1 - d(v_m)$, where $d(v)$ is the number of descendants of v in the lattice. The values of $d(v)$ and $d(v_m)$ are computed in each nomination phase.
- The benefit of v is $(C(a_m) - C(v)) \times d$.

Example 7.3
Let us consider again the lattice



and let $k = 3$. The following table, for each iteration, shows the two algorithm's phases and the candidates' views. The nominated views at each iteration are marked with a $\surd$.

Views	Iteration 1 Selection		Iteration 2 Selection		Iteration 3 Selection	
$v_2(P,S)$					$\surd$	0
$v_3(P,D)$	$\surd$	$\mathbf{500 \times 4 = 2\,000}$				
$v_4(S,D)$			$\surd$	$300 \times 2 = 600$	$\surd$	$\mathbf{300 \times 2 = 600}$
$v_5(P)$					$\surd$	$300 \times 1 = 300$
$v_6(S)$			$\surd$	$\mathbf{900 \times 2 = 1\,800}$		
$v_7(D)$	$\surd$	$850 \times 2 = 1\,700$	$\surd$	$350 \times 2 = 700$	$\surd$	$350 \times 1 = 350$
$v_8()$	$\surd$	$999 \times 1 = 999$	$\surd$	$499 \times 1 = 499$	$\surd$	$99 \times 1 = 99$

At the first iteration PGA nominates the smallest node v_3 from amongst the children of the root of the lattice v_1. PGA then examines the children of v_3 and nominates the smallest child v_7. The process continues until the bottom of the lattice is reached. The candidate set is then $\{v_3, v_7, v_8\}$.

The selection phase evaluates each view v in the candidate set, and selects the one with the highest benefit if materialized. Since the smallest ancestor a_m of v in M is v_1, and there is no materialized view v_m in M smaller than v and with the maximum number of descendants $d(v_m)$ in common in the lattice, v_3 is selected with the highest benefit $(C(v_1) - C(v_3)) \times (d(v_3) + 1) = 500 \times 4$, and it is eliminated from the candidate set.

At the second iteration PGA nominates v_4 and v_6, and the candidate set becomes $\{v_4, v_6, v_7, v_8\}$.

In the selection phase for v_7 and v_8 the view a_m is v_3, materialized in the previous step, while for v_4 and v_6 the view a_m is v_1. In calculating the benefit of v_4, the smallest ancestor in M is v_1 and the view v_m is v_3 with two descendants in common, therefore $(C(v_1) - C(v_4)) \times (d(v_4) + 1 - d(v_m)) = 300 \times 2$. In calculating the benefit of v_6, v_7 and v_8, since there is no smallest view a_m to consider, v_6 is selected with the highest benefit, and it is eliminated from the candidate set.

At the third iteration the candidate views are $\{v_2, v_4, v_5, v_7, v_8\}$. v_4 is selected and the result is $M = \{v_1, v_3, v_6, v_4\}$, different from that found previously with the algorithm HRU.

In general, the algorithms HRU and PGA do not find the same set M, because they use different heuristics. In addition, for the algorithm PGA the existence of a lower limit on the quality of the solution has not been proved, as for the HRU, but it has been shown experimentally that, as the number of dimensions increases, the algorithm finds solutions with a total benefit not much lower than that found by HRU, but in less time.

7.5.2 Algorithm with an Upper Bound on the Memory Available

Let us assume that instead of having a limit on the number of views k that can be materialized, there is an upper bound on the total storage space S that the set of materialized views M can occupy [Harinarayan et al., 1996].

Let $S(v) = |v|$ be the space occupied by the view v. The algorithm in Figure 7.4 (*Benefit for Unit of Space*), in the i-th iteration chooses a view not based on the absolute benefit $B(v, M)$, but on the benefit per space-unit that v occupies $B(v, M)/|v|$.

Algorithm BPUS(S)

```
% Let v1 the lattice root, which is always stored
M = {v1};
N = V − M;
while S > 0
{ v = the view in N with the maximum benefit Bs;
  if (S − S(v) > 0) then
     {S = S − S(v);
      M = M ∪ {v};
      N = N − {v}};
  else S = 0;
}return M;
```

Figure 7.4: A *greedy* algorithm with an upper bound on the total storage space occupied by the materialized views

In [Shukla et al., 1998] a variation of BPUS to reduce its complexity has been proposed, shown in Figure 7.5 (*Pick By Size*), with complexity $O(n \log n)$.

It has been proved that for lattices called *Size Restricted* BPS and BPUS produce the same result. A lattice is *Size Restricted* when for each view v with k children, for each ancestor z the following relation holds

$$|z| \geq (1 + k) \times |v|$$

when $|z| \neq |v_1|$. For example, the algorithm applied to the previous example, with $S = 700$, finds the solution $M = \{v_1, v_8, v_6, v_7, v_5\}$.

Algorithm PBS(S)

```
% Let v1 the lattice root, which is always stored
M = {v1};
N = V − M;
while S > 0
{ v = the smallest view in N;
  if (S − S(v) > 0) then
     {S = S − S(v);
      M = M ∪ {v};
      N = N − {v} };
  else S = 0;
}return M;
```

Figure 7.5: Another *greedy* algorithm with an upper bound on the total storage space occupied by the materialized views

7.5.3 Algorithm for a Particular Workload

If the queries of the workload are not equally likely, and then some views are more used than others, the algorithm HRU can be easily modified in the step of calculating the benefit, weighting each view with its likelihood of being used.

If the workload Q is not the set of queries defining the views of the lattice, each query has a natural view from which it is most easily answered. In [Baralis et al., 1997] it has been shown that it is not necessary to apply the selection algorithm to the set of lattice nodes, but it is enough to consider only the so-called *candidate views* set, defined as follows.

Definition 7.2

Given a workload Q, a lattice node v belongs to the candidate-view set, if it satisfies one of the following two conditions:

1. There is a query $q \in Q$ such that $\mathcal{A}(q) = g(v)$.
2. There are two candidate views v_i and v_j such that $g(v) = g(v_i) \cup g(v_j)$, with $g(v_i) \cup g(v_j)$ the attributes of the most specialized common ancestor v_i and v_j.

An algorithm to compute the set C of candidate views is shown in Figure 7.6.

Algorithm C(Q)

% Top is the set of attributes of the lattice root
C = the set of $g(v)$ such that $g(v) = \mathcal{A}(q)$ for each $q \in Q$;
while (changes to C)
{ **for** $g(v_i) \in C$ **with** $g(v_i) \neq Top$
{**for** $g(v_j) \in C$ **with** $g(v_j) \neq Top$
and $g(v_j) \neq g(v_i)$ **and** $g(v_i) \cup g(v_j) \notin C$
{ $C = C \cup g(v_i) \cup g(v_j)$}};
}**return** C;

Figure 7.6: An algorithm to compute the set of candidate views

Example 7.4

Let us consider the lattice in Figure 7.7a. Let the workload Q be the following queries on the fact table v_1:

- $q_1 : {}_P\gamma$
- $q_2 : {}_D\gamma$
- $q_3 : {}_D\gamma\,\sigma_S$

C is initialized to the set of nodes associated with the queries in Q:

- $g(v_5) = \mathcal{A}(q_1) = \{P\}$
- $g(v_7) = \mathcal{A}(q_2) = \{D\}$
- $g(v_4) = \mathcal{A}(q_3) = \{S, D\}$

C is extended with the nodes that are the most specialized common ancestors of node pairs that already belong to C.

$$\{P\} \cup \{D\} = \{P, D\}$$

The algorithm terminates when C has not changed in the last iteration, with the result of Figure 7.7b.

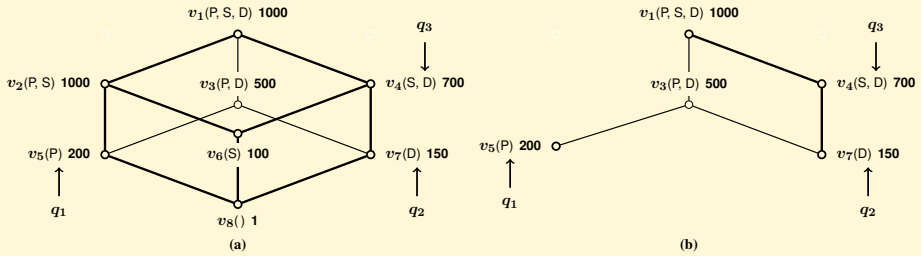


Figure 7.7: Candidate views for the workload Q New

Once the set of candidates views has been defined, we then proceed with one of the algorithms previously seen for the choice of the candidate views to materialize.

7.5.4 Algorithm for Dimensional Attributes

The presence of dimensional attributes increases the number of potential grouping attributes, and therefore an exponential increase in the number of nodes in the lattice. However, since the dimensional attributes are functionally dependent on the key of the corresponding table, some of the nodes of the lattice are redundant. For example, if a dimensional table has the key K and the attribute A, the grouping on (K) produces the same groups as a grouping on (K, A).

Example 7.5
Let us consider the fact table F with the foreign key A for the dimensional table T:

F(D, A)
T(A, B, C)

Figure 7.8a shows the views lattice with the natural join $F \bowtie T$ as root, and Figure 7.8b shows how the lattice is simplified because of the functional dependency $A \rightarrow BC$.

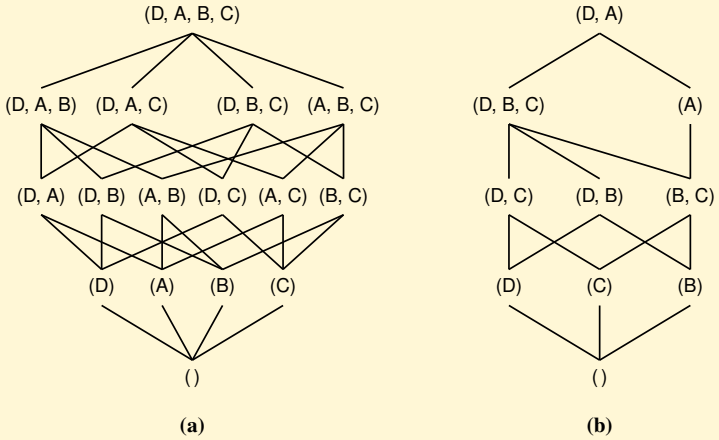


Figure 7.8: Lattice with dimensional attributes

7.5.5 Algorithm for Dimensional Attributes and Hierarchies

For simplicity, let us assume that total ordering only exists over the dimension hierarchies, that is, each attribute has at most one child. A hierarchy among a more specific attribute s and another more general g of a table is equivalent to a functional dependence $s \rightarrow g$.

The presence of hierarchies between dimensional attributes can remove some elements from the lattice. For instance, if there is the hierarchy $s \rightarrow g$, a view defined with grouping attributes (s, g) is redundant, because a grouping on (s, g) produces the same groups as a grouping on (s).

Example 7.6
Let us consider the fact table F with attribute A the external key for the dimensional table T, which is defined with the hierarchy $B \rightarrow C$:

F(D, A)
T(A, B, C)

Figure 7.9a shows the lattice of views without considering the hierarchy of $B \rightarrow C$, and Figure 7.9b shows how the lattices change by considering the dimensional hierarchy.

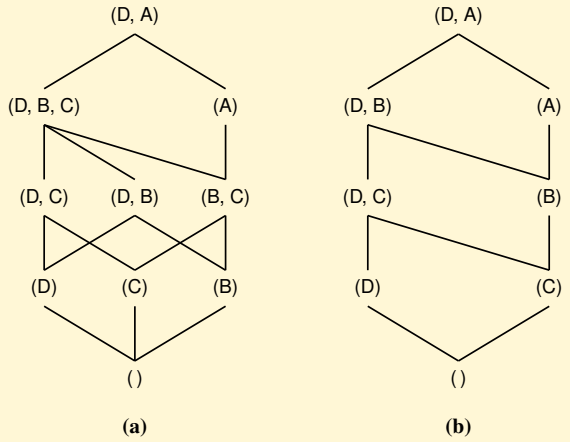


Figure 7.9: Lattice in the presence of hierarchies with total ordering

7.6 The Selection of Indexes on Materialized Views

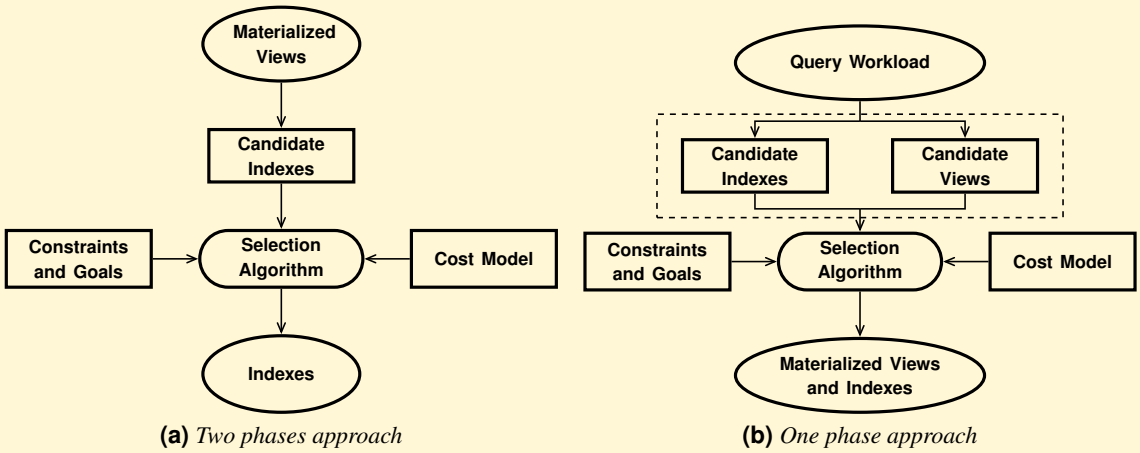
In general to reduce the execution time of the queries that use materialized views it is useful to put one or more indexes on a view.

For example, let us consider the execution of the query $q = \sigma_B$ using a view v with attributes (A, B). Without indexes, let us assume that the execution cost estimate is the number of records $|v(A, B)|$. If there is the index I_B on v, the execution cost estimate, without considering the number of index nodes processed, becomes $|v(A, B)|/N_{key}(I_B) = |v(A, B)|/|\pi_B(v)|$.

Two approaches can be adopted for the selection of both the materialized views and the indexes on them:

1. *Two steps approach*: The selection of materialized views is made first, and then the selection of indexes is made using the techniques adopted for selecting indexes on a relational database (Figure 7.10a).

2. *One step approach*: Indexes and materialized views selection is made together because the two problems can interact with one another, i.e., the presence of an index can make a materialized view more convenient and vice versa (Figure 7.10b).



(a) *Two phases approach* **(b)** *One phase approach*

Figure 7.10: Approaches for indexes and materialized views selection

In [Gupta et al., 1997] an example is shown of the poor performance of the two-step approach, and a family of one-step greedy algorithms, given space constraints, that produce better solutions. A similar proposal for the system Microsoft SQL Server 2000 is presented in [Agrawal et al., 2000].

7.7 Summary

- Instead of treating a view as a query on the database, the technique of materialized views plans to store the result of the query that defines it. Then to execute a complex query it is possible that a materialized view exists such that some or all of the more complex operations have already been performed. Therefore, using the previously stored data, the query can be appropriately rewritten automatically to use the view, without the user knowing of the existence of materialized views, with significant improvements in response times.
- The choice of the views to materialize, and the indexes on them, by the data base administrator, or automatically by the system according to the type of use that is made of data, requires the use of appropriate algorithms.
- The basic algorithm HRU has been presented, and some of its interesting extensions for the selection of views to materialize.

Chapter 8

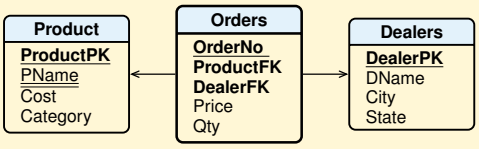
OPTIMIZATION OF STAR QUERIES WITH GROUPING

The data warehouse systems, because of the quantities of data managed and the different sizes of fact and and dimensional tables, require new techniques for star query optimization. In particular, new specific solutions have been studied for the following problems: (a) how to optimize the execution of the GROUP BY and (b) how to rewrite queries to generate query plans using materialized views. In the following we consider the first problem and the second one is considered in the next chapter.

8.1 Introduction

The standard way to evaluate queries with group-by is to perform the joins first and then the group-by. However, several authors have shown that the optimizer should also consider doing the group-by before the join to produce cheaper access plans [Chaudhuri and Shim, 1994], [Yan and Larson, 1995], [Galindo-Legaria and Joshi, 2001], [Tsois and Sellis, 2003]. We will considered the three cases studied in [Chaudhuri and Shim, 1994], [Galindo-Legaria and Joshi, 2001], for which the authors have given sufficient conditions for doing the group-by first on the basis of the query structure. In what follows, we will first reformulate and revise the conditions, and then we will show how they can be used to optimize star queries that exploits the benefits of bringing forward the group-by on the fact table.

The examples will be given using the following simple star schema, where the keys are underlined (different keys of the same table are underlined differently):



The sufficient conditions to bring forward the group-by are given in the form of equivalence rules of relational algebra. The optimizer considers two ways to evaluate the group-by from the global rowset: the first one evaluates the group-by using the result of the join operator, the second evaluates the group-by before the join operator. Based on the cost estimates of the two alternative query plans, the optimizer will then decide which one to use to evaluate the query.

Before presenting the equivalence rules that are useful for bringing forward the group-by, let us give the reader a brief review of some basic properties both of functional dependencies and of the group-by operator that will be also used in the next chapter.

8.2 Properties of Functional Dependencies and of the Group-by Operator

For simplicity, we assume that

1. The database tables are without null values and are *sets* because have been defined with keys.
2. The **FROM** clause uses a set of tables $\mathcal{R}$, where no attribute appears in two different tables.
3. The condition C in the **WHERE** clause is a conjunctive normal form (CNF) of predicates involving attributes of the tables in $\mathcal{R}$.
4. All the grouping attributes are present in the **SELECT** clause.

We will use a basic algorithm of functional-dependency theory for determining if an *interesting* functional dependency can be inferred from the set F of functional dependencies which hold in the result of a query.

Let us briefly recall some basic properties of functional dependencies.

Definition 8.1 *Functional Dependency*
Given a relation schema R and X, Y subsets of attributes of R, a functional dependency $X \rightarrow Y$ (X *determines* Y) is a constraint that specifies that for every possible instance r of R and for any two tuples $t_1, t_2 \in r$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$.

A peculiar example of functional dependency, that will be used in the following, is $\emptyset \rightarrow Y$: *it specifies that the value of* Y *is the same for every tuple of an instance* r *of* R.

Example 8.1
Let us consider the following relation containing information about students and exams at the University of Pisa.

StudentsExams

StudCode	Name	City	Region	BirthYear	Subject	Grade	University
1234567	N1	C1	R1	1995	DB	30	Pisa
1234567	N1	C1	R1	1995	SE	28	Pisa
1234568	N2	C2	R2	1994	DB	30	Pisa
1234568	N2	C2	R2	1994	SE	28	Pisa

Let us see if the following functional dependencies are properties of the meaning of the attributes in the relation:

- StudCode $\rightarrow$ Name, City, Region, BirthYear holds because each student code has the same name, city, region and birth year.
- City $\rightarrow$ Region holds because each city is in a region.
- StudCode, Subject $\rightarrow$ Grade holds because each students receive one grade in each subject.
- $\emptyset \rightarrow$ University holds because the data are only about students and exams at the University of Pisa.
- Subject $\rightarrow$ Grade does not hold because a subject may have different grades.
- Subject $\rightarrow$ Subject is not useful because always holds, and it is called "trivial".

Given a set F of functional dependencies, we can prove that certain other ones also hold. We say these ones are *logically implied* by F.

Definition 8.2 *Logical Implication*
Given a set F of functional dependencies on a relation schema R, another functional dependency $X \rightarrow Y$ is *logically implied* by F if every instance of R that satisfies F also satisfies $X \rightarrow Y$:

$$F \vdash X \rightarrow Y$$

This property holds if $X \rightarrow Y$ can be derived by F using the following set of inference rule, known as the *Armstrong's axioms*:

$$\begin{aligned} (\textit{Reflexivity}) \quad & \text{If } Y \subseteq X, \text{ then } X \rightarrow Y \\ (\textit{Augmentation}) \quad & \text{If } X \rightarrow Y, Z \subseteq T, \text{ then } XZ \rightarrow YZ \\ (\textit{Transitivity}) \quad & \text{If } X \rightarrow Y, Y \rightarrow Z, \text{ then } X \rightarrow Z \end{aligned}$$

A simpler way of solving the implication problem follows from the following notion of *closure* of an attribute set.

Definition 8.3 *Closure of an Attribute Set*
Given a schema $R < T, F >$, and $X \subseteq T$, the *closure* of X, denoted by X^+, is

$$X^+ = \{A_i \in T | F \vdash X \rightarrow A_i\}$$

The closure of attributes is used in the following result.

Theorem 8.1
$F \vdash X \rightarrow Y$ iff $Y \subseteq X^+$

Thus testing whether a functional dependency $X \rightarrow Y$ is implied by F can be accomplished by computing the closure X^+. The following simple algorithm can be used to compute this set, although a more efficient and complex one exists.[1]

Algorithm 8.1 *Computing the Closure of an Attribute Set X*

$X^+ = X$;
while (changes to X^+) **do**
    for each $W \rightarrow V$ **in** F **with** $W \subseteq X^+$ **and** $V \not\subseteq X^+$
        do $X^+ = X^+ \cup V$;

Functional dependencies are commonly used to normalize a database schema, but in the following we will show that they are also useful to reason about the properties of a query result.

A set of functional dependencies F which hold in the result of a query with joins and selections is found as follows:

1. Let F be the initial set of functional dependencies where their determinants are the keys of every table used in the query.
2. Let C the condition of the $\boldsymbol{\sigma}$ operator. If a conjunct of C is a predicate $A_i = c$, where c is a constant, F is extended with the functional dependency $\emptyset \rightarrow A_i$.
3. If a conjunct of C is a predicate $A_j = A_k$, e.g., a join condition, F is extended with the functional dependencies $A_j \rightarrow A_k$ and $A_k \rightarrow A_j$.

Note that the following properties hold.

- If $X \rightarrow Y$ in R or in S, then this is still the case in $R \times S$.
- If X is a key for R and Y is a key for S then $X \cup Y$ is a key for $R \times S$.

1. For an example see the application implemented by R. Orsini available at this URL: http://dblab.dsi.unive.it:8080

A way for determining if an *interesting* functional dependency $X \rightarrow Y$ may be inferred from the set F is to use the previous algorithm to compute X^+. Instead, in the following examples we will use the following version of the algorithm to compute X^+, which does not use explicitly the functional dependencies.

Algorithm 8.2 *Computing the Closure of an Attribute Set X*

1. Let $X^+ = X$.
2. Add to X^+ all attributes A_i such that the predicate $A_i = c$ is a conjunct of $\boldsymbol{\sigma}$, where c is a constant.
3. Repeat until X^+ is changed:
   (a) Add to X^+ all attributes A_j such that the predicate $A_j = A_k$ is a conjunct of $\boldsymbol{\sigma}$ and $A_k \in X^+$.
   (b) Add to X^+ all attributes of a table if X^+ contains a key for that table.

8.2.1 Properties of the Group-by Operator

We consider only some basic equivalence rules on relational algebra expressions involving the group-by operator which will also be useful for proving more complex ones later on. For the sake of brevity, we only give an informal justification of these equivalence rules.

1. If Y contains the grouping attributes X and the aggregation attributes in F, then

$$ {}_X\boldsymbol{\gamma}_F(\boldsymbol{\pi}^b_Y(E)) \equiv {}_X\boldsymbol{\gamma}_F(E) \tag{8.1}$$

   The equivalence follows from the hypothesis.
2. A restriction can be moved before the grouping operator in the following cases.

   (a) If θ uses only attributes from X and F is a set of aggregate functions that use only attributes from E, then

$$ \boldsymbol{\sigma}_\theta({}_X\boldsymbol{\gamma}_F(E)) \equiv {}_X\boldsymbol{\gamma}_F(\boldsymbol{\sigma}_\theta(E)) \tag{8.2}$$

   The restriction of the left hand side eliminates a record r from the $\boldsymbol{\gamma}$ result if and only if it eliminates all the records from the group that has generated r.

   (b) If X and B are attributes from E, with $B \notin X$, v is a B value and MIN *is the only aggregate function*, then

$$ \boldsymbol{\sigma}_{\mathsf{mB}<v}({}_X\boldsymbol{\gamma}_{\mathsf{MIN}(B)\,\mathbf{AS}\,\mathsf{mB}}(E)) \equiv {}_X\boldsymbol{\gamma}_{\mathsf{MIN}(B)\,\mathbf{AS}\,\mathsf{mB}}(\boldsymbol{\sigma}_{B<v}(E)) \tag{8.3}$$

   If the restriction of the left hand side eliminates a record r such that the attribute value $r.\mathsf{mB}$ is the minimum among those of its group, then each record r_i of the group has $r_i.B \geq r.\mathsf{mB} \geq v$, and therefore all of them will be eliminated by the restriction of the right hand side. The equivalence also holds if $<$ is $\leq$.

   (c) If X and B are attributes from E, with $B \notin X$, v is a B value and MAX *is the only aggregate function*, then

$$ \boldsymbol{\sigma}_{\mathsf{MB}>v}({}_X\boldsymbol{\gamma}_{\mathsf{MAX}(B)\,\mathbf{AS}\,\mathsf{MB}}(E)) \equiv {}_X\boldsymbol{\gamma}_{\mathsf{MAX}(B)\,\mathbf{AS}\,\mathsf{MB}}(\boldsymbol{\sigma}_{B>v}(E)) \tag{8.4}$$

   If the restriction of the left hand side eliminates a record r such that the attribute value $r.\mathsf{MB}$ is the maximum of those of its group, then each record r_i of the group has $r_i.B \leq r.\mathsf{MB} \leq v$, and therefore all of them will be eliminated by the restriction of the right hand side. The equivalence also holds if $>$ is $\geq$.

3. If X and Y are attributes from E, $Y \not\subseteq X$, $X \rightarrow Y$ and F is a set of aggregate functions that use only attributes from E, then

$$ {}_X\boldsymbol{\gamma}_F(E) \equiv \boldsymbol{\pi}^b_{X \cup F}({}_{X \cup Y}\boldsymbol{\gamma}_F(E)) \tag{8.5}$$

   Each record of a group that has generated a record r of the $\boldsymbol{\gamma}$ in the left hand side belongs to the same group that has generated a record r of the $\boldsymbol{\gamma}$ in the right hand side.
4. An aggregate function f is called *decomposable* if there is a *local* aggregate function f_l and a *global* aggregate function f_g, such that for each multiset V and for any partition of it $\{V_1, V_2\}$ we have

$$ f(V_1 \cup^{all} V_2) = f_g(\{f_l(V_1), f_l(V_2)\}) $$

   where $\cup^{all}$ is the SQL's union-all operator without duplicate elimination.
   For example, SUM, MIN, MAX and COUNT are decomposable:

   - SUM($V_1 \cup^{all} V_2$) = SUM({SUM(V_1), SUM(V_2)})
   - MIN($V_1 \cup^{all} V_2$) = MIN({MIN(V_1), MIN(V_2)})
   - MAX($V_1 \cup^{all} V_2$) = MAX({MAX(V_1), MAX(V_2)})
   - COUNT($V_1 \cup^{all} V_2$) = SUM({COUNT(V_1), COUNT(V_2)})[2]

   If the aggregate functions in F are decomposable, X and Y are attributes from E, $Y \not\subseteq X$, $X \not\rightarrow Y$, then

$$ {}_X\boldsymbol{\gamma}_F(E) \equiv {}_X\boldsymbol{\gamma}_{F_g}({}_{X \cup Y}\boldsymbol{\gamma}_{F_l}(E)) \tag{8.6}$$

   Adding new grouping attributes to the internal $\boldsymbol{\gamma}$ of the right hand side produces smaller groups on which the local aggregate functions are computed, but then external $\boldsymbol{\gamma}$ combines these partial results to compute the global aggregate functions, and the result is the same as the $\boldsymbol{\gamma}$ of the left hand side.
5. Let $A \notin X$ be an attribute from E such that $X \rightarrow A$ and F is a set of aggregate functions that use A, renamed with **AS** Ide, then

$$ {}_X\boldsymbol{\gamma}_F(E) \equiv \boldsymbol{\pi}^b_{X \cup Z}({}_{X \cup \{A\}}\boldsymbol{\gamma}_{\text{COUNT}(*)\ \textbf{AS}\ \text{GBCount}}(E)) \tag{8.7}$$

   where Z is a set of expressions such that

   - $A \times$ GBCount **AS** Ide $\in Z$, if $f(A) \in F$ is SUM(A) **AS** Ide,
   - A **AS** Ide $\in Z$, if $f(A) \in F$ is MIN(A) **AS** Ide, MAX(A) **AS** Ide, AVG(A) **AS** Ide,
   - GBCount **AS** Ide $\in Z$, if $f(A) \in F$ is COUNT(A) **AS** Ide.

   The groups generated from the $\boldsymbol{\gamma}$ in the right hand side are exactly the same as those generated by the $\boldsymbol{\gamma}$ in the left hand size, and they have equal A values. Consequently the aggregate functions $f \in F$ can be removed from the $\boldsymbol{\gamma}$ in the right hand side and their values can be computed with the $\boldsymbol{\pi}$ using the given rules since the a value is the same in each group.

   As a special case, the equivalence rule also applies when $A \in X$.
6. If $X = X_{E_1} \cup X_{E_2}$, with X_{E_1} and X_{E_2} not empty sets of attributes from E_1 and E_2, X_{E_2} a superkey of E_2 and F is a set of aggregate functions that use only attributes from E_1, then

$$ {}_X\boldsymbol{\gamma}_F(E_1 \times E_2) \equiv \boldsymbol{\pi}^b_{X \cup F}(({}_{X_{E_1}}\boldsymbol{\gamma}_F(E_1)) \times E_2) \tag{8.8}$$

2. AVG is not decomposable according the given definition, but it can be computed with a different rule from two local functions SUM and COUNT:
   AVG($V_1 \cup^{all} V_2$) = SUM({SUM(V_1), SUM(V_2)}) / SUM({COUNT(V_1), COUNT(V_2)})

Let n_1 and n_2 be the number of different values of the grouping attributes in E_1 and E_2, with $n_2 = |E_2|$. For each value $X^i_{E_1}$ of X_{E_1}, if k_i records have the same values of $X^i_{E_1}$, in $E_1 \times E_2$ there are $k_i \times n_2$ records, and k_i of them belong to the same group because they have the same values of X. The result has $n_1 \times n_2$ records.
The same records are generated by the right hand side expression where first the relation W is computed with n_1 different records by grouping E_1 on X_{E_1} and then $W \times E_2$ is computed to get $n_1 \times n_2$ records.

8.3 First Case: Invariant Grouping

This first case for doing the group-by before a join is called *invariant grouping* since the operator can be brought forward without modifications, but the transformation may require an additional projection to produce the final result.

Let $\mathcal{A}(\alpha)$ be the set of attributes in α and $R \overset{\bowtie}{\underset{C_j}{}} S$ an equi-join using the primary key p_k of S and the foreign key f_k of R.

Theorem 8.2 *Invariant Grouping*
R has the *invariant grouping* property

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{C_j}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}(({}_{X \cup \mathcal{A}(C_j) - \mathcal{A}(S)}\boldsymbol{\gamma}_F(R)) \overset{\bowtie}{\underset{C_j}{}} S) \tag{8.9} $$

if the following conditions are true:

1. $X \rightarrow f_k$, i.e. the foreign key of R is functionally determined by the grouping attributes X in $R \overset{\bowtie}{\underset{C_j}{}} S$.
2. Each aggregate function in F uses only attributes from R.

Proof

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv {}_X\boldsymbol{\gamma}_F(\boldsymbol{\sigma}_{f_k=p_k}(R \times S)) $$

if in $R \overset{\bowtie}{\underset{f_k=p_k}{}} S$, $X \rightarrow f_k$ holds (condition 1), then $X \rightarrow p_k$ because $f_k = p_k$ and so using rule 8.5:

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}({}_{X \cup \{f_k, p_k\}}\boldsymbol{\gamma}_F(\boldsymbol{\sigma}_{f_k=p_k}(R \times S))) $$

then using rule 8.2:

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}(\boldsymbol{\sigma}_{f_k=p_k}({}_{X \cup \{f_k, p_k\}}\boldsymbol{\gamma}_F(R \times S))) $$

since each attribute in F is from R (condition 2), then using rule 8.8:

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}(\boldsymbol{\sigma}_{f_k=p_k}(\boldsymbol{\pi}^b_{X \cup \{f_k, p_k\} \cup F}(({}_{X \cup \{fk\} - \mathcal{A}(S)}\boldsymbol{\gamma}_F(R)) \times S))) $$

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}(\boldsymbol{\pi}^b_{X \cup \{f_k, p_k\} \cup F}(\boldsymbol{\sigma}_{f_k=p_k}(({}_{X \cup \{fk\} - \mathcal{A}(S)}\boldsymbol{\gamma}_F(R)) \times S))) $$

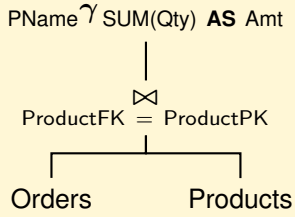
Finally, since the internal $\boldsymbol{\pi}$ is superfluous, we end up with:

$$ {}_X\boldsymbol{\gamma}_F(R \overset{\bowtie}{\underset{f_k=p_k}{}} S) \equiv \boldsymbol{\pi}^b_{X \cup F}(({}_{X \cup \{f_k\} - \mathcal{A}(S)}\boldsymbol{\gamma}_F(R)) \overset{\bowtie}{\underset{f_k=p_k}{}} S) $$ ■

Example 8.2
Consider the query

```
SELECT    PName, SUM(Qty) AS Amt
FROM      Orders, Products
WHERE     ProductFK = ProductPK
GROUP BY  PName;
```

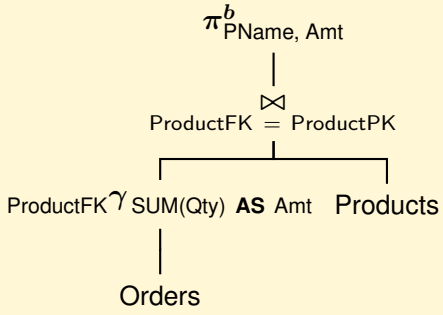
represented with the logical tree



To decide whether Orders has the invariant grouping property, since the Condition 2 holds, let us check whether PName $\rightarrow$ ProductFK, with PName a key in Products:

$$\begin{aligned} \text{PName}^{+} &= \{\text{PName,ProductPK, Cost, Category}\} \\ &= \{\text{PName,ProductPK, Cost, Category, ProductFK}\} \end{aligned}$$

Since PName^{+} contains ProductFK, PName $\rightarrow$ ProductFK holds and the logical query tree can be transformed as follows:



Observation
The invariant grouping property can also be exploited in the following cases to do a group-by before the join:

– *Having*. An SQL query with a HAVING clause, is represented with the following algebraic expression:

$$\sigma_{\theta}({}_{X}\gamma_{F}(R \overset{\bowtie}{}_{C_j} S))$$

We assume that (a) the restriction cannot be moved before the grouping operator using the equivalence rules 8.2, 8.3 or 8.4, and (b) the conjunctive terms in θ that do not contain aggregate functions have been moved in a restriction before the grouping operator using the equivalence rules 8.2. If

the invariant grouping property holds, then the restriction due to the HAVING clause can be brought forward together with the γ operator.

– *Star query*. Let R be the fact table and S_i a set of n dimensional tables joined with R:

$$R \underset{C_{j_1}}{\bowtie} S_1 \underset{C_{j_2}}{\bowtie} S_2 \ldots \underset{C_{j_n}}{\bowtie} S_n$$

We represent the star query in the form

$$R \underset{C_j}{\bowtie} \{S_1, \ldots, S_n\}$$

with $C_j = C_{j_1} \wedge C_{j_2} \cdots \wedge C_{j_n}$.

R has the *invariant grouping* property

$$_X\boldsymbol{\gamma}_F(R \underset{C_j}{\bowtie} \{S_1, \ldots, S_n\}) \equiv \boldsymbol{\pi}^b_{X \cup F}(({}_{X \cup \mathcal{A}(C_j) - \mathcal{A}(\{S_1,\ldots,S_n\})}\boldsymbol{\gamma}_F(R)) \underset{C_j}{\bowtie} \{S_1, \ldots, S_n\})$$

if the following conditions are true:

1. $X \rightarrow f_{k_1} f_{k_2} \ldots f_{k_n}$, with $f_{k_1} f_{k_2} \ldots f_{k_n}$ the foreign keys of R for the dimensional tables used in the joins.
2. Each aggregate function in F uses only attributes from R.

8.4 Second Case: Double Grouping

The invariant grouping property is interesting in terms of its simplicity, but its applicability restrictions limit its use. For example, in the following query it is not possible to move the group-by because the condition 1 of Proposition 8.2 does not hold:

```
SELECT      Category, SUM(Qty) AS Amt
FROM        Orders, Products
WHERE       ProductFK = ProductPK
GROUP BY    Category;
```

The following theorem gives a sufficient condition to move the group-by on the fact table in this kind of queries as well, but then another group-by is required to compute the final aggregations [Chaudhuri and Shim, 1994], [Galindo-Legaria and Joshi, 2001], [Yan and Larson, 1995]. The group-by operation is therefore done in two stages: the first one creates smaller groups and does part of the aggregations, then the second stage combines multiple groups into a single one to get the final result. Such a staged aggregation requires that the aggregate functions satisfy the following property.

Definition 8.4 *Early Partial Aggregation*
In ${}_X\boldsymbol{\gamma}_F(R \underset{C_j}{\bowtie} S)$, R has the *early partial aggregation* property if all the aggregate functions are *decomposable* and they use attributes from R.

Theorem 8.3 *Double Grouping*
If R does not have the invariant grouping property because Condition 1 does not hold, but it has the *early partial aggregation* property, then

$$_X\boldsymbol{\gamma}_F(R \underset{C_j}{\bowtie} S) \equiv {}_X\boldsymbol{\gamma}_{F_g}(({}_{X \cup \mathcal{A}(C_j) - \mathcal{A}(S)}\boldsymbol{\gamma}_{F_l}(R)) \underset{C_j}{\bowtie} S) \tag{8.10}$$

Proof
For the early partial aggregation property and the rule 8.6, with $B = A \cup \{f_k\}$:

$$ {}_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv {}_X\gamma_{F_g}\ ({}_{X\cup\{f_k\}}\gamma_{F_l}(R \underset{f_k=p_k}{\bowtie} S)) $$

for Theorem 8.2

$$ {}_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv {}_X\gamma_{F_g}(\pi^b_{X\cup\{f_k\}\cup F_l}(({}_{X\cup\{f_k\}-\mathcal{A}(S)}\gamma_{F_l}(R)) \underset{f_k=p_k}{\bowtie} S)) $$

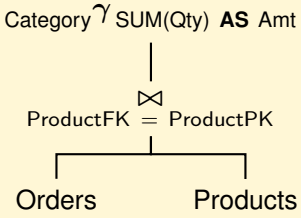
Finally, using rule 8.1 we end up with:

$$ {}_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv {}_X\gamma_{F_g}({}_{X\cup\{f_k\}-\mathcal{A}(S)}\gamma_{F_l}(R)) \underset{f_k=p_k}{\bowtie} S) \blacksquare $$

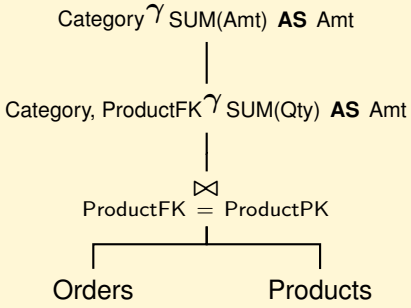
Example 8.3
Consider the query

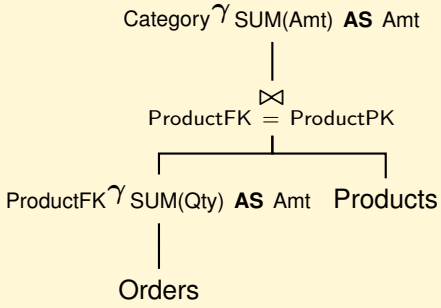
```
SELECT      Category, SUM(Qty) AS Amt
FROM        Orders, Products
WHERE       ProductFK = ProductPK
GROUP BY    Category;
```

represented with the logical tree



Since the fact table Orders does not have the invariant grouping property, but it does have the early partial aggregation property, the logical query tree can be transformed as follows:





Observation
The early partial aggregation property can also be exploited in the following cases to do a group-by before the join:

– *Having*. We assume that the SQL query has a HAVING clause that cannot be transformed into a restriction before the group-by. If the early partial aggregation property holds, then the restriction due to the HAVING clause cannot be moved together with the γ operator, but must be applied to the result of the second one.
– *Star query*. Let R be the fact table and S_i a set of n dimensional tables joined with R:

$$ {}_X\gamma_F(R \overset{\bowtie}{}_{C_j} \{S_1, \ldots, S_n\}) $$

If (a) $X \not\rightarrow f_{k_1} f_{k_2} \ldots f_{k_n}$, with X the grouping attributes of $R \overset{\bowtie}{}_{C_j} \{S_1, \ldots, S_n\}$ and $f_{k_1} f_{k_2} \ldots f_{k_n}$ the foreign keys of R for the dimensional tables used in the joins, and (b) R has the *early partial aggregation* property, then

$$ {}_X\gamma_F(R \overset{\bowtie}{}_{C_j} \{S_1, \ldots, S_n\}) \equiv $$
$$ {}_X\gamma_{F_g}({}_{(X \cup \mathcal{A}(C_j) - \mathcal{A}(\{S_1, \ldots, S_n\}))}\gamma_{F_l}(R)) \overset{\bowtie}{}_{C_j} \{S_1, \ldots, S_n\}) $$

8.5 Third Case: Grouping and Counting

The following theorem considers the case called *foreign relation aggregate* in [Chaudhuri and Shim, 1994] and *eager count* in [Yan and Larson, 1995], when neither the invariant grouping property, nor the partial aggregation property hold because of aggregate functions that use attributes not in R, the table where the group-by must be moved, as in the following example:

```
SELECT    PName, SUM(Cost) AS C
FROM      Orders, Products
WHERE     ProductFK = ProductPK
GROUP BY  PName;
```

To move the group-by on the fact table in this kind of queries too, the idea is to exploit the equivalence 8.7 to derive the aggregate functions values from the number of elements of the partitions generated by the moved group-by.

Theorem 8.4 *Grouping and Counting*
If R does not have the invariant grouping property because there are decomposable aggregate functions in $G \subseteq F$ that use attributes in S (the Condition 2 does not hold), then

$$ {}_X\gamma_F(R \overset{\bowtie}{}_{C_j} S) \equiv \pi^b_{X \cup (F-G) \cup Z}(({}_{X \cup \mathcal{A}(C_j) - \mathcal{A}(S)}\gamma_{(F-G) \cup \{\text{COUNT}(*)\ \textbf{AS}\ \text{GBCount}\}}(R)) \overset{\bowtie}{}_{C_j} S) \tag{8.11} $$

where Z is a set of expressions such that [3]

– $A_i \times$ GBCount **AS** Ide$_j \in Z$, if $f(A_i) \in G$ is SUM(A_i) **AS** Ide$_j$,
– Ide$_j \in Z$, if $f(A_i) \in G$ is MIN(A_i) **AS** Ide$_j$, MAX(A_i) **AS** Ide$_j$, AVG(A_i) **AS** Ide$_j$,
– GBCount **AS** Ide$_j \in Z$, if $f(A_i) \in G$ is COUNT(A_i) **AS** Ide$_j$.

Proof
For the sake of simplicity, assume that there is only one attribute A of S, and $F = G$. Since $X \rightarrow f_k \rightarrow p_k \rightarrow A$, using rule 8.7:

3. We assume that an aggregate function f is renamed with f **AS** Ide

$$_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv \pi^b_{X\cup Z}({}_{X\cup\{A\}}\gamma_{\text{COUNT(*) AS GBCount}}(R \underset{f_k=p_k}{\bowtie} S))$$

for Theorem 8.2

$$_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv$$
$$\pi^b_{X\cup Z}(\pi^b_{X\cup\{A\}\cup\{\text{GBCount}\}}(({}_{X\cup\mathcal{A}(C_j)-\mathcal{A}(S)}\gamma_{\text{COUNT(*) AS GBCount}}(R)) \underset{f_k=p_k}{\bowtie} S))$$

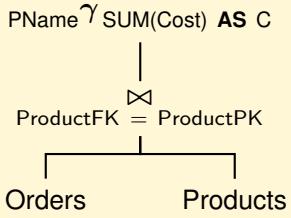
since $X\cup Z$ is a set of expressions that use only attributes from $X\cup\{A\}\cup\{\text{GBCount}\}$, the internal π is superfluous, and we end up with:

$$_X\gamma_F(R \underset{f_k=p_k}{\bowtie} S) \equiv \pi^b_{X\cup Z}(({}_{X\cup\mathcal{A}(C_j)-\mathcal{A}(S)}\gamma_{\text{COUNT(*) AS GBCount}}(R)) \underset{f_k=p_k}{\bowtie} S) \blacksquare$$

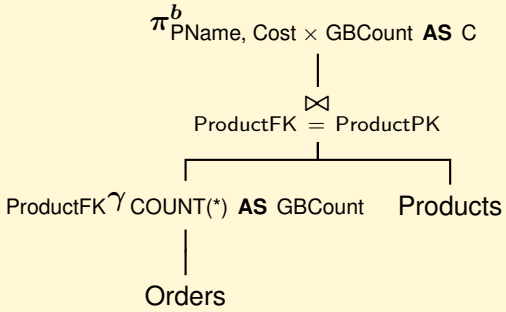
Example 8.4
Consider the query:

```
SELECT      PName, SUM(Cost) AS C
FROM        Orders, Products
WHERE       ProductFK = ProductPK
GROUP BY    PName;
```

represented with the logical tree



Because of the *foreign relation aggregate*, and PName $\rightarrow$ ProductFK holds, the logical query tree can be transformed as follows:



Observation
The equivalence 8.11 can also be exploited in the following cases to do a group-by before the join:

– *Having*. We assume that the SQL query has a HAVING clause that cannot be transformed into a restriction before the group-by. If the restriction due to the HAVING clause is different from a predicate

on COUNT(∗), it cannot be moved together with the γ operator, but must be applied to the result of the projection operator.

– *Star query*. Let R be the fact table and S_i a set of n dimensional tables joined with R:

$$_X\gamma_F(R \overset{\bowtie}{_{C_j}} \{S_1, \ldots, S_n\})$$

If (a) $X \rightarrow f_{k_1} f_{k_2} \ldots f_{k_n}$, with $f_{k_1} f_{k_2} \ldots f_{k_n}$ the foreign keys of R for the dimensional tables used in the joins, and (b) $F = H \cup G$, with H a set of aggregate functions that use attributes in R, and G a set of decomposable aggregate functions $\{f(A_i)\}$ that use dimensional attributes A_i, then

$$_X\gamma_F(R \overset{\bowtie}{_{C_j}} \{S_1, \ldots, S_n\})) \equiv$$
$$\pi^b_{X \cup H \cup Z}((_{X \cup \mathcal{A}(C_j) - \mathcal{A}(\{S_1,\ldots,S_n\})}\gamma_{H \cup \{\text{COUNT(*) AS GBCount}\}}(R)) \overset{\bowtie}{_{C_j}} \{S_1, \ldots, S_n\})$$

8.5.1 Finale

In the case of an expression $_X\gamma_F(R \overset{\bowtie}{_{f_k=p_k}} S)$ that does not satisfy the condition for the application of the equivalence rule 8.11 because the join attribute f_k from R is not determined by the grouping attributes X from $R \overset{\bowtie}{_{C_j}} S$, but the aggregate functions are decomposable, the group-by can be brought forward using the equivalence rules 8.10 and 8.11:

$$\begin{aligned} _X\gamma_F(R \overset{\bowtie}{_{f_k=p_k}} S) &\equiv {_X\gamma_{F_g}}(_{X \cup \{f_k\}}\gamma_{F_l}(R \overset{\bowtie}{_{f_k=p_k}} S)) \text{ using 8.10} \\ &\equiv {_X\gamma_{F_g}}(\pi^b_{X \cup H \cup Z}((_{X \cup \{f_k\} - \mathcal{A}(S)}\gamma_{H \cup \{\text{COUNT(*) AS GBCount}\}}(R)) \\ &\quad \overset{\bowtie}{_{f_k=p_k}} S)) \text{ using 8.11} \end{aligned}$$

8.6 Summary

– The star query optimizer can produce better access plans when considering the possibility of bringing forward group-by, usually not considered by traditional optimizers of relational systems.
– If the pre-grouping transformation is possible, the optimizer chooses the best physical plan for two logical plans with or without the pre-grouping.
– Three cases were shown in which the pre-grouping transformation is possible for star queries: *invariant grouping*, *double grouping* and *grouping and counting*.

Chapter 9

QUERY REWRITING USING MATERIALIZED VIEWS

Materialized views can yield substantial reductions in queries' execution time, especially in the case of grouping operations of large fact tables. To exploit this possibility, the query optimizer has to determine which view is useful to perform a query rewrite and how to use the selected view. We present two algorithms to solve the problem by making some simplifying assumptions.

9.1 Introduction

In traditional DBMSs views are used frequently for different reasons and especially to simplify particularly complex query writing or to write queries that would not be possible without using views. The use of views is explicit in queries and, when possible, they are automatically rewritten before being executed by replacing the expression defining the view instead of using it before optimizing queries.

OLAP queries are typically aggregate queries. Analysts want fast answers and over large data sets. This is why it is normal to choose a subset of aggregate queries, to store their results (*views materialization*), and then to use them to process expensive OLAP queries efficiently by doing just few additional computations (*query rewriting*). Unlike how views are traditionally used, query rewriting is done automatically by the system, without the user being aware of the existence of materialized views.

The query optimizer is responsible for determining if it is possible to rewrite a query Q, defined on the data warehouse tables, in an equivalent form Q', which uses a materialized view V and produces a cheaper execution plan. Intuitively, V can be used to rewrite Q if (a) it has the attributes that are needed by Q and (b) it calculates all the records that are needed to produce the result of Q. The problem has been studied from different points of view and for a review see [Halevy, 2001]. In the following, the idea of two solutions will be presented making the following simplifying assumptions:[1]

1. The *data warehouse* has a star schema, with foreign keys in the fact table and primary keys in the dimensional tables without null values and *surrogates keys*, i.e. each of them with a single attribute. Foreign and primary keys are the only attributes of the star schema with the same names and data types.

   For simplicity, we assume also that

   a) The fact and dimensional tables have attributes without null values.
   b) The fact table foreign keys are used for different dimensional tables.
   c) For each dimensional table D_i only the information on the functional dependencies $X \rightarrow Y$, with X a key of D_i, is available. In other words we will not consider dimensional attributes hierarchies.
   d) A materialized view V is defined in SQL with the command:

   CREATE MATERIALIZED VIEW V
   BUILD IMMEDIATE ENABLE QUERY REWRITE AS <SQL query>;

1. We are grateful to Enrico Detti for his contribution to the preparation of this chapter.

but in the following, for brevity, the full command will not be used and the view defining query only will be used.

2. The queries for Q and the view V definitions always use the fact table or they are *star queries*, that is they use *natural joins* between the fact and dimension tables. The joins cannot be eliminated by query rewriting.
3. The queries for Q and the view V definitions have the structure **SELECT-FROM-WHERE**, usually with **GROUP BY** and **HAVING**, but without **ORDER BY**, **DISTINCT**, subqueries and range variables. The **SELECT** and **HAVING** clauses may contain the aggregate functions MIN, MAX, SUM and COUNT. The AVG function is not considered because it can be computed using SUM and COUNT.

   In the **SELECT** attributes renaming is allowed only for aggregate functions.

   A condition in the **WHERE** is a logical product of predicate of the kind $A\,\theta\,c$, with A an attribute, c a constant and $\theta \in \{=, <, \leq, >, \geq\}$.
4. The aggregate functions in the view definition have argument '$*$' or one attribute not in the set of the grouping attributes. We do not consider aggregation computations over multiple attributes, for example SUM(A $*$ B + C).
5. The rewriting of a query uses one view only.

A solution for the problem of how to rewrite a query will be presented considering the logical query and view trees. But once it has been established that a query can be rewritten, an optimizer task is to find the best access plan to execute it, comparing the cost of the rewritten query and original query to choose the cheaper execution plan. In the algebraic representation of a query we assume that:

– The queries for Q and V are represented with a logical tree A with the following structure, called *normal form*, that has all aggregations and selections above all the joins. The normal form makes reasoning about query rewriting much easier.

$$A_T = \boldsymbol{\pi}^b_{L_T}(\boldsymbol{\sigma}_{H_T}(\boldsymbol{\gamma}_{G_T}(\boldsymbol{\sigma}_{C_T}(\bowtie R_T)x)))$$

where T is either Q or V and

  – $\bowtie R_T$ is the natural join of all tables in the set R_T, or a fact table.
  – C_T is a conjunctive condition and $\mathcal{A}(C_T)$ is the set of attributes in C_T.
  – $\boldsymbol{\gamma}_{G_T}$ is an abbreviation for ${}_{g(T)}\boldsymbol{\gamma}_{a(T)}$, where $g(T)$ is the set of grouping attributes and $a(T)$ is the set of the aggregate functions, renamed with AS;
  – H_T is a conjunctive condition on the grouping result.
  – L_T is the set of result attributes.

Some relational algebra operator may not be in the logical tree A.

From the assumption made, it follows that the joins in Q and V are *lossless* and *non-duplicating*:

Definition 9.1 *Lossless Join*

A join $F \bowtie D$ is lossless if for each record of F the join condition is satisfied by at least a record of D.

Definition 9.2 *Non-duplicating Join*

A join $F \bowtie D$ is non-duplicating if for each record of F the join condition is satisfied by at most a record of D.

In a star query the join $F \bowtie D$ is lossless because the foreign keys of F do not have null values, and it is non-duplicating because the join attribute of D is a key.

In the following we will use the star schema in Figure 9.1 (keys are in boldface) and the notations:

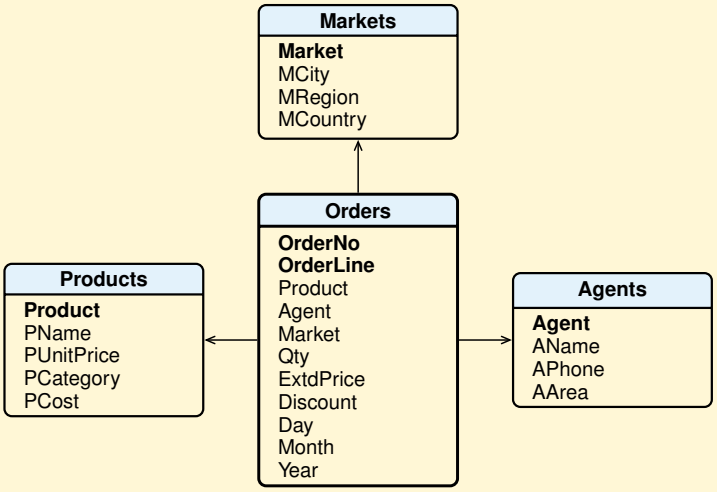
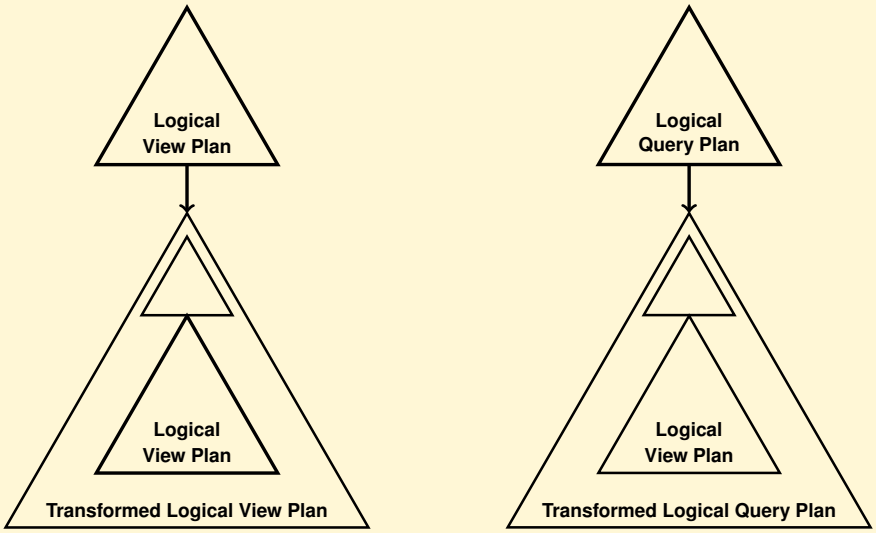


Figure 9.1: A star schema example

- A_Q is the logical query tree, and A_V is the logical view tree.
- ε_Q is a node of A_Q, and μ_V is a node of A_V.
- $A_Q(\varepsilon_Q)$ is a subtree of A_Q with root ε_Q, and $A_V(\mu_V)$ is a subtree of A_V with root μ_V.
- The *records* of $A_Q(\varepsilon_Q)$ are those returned by the logical tree.
- *Node* and *algebraic operator* of a logical tree are equivalent terms.

Among the approaches to rewriting a query Q in terms of a materialized view V, we will consider the following ones (Figure 9.2):



(a) *With a compensation on the view* **(b)** *With a transformation of the query*

Figure 9.2: Query rewriting approaches

1. *Approach with a compensation on V*. The idea is to match pairs of nodes of A_Q and A_V from the leaves to the roots to find a *compensation* $\alpha(A_V)$ on the A_V root, that is a set of logical operations in order to get a new logical tree $\alpha(A_V)$ equivalent to that of A_Q. Then A_V in $\alpha(A_V)$ is replaced with view V to get the rewriting of Q with the use of V [Zaharioudakis et al., 2000].
2. *Approach with a transformation of Q*. The idea is to rewrite A_Q, using the relational algebra equivalence rules, in order to get a new logical tree $\alpha(A_Q)$ with a subtree equivalent to the logical view tree A_V. Then A_V in $\alpha(A_Q)$ is replaced with view V to get the rewriting of Q with the use of V [Gupta et al., 1995].

Let us see first an algorithm using the first approach, and then we will only see with an example how to proceed using the second approach.

9.2 Approach with a Compensation on the View

The algorithm to rewrite Q using V is based on the notions of *match* and *compensation*, between the logical trees A_Q and A_V, defined as follows.

Definition 9.3 *Logical Operators Match*
A logical operator ε_Q matches a logical operator μ_V if the records of $A_Q(\varepsilon_Q)$ and $A_V(\mu_V)$ are the same.

Definition 9.4 *Logical Operators Compensation*
A logical operator ε_Q partially matches a logical operator μ_V if the records $A_Q(\varepsilon_Q)$ and $A_V(\mu_V)$ are different, but it is possible to add on the root of the subtree $A_V(\mu_V)$ a logical tree $\alpha(\mu_V)$, called *compensation*, such that the result of $\alpha(\mu_V)$ is the same as $A_Q(\varepsilon_Q)$.

Definition 9.5 *Query and View Match*
We say that Q matches V, if there is a compensation between the roots of A_Q and A_V.

Definition 9.6
Let C_Q and C_V be some logical conditions. We say that C_V is *less restrictive* than C_Q, if the records satisfying C_Q are a subset of those satisfying C_V (C_Q logically implies C_V).

To find a rewriting of Q using the view V, the algorithm looks for a compensation between the roots of A_Q and A_V. The compensation is determined by matching the nodes of the two logical trees, from the leaves up to the roots.

When a node ε_Q of A_Q does not match a node μ_V of A_V, the algorithm checks if certain conditions are satisfied, depending on the kind of nodes, as it will be shown in the following. If on the operand of μ_V there is a compensation α, it must be possible to *float* it on μ_V (Figure 9.3).

Definition 9.7
The float of a compensation α on μ_V requires only the execution of the operations in α necessary to compute the result of ε_Q using that of μ_V. If some necessary attributes are not available, before the float of α, they are retrieved with a compensation on the the result of μ_V using *lossless and non-duplicating joins* with the tables that contain them.

Let us define an algorithm *Rewrite* to find the compensation on the root of V to rewrite Q, by considering only certain cases, and for each of them under which conditions it is possible to proceed with the rewriting of Q. The algorithm for the general case is more complex, and its description is outside the scope of this text.

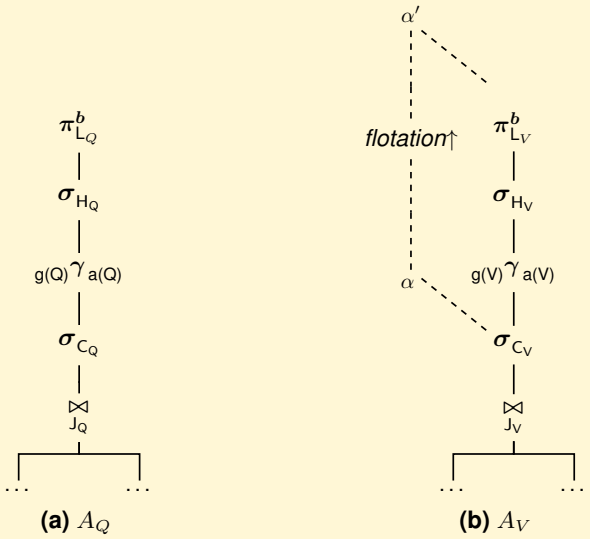


Figure 9.3: The float of a compensation

9.2.1 First Case: Q and V with Groupings

The Q and V logical trees have the following normal form

$$A_Q = \gamma_{G_Q}(\sigma_{C_Q}(\bowtie R_Q)) \qquad A_V = \gamma_{G_V}(\sigma_{C_V}(\bowtie R_V))$$

and, for simplicity, let us assume that (a) the logical operators are all present, (b) the set grouping attributes and the set of aggregate functions in γ_{G_Q} and γ_{G_V} are not empty, (c) $R_V \subseteq R_Q$.

If γ_{G_Q} and γ_{G_V} do not match, the problem is to decide whether a compensation exists to rewrite γ_{G_Q} from γ_{G_V}. We will use the algorithm for computing the closure of a set of attributes presented in the previous chapter to establish if a particular functional dependency can be inferred from the set F of functional dependencies which hold in Q.

Rewriting Algorithm

The algorithm proceeds as follows:[2]

Algorithm *Rewriting*
Input Logical trees of Q and V
Output Logical tree for rewriting of Q

1. ($\bowtie$) Under the hypothesis that $R_V \subseteq R_Q$, if the joins do not match, then in R_Q there are the tables W not in R_V, $W = R_Q - R_V$, and a compensation $\alpha_{\bowtie}$ is added to the view operator as follows:

$$\alpha_{\bowtie} = (\bowtie W(A_V(\bowtie)))$$

2. A general algorithms is beyond the scope of the book, because of its complexity. The cases treatable, however, are among the most common in practice.

The compensation is a join on $(\bowtie R_V)$, because in the fact table in R_V there are the foreign keys for the tables W.

The compensation can float on $\boldsymbol{\sigma}_{C_V}$ and then on $\boldsymbol{\gamma}_{G_V}$, if in $g(V)$ there are the foreign keys for the relations $W = R_Q - R_V$, *otherwise the query is not rewritable.*

2. $(\boldsymbol{\sigma})$ If on the operand there is a compensation $\alpha_{\bowtie}$, it floats on the operator $\boldsymbol{\sigma}_{C_V}$. Let $A_V(\boldsymbol{\sigma}_{C_V})$ be the root of the compensation tree on $\boldsymbol{\sigma}_{C_V}$.

   If the selections do not match, and $C_Q = C_V \wedge C$, with C a condition on the result of $A_V(\boldsymbol{\sigma}_{C_V})$, then the following compensation is added

$$\alpha_{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_C(A_V(\boldsymbol{\sigma}_{C_V}))$$

   The compensation can float on $\boldsymbol{\gamma}_{G_V}$ if it uses only attributes in $g(V)$, or attributes of relations with foreign keys in $g(V)$, *otherwise the query is not rewritable.*

3. $(\boldsymbol{\gamma})$ If on the operand there is a compensation $\alpha_{\boldsymbol{\sigma}}$, it floats on the operator $\boldsymbol{\gamma}_{G_V}$. Let $A_V(\boldsymbol{\gamma}_{G_V})$ be the root of the compensation tree on $\boldsymbol{\gamma}_{G_V}$, and $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ be the set of the attributes in $A_V(\boldsymbol{\gamma}_{G_V})$.

   Let us consider the following cases for matching first the *grouping attributes* and then the *aggregate functions* to define a compensation on $A_V(\boldsymbol{\gamma}_{G_V})$, under the hypothesis that $R_V \subseteq R_Q$.

   Grouping compensation
   Let us consider the possible cases as follows:

   (a) If

$$g(V) \rightarrow g(Q)$$

   does not holds in Q then *the query is not rewritable*.

   (b) The rewriting is *without grouping* when the groupings in Q and V partition data into the same number of groups, i.e. if

$$g(Q) \rightarrow g(V) \wedge g(V) \rightarrow g(Q)$$

   holds in Q and the Q attributes $A \in \mathcal{A}(R)$ not in $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ can be retrieved with a lossless and non-duplicating join of $A_V(\boldsymbol{\gamma}_{G_V})$ with R. For example, $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ contains the foreign key of R.

   (c) The rewriting is *with grouping* $g(Q)$ when the grouping in V partitions data into more groups than the grouping in Q, i.e., if only

$$g(V) \rightarrow g(Q)$$

   holds in Q and the Q attributes $A \in \mathcal{A}(R)$ not in $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ can be retrieved with a lossless and non-duplicating join of $A_V(\boldsymbol{\gamma}_{G_V})$ with R.

   Aggregate compensation.
   Let $f_Q =$ AGG(A) **AS** Ide_Q be an aggregate function in $a(Q)$, with AGG one of the aggregate functions MIN, MAX, SUM, and COUNT.

   We consider the following cases to rewrite f_Q with a function $f_V =$ AGG(A) **AS** Ide_V in $a(V)$, and to define the *compensation* on $A_V(\boldsymbol{\gamma}_{G_V})$.

   (a) If the grouping compensation is *without grouping*, then f_Q is rewritten with the following rules:

i. If AGG(A) of f_Q is COUNT,[3] COUNT(DISTINCT A), SUM(A), SUM(DISTINCT A), MAX(A) or MIN(A), and the *same aggregate function* exists in $a(V)$, then

$$\alpha_{\boldsymbol{\gamma}} = \boldsymbol{\pi}^{b}_{g(Q)\cup(a(Q)-\{f_Q\}\cup\{\mathsf{Ide}_Q\})}(A_V(\boldsymbol{\gamma}_{G_V}))$$

if $\mathsf{Ide}_Q = \mathsf{Ide}_V$, otherwise

$$\alpha_{\boldsymbol{\gamma}} = \boldsymbol{\pi}^{b}_{g(Q)\cup(a(Q)-\{f_Q\}\cup\{\mathsf{Ide}_V\ \textbf{AS}\ \mathsf{Ide}_Q\})}(A_V(\boldsymbol{\gamma}_{G_V}))$$

ii. If AGG(A) of f_Q is SUM(A), a COUNT **AS** Ide_V exists in $a(V)$, and
- $A \in \mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$, then

$$\alpha_{\boldsymbol{\gamma}} = \boldsymbol{\pi}^{b}_{g(Q)\cup(a(Q)-\{f_Q\}\cup\{(A\times\mathsf{Ide}_V)\ \textbf{AS}\ \mathsf{Ide}_Q\})}(A_V(\boldsymbol{\gamma}_{G_V}))$$

- $A \notin \mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$, $A \in \mathcal{A}(R)$ and $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ contains the foreign key $f_k R$ for R, then

$$\alpha_{\boldsymbol{\gamma}} = \boldsymbol{\pi}^{b}_{g(Q)\cup(a(Q)-\{f_Q\}\cup\{(A\times\mathsf{Ide}_V)\ \textbf{AS}\ \mathsf{Ide}_Q\})}(A_V(\boldsymbol{\gamma}_{G_V}) \bowtie R))$$

(b) If the grouping compensation is *with grouping*, then f_Q is rewritten with the following rules:

i. If $f_Q =$ MAX(A) **AS** Ide_Q and

A. $f_V =$ MAX(A) **AS** Ide_V, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\mathsf{MAX}(\mathsf{Ide}_V)\ \textbf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

B. $A \in g(V)$ then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{(\mathsf{MAX}(\mathsf{A})\ \textbf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

C. $A \notin \mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$, $A \in \mathcal{A}(R)$ and $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ contains the foreign key $f_k R$ for R, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\mathsf{MAX}(\mathsf{A})\ \textbf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}) \bowtie R))$$

ii. If $f_Q =$ MIN(A), then the rewriting of MIN(A) is similar to MAX(A).

iii. If $A \in \mathcal{A}(R_V)$ and

A. $f_Q =$ COUNT(A) **AS** Ide_Q and $f_V =$ COUNT **AS** Ide_V, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\mathsf{SUM}(\mathsf{Ide}_V)\ \mathsf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

B. $f_Q =$ SUM(A) **AS** Ide_Q, $f_V =$ COUNT **AS** Ide_V and
- $A \in \mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\mathsf{SUM}(\mathsf{A}\times\mathsf{Ide}_V)\ \textbf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

- $A \notin \mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$, $A \in \mathcal{A}(R)$ and $\mathcal{A}(A_V(\boldsymbol{\gamma}_{G_V}))$ contains the foreign key $f_k R$ for R, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\mathsf{SUM}(\mathsf{A}\times\mathsf{Ide}_V)\ \textbf{AS}\ \mathsf{Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}) \bowtie R))$$

3. For the hypothesis that aggregate attributes A does not have null values, COUNT may be COUNT(∗) or COUNT(A).

C. f_Q = SUM(A) **AS** Ide_Q and f_V = SUM(A) **AS** Ide_V, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\text{SUM(Ide}_V\text{) AS Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

iv. If $A \notin \mathcal{A}(R_V)$, or $A = *$, a f_V = COUNT **AS** Ide_V exists in $a(V)$, and
A. f_Q = SUM(A) **AS** Ide_Q, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\text{SUM(A}\times\text{ Ide}_V\text{) AS Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

B. f_Q = COUNT **AS** Ide_Q, then

$$\alpha_{\boldsymbol{\gamma}} = {}_{g(Q)}\boldsymbol{\gamma}_{a(Q)-\{f_Q\}\cup\{\text{SUM(Ide}_V\text{) AS Ide}_Q\}}(A_V(\boldsymbol{\gamma}_{G_V}))$$

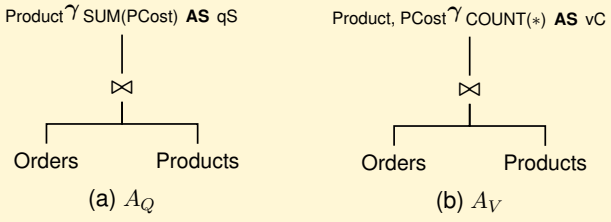
4. **Rewriting**
The compensation $\alpha_{\boldsymbol{\gamma}}(A_V)$ is the rewriting of Q.

Example 9.1
Let us consider the following Q and V

```
Q:  SELECT    Product, SUM(PCost) AS qS
    FROM      Orders NATURAL JOIN Products
    GROUP BY  Product;

V:  SELECT    Product, PCost, COUNT(*) AS vC
    FROM      Orders NATURAL JOIN Products
    GROUP BY  Product, PCost;
```

represented with the logical trees

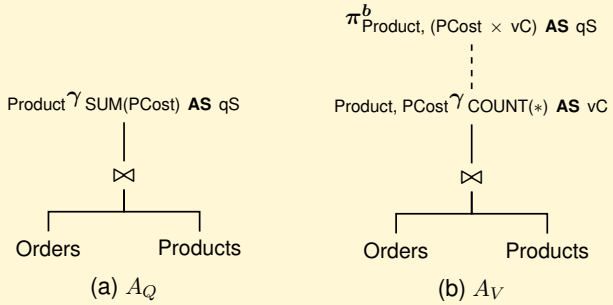


(a) A_Q (b) A_V

The joins match, and a grouping compensation is required without grouping because

- $g(V) \to g(Q)$ trivially holds since $g(Q) \subset g(V)$.
- $g(Q) \to g(V)$ holds in Q since $\text{Product}^+ = \{\text{Product}, \ldots, \text{PCost}\}$

The rewriting requires an aggregate compensation of SUM(PCost) in Q as shown in the figure:



The compensation on V is the rewriting of Q.

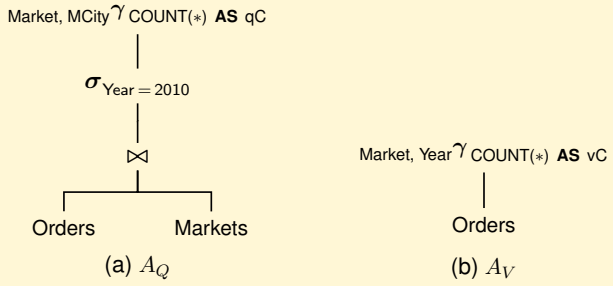
Rewriting of Q: **SELECT** Product, (PCost × vC) **AS** qS
FROM V;

Example 9.2
Let us consider the following Q and V

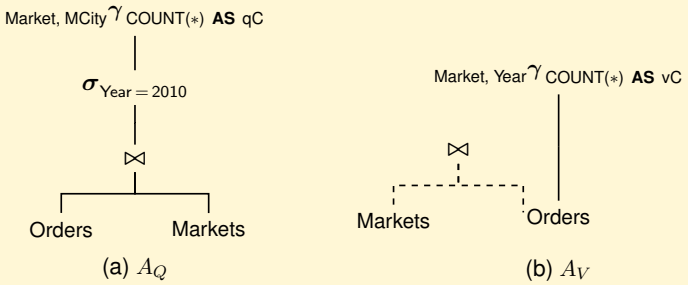
Q: **SELECT** Market, MCity, COUNT(∗) **AS** qC
FROM Orders **NATURAL JOIN** Markets
WHERE Year = 2010
GROUP BY Market, MCity;

V: **SELECT** Market, Year, COUNT(∗) **AS** vC
FROM Orders
GROUP BY Market, Year;

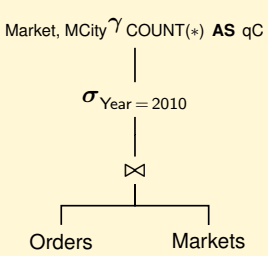
represented with the logical trees



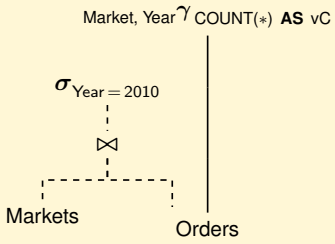
Since the joins do not match, a compensation is needed, as shown in the figure:



Since the selections do not match, a compensation is needed, as shown in the figure:



(a) A_Q



(b) A_V

The compensation on the operand of γ_V floats and a grouping compensation is required without grouping because

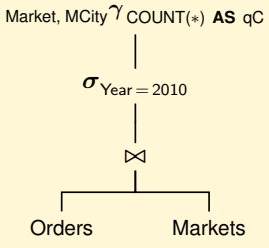
– $g(V) \rightarrow g(Q)$ holds in Q since

$$(\text{Market, Year})^+ = \{\text{Market, Year, MCity, } \ldots\}$$

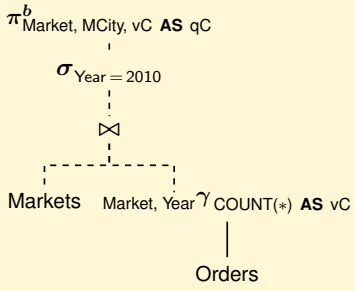
– $g(Q) \rightarrow g(V)$ holds in Q since Year = 2010 and

$$(\text{Market, MCity})^+ = \{\text{Market, MCity, Year, } \ldots\}$$

The rewriting requires an aggregate compensation of COUNT(*) in Q as shown in the figure:



(a) A_Q



(b) A_V

The compensation on V is the rewriting of Q.

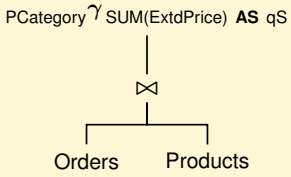
Rewriting of Q:
```
SELECT    Market, MCity, vC AS qC
FROM      V NATURAL JOIN Markets
WHERE     Year = 2010;
```

Example 9.3
Let us consider the following Q and V

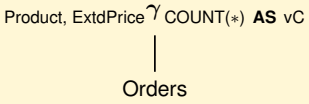
Q:
```
SELECT    PCategory, SUM(ExtdPrice) AS qS
FROM      Orders NATURAL JOIN Products
GROUP BY  PCategory;
```

V:
```
SELECT    Product, ExtdPrice, COUNT(*) AS vC
FROM      Orders
GROUP BY  Product, ExtdPrice;
```

represented with the logical trees

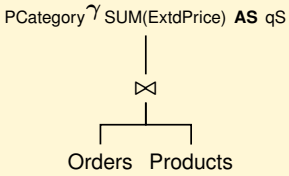


(a) A_Q

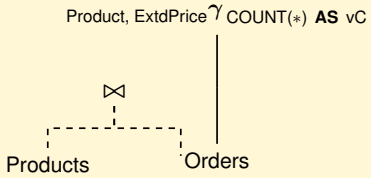


(b) A_V

Since the joins do not match, a compensation is added as shown in the figure:



(a) A_Q

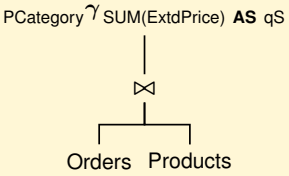


(b) A_V

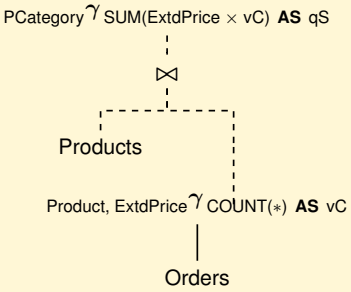
The compensation on the operand of γ_V floats and a grouping compensation is required with grouping because

- $g(V) \rightarrow g(Q)$ only holds in Q since $(\text{Product, ExtdPrice})^+ = \{\text{Product, ExtdPrice, } \ldots, \text{ PCategory}\}$,
- in $g(Q)$ there are attributes from Products not in $g(V)$, but in $g(V)$ there is the foreign key for Products, and the join is already in the compensation on γ_{G_V},
- the aggregation attribute ExtdPrice in Q is in $g(V)$.

The rewriting requires an aggregate compensation of SUM(ExtdPrice) in Q as shown in the figure:



(a) A_Q



(b) A_V

The compensation on V is the rewriting of Q.

Rewriting of Q:

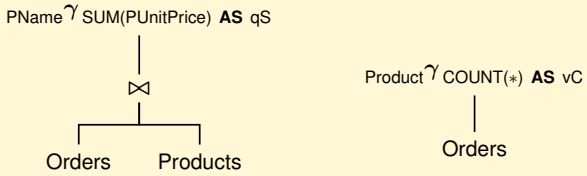
```
SELECT    PCategory, SUM(ExtdPrice × vC) AS qS
FROM      V NATURAL JOIN Products
GROUP BY  PCategory;
```

Example 9.4
Let us consider the following Q and V

Q: **SELECT** PName, SUM(PUnitPrice) **AS** qS,
FROM Orders **NATURAL JOIN** Products
GROUP BY PName;

V: **SELECT** Product, COUNT(∗) **AS** vC
FROM Orders
GROUP BY Product;

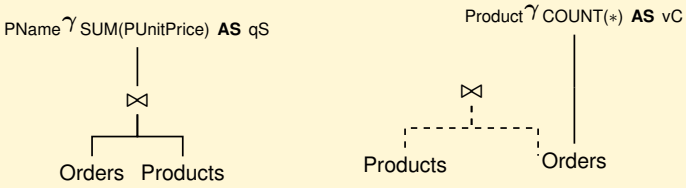
represented with the logical trees:



(a) A_Q (b) A_V

Let us assume that PName *is another key of the table* Products.

The join operations do not match. Therefore, a compensation is added as shown in the figure:

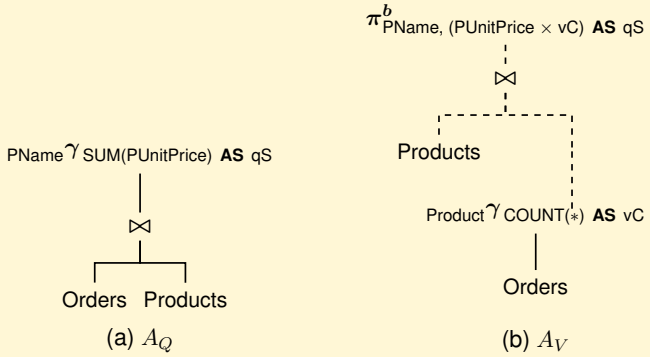


(a) A_Q (b) A_V

The compensation on the operand of γ_V floats and a grouping compensation is required without grouping because

- $g(V) \rightarrow g(Q)$ holds in Q since $\mathsf{Product}^+ = \{\mathsf{Product}, \mathsf{PName}, \dots\}$.
- $g(Q) \rightarrow g(V)$ holds in Q since $\mathsf{PName}^+ = \{\mathsf{PName}, \mathsf{Product}, \dots\}$

The rewriting requires an aggregate compensation only with a project, as shown in the figure:



(a) A_Q (b) A_V

The compensation on V is the rewriting of Q.

Q': **SELECT** PName, (PUnitPrice × vC) **AS** qS
FROM V **NATURAL JOIN** Products;

9.2.2 Second Case: Q and V with Groupings and HAVING

The Q and V logical trees have the following normal form

$$A_Q = \pi^b_{L_Q}(\sigma_{H_Q}(\gamma_{G_Q}(\sigma_{C_Q}(\bowtie R_Q)))) \quad A_V = \pi^b_{L_V}(\sigma_{H_V}(\gamma_{G_V}(\sigma_{C_V}(\bowtie R_V))))$$

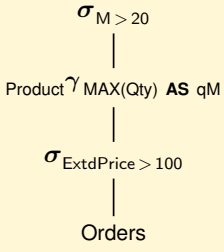
The following example shows how the **HAVING** elimination might be necessary to rewrite a query.

Example 9.5
Let us consider the following Q and V

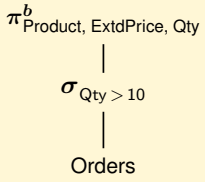
Q: **SELECT** Product, MAX(Qty) **AS** qM
FROM Orders
WHERE ExtdPrice > 100
GROUP BY Product
HAVING MAX(Qty) > 20;

V: **SELECT** Product, ExtdPrice, Qty
FROM Orders
WHERE Qty > 10;

represented with the logical trees

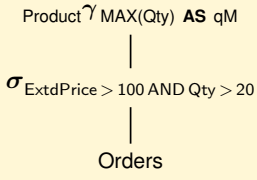


(a) A_Q

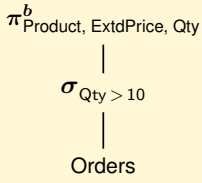


(b) A_V

Since the selections do not match, Q is not rewritable using V. However, applying the algebraic equivalence rule that allows us to delete the **HAVING** by pushing down the selection condition on the grouping, we obtain the following logical trees:

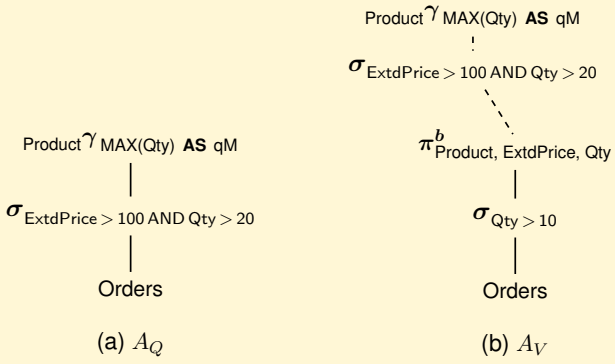


(a) A_Q



(b) A_V

Since the selection in V is not more restrictive than that in Q, Q is rewritable using V with the following compensation:



To decide whether Q is rewritable using V it is necessary to establish whether there is a correspondence between the roots of their logical trees, considering that:

1. If Q has the structure $\boldsymbol{\sigma}_Q(\boldsymbol{\gamma}_Q(E_Q))$ and V has the structure $\boldsymbol{\sigma}_V(\boldsymbol{\gamma}_V(E_V))$, then
   (a) If the nodes $\boldsymbol{\gamma}_Q$ and $\boldsymbol{\gamma}_V$ match, then we must check whether $\boldsymbol{\sigma}_Q$ and $\boldsymbol{\sigma}_V$ use the same aggregate function and the condition $\boldsymbol{\sigma}_V$ is not more restrictive than that in $\boldsymbol{\sigma}_Q$.
   (b) If $\boldsymbol{\gamma}_Q$ and $\boldsymbol{\gamma}_V$ do not match, we must check whether the compensation α might float on $\boldsymbol{\sigma}_V$, and then if there is a compensation between $\boldsymbol{\sigma}_Q$ and $\boldsymbol{\sigma}_V$.
   For the floatation of α on $\boldsymbol{\sigma}_V$ the following rules hold:
      i) If α does not contain a $\boldsymbol{\gamma}$, then α always floats on $\boldsymbol{\sigma}_V$.
      ii) If α contains a $\boldsymbol{\gamma}$, then α floats on $\boldsymbol{\sigma}_V$ only if $\boldsymbol{\gamma}$ is applied to a selection $\boldsymbol{\sigma}_{\varphi_c}$, with φ_V not more restrictive than φ_c, because $\boldsymbol{\sigma}_{\varphi_c}$ guarantees that with $\boldsymbol{\sigma}_V$ the records of $\boldsymbol{\gamma}_V$, necessary to calculate the functions in $\boldsymbol{\gamma}$, are not deleted.
   (c) If on the operand of $\boldsymbol{\gamma}_V$ there is a compensation α with a selection $\boldsymbol{\sigma}_{\varphi_c}$, then
      iii) $\boldsymbol{\sigma}_{\varphi_c}$ floats on $\boldsymbol{\gamma}_V$ if it satisfies the condition for the algebraic equivalence rules for doing the $\boldsymbol{\sigma}$ before a $\boldsymbol{\gamma}$, without considering the aggregate functions in $\boldsymbol{\gamma}_V$ which are not used to compute the aggregation functions of $\boldsymbol{\gamma}_Q$.
      This rule follows from the following algebraic equivalence rule, where F is an aggregation function different from MAX, X a set of attributes of R, B and D attributes of R:

$$\boldsymbol{\pi}^b_{X,M}(\boldsymbol{\sigma}_{M>v}(X\boldsymbol{\gamma}_{\mathsf{MAX}(B)\,\mathsf{AS}\,M,F(D)\,\mathsf{AS}\,f}(R))) \equiv \boldsymbol{\pi}^b_{X,M}(X\boldsymbol{\gamma}_{\mathsf{MAX}(B)\,\mathsf{AS}\,M,F(D)\,\mathsf{AS}\,f}(\boldsymbol{\sigma}_{B>v}(R)))$$

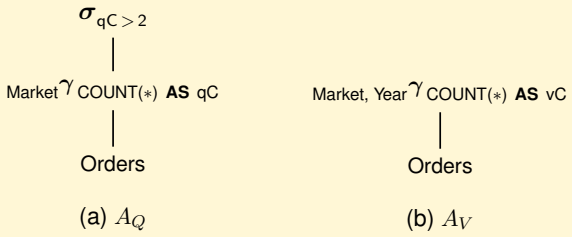
      Note that a function different from MAX can have different values in the two expressions, but they are eliminated with the projections.
      The equivalence rule holds also replacing $>$ with $\geq$, or replacing MAX with MIN and $>$ with $<$ or $\leq$.
      iv) If $\boldsymbol{\sigma}_{\varphi_c}$ uses attributes Y different from the grouping attributes of $\boldsymbol{\gamma}_V$, $\boldsymbol{\sigma}_{\varphi_c}$ can float on $\boldsymbol{\gamma}_V$ if Y can be retrieved from the results of $\boldsymbol{\gamma}_V$ with a lossless and non-duplicating join compensation with the relations containing them, and $\boldsymbol{\sigma}_{\varphi_c}$ floats on the compensation expression.
2. If Q has the structure $\boldsymbol{\sigma}_Q(\boldsymbol{\gamma}_Q(E_Q))$ and V has the structure $\boldsymbol{\gamma}_V(E_V)$, then it must be possible to add $\boldsymbol{\sigma}_Q$ to any compensation on $\boldsymbol{\gamma}_V$.
3. If Q has the structure $\boldsymbol{\gamma}_Q(E_Q)$ and V has the structure $\boldsymbol{\sigma}_V(\boldsymbol{\gamma}_V(E_V))$, then it must be possible to float on $\boldsymbol{\sigma}_V$ any compensation on $\boldsymbol{\gamma}_V$. If E_Q is without a selection, then V cannot be used to rewrite Q because the having clause in the view may eliminates certain groups of data needed by Q.

Example 9.6
Let us consider the following Q and V

Q: **SELECT** Market, COUNT($*$) **AS** qC
FROM Orders
GROUP BY Market
HAVING COUNT($*$) $>$ 2;

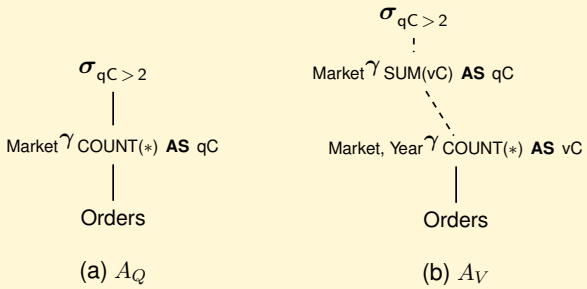
V: **SELECT** Market, Year, COUNT($*$) **AS** vC
FROM Orders
GROUP BY Market, Year;

represented with the logical trees



(a) A_Q (b) A_V

A grouping compensation is required on the root of V because $g(V) \rightarrow g(Q)$ only trivially holds. A rewriting with grouping and an aggregate compensation of COUNT($*$) in Q are performed, and then a selection is added to match σ_Q, as is shown in the figure:



(a) A_Q (b) A_V

The compensation on V is the rewriting of Q.

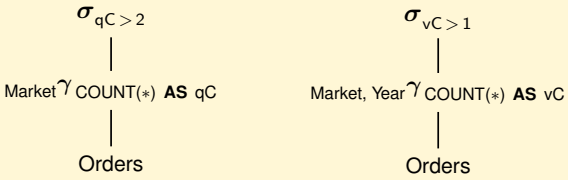
Rewriting of Q: **SELECT** Market, SUM(vC) **AS** qC
FROM V
GROUP BY Market
HAVING SUM(vC) $>$ 2;

Example 9.7
Let us consider the following Q and V

Q: **SELECT** Market, COUNT(∗) **AS** qC
FROM Orders
GROUP BY Market
HAVING COUNT(∗) > 2;

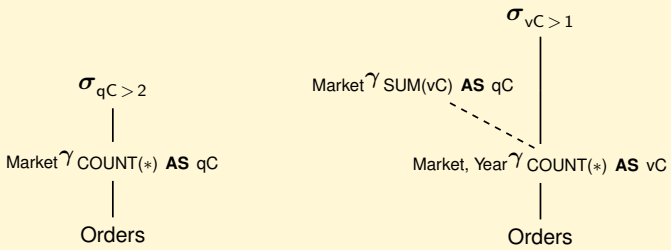
V: **SELECT** Market, Year, COUNT(∗) **AS** vC
FROM Orders
GROUP BY Market, Year
HAVING COUNT(∗) > 1;

represented with the logical trees



(a) A_Q (b) A_V

Since $g(V) \rightarrow g(Q)$ only trivially holds, a rewriting with grouping and an aggregate compensation of COUNT(∗) in Q are performed, as is shown in the figure:



(a) A_Q (b) A_V

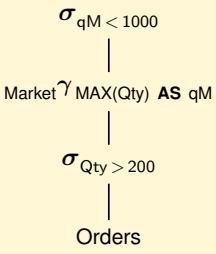
Considering then the σ, the compensation cannot float on σ_V, and so the rewriting of Q is not possible.

Example 9.8
Let us consider the following Q and V

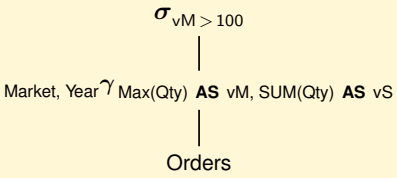
Q: **SELECT** Market, MAX(Qty) **AS** qM
FROM Orders
WHERE Qty > 200
GROUP BY Market
HAVING MAX(Qty) < 1000;

V: **SELECT** Market, Year, MAX(Qty) **AS** vM, SUM(Qty) **AS** vS
FROM Orders
GROUP BY Market, Year
HAVING MAX(Qty) > 100;

represented with the logical trees

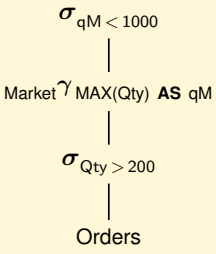


(a) A_Q

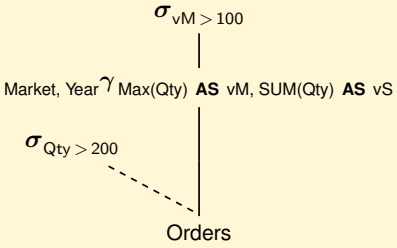


(b) A_V

Since the selections do not match, a compensation is needed, as shown in the figure:

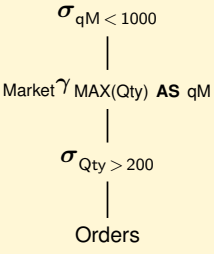


(a) A_Q



(b) A_V

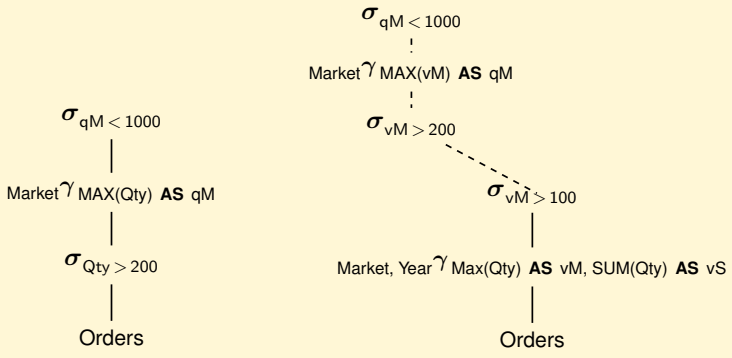
The compensation $\sigma_{\text{Qty} > 200}$ can then float on γ_V (condition 1.c.iii), because the aggregate function SUM(Qty) **AS** vS is not used in γ_Q, and so the selection satisfies the condition for being pulled up above the group-by with respect to MAX(Qty) AS vM. Then, since $g(V) \rightarrow g(Q)$ only trivially holds, a rewriting with grouping is performed too.



(a) A_Q

(b) A_V

The compensation on γ_V can float on σ_V (condition 1.b.ii), and so the rewriting algorithm finds the following match of Q with V:



The compensation on V is the rewriting of Q.

Rewriting of Q:

```
SELECT      Market, MAX(vM) AS qM
FROM        V
WHERE       vM > 200
GROUP BY    Market
HAVING      MAX(vM) < 1000;
```

9.3 Approach with a Transformation of the Query

The idea is to rewrite the logical tree of Q using algebraic equivalence rules, in particular those to push the group-by operation past one or more joins, in order to obtain a lower portion of the tree equivalent to the query tree for V, and the upper portion as the transformed query tree Q using V.

For the sake of simplicity, let us assume that the queries Q and V have the following normal form:

$$A_Q = \gamma_{G_Q}(\sigma_{C_Q}(\bowtie R_Q)) \qquad A_V = \gamma_{G_V}(\sigma_{C_V}(\bowtie R_V))$$

with $g(Q) \subset g(V)$, so that $g(V) \rightarrow g(Q)$ only holds in Q, and the aggregate functions of γ_{G_Q} are computable from the aggregate function of γ_{G_V}.

The steps of the algorithm are simply outlined below.

1. If the selections σ do not match, check that $C_Q = C_V \wedge C$.
2. If $R_Q \neq R_V$, rewrite the joins in the query so that the left subtree contains only the $(\bowtie R_V)$.
3. If C_V is not empty, push down its predicates in the selection conditions on the left subtree, which becomes $\sigma_{C_V}(\bowtie R_V)$.
4. Rewrite the grouping operator γ_{G_Q} as $\gamma_{G_{Qtop}}$ and $\gamma_{G_{Qbot}}$, such that $\gamma_{G_{Qtop}}$ is equivalent to γ_{G_Q}, and $\gamma_{G_{Qbot}}$ is equivalent to γ_{G_V} and can be pushed on $\sigma_{C_V}(\bowtie R_V)$.
5. Replace the subtree rooted at $\gamma_{G_{Qbot}}$ with view V to get the rewriting of Q in SQL.

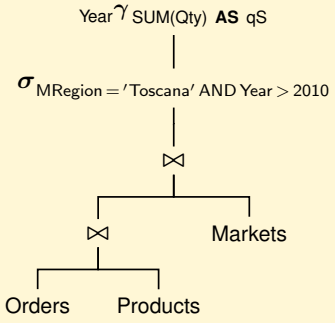
Let us only illustrate how the algorithm is applied to a simple case. We leave to the reader the task of applying the algorithm to any of the previous examples, and to incorporate enhancements when necessary.

Example 9.9
Let us consider the following Q and V

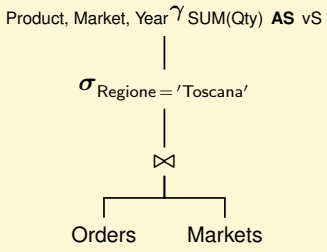
Q: **SELECT** Year, SUM(Qty) **AS** qS
FROM Orders **NATURAL JOIN** Products **NATURAL JOIN** Markets
WHERE Year $>$ 2010 **AND** MRegion = 'Toscana'
GROUP BY Year;

V: **SELECT** Product, Market, Year, SUM(Qty) **AS** vS
FROM Orders **NATURAL JOIN** Markets
WHERE MRegion = 'Toscana'
GROUP BY Product, Market, Year;

represented with the logical trees



(a) A_Q

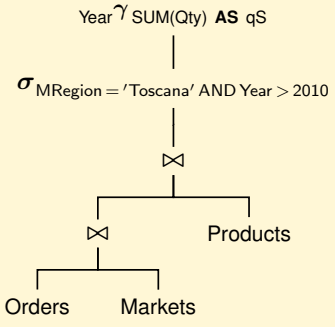


(b) A_V

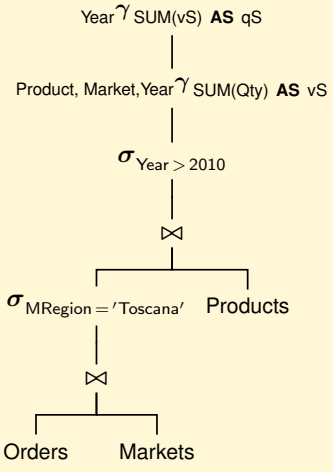
The selection condition σ_Q is more restrictive than σ_V, and since $R_Q \neq R_V$, the joins in the query are rewritten so that the left subtree contains only the R_V (logical query tree $A_{Q'}$).

C_V is pushed down on the left subtree (logical query tree $A_{Q''}$).

The grouping attributes of γ_{G_Q} (Year) are a proper subset of the grouping attributes of γ_{G_V}, and so it is possible to split γ_{G_Q} to have a $\gamma_{G_{Qbot}} = \gamma_{G_V}$, as shown in the figure (logical tree $A_{Q''}$). The aggregate function of $\gamma_{G_{Qtop}}$ can be obtained from that of $\gamma_{G_{Qbot}}$.

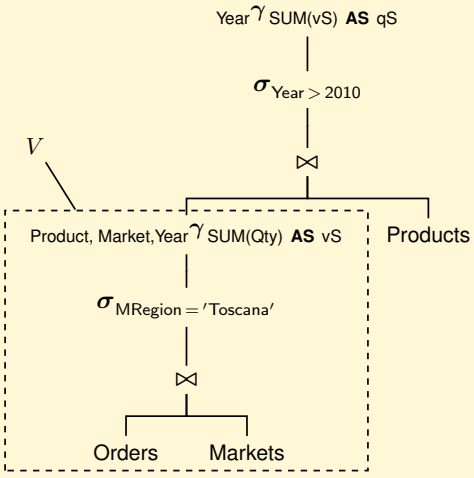


(c) $A_{Q'}$



(d) $A_{Q''}$

The $\gamma_{G_{Qbot}}$ can be pushed on the left subtree (logical query tree $A_{Q'''}$).



(e) $A_{Q'''}$

The subtree rooted at $\gamma_{G_{Qbot}}$ is identical to V, so the rewriting of Q succeeds.

Rewriting of Q:

```
SELECT      Year, SUM(vS) AS qS
FROM        V NATURAL JOIN Products
WHERE       Year > 2010
GROUP BY    Year;
```

9.4 Summary

- Analytic queries are typically aggregate queries. Analysts want fast answers and over large data sets. This is why it is normal to choose a subset of aggregate queries, to store their results (*views materialization*), and then to use them to process expensive analytic queries efficiently by performing only a few additional computations (*query rewriting*). Unlike how views are traditionally used, query rewriting is done automatically by the system, without the user being aware of the existence of materialized views. Methods have been proposed to rewrite a given query using materialized views. Among the possible approaches to the problem, two of them have been presented: *approach with a compensation on the view* and *approach with a transformation of the query*.
- The *approach with a compensation on the view* tries to add an algebraic expression, called *compensation*, on the root of the logical view tree in order to get a new logical tree equivalent to the logical query tree.
- The *approach with a transformation of the query* tries to rewrite the logical query tree, using relational algebra equivalence rules, in order to have a logical query tree with a subtree equivalent to the logical view tree.
- Two general algorithms for both approaches are beyond the scope of the book, because of their complexity. With appropriate simplifying assumptions only an algorithm for the *approach with a compensation on* V has been presented. The cases treatable, however, are among the most common in practice.
- The knowledge of an algorithm for rewriting queries to use a materialized view is useful to understand, by analyzing the physical query plans of the system available, why a view is not used for some critical queries and how to redefine it.

Appendix A

CASE STUDIES

A.1 Hospital

A hospital uses the database shown in Figure A.1 to store the following information about inpatients' treatments.

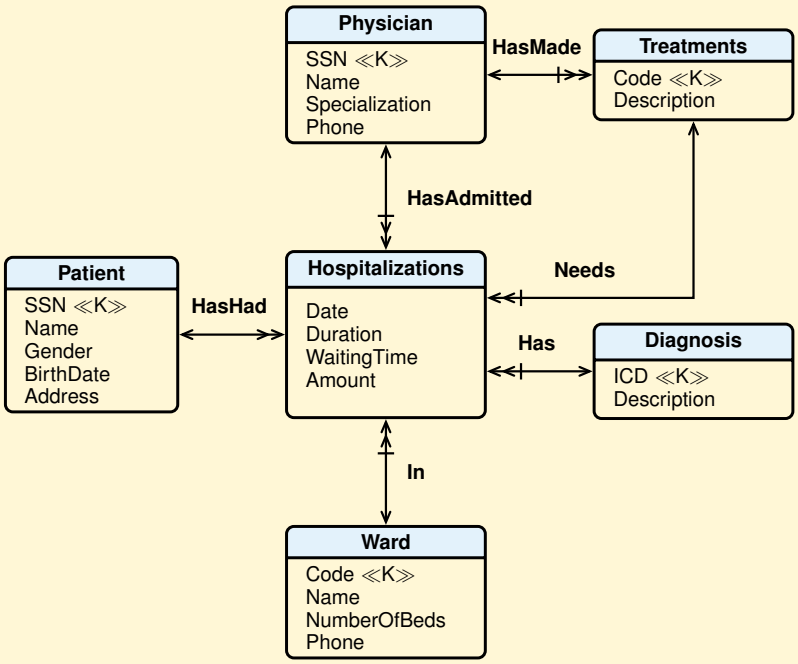


Figure A.1: Conceptual design of a database for inpatients' treatments

For each patient the information of interest is the SSN (*Social Security Number*), which is unique, the name and the address.

A patient may be hospitalized several times, and each time the information of interest is the date, the physician who admitted the patient, the ward assigned, the diagnosis, the duration in days of hospitalization, the number of days of waiting time for the hospitalization, the received treatment, and the billed amount. For simplicity, let us assume that each patient may receive only one treatment from a given physician on a given hospitalization.

For each hospital ward the information of interest is the code, which is unique, the name, the number of beds and the phone.

For each treatment the information of interest is the code, which is unique, the description, and the physician who carried it out.

For each diagnosis the information of interest is the ICD code, *International Classification of Diseases*, which is unique, and its description.

Give a conceptual and logical designs of a data mart assuming that the following examples of business questions heve been collected during the user interviews:

1. Total billed amount for hospitalizations by diagnosis code and description, by month (year).
2. Total number of hospitalizations and billed amount by ward, by patient gender (age at date of admission, city, region).
3. Total billed amount, average length of stay and average waiting time by diagnosis code and description, by name (specialization) of the physician who admitted the patient.
4. Total billed amount, and average waiting time of admission by patient age (region), by treatment code (description).

A.2 Airline Companies

We want to analyze airline companies' flights to compare them from the point of view of their ability to fly with occupied seats and therefore to make profits.

For each flight the information of interest is the company name, the departure and the destination cities, the departure time (hour, day, month, year), the number of unoccupied seats in each class (economic, business, first), the revenue of each class.

A flight code (a combination of the *ICAO airline designator* with the flight number) identifies a flight of an airline company from a departure airport to a destination airport (e.g. AP2701 is an Alitalia flight from Malpensa to Fiumicino, available on certain days a week).

A flight is identified by the flight code and the departure time.

For each city the information of interest is the city's name, the country and the continent.

For each company the information of interest is the name and the type (private or national).

Give a conceptual and logical data mart designs assuming that the following examples of business questions have been collected during the user interviews:

1. Number of unoccupied seats in a given year, by flight code, by company name (or type), by class, by departure time (hour, day, month, year)
2. Number of unoccupied seats in a given class and year, by flight code, by company name, by class, by departure (destination) city (country, continent)
3. Number of unoccupied seats and income of the Alitalia company, by year, by month, by destination country.

A.3 Airline Flights

An airline uses the database shown in Figure A.2 to store the following traffic information on passengers from its flights.

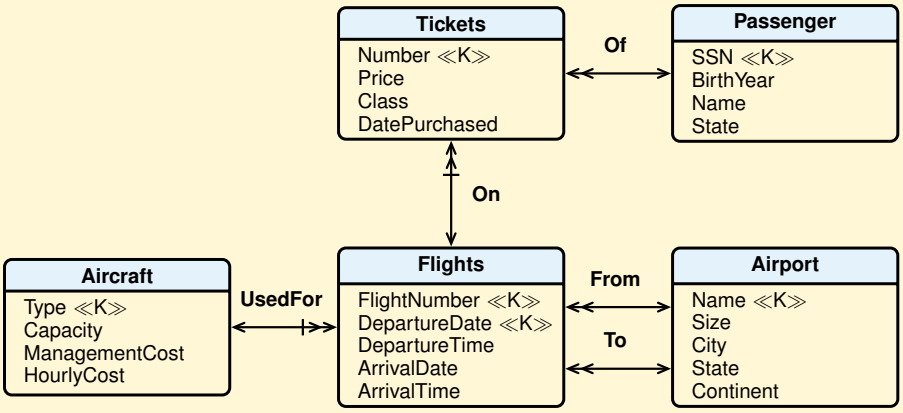


Figure A.2: The conceptual design of a database for airline flights

For each flight the information of interest is the flight code, the date and time of departure, the date and time of arrival, the departure and destination airport, and the aircraft used. The flight code identifies a possible flight of an airline from a departure airport to a destination airport (e.g. AP2701 is an Alitalia flight from Malpensa to Fiumicino, available on certain days a week). A specific flight is identified by a flight code and the departure date.

A flight is used by a set of passengers who bought a ticket. For each ticket the information of interest is the number, which is unique, the price, the class and the date of purchase. For each passenger the information of interest is the SSN, which is unique, the name and country.

A flight is made with an aircraft for which the information of interest is the type, which is unique, the capacity, the monthly management cost (management cost for brevity) and the hourly operating cost (fuel and crew).

For each airport the information of interest is the name, which is unique, the country, the continent, and the size, with values “small”, “medium”, “large”.

Give a conceptual and logical data mart designs and the SQL queries for the following business questions collected during the user interviews:

1. Number of first-class passengers in a given month and year, by country and by age range of passengers.
2. Number of passengers from Europe to the U.S. in a given month and year, and the total revenue, by country and by age range of passengers.
3. Number of flights, by departure city, by destination city.
4. Average number of airline passengers, by month, by aircraft type, by country of destination.
5. Average number of airline passengers, by class, by holiday date.
6. Number of passengers, by year, by size of the destination airport.
7. Number of flights to airports in Germany from the October to December quarter of a given year, and total management cost of the aircraft, by aircraft type.
8. Average profit of all flights, by country of departure, by destination country. The profit of a flight is the total passenger price minus the total flight cost.
9. Total revenue in a given year of flights by month, by destination country. The total revenue by month, total revenue by destination country, and the total revenue are also of interest.

A.4 Inventory

A company has a set of warehouses in different cities containing some of the products for sale. Proper inventory management is very important for a company and requires reconciling two conflicting demands: keeping up their level to meet customer demands, and minimizing their level to reduce the capital investment and space required for storage.

The quantity of a product that is added to the warehouse is called QtyAcquired (QtyA), that which is removed from the warehouse and shipped to customers is called QtyShipped (QtyS) and that present on a given day is called QtyOnHand (QtyOH). For example:

Date	Description	QtyA	QtyS	QtyOH	Days	QtyOH × Days	Average monthly QtyOH
01/01/2008	Initial QtyOnHand			100	14	1 400	
15/01/2008	QtyReceived	120		220	6	1 320	
21/01/2008	QtyShipped		80	140	4	560	
25/01/2008	QtyShipped		60	80	7	560	
	Totals January	**120**	**140**		**31**	**3 840**	**123.87**
01/02/2008				80	6	480	
07/02/2008	QtyShipped		20	60	8	480	
15/02/2008	QtyReceived	150		210	10	2 100	
25/02/2008	QtyShipped		50	160	4	640	
29/02/2008	QtyShipped		100	60	1	60	
	Totals February	**150**	**170**		**29**	**3 760**	**129.66**

Among the possible models for the analysis of inventory products, and their handling, the *model for periodic snapshots* is considered, simplified as follows: at the end of each month, for each deposit, the following quantities are considered: (a) the monthly average QtyOnHand of each product, (b) the total quantity of each product acquired in the month, and (c) the total quantity of each product shipped during the month. The monthly average QtyOnHand is calculated as the monthly arithmetic average of the various values of existing stocks for one month, weighted by their durations:

$$\frac{\sum_{i=1}^{n} q_i \times d_i}{T}$$

where q_i is the value of the quantity on hand for d_i days and T is the number of days of the month.

For the sake of simplicity, we will use the terms QtyOnHand, QtyAcquired and QtyShipped instead of Monthly average QtyOnHand, Monthly QtyAcquired and Monthly QtyShipped.

The company is interested in analyzing the QtyOnHand on a volume basis, and not on a financial basis, by considering the following metrics in a given *time period* T_m measured in months:

– *Inventory Turns* (*Inventory Turnover Ratio* or *Turns*) is the top inventory metric used by any business. This metric measures how fast a product moves in and out of the warehouse, and is calculated with the following ratio:

$$\frac{\text{Total QtyShipped}}{\text{Average QtyOnHand}}$$

A high value of the *Inventory Turns* means that products are more frequently sold, and so their sale allow the purchase of new quantities on hand.

– *Days in Inventory* of a product (*Days Sales in Inventory*, *Average Turnover Period* or *Days Inventory Outstanding*) measures the average time (in days) of the product in stock, and is calculated with the following ratio:

$$\frac{\text{Number of days}}{\text{Inventory Turns}}$$

A low value of the *Days in Inventory* means that the recovery of capital invested in stocks is more rapid.[1]

The following are some examples of business questions collected during the user interviews. For each report there are others similar with QtyAcquired and QtyShipped, by city or region of warehouses, by product category, by quarter or year.

Report 1. Total of *Quantity on Hand* (QtyOnHand) in January 2010, by product (SKU and Product Name), by region. The subtotal, by all regions, is also of interest.

Quantity on Hand
January 2010

SKU Product	Product Name	Region	Total of QtyOnHand
1	P1	North	200
		South	150
		East	50
		West	100
		All	...
2	P2	North	400
	...	...	...

Report 2. Total of *Quantity on Hand* in the first quarter of 2010, by product category, by month name.

Product Category Quantity on Hand
First Quarter 2010

Product Category	Month	Total of QtyOnHand
C1	January	900
	February	300
	March	500
C2	January	400
	...	...

Report 3. Values of the *Inventory Turns* and *Days in Inventory* in the year 2010, by product category, by quarter name.

1. For simplicity, we assume that a month is 30 days, a quarter is 90 days and one year is 365 days.

Inventory Turns and Days in Inventory Year 2010			
Product Category	**Quarter**	**Inventory Turns**	**Days in Inventory**
C1	Q1	...	...
	Q2	...	...
	Q3	...	...
	Q4	...	...
C2	Q1	...	...
	...	...	...

Give a conceptual and logical data mart designs. For each measure, specify if it is additive, semi-additive or non-additive. Give the SQL queries to produce the data of the reports.

A.5 Hotels

The managers of an international hotel chain are interested in analyzing the degree of use of different types of hotel rooms to determine how to price them.

Every day the rooms may be *occupied*, *vacant* or *unavailable* for maintenance reasons. There are different types of rooms on the basis of the following properties: the type (standard, suite, deluxe), the number of beds, the maximum number of occupants, and optional features, such as Minibar, satellite TV, Internet, whirlpool bath, kitchenette, suite.

For each hotel the information of interest is the name, the location and the category (5 star, 4 star, . . . , 1 star).

The managers are interested in analyzing the daily capacity utilization (*occupancy rate*) of each *room type* using the following metrics:

- The *room occupancy rate*, defined as the ratio of the number of rooms occupied to the total number of rooms (occupied, free and unavailable).
- The *average room revenue*, defined as the ratio of the total revenue for rooms occupied to the number of rooms occupied.
- The *revenue per available room*, defined as the ratio of the total revenue for rooms occupied to the number of rooms avaliable, equivalent to the *average room revenue* $\times$ *room occupancy rate*.

The following are some examples of business questions collected during the user interviews, of interest also by category, region and country of the hotel, and by month, year and day of a holiday date.

1. The *room occupancy rate* of hotels of a given city and day, by hotel.

Occupancy Rate
Hotel Best, Florence
July 17, 2010

Hotel	Occupancy Rate
Best 1	47%
Best 2	53%
Best 3	19%

2. The *room occupancy rate* of hotels of a given region and day, by room type.

Occupancy Rate
Hotel Best, Tuscany
July 17, 2010

Room Type	Occupancy Rate
Standard	47%
Suite	53%
Deluxe	61%

3. The *room occupancy rate* at a given month and year, by hotel of a given city.

Occupancy Rate
Hotel Best, Florence
July 2010

Hotel	Occupancy Rate
Best 1	74%
Best 2	79%
Best 3	60%

4. The *room occupancy rate* and *average room revenue* of hotels in a given city, at a given month and year, by hotel.

Occupancy Rate and Average Room Revenue
Hotel Best, Milan
July 2010

Hotel	Occupancy Rate	Average Room Revenue
Best 1	74%	145
Best 2	79%	60
Best 3	60%	75

5. The monthly revenue and the cumulative revenue of 4-star hotels in a given year, by country and by month.
6. In a given year, the total revenue, and the cumulative revenue, of the rooms with the maximum number of occupants and whirlpool bath, by hotel.

Give a conceptual and logical data mart designs to analyze the room type utilization. For each measure, specify if it is additive, semi-additive or non-additive. Give the SQL queries to produce the data of the reports.

A.6 Mortgage Applications

A mortgage application process has a duration depending on the complexity of the mortgage, current market conditions, and whether the applicant has to provide additional information.

Each bank has its own processes and, for simplicity, we assume that:

- The process takes place in four phases, a number between 1 and 4 (*1. Submitting*, *2. Reviewing*, *3. Underwriting*, *4. Closing*).
- The applications are only those under evaluation (current phase number less than 4) or approved (phase number equal to 4) in the current year, and are identified by a numerical code.
- The applications sent back to a previous phase, because of an error detected during processing, are not considered.
- Each phase of the process has a start date, represented by the current day number, and an end date, which is the start day number of the next phase. The *Closing* phase has only the start day number, used to know the duration in days of the *Underwriting* phase.

Give a conceptual and logical data mart designs and the SQL queries for the following sample list of business questions about *processing volumes* (1–3) and *process efficiency* (4–6). The examples of data analysis results refer to the following sample data:

Mortgage Applications Current Year

Application Code	Day Start Phase	Phase	Amount
1	100	1	100
1	105	2	100
1	130	3	100
1	150	4	100
2	110	1	200
2	120	2	200
2	150	3	200
2	170	4	200
3	120	1	300
3	140	2	300
3	170	3	300
4	120	1	400
5	115	1	500
5	135	2	500

1. Number of applications, by phase.

Phase	No
1	5
2	4
3	3
4	2

2. Number of closed applications and total amount of applications.

No	TotalAmount
2	300

3. Number of applications not yet closed and total amount of applications.

No	TotalAmount
3	1200

4. Number of applications and average processing time, by phase completed.

Phase	No	AvgProcTime
1	4	14
2	3	28
3	2	20

5. Total processing time of closed applications, by application.

ApplicationID	ProcTime
1	50
2	60

6. Number of closed applications and average processing time.

No	AvgProcTime
2	55

Appendix B

CASE STUDIES: SOLUTIONS

It is likely that the solutions shown here will turn out to be not perfect. If you disagree with an answer, please feel free to mail us.

B.1 Hospital

Requirements specification

Each business question is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Hospitalization

Requirements analysis	Dimensions	Measures	Metrics
Total billed amount for hospitalizations, by diagnosis code and description, by month (year).	Diagnosis (ICD, Description), Date (Month, Year)	Amount	Total Amount
Total number of hospitalizations and billed amount, by ward, by patient gender (age at date of admission, city, region).	Ward, Patient (Gender, Age, City, Region)	Amount	Total number Total Amount
Total billed amount, average length of stay and average waiting time by diagnosis code and description, by name (specialization) of the physician who admitted the patient.	Diagnosis (ICD code, Description), Physician (Name, Specialization)	Amount, Duration, WaitingTime	Total Amount Average Duration Average WaitingTime
Total billed amount, and average waiting time for admission by patient age (region), by treatment code (description).	Patient (Age, Region), Treatment (Code, Description)	Amount, Duration, WaitingTime	Total Amount Average WaitingTime

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is a hospitalization of a patient, assuming that they may require one treatment only
Preliminary dimensions	Patient, Date, Ward, Diagnosis, Treatment, Physician
Preliminary measures	Duration, WaitingTime, Amount

The measure **Amount** is **additive**. The measures **Duration** and **WaitingTime** are **non-additive** because in the analysis they are used only to average them.

Conceptual Design

The data mart conceptual design is shown in Figure B.1.

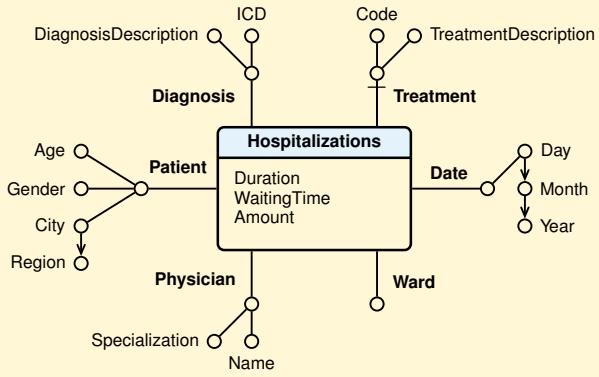


Figure B.1: The conceptual design of a data mart for the hospitalizations

Logical design

In the logical design, the facts are stored in the relation Hospitalizations, with the measures, the degenerate dimension Ward and a foreign key for each dimension table, with its own surrogate primary key (Figure B.2). The surrogate primary key for the Date dimension is a day, an integer of the form YYYYMMDD.

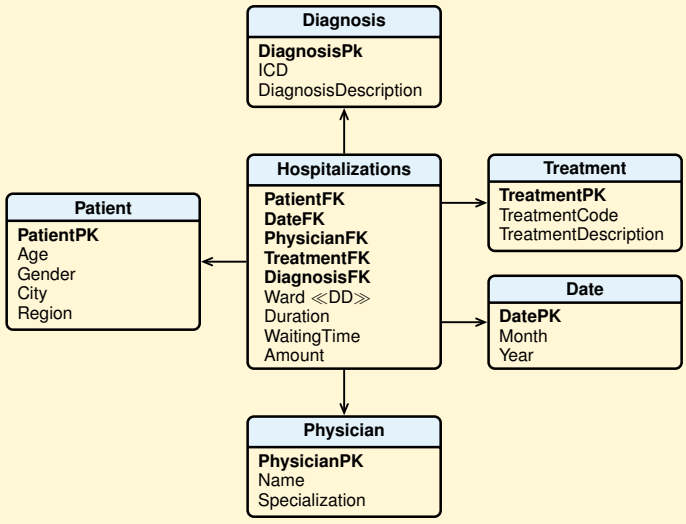


Figure B.2: The initial logical design of a data mart for the hospitalizations

This solution is correct, assuming that if a patient is hospitalized several times with different values of age, its value in the dimension Patient is that of the last hospitalization. If we are interested in storing the value of a patient age at each hospitalization, as desired by the requirements, with the admission of

a patient with an age different from the one already present in Patient, a new record is created in the table Patient with a different surrogate primary key (changes dealt with mode Type 2). To find out which data refer to the same patient hospitalizations (for example, to count the different patients hospitalized), InitialPatientKey is added as the attribute in the fact table, with the first surrogate key value assigned to a patient (Figure B.3). This solution also allows us to deal with cases in which, at each new hospitalization, the patient also changes the city and region of residence.

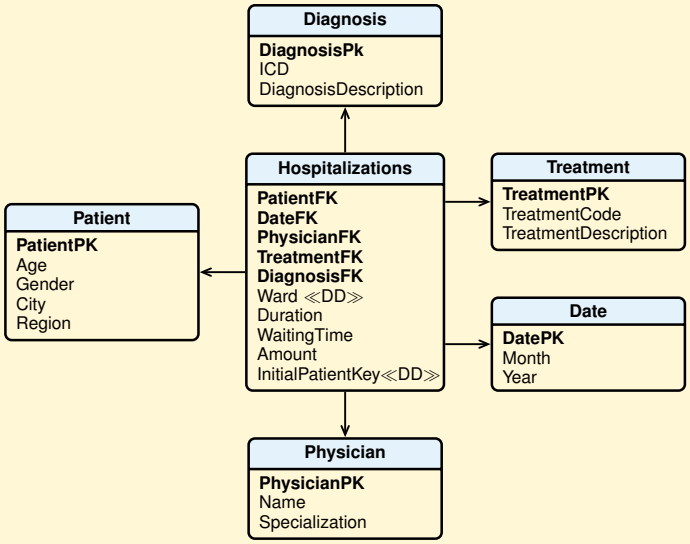


Figure B.3: The final logical design of a data mart for the hospitalizations

B.2 Airline Companies

Requirements specification

Each business requirement analysis is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Airline companies

Requirements analysis	Dimensions	Measure	Metrics
Number of unoccupied seats in a given year, by flight code, by company name (or type), by class, by departure time (time, day, month, year)	FlightCode, Class, Company(Name, Type), DepartureTime (Time, Day, Month, Year)	UnoccupiedSeats	Total UnoccupiedSeats
Number of unoccupied seats in a given class and year, by flight code, by company name, by class, by departure (destination) city (country, continent).	FlightCode, Class, Company(Name), DepartureCity (Country, Continent), DestinationCity (Country, Continent)	UnoccupiedSeats	Total UnoccupiedSeats
Number of unoccupied seats and revenue of the Alitalia company, by year, by month, by destination country.	Company(Name), DepartureTime (Month, Year), DepartureCity(Country)	UnoccupiedSeats Revenue	Total UnoccupiedSeats, Revenue

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is the information on the number of unoccupied seats on a flight of a class of a company
Preliminary dimensions	Class, FlightCode, Company, Departure time, Departure city, Destination city
Preliminary measures	UnoccupiedSeats, Revenue

The measures are **additive**.

Conceptual Design

The data mart conceptual design is shown in Figure B.4.

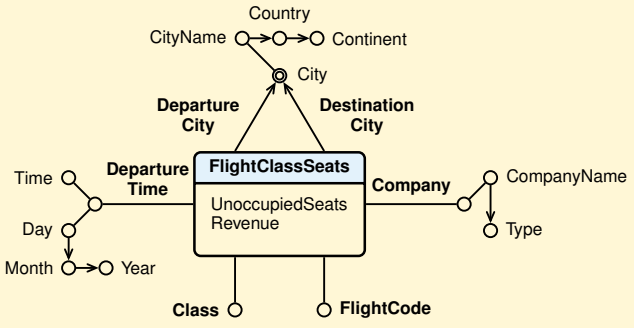


Figure B.4: The conceptual design of a data mart for the airline companies

Logical design

In the logical design, the facts are stored in the relation FlightClassSeats, with the measures, the degenerate dimensions Class, FlightCode and a foreign key for each dimension table, with its own surrogate primary key (Figure B.5).

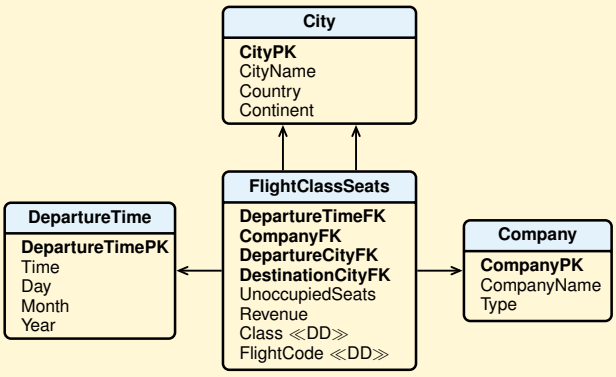


Figure B.5: The logical design of a data mart for the airline companies

B.3 Airline Flights

Requirements specification

Each business requirement analysis is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Flight Process

Requirements analysis	Dimensions	Measures	Metrics
Number of first-class passengers in a given month and year, by country, by age range of passengers.	Passenger (Nationality, AgeRange), Class, DepartureDate(Month, Year)		Number of passengers
Number of passengers from Europe to the U.S. in a given month and year, and the total revenue, by country, by age range of passengers.	Passenger (Nationality, AgeBand), DepartureDate(Month, Year), DepartureAirport(Continent), DestinationAirport(Country)	Price	Number of passengers, Total price
Number of flights by departure city, by destination city.	DepartureAirport(City), DestinationAirport(City)		Number of flights
Average number of airline passengers by month, by aircraft type, by destination country.	DepartureDate(Month), Aircraft(Type), DestinationAirport(Country)		Average number of passengers
Average number of airline passengers by class, by holiday date.	Class, DepartureDate(HolidayFlag)		Average number of passengers
Number of passengers per year, by size of the destination airport.	DestinationAirport(Size), DepartureDate(Year)		Number of passengers
Number of flights to airports in Germany from the October to December quarter of a given year, and total management cost of the aircraft, by aircraft type.	DepartureDate(Month, Year), DestinationAirport(Country), Aircraft(Type, ManagementCost)		Number of flights, Total management cost
Average profit of all flights, by country of departure, by destination country. The profit of a flight is the total passenger price minus the total flight cost.	DepartureAirport(Country), DestinationAirport(Country), Flight(Duration), Aircraft(ManagementCost, HourlyOperatingCost)	Price	Average profit
Total revenue in a given year of flights, by month, by destination country. The total revenue by month, total revenue by destination country, and the total revenue are also of interest.	DestinationAirport(Country), DepartureDate(Month, Year)	Price	Total Price

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is the information on the ticket of a passenger flight
Preliminary dimensions	Passenger, Flight, Class, Aircraft, Departure Airport, Destination Airport
Preliminary measures	Price

The measure **Price** is **additive**.

Conceptual Design

The data mart conceptual design is shown in Figure B.6:

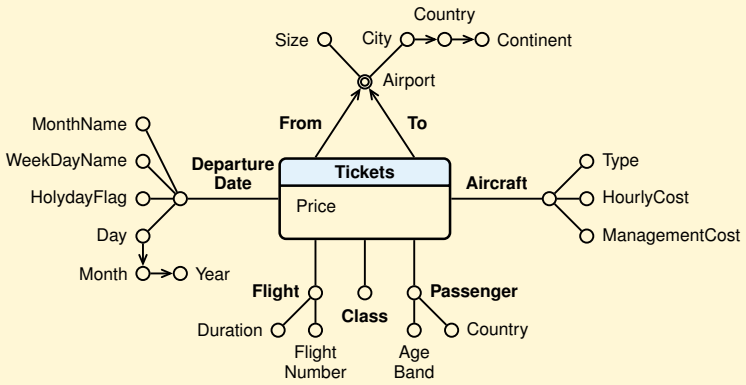


Figure B.6: The conceptual design of a data mart for the airline flights

Logical design

In the logical design, the facts are stored in the relation Tickets, with the measures, the degenerate dimension Class and a foreign key for each dimension table, with its own surrogate primary key (Figure B.7). The surrogate primary key for the DepartureDate dimension is a day, an integer of the form YYYYMMDD.

To simplify the SQL analysis, the degenerate dimension FlightID has been added to the fact table to identify the flight of a ticket, with a value the chaining together of the FlightFK and DepartureDateFK values. If FlightID is not used, in the SQL analysis it will be substituted by the expression:

(**CAST** (FlightFK **AS** varchar) + **CAST** (DepartureDateFK **AS** varchar))

The table Passenger has as many elements as are the different combinations of Nationality and AgeBand values.

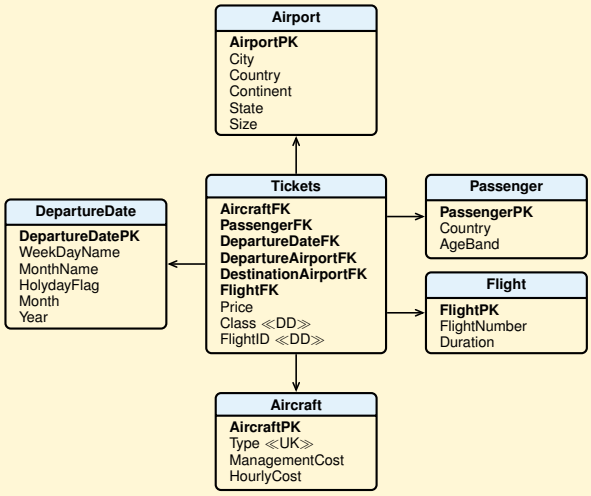


Figure B.7: The logical design of a data mart for the airline flights

Data Analysis

Let us assume that and a month is represented as the integer YYYYMM, and a holiday date has the HolidayFlag = true.

1. Number of first-class passengers in a given month and year, by country, by age range of passengers.

```
SELECT    Country, AgeBand, COUNT(*) AS NoOfPassengers
FROM      Tickets, DepartureDate, Passenger
WHERE     PassengerFK = PassengerPK AND DepartureDateFK = DepartureDatePK
          AND  Month = 200812 AND Class = 1
GROUP BY  Country, AgeBand;
```

2. Number of passengers from Europe to the U.S. in a given month and year, and the total revenue, by country, by age range of passengers.

```
SELECT    Passenger.Country, AgeBand
          , COUNT(*) AS NoOfPassengers, SUM(Price) AS Revenue
FROM      Tickets, Airport FRM, Airport TO, DepartureDate, Passenger
WHERE     DepartureAirportFK = FRM.AirportPK
          AND DestinationAirportFK = TO.AirportPK
          AND PassengerFK = PassengerPK AND DepartureDateFK = DepartureDatePK
          AND Month = 200812 AND FRM.Continent = 'Europa' AND TO.Country = 'USA'
GROUP BY  Passenger.Country, AgeBand;
```

3. Number of flights, by departure city, by destination city.

```
SELECT    FRM.City AS DepartureCity, TO.City AS DestinationCity
          , COUNT(DISTINCT FlightID) AS NoOfFlights
FROM      Tickets, Airport FRM, Airport TO
WHERE     DepartureAirportFK = FRM.AirportPK
          AND DestinationAirportFK = TO.AirportPK
GROUP BY  FRM.City, TO.City;
```

4. Average number of airline passengers by month, by aircraft type, by destination country.

```
SELECT    MonthName , Type AS AircraftType, Country AS DestinationCountry
          , COUNT(*) / COUNT(DISTINCT FlightID) AS AvgNoOfPassengers
FROM      Tickets, Airport, DepartureDate, Aircraft
WHERE     DestinationAirportFK = AirportPK
          AND DepartureDateFK = DepartureDatePK AND AircraftFK = AircraftPK
GROUP BY  MonthName, Type, Country;
```

5. Average number of airline passengers, by class, by holiday date.

```
SELECT    Class, DepartureDateFK AS HolydayDate
          , COUNT(*) / COUNT(DISTINCT FlightID) AS AvgNoOfPassengers
FROM      Tickets, DepartureDate
WHERE     DepartureDateFK = DepartureDatePK AND HolydayFlag
GROUP BY  Class, DepartureDateFK;
```

6. Number of passengers, by year, by size of the destination airport.

```
SELECT      Year, Size AS SizeDestinationAirport
            , COUNT(*) AS NoOfPassengers
FROM        Tickets, Airport, DepartureDate
WHERE       DestinationAirportFK = AirportPK
            AND DepartureDateFK = DepartureDatePK
GROUP BY    Year, Size;
```

7. Number of flights to airports in Germany from the October to December quarter of a given year, and total management cost of the aircraft, by aircraft type.

```
SELECT      Type
            , COUNT(DISTINCT FlightID) AS NoOfFlights
            , COUNT(DISTINCT Month)*ManagementCost AS TotalManagementCost
FROM        Tickets, Airport, DepartureDate, Aircraft
WHERE       DestinationAirportFK = AirportPK
            AND DepartureDateFK = DepartureDatePK AND AircraftFK = AircraftPK
            AND Country = 'Germania' AND Month IN (200710 , 200711 , 200712)
GROUP BY    Type, ManagementCost;
```

8. Average profit of all flights by country of departure and by destination country. The profit of a flight is the total passenger price minus the total flight cost.

```
WITH        Price-FlightHourlyCost-FlighManagementCost AS
            ( SELECT      Type
                          , FRM.Country AS DepartureCountry
                          , TO.Country AS DestinationCountry
                          , SUM(Price) AS TotalPrice
                          , HourlyCost*Duration*COUNT(DISTINCT FlightID)
                              AS FlightHourlyCost
                          , ManagementCost*COUNT(DISTINCT Month)
                              AS FlighManagementCost
              FROM        Tickets, Airport FRM, Airport TO, Flight, Aircraft, DepartureDate
              WHERE       DepartureAirportFK = FRM.AirportPK
                          AND DestinationAirportFK = TO.AirportPK
                          AND FlightFK = FlightPK
                          AND AircraftFK = AircraftPK
                          AND DepartureDateFK = DepartureDatePK
              GROUP BY    FlightFK, Type, FRM.Country, TO.Country, HourlyCost,
                          ManagementCost, Duration
            )

SELECT      DepartureCountry
            , DestinationCountry
            , (SUM(TotalPrice) -
                SUM(FlightHourlyCost) - SUM(FlighManagementCost))/COUNT(*)
                  AS FlightsAvgProfit
FROM        Price-FlightHourlyCost-FlighManagementCost
GROUP BY    DepartureCountry, DestinationCountry;
```

9. Total revenue in a given year of flights by month and by destination country. The total revenue by month, total revenue by destination country, and the total revenue are also of interest.

```
SELECT      MonthName, Country AS DestinationCountry
            , SUM(Price) AS TotalRevenue,
FROM        Tickets, Airport, DepartureDate
WHERE       DestinationAirportFK = AirportPK AND DepartureDateFK = DepartureDatePK
            AND Year = 2008
GROUP BY    CUBE (MonthName, Country);
```

B.4 Inventory

Requirements specification

From the examples of business questions the following fact granularity arises:

	Fact granularity
Description	A fact is the information on the monthly values of the quantities of products on hand, acquired and shipped
Preliminary dimensions	Product (SKUProduct, Name, Category), Date (Month, Quarter, Year) Warehouse (Name, City, Region, Area)
Preliminary measures	QtyOnHand, QtyAcquired, QtyShipped

The measures **QtyAcquired** and **QtyShipped** are **semi-additive** with respect to the dimension **Product**. In fact, for analysis of inventories, it makes sense to aggregate quantities of a specific product, e.g., in order to calculate the Inventory turns of a specific product. Aggregation of different products makes sense instead when considering measures of weight, space, or monetary value.

The measure **QtyOnHand** is **semi-additive** with respect to both the dimension **Date** and the dimension **Product**.

The metrics **Inventory Turns** and **Days in Inventory**, defined with a ratio, are **non-additive** and cannot be considered as measures.

Conceptual Design

The data mart conceptual design is shown in Figure B.8.

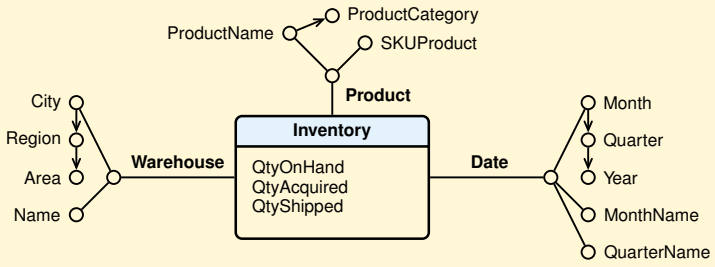


Figure B.8: The conceptual design of a data mart for the Inventory

Logical design

In the logical design, the facts are stored in the relation Inventory, with the measures, and a foreign key for each dimension table, with its own surrogate primary key (Figure B.9). The surrogate primary key for the Date dimension is a month, an integer of the form YYYYMM.

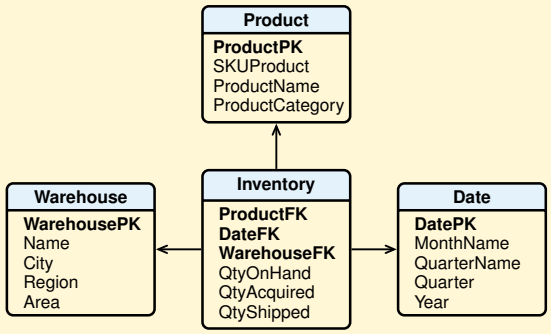


Figure B.9: The logical design of a data mart for the Inventory

Data Analysis

1. **Report 1.** Total of **QtyOnHand** in January 2010, by product (SKU and Product Name), by region. The subtotal by all regions is also of interest.

```
SELECT    SKUProduct, ProductName, Region
          , SUM(QtyOnHand) AS TotalQtyOnHand
FROM      Inventory, Product, Warehouse
WHERE     ProductFK = ProductPK AND WarehouseFK = WarehousePK
          AND DateFK = 201001
GROUP BY  SKUProduct, ProductName, ROLLUP(Region);
```

2. **Report 2.** Total of **QtyOnHand** in the first quarter 2010, by product category, by month name.

   A value of the attribute Quarter is an integer of the form YYYYQ.

   This report has no subtotals, as the previous one, because a subtotal for each category would require totalling the quantities over time, which is meaningless. Adding together the month-end quantities for January, February, and March produces a number that has no meaning. It does not represent the quantity on hand at the end of the period; the March value alone tells us that.

   When summing a semi-additive measure such as **QtyOnHand**, the dimension across which it is not additive (time) must be used to constrain the query, as in **Report 1**, or the semi-additive measure must be grouped by the dimension in question, as in this report, without a further total or subtotal.

   As "subtotals" we can compute the average of the values, but attention is required in correctly computing the average of a set of values as the sum of the values divided by the number of values. In this example the standard SQL average function will not perform this calculation correctly because it assumes as cardinality of a set of values the number of elements of a group of records. For example, if we have two products of the same category available in two warehouses every month of a quarter,

QtyOnHand of product category C1
First Quarter 2010

Product Category	DateFK	WarehouseFK	ProductFK	Month Name	QtyOnHand
C1	201001	1	1	January	500
C1	201001	2	2	January	400
C1	201002	1	1	February	100
C1	201002	2	2	February	100
C1	201003	1	1	March	200
C1	201003	2	2	March	300

   grouping the data on ProductCategory, C1 will appear in 6 records and so

$$\text{AVG(QtyOnHand)} = \frac{\text{SUM(QtyOnHand)}}{6}$$

   while the correct value is

$$\text{AVG(QtyOnHand)} = \frac{\text{SUM(QtyOnHand)}}{3\ \text{(months of the quarter)}}$$

   The problem is avoided by computing the average without using the SQL average function, as follows.

```
SELECT    ProductCategory, MonthName AS Month
          , SUM(QtyOnHand) / COUNT(DISTINCT DateFK) AS TotalQtyOnHand
FROM      Inventory, Product, Date
WHERE     ProductFK = ProductPK AND DateFK = DatePK AND Quarter = 20101
GROUP BY  ProductCategory, ROLLUP(MonthName);
```

3. **Report 3.** Values of the *Inventory Turns* and *Days in Inventory* in the year 2010, by product category, by quarter name.

   The **non-additive** metrics *Inventory Turns* and *Days in Inventory*, must be computed by a "*ratio of sum* and not by a *sum of ratio*".

```
SELECT    ProductCategory, QuarterName AS Quarter
          , SUM(QtyShipped) / (SUM(QtyOnHand) / COUNT(DISTINCT DateFK))
               AS InventoryTurns
          , 90 * ( SUM(QtyOnHand) / COUNT(DISTINCT DateFK) )
               /   SUM(QtyShipped)
               AS DaysInInventory
FROM      Inventory, Product, Date
WHERE     ProductFK = ProductPK AND DateFK = DatePK AND Year = 2010
GROUP BY  ProductCategory, QuarterName;
```

B.5 Hotels

Requirements specification

From the requirements the following fact granularity arises :

	Fact granularity
Description	A fact is the information on the daily room type utilization and revenue of each hotel
Preliminary dimensions	Room type, Date, Hotel
Preliminary measures	NOccupiedRooms, NVacantRooms, NUnavailableRooms, NOccupants, Revenue

The dimension **Room type** has as many attributes as the properties of a room, with the attributes for the optional features available with values 'Y' or 'N'.

The measures **NOccupants** and **Reveue** are **additive**, while **NoOccupiedRooms**, **NVacantRooms** and **NUnavailableRooms** are **semi-additive** with respect to **Date**.

The metrics **Occupancy Rate**, **Average Available Room Revenue** and **Average Room Revenue** are **non-additive** and must not be defined as measures.

Conceptual Design

The conceptual design of a data mart is shown in Figure B.10.

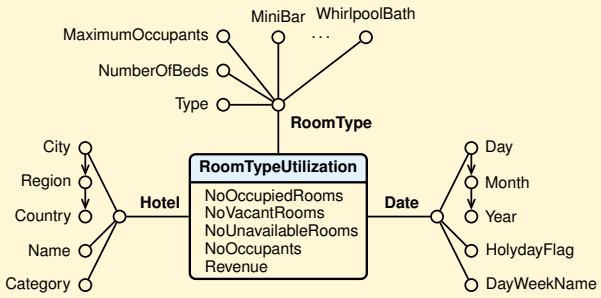


Figure B.10: The conceptual design of a data mart for the hotel room type utilization

Logical design

In the logical design, the facts are stored in the relation RoomTypeUtilization, with the measures, and a foreign key for each dimension table, with its own surrogate primary key (Figure B.11). The surrogate primary key for the Date dimension is a day, an integer of the form YYYYMMDD.

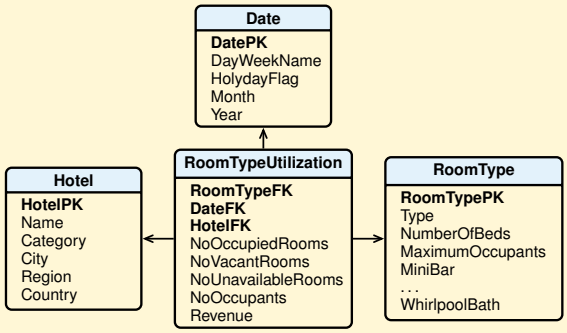


Figure B.11: The logical design of a data mart for the hotel room type utilization

Data Analysis

1. The *room occupancy rate* of hotels of a given city and day, by hotel.

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate
FROM      RoomTypeUtilization F, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = 20100717
          AND H.City = 'Florence'
GROUP BY  F.HotelFK, H.Name;
```

2. The *room occupancy rate* of hotels of a given region and day, by room type.

```
SELECT    R.Type
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate
FROM      RoomTypeUtilization F, RoomType R, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.RoomTypeFK = R.RoomTypePK
          AND H.Region = 'Tuscany' AND F.DateFK = 20100717
GROUP BY  R.Type;
```

3. The *room occupancy rate* at a given month and year, by hotel in a given city.

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate,
FROM      RoomTypeUtilization F, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
          AND D.Month = 201007 AND H.City = 'Florence'
GROUP BY  F.HotelFK, H.Name;
```

4. The *room occupancy rate* and *average room revenue* of hotels in a given city, at a given month and year, by hotel.

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate
          , SUM(F.Revenue)/SUM(F.NOccupiedRooms) AS AvgRevenueByRoom
FROM      RoomTypeUtilization F, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
          AND D.Month = 201007 AND H.City= 'Milan'
GROUP BY  F.HotelFK, H.Name;
```

5. The monthly revenue and the cumulative revenue of 4-star hotels in a given year, by country and by month.

```
SELECT      H.Country, D.Month
            , SUM(F.Revenue) AS MonthlyRevenue
            , SUM(SUM(F.Revenue)) OVER
                (PARTITION BY H.Country ORDER BY D.Month
                    ROWS UNBOUND PRECEDING)
                        AS CumulativeRevenue
FROM        RoomTypeUtilization F, Date D, Hotel H
WHERE       F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
            AND H.Category = '4-star' AND D.Year = 2010
GROUP BY    H.Country, D.Month;
```

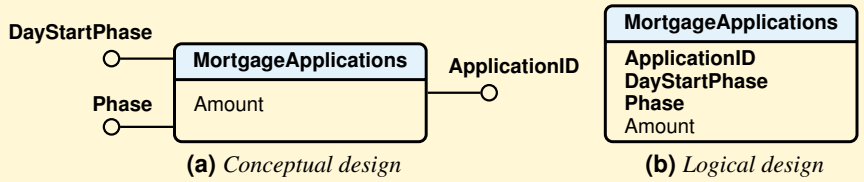
6. In a given year, the total revenue, and the cumulative revenue, of the rooms with the maximum number of occupants and whirlpool bath, by hotel.

```
SELECT      F.HotelFK, H.Name, SUM(F.Revenue) AS TotalRevenue
            , SUM(SUM(F.Revenue)) OVER
                (ROWS UNBOUND PRECEDING)
                    AS CumulativeRevenue
FROM        RoomTypeUtilization F, RoomType R, Date D, Hotel H
WHERE       F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
            AND F.RoomTypeFK = R.RoomTypePK
            AND D.Year = 2010 AND R.WhirpoolBath = 'Y'
            AND F.NOccupants = R.MaximumOccupants
GROUP BY    F.HotelFK, H.Name;
```

B.6 Mortgage Applications

First Solution: A Transaction Fact

Let us assume that a fact is a phase of the mortgage application and the data mart conceptual and logical designs are those in Figure B.12. When a mortgage application is submitted, a row is inserted into the fact table, with the application code, the phase number, and the start day number. Each time the application enters the next phase, an additional row will be added for the application, with the start day number of the new phase, and so on.



(a) *Conceptual design* **(b)** *Logical design*

Figure B.12: A transaction fact

The business questions about *processing volumes* are easy to write in SQL.

1. Number of applications, by phase.

```
SELECT    Phase
          , COUNT(*) AS No
FROM      MortgageApplications
GROUP BY  Phase;
```

2. Number of closed applications and total amount of applications.

```
SELECT    COUNT(*) AS No
          , SUM(Amount) AS TotalAmount
FROM      MortgageApplications
WHERE     Phase = 4;
```

3. Number of applications not yet closed and total amount of applications.

```
SELECT    COUNT(*) AS No
          , SUM(Amount) AS TotalAmount
FROM      MortgageApplications A
WHERE     A.Phase = 1
          AND NOT EXISTS (
          SELECT    *
          FROM      MortgageApplications B
          WHERE     A.ApplicationID = B.ApplicationID AND B.Phase = 4);
```

The business questions about *process efficiency* are neither easy to write in SQL nor likely to be efficient on large fact tables, because it is necessary to correlate fact rows that represent the phase changes. For example, to find the duration of an approved application requires computing the number of days between its submission and its closing phase start days, and this information is stored in separate rows.

4. Number of applications and average processing time, by phase completed.

```
SELECT    B.Phase AS Phase
          , COUNT(*) AS No
          , AVG(A.DayStartPhase − B.DayStartPhase) AS AvgProcTime
FROM      MortgageApplications A, MortgageApplications B
WHERE     A.ApplicationID = B.ApplicationID AND B.Phase = A.Phase − 1
GROUP BY  B.Phase;
```

5. Total processing time of closed applications, by application.

```
SELECT    A.ApplicationID AS ApplicationID
          , MAX(A.DayStartPhase) − MIN(A.DayStartPhase) AS ProcTime
FROM      MortgageApplications A
WHERE     EXISTS (
          SELECT    *
          FROM      MortgageApplications B
          WHERE     A.ApplicationID = B.ApplicationID AND B.Phase = 4)
GROUP BY  A.ApplicationID;
```

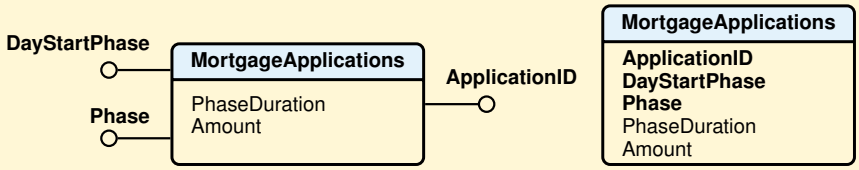
6. Number of closed applications and average processing time.

```
WITH      DurationClosedApplications AS (
          SELECT    A.ApplicationID AS ApplicationID
                    , MAX(A.DayStartPhase) − MIN(A.DayStartPhase) AS ProcTime
          FROM      MortgageApplications A
          WHERE     EXISTS (
                    SELECT    *
                    FROM      MortgageApplications B
                    WHERE     A.ApplicationID = B.ApplicationID AND B.Phase = 4)
          GROUP BYA.ApplicationID
          )

SELECT    COUNT(*) AS No,
          , AVG(ProcTime) AS AvgProcTime
FROM      DurationClosedApplications;
```

Second Solution: A Transaction Fact with Duration of Phases

The business questions about *process efficiency* can be simplified in SQL by using two pieces of information for each application phase: the day on which the phase begins and duration of the phase (the number of days) (Figure B.13).



(a) *Conceptual design* **(b)** *Logical design*

Figure B.13: A transaction fact with duration of phases

When a mortgage application is submitted, a row is inserted into the fact table, with the application code, the phase number, the start day number, and the duration of the phase have the value 0. Each time the

application enters a new phase, an additional row will be added for the application, with the start day number, and the duration in the fact table row of the previous phase is updated with its value.

The duration of the *Closing* phase has the value 0.

The following table displays the rows of a simple Mortgage Applications transaction fact table with the duration of phases.

Mortgage Applications
Current Year

Application Code	Day Start Phase	Phase	Phase Duration	Amount
1	100	1	5	100
1	105	2	25	100
1	130	3	20	100
1	150	4	0	100
2	110	1	10	200
2	120	2	30	200
2	150	3	20	200
2	170	4	0	200
3	120	1	20	300
3	140	2	30	300
3	170	3	0	300
4	120	1	0	400
5	115	1	20	500
5	135	2	0	500

The SQL queries for the first three business questions are the same. Let us see those that change.

4. Number of applications and average processing time, by phase completed.

```
SELECT    Phase, COUNT(*) AS No, AVG(PhaseDuration) AS AvgProcTime
FROM      MortgageApplications
WHERE     PhaseDuration > 0
GROUP BY  Phase;
```

5. Total processing time of closed applications, by application.

```
SELECT    ApplicationID, SUM(A.DayStartPhase − B.DayStartPhase) AS ProcTime
FROM      MortgageApplications
WHERE     EXISTS (
          SELECT   *
          FROM     MortgageApplications B
          WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4)
GROUP BY  ApplicationID;
```

6. Number of closed applications and average processing time.

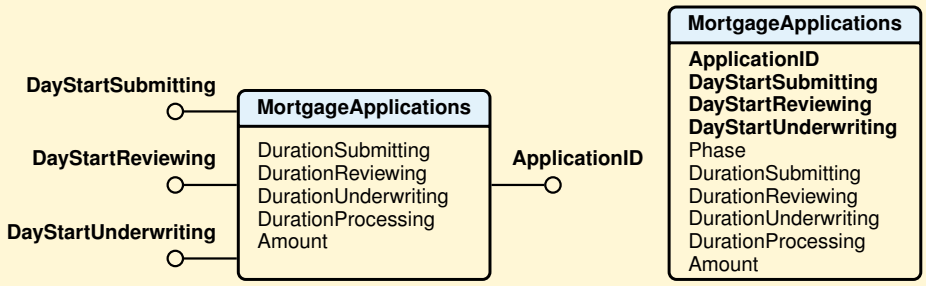
```
SELECT    COUNT(DISTINCT A.ApplicationID) AS No
          , SUM(A.PhaseDuration) / COUNT(DISTINCT A.ApplicationID)
               AS AvgProcTime
FROM      MortgageApplications A
WHERE     EXISTS (
          SELECT   *
          FROM     MortgageApplications B
          WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4);
```

Unfortunately, this approach does not eliminate the correlated subquery when looking at the time spent across multiple phases. Let us see another solution to simplify the SQL queries.

Third Solution: An Accumulating Snapshot Fact

Phase start day and phase duration do not simplify all the business questions about *process efficiency* in SQL using a *transaction fact* table. A better solution is a different data mart design based on an *accumulating snapshot fact*: there is one row for each application, independently of the number of phases, and the fact table rows are updated when a phase begins or terminates (Figure B.14).

Constructed in this manner, the accumulating snapshot fact is a useful and powerful tool for studying time spent at any phase or any combination of phases. Duration of phases can be studied in terms of their minimums, maximums, or averages across any relevant dimensions, simply by aggregating the appropriate measures as required.



(a) *Conceptual design* **(b)** *Logical design*

Figure B.14: An accumulating snapshot fact

The following table displays the rows of a simple Mortgage Applications accumulating snapshot fact table.

Mortgage Applications Current Year				
Application ID	**Day Start Submitting**	**Day Start Reviewing**	**Day Start Underwriting**	**Phase**
1	100	105	130	4
2	110	120	150	4
3	120	140	170	3
4	120	0	0	1
5	115	135	0	2

Mortgage Applications Current Year				
Duration Submitting	**Duration Reviewing**	**Duration Underwriting**	**Duration Processing**	**Amount**
5	25	20	50	100
10	30	20	60	200
20	30	0	50	300
0	0	0	0	400
20	0	0	20	500

Let us assume that the following table exists about the phases of mortgage applications.

Phases

PhaseNo	Description
1	Submitting
2	Reviewing
3	Underwriting
4	Closing

Let us see how the SQL queries change and perform better on large fact tables.

1. Number of applications, by phase.

```
SELECT    PhaseNo AS Phase, COUNT(*) AS No
FROM      MortgageApplications, Phases
WHERE     Phase >= PhaseNo
GROUP BY  PhaseNo ORDER BY PhaseNo;
```

2. Number of closed applications and total amount of applications.

```
SELECT    COUNT(*) AS No
          , SUM(Amount) AS TotalAmount
FROM      MortgageApplications
WHERE     Phase = 4;
```

3. Number of applications not yet closed and total amount of applications.

```
SELECT    COUNT(*) AS No
          , SUM(Amount) AS TotalAmount
FROM      MortgageApplications
WHERE     Phase < 4;
```

4. Number of applications and average processing time, by phase completed.

```
SELECT    PhaseNo AS Phase, COUNT(*) AS No
          , AVG(CASE PhaseNo
                WHEN 1 THEN DurationSubmitting
                WHEN 2 THEN DurationReviewing
                WHEN 3 THEN DurationUnderwriting END) AS AvgProcTime
FROM      MortgageApplications, Phases
WHERE     Phase > PhaseNo
GROUP BY  PhaseNo ORDER BY PhaseNo;
```

5. Total processing time of closed applications, by application.

```
SELECT    ApplicationID, DurationProcessing
FROM      MortgageApplications
WHERE     Phase = 4;
```

6. Number of closed applications and average processing time.

```
SELECT    COUNT(*) AS No
          , AVG(DurationProcessing) AS AvgProcTime
FROM      MortgageApplications
WHERE     Phase = 4;
```

Fourth Solution: A More General Accumulating Snapshot Fact

Let us assume that the bank is also interested in the following business questions:

- Number of mortgage applications, and total funding requested, by interest rate, by year, by quarter, by month.
- Number of mortgages underwritten, the total amount underwritten, the average difference between the amount requested by the application and the amount underwritten, by type rate, by year, by quarter, by month.
- Number of mortgage applications denied, and the average duration of the review and processing phase, by the applicant's income range.
- Minimum, maximum and average duration of mortgage applications underwritten, by the employee who reviewed and processed the application.

Figure B.15 shows the conceptual design of a possible data mart to support in addition the new business questions.

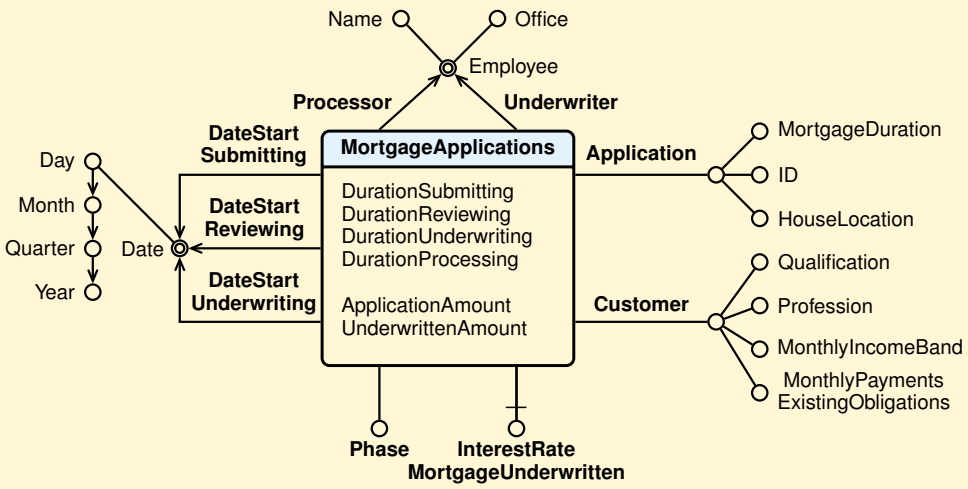


Figure B.15: The final data mart Mortgage Applications conceptual design

Appendix C

GLOSSARY

Aggregation
The result of an aggregate function (sum, count, average, minimum, maximum, etc.) applied to a bag of values.

Business Intelligence
A set of methods and tools for interactive data analysis used primarily by business administrative staff to understand and analyze business performance in order to obtain useful information to support unstructured decision making.

The term *intelligence* is used with the meaning of investigating to find out something interesting, like in *Intelligence Service*.

The *business intelligence* methods and tools are of the following types:

- *Reports.* Reporting is considered the basic level of decision support.
- *Multidimensional data analysis.* Data analysis is usually accomplished interactively with some kind of data analysis tool.
- *Exploratory data analysis.* This data analysis technique is very different from reports and multi-dimensional analysis: it uses what is called a *discovery technique of useful data models* with *data mining* algorithms.

Computerized Information System
A subset of an information system that use a variety of technologies to process information.

Conformed Dimension
A dimension shared by several fact tables.

Constellation schema
The relational schema of a data warehouse with several fact tables that share dimensional tables.

Cube
A multidimensional cube model (data cube) represent facts with n dimensions by points (a cell) in an n-dimensional space. The cells of the cube contain data measures and the edges of the cube represent the data dimensions.

Although a cube implies only 3 dimensions in geometry, a data cube may represent any number of dimensions.

Some vendors provide OLAP servers that implement the fact table as a data cube using a specialized data structure. Such implementations are referred to as MOLAP (Multidimensional OLAP).

Cuboid
Let us assume that each dimension domain is extended with an additional value "$*$". This value has the intuitive meaning "all", and it represents summarization along the dimension in which it appears, called *cuboid*. A cube can be extended with new "borders" made of cells containing the value of aggregate functions.

The extended cube is a generalization of a cross-tabulation, which is 2-dimensional, to n dimensions.

To speed up data analysis, commercial data cube systems precompute all or some of the cuboids and store them as *materialized views* of the data cube.

Data, Information, Knowledge

Data is the representation of certain facts that a computer records, stores and processes. Data, or a condensed form of them, become information when are interpreted in a certain context. Information becomes knowledge when it provides insight upon which the recipient, on the base of his experience, competence, and attitude, can make informed and effective decisions and take proper actions.

Data Mart

A database that has the same characteristics of a data warehouse, but it is focused on a single measurable business process to analyze, and so it has only one fact.

Data Mining

An exploratory data analysis technique to discovery useful data models with specialized algorithms.

Data Warehouse

A decision support database with historical, nonvolatile data, pulled together primarily from operational business systems, structured and tuned to facilitate analysis of the performance of key business processes, worthy of improvement.

The first and still now the most widely cited definition of data warehouse was provided by William Inmon in 1990: "A data warehouse is a subject-oriented, integrated, nonvolatile, and time-varying collection of data in support of management's decisions."

A fundamental axiom of the data warehouse is that data is both read-only and non-volatile. As the amount of data within the data warehouse grows, the value of the data increases, allowing a user to perform longer-term analyses of the data.

Whereas the operational data is generally real-time or near real-time, data within the data warehouse is historical, since the data warehouse is used primarily for reporting and analyzing relatively large volumes of historical data in an effort to decide what to do in the future.

Data Warehousing

The process used to organize data in a data warehouse and then allow users to analyze them with business intelligence tools.

Data Warehouse Management System (DWMS)

A specialized software for creating and managing large amount of nonvolatile data efficiently and allowing it to be analyzed with OLAP queries. There are three broad directions that have been taken to develop this specialized systems: Relational OLAP (ROLAP), Multidimensional OLAP (MOLAP), Column-Oriented OLAP.

DSS, Decision Support System

A software system used to support decision-making processes within an organization. While an operational system is for performing the business, a decision support systems is for analyzing the business.

Dice

An operator to selects a subcube of a given cube with a selection on two or more dimensions. The operator does not make aggregations on the data cube.

Dimensional Data Model

A data model that represents measurements of a process and the independent variables that may affect that process. In a dimensional model, data are organized into multiple dimensions and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides the users with the flexibility to view data from different perspectives.

Dimensional Fact Model (DFM)

A conceptual dimensional data model.

Dimension

One of the perspectives that can be used to analyze the data in a data warehouse.

Dimensional Table

The table of a relational database which contains the data for one of the dimensions. The dimensional attributes describe individual characteristics of a dimension.

The dimension table has a primary key (usually a surrogate one) which is used to connect it to the fact table. The dimension tables in a star schema are intentionally de-normalized.

DOLAP (Desktop OLAP)
A system which manage on a personal computer small amount of data extracted from a multidimensional OLAP server, a DW or an operational DBMS.

Drill-down or Roll-down
An operator to have an aggregated view of the data to a higher level of detail in two ways: by moving down along a dimensional hierarchy level or by adding a dimension of analysis.

ERP (Enterprise Resource Planning)
The meaning of the acronym ERP does not explain the purpose of these systems, which is not the enterprise resource planning, but the integration of business processes in a single software system that can meet all the information requirements of the company using a centralized database .

ETL (Extract, Transform, Load)
A set of back-end data staging steps that are used to (1) obtain data from operational sources (i.e. the extraction step), (2) cleanse and prepare data for import into the data warehouse (i.e. the transformation step), and (3) actually importing the transformed data into the data warehouse (i.e. the loading step).

Fact
A collection of related data items, consisting of measures and context data. Each fact typically represent a business item, a business transaction, or an event that can be used in analyzing the business or key business processes. The most useful data items are indeed numeric and often additive.

Fact Table
The table of a relational database which contains the individual facts being stored in the data warehouse.

There are two types of fields in a fact table: a) The fields storing the foreign keys which connect each particular fact to the appropriate value in each dimension; b) The fields storing the individual fact measures, such as number, amount, or price.

The granularity of the fact table is one of the most significant design decisions in creating a data warehouse. The facts should be as detailed as possible to allow for the data to be viewed from the greatest number of perspectives.

Granularity
The level of detail of the facts stored in a data warehouse, and so the meaning of a single record in a fact table.

Hierarchy
Dimensional attributes can be arranged into one or more logical structures to analyze data at various levels of detail.

For example, the hierarchy among the attributes City and Region of the dimension Location, states that each city belongs to one region and a region generally contains several cities. The multidimensional analysis usually exploits the hierarchy among the dimensional attributes to perform aggregations of the measures at various levels of detail along the dimensions of the data warehouse. For example, a typical hierarchy is the dimension of time to analyze the facts by year, by quarter, by month or by day.

HOLAP (Hybrid On Line Analytical Processing)
A combined use of Relational OLAP (ROLAP) and Multidimensional OLAP (MOLAP).

Information System
A system whose purpose is to store, process, and communicate information.

Key Business Process
A business process that can be clearly defined, is measurable, and is worthy of improvement.

Measure
A numerical property of a fact useful for evaluating the performance of the processes to be analyzed.

Materialized View
The results of a query stored and automatically used to facilitate the execution of other more complex queries.

Metadata
It is referred to as being the data about data, which defines all aspects of the data contained in a data warehouse including where it originally comes from, its type, what transformations it has been subjected to, where it has been used and what it means from a business perspective.

MOLAP (Multidimensional On Line Analytical Processing)
OLAP systems that store cuboids in a specialized data structures.

OLAP (On Line Analytical Processing)
A category of database software systems that primarily involves aggregating large amounts of data from a data warehouse. The term was introduced to distinguish the activities of data analysis from daily activities on business data organized in databases.

OLAP Client
A system that provides interactive tools for multi-dimensional analysis.

OLAP Server
A system that provides a vision of data to be analyzed as a cube.

OLTP (On Line Transaction Processing)
A category of database software systems that typically involves processing transactions in real time.

Operational Systems
A transaction processing systems to process operational data.

ROLAP (Relational On Line Analytical Processing)
An OLAP system that store data and materialized views in a relational DBMS.

Roll-up
The operator performs aggregation on a data cube, either climbing up a concept hierarchy for a dimension or by dimension reduction.

Schema
The definition of the logical structure of a database or a data warehouse.

Slice
An operator to selects a cross section that cut across a cube with a selection on one dimension. The result is a subcube, and so the operator does not make aggregations on a data cube.

Snowflake Schema
A variant of the star schema, where some dimension tables are normalized, thereby further splitting the data into additional tables.

Star Schema
The relational schema of a data warehouse with (1) a large central table (fact table) containing the bulk of the data without redundancy, and (2) a set of smaller attendant tables (dimension tables), one for each dimension.

BIBLIOGRAPHY

Adamson, C. and Venerable, M. (1998). *Data Warehouse Design Solutions*. J. Wiley & Sons, New York. 20, 65

Agrawal, S., Chaudhuri, S., and Narasayya, V. (2000). Automated selection of materialized views and indexes for SQL databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 496–505, Cairo, Egypt. 151

Albano, A., Rosa, L. D., Goglia, L., Goglia, R., Minei, V., and Dumitrescu, C. (2006). Another Example of a Data Warehouse System Based on Transposed Files. In *Proceedings of the International Conference on Extending Database Technology*, pages 1110–1114, Munich, Germany. 135

Artz, J. M. (2005). Data driven versus metric driven data warehouse design. In Wang, J., editor, *Encyclopedia of Data Warehousing and Mining*, pages 223–227. IDEA Group Reference, Hershey, PA, USA. 10, 36

Ballard, C., Farrell, D. M., Gupta, A., Mazuela, C., and Vohnik, S. (2006). *Dimensional Modeling: In a Business Intelligence Environment*. IBM, http://www.redbooks.ibm.com/redbooks/pdfs/sg247138.pdf. 36

Baralis, E., Paraboschi, S., and Teniente, E. (1997). Materialized views selection in a multidimensional database. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 156–165, Athens, Greece. 148

Batory, D. S. (1979). On searching transposed files. *ACM Transactions on Database Systems*, 4(4):531–544. 133

Boncz, P. A. and Kersten, M. L. (1999). MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2):101–119. 133

Boncz, P. A., Zukowski, M., and Nes, N. (2005). Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, Asilomar, CA, USA. 133

Chaudhuri, S. and Shim, K. (1994). Including Group-By in Query Optimization. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 354–366, Santiago, Chile. 153, 160, 162

Copeland, G. P. and Khoshafian, S. N. (1985). A Decomposition Storage Model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 268–279, Austin, Texas, US. 135

Datta, A., Moon, B., Ramamritham, K., Thomas, H., and Viguier, I. (1998). "Have your data and index it, too" Efficient storage and indexing for data warehouses. Technical Report 98-7, Department of Computer Science, University of Arizona, US. 135

Datta, A., Ramamritham, K., and Thomas, H. M. (1999). Curio: A novel solution for efficient storage and indexing in data warehouses. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 730–733, Edinburgh, Scotland. 135

French, C. D. (1995). "One Size Fits All" Database Architectures do not Work for DDS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 449–450, San Jose,

California, USA. 133

French, C. D. (1997). Teaching an OLTP Database Kernel Advanced Datawarehousing. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 194–198. 133

Galindo-Legaria, C. A. and Joshi, M. M. (2001). Orthogonal optimization of subqueries and aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 571–581, Santa Barbara, California, USA. 153, 160

Golfarelli, M., Maio, D., and Rizzi, S. (1998). Conceptual design of data warehouses from E/R schemes. In *Proc. Hawaii Int. Conf. on System Sciences, vol. VII*, pages 334–343, Kona, Hawaii. 17

Gupta, A., Harinarayan, V., and Quass, D. (1995). Aggregate-query processing in data warehousing environments. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 358–369, Zurich, Switzerland. 168

Gupta, H., Harinarayan, V., Rajaraman, A., and Ullman, J. D. (1997). Index selection for OLAP. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 208–219, Birmingham, UK. 151

Halevy, A. Y. (2001). Answering queries using views: A survey. *The VLDB Journal*, 10:270–294. 165

Harinarayan, V., Rajaraman, A., and Ullman, J. D. (1996). Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 205–216, Montreal, Canada. 141, 142, 145, 147

Kimball, R. and Ross, M. (2002a). *Data warehouse. La guida completa*. Hoepli Informatica, Milano. 20

Kimball, R. and Ross, M. (2002b). *The Data Warehouse Toolkit: How to Design Dimensional Data Warehouses. Second Edition*. J. Wiley & Sons, New York. 43

Moody, D. L. and Kortink, M. A. R. (2000). From enterprise models to dimensional models: A methodology for Data Warehouse and Data Mart design. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)*, pages 1–12, Stockholm, Sweden. 41

Morfonios, K., Konakas, S., Ioannidis, Y., and Kotsis, N. (2007). ROLAP Implementation of the Data Cube. *ACM Computing Surveys*, 39(4):12:1–12:53. 145

Nadeau, T. P. and Teorey, T. J. (2001). A Pareto Model for OLAP View Size Estimation. In *Proc. CASCON 2001 Conference (best paper award), (also in: Information Systems Frontiers 5,2 (2003), pp. 137-147)*, pages 1–13, Toronto, Canada. 142

Nadeau, T. P. and Teorey, T. J. (2002). Achieving scalability in OLAP materialized view selection. In *Proceedings of the International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 28–34, McLean, Virginia, USA. 145

O'Neil, P. E. and Graefe, G. (1995). Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11. 125

Shukla, A., Deshpande, P., Naughton, J. F., and Ramasamy, K. (1996). Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Bombay, India. 142

Shukla, A., Deshpande, P. M., and Naughton, J. F. (1998). Materialized view selection for multidimensional datasets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 488–499, New York City, NY, USA. 147

Song, I., Rowe, W., Mesker, C., and Ewen, E. (2001). An analysis of many-to-many relationships between fact and dimension table in dimensional modeling. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW 2001)*, pages 6.1–6.13, Interlaken, Switzerland. 50

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E. J., O'Neil, P. E., Rasin, A., Tran, N., and Zdonik, S. B. (2005). C-Store: A Column-oriented DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 553–564, Trondheim, Norway. 135

Tsois, A. and Sellis, T. K. (2003). The generalized pre-grouping transformation: Aggregate-query optimization in the presence of dependencies. In *Proceedings of the International Conference on Very*

Large Data Bases (VLDB), pages 644–655, Berlin, Germany. 153

Turner, M. J., Hammond, R., and Cotton, P. (1979). A DBMS for large statistical databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 319–327, Rio de Janeiro, Brazil. 133

Valduriez, P. (1987). Join indices. *ACM Transactions on Database Systems*, 12(2):218–246. 124

Yan, W. P. and Larson, P. A. (1995). Eager aggregation and lazy aggregation. In *The VLDB Journal*, pages 345–357. 153, 160, 162

Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., and Urata, M. (2000). Answering complex SQL queries using automatic summary tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 105–116, Dallas, Texas, USA. 168

SUBJECT INDEX