Data Warehousing and OLAP

Barcelona
September 9th, 2011

Foreword

Information assets are immensely valuable to any enterprise, and because of this, these assets must be properly stored and readily accessible when they are needed. However, the availability of too much data makes the extraction of the most important information difficult, if not impossible. View results from any Google search, and you'll see that the data = information equation is not always correct. That is, too much data is simply too much.

Data warehousing is a phenomenon that grew from the huge amount of electronic data stored in recent years and from the urgent need to use that data to accomplish goals that go beyond the routine tasks linked to daily processing. In a typical scenario, a large corporation has many branches, and senior managers need to quantify and evaluate how each branch contributes to the global business performance. The corporate database stores detailed data on the tasks performed by branches. To meet the managers' needs, tailor-made queries can be issued to retrieve the required data. In order for this process to work, database administrators must first formulate the desired query (typically an aggregate SQL query) after closely studying database catalogs. Then the query is processed. This can take a few hours because of the huge amount of data, the query complexity, and the concurrent effects of other regular workload queries on data. Finally, a report is generated and passed to senior managers in the form of a spreadsheet.

Many years ago, database designers realized that such an approach is hardly feasible, because it is very demanding in terms of time and resources, and it does not always achieve the desired results. Moreover, a mix of analytical queries with transactional routine queries inevitably slows down the system, and this does not meet the needs of users of either type of query. Today's advanced data warehousing processes separate online analytical processing (OLAP) from online transactional processing (OLTP) by creating a new information repository that integrates basic data from various sources, properly arranges data formats, and then makes data available for analysis and evaluation aimed at planning and decision-making processes.

For example, some fields of application for which data warehouse technologies are successfully used are:

- Trade: Sales and claims analyses, shipment and inventory control, customer care and public relations
- Craftsmanship: Production cost control, supplier and order support
- Financial services: Risk analysis and credit cards, fraud detection
- Transport industry: Vehicle management

- Telecommunication services: Call flow analysis and customer profile analysis
- Health care service: Patient admission and discharge analysis and bookkeeping in accounts departments

The field of application of data warehouse systems is not only restricted to enterprises, but it also ranges from epidemiology to demography, from natural science to education. A property that is common to all fields is the need for storage and query tools to retrieve information summaries easily and quickly from the huge amount of data stored in databases or made available on the Internet. This kind of information allows us to study business phenomena, learn about meaningful correlations, and gain useful knowledge to support decision-making processes.

In the following, we present an introduction to such systems and some insights to the techniques and methods underneath.

Contents

Chapter 1

Introduction

Nowadays, the *free market economy* is the basis of *capitalism* (the current global economic system) in which the production and distribution of goods are decided by market businesses and consumers; giving rise to the *supply and demand* concept. In this scenario, being more competitive than the other organizations becomes essential, and *decision making* raises as a key factor for the organization success.

Decision making is based on *information*. The more accurate information I get, the better decisions I can make to get competitive advantages. That is the main reason why information (understood as the result of processing, manipulating and organizing data in a way that adds new knowledge to the person or organization receiving it) has become a key piece in any organization. In the past, managers' ability for foreseeing upcoming trends was crucial, but this largely subjective scenario changed when the world *became digital*. Actually, any event can be recorded and stored for later analysis, which provides new and objective business perspectives to help managers in the decision making process. Hence, (digital) information is a valuable asset to organizations, and it has given rise to many well-known concepts such as *Information Society*, *Information Technologies* and *Information Systems* among others.

For this reason, today, decision making is a research hot topic. In the literature, those applications and technologies for gathering, providing access to, and analyzing data for the purpose of helping organization managers make better business decisions are globally known as *Decision Support Systems*, and those computer-based techniques and methods used in these systems to extract, manipulate and analyze data as *Business Intelligence* (BI). Specifically, BI can be defined as a broad category of applications and technologies for gathering, integrating, analyzing, and providing access to data to help enterprise users make better business decisions. BI applications include the activities of decision support systems, query and reporting, online analytical processing (OLAP), statistical analysis, forecasting, and data mining.

BI implies having a comprehensive knowledge of any of the factors that affect an organization business with one main objective: the better decisions you make, the more competitive you are. Under the BI concept we embrace many different disciplines such as *Marketing*, *Geographic Information Systems* (GIS), *Knowledge Discovery* or *Data Warehousing*.

In this work we focus on the latter, which is, possibly, the most popular, generic solution to

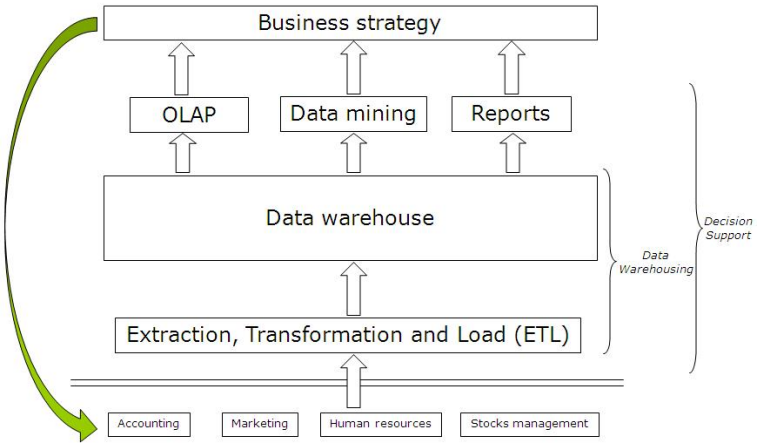


Figure 1.1: Business Intelligence cycle

give answer to extract, store (conciliate) and analyze data (what is also know as the BI cycle). Figure 1.1 depicts a data warehousing system supporting decision-making. There, data extracted from several data sources is first transformed (i.e., cleaned and homogenized) prior to be loaded in the data warehouse. The data warehouse is a read-only database (meaning that data loading is not performed by the end-users, who only query it), intended to be exploited by end-users by means of the exploitations tools (such as query and reporting, data mining and on-line analytical processing -OLAP-). The analysis carried out over the data warehouse represents valuable, objective input used in the organizations to build their business strategy. Eventually, these decisions will impact on the data sources collecting data about organizations and, again, we repeat the cycle to analyze our business reality and come up with an adequate strategy suiting our necessities.

In the following sections we discuss in detail data warehousing systems and we will also focus on their main components, such as the data warehouse, the meta-data repository, the ETL tools and the most relevant exploitation tool related to these systems: OLAP tools.

1.1 Data Warehousing Systems

Data warehousing systems are aimed at exploiting the organization data, previously integrated in a huge repository of data (the *data warehouse*), to extract relevant knowledge of the organization.

A formal definition would be: *Data Warehousing is a collection of methods, techniques and tools used to support* knowledge workers *-senior managers, directors, managers and analysts- to conduct data analyses that help with performing decision-making processes and improving information resources*. This definition gives a clear idea of these systems final aim: give support to decision making without regard of technical questions like data heterogeneity or data sources implementation. This is a key factor in data warehousing. Nowadays, any event can be recorded within organizations. However, the way each event is stored differs in every organization, and it

depends on several factors such as relevant attributes for the organization (i.e., their daily needs), technology used (i.e., implementation), analysis task performed (i.e., data relevant for decision making), etc. Thus, these systems must gather and assemble all (relevant) business data available from various (and possibly heterogeneous) sources in order to gain a single and detailed view of the organization that later will be properly managed and exploited to give support to decision making. The role of data warehousing can be better understood with five claims introduced by Kimball in [KRTR98]:

- *We have heaps of data, but we cannot access it.* Loads of data are available. However, we need the appropriate tools to effectively exploit (in the sense of query and analyze) it.
- *How can people playing the same role achieve substantially different results?* Any organization may have several databases available (devoted to specific business areas) but they are not conceptually integrated. Providing a single and detailed view of the business process is a must.
- *We want to select, group and manipulate data in every possible way.* This claim underlines the relevance of providing powerful and flexible analysis methods.
- *Show me just what matters.* Too much information may be, indeed, too much. The end-user must be able to focus on relevant information for his / her current decision making processes.
- *Everyone knows that some data is wrong.* A sensitive amount of transactional data is not correct and it has to be properly cleaned (transformed, erased, filtered, etc.) in order to avoid misleading results.

Data warehousing systems have three main components: the data warehouse, the ETL *(Extraction, Transformation and Load*) tools and the exploitation tools. The data warehouse is a huge repository of data; i.e., a database. It is the data warehousing system core and that is why these systems are also called *data warehouse systems*. However, it is not just another traditional database: it depicts a single and detailed view of the organization business. By means of the ETL tools data from a variety of sources is loaded (i.e., homogenized, cleaned and filtered) into the data warehouse. Once loaded, it is ready to be exploited by means of the exploitation tools.

1.1.1 The Data Warehouse

The data warehouse term was coined by B. Inmon in 1992 in [Inm92], which defined it as: "*a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process*". Where:

- *Subject oriented* means that data stored gives information about a particular subject instead of the daily operations of an organization. This data is clustered together in order to undertake different analysis processes over it. This fact is represented in Fig. 1.2. There, tables from the operational sources (which were thought to boost transaction performance) are broke into small pieces of data before loading the data warehouse, which is interested

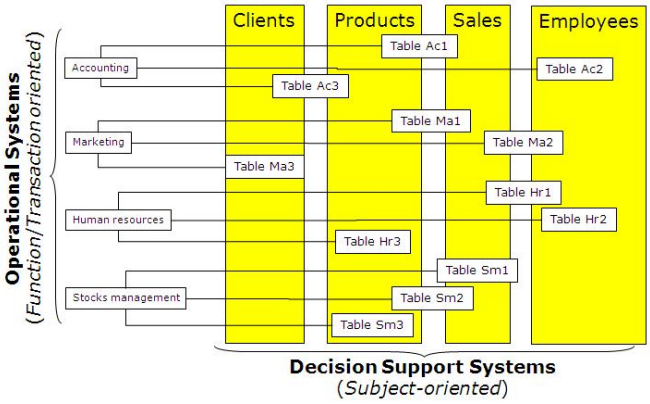


Figure 1.2: Subject oriented Vs. transaction oriented

in concepts such as clients or products whose data can be spread all over different transactional tables. Now, we can analyze concepts such as clients that were blurred on the transactional system;

- *Integrated* means that data have been gathered into the data warehouse from a variety of sources and merged into a coherent whole;

- *time-variant* means that all data in the data warehouse is identified with a particular time period and for example, historical data (which is of no interest for OLTP systems) is essential and finally,

- *non-volatile* means that data is stable in the data warehouse. Thus, more data is added but data is never removed. This enables management to gain a consistent picture of the business.

Despite this definition was introduced almost 20 years ago, it still remains reasonably accurate. However, a single-subject data warehouse is currently referred to as a *data mart* (i.e., a local or departmental data warehouse), while data warehouses are more global, giving a general enterprise view. In the literature, we can find other definitions like the one presented in [KRTR98], where a data warehouse is defined as "*a copy of transaction data specifically structured for query and analysis*"; this definition, despite being simpler, is not less compelling, since it underlines the relevance of querying in a data warehousing system. The data warehouse design is focused on improving queries performance instead of improving update statements (i.e., insert, update and delete) like transactional databases do. Moreover, the data warehousing system end-users are high-ranked people involved in decision making rather than those low/medium-ranked people maintaining and developing the organization information systems. Next table summarizes main differences between an operational database and a data warehouse (or, in general, a decisional system):

	Operational	Decisional
Objective	Business operation	Business analysis
Main functions	Daily oper. (OLTP)	Decision Support System (OLAP)
Usage	Repetitive (predefined)	Innovative (unexpected)
Design orientation	Functionality	Subject
Kind of users	Clerks	Executives
Number of users	Thousands	Hundreds
Accessed tuples	Hundreds	Thousands
Data sources	Isolated	Integrated
Granularity	Atomic	Summarized
Time coverage	Current	Historical
Access	Read/Write	Read-only
Work units	Simple transactions	Complex queries
Requirements	Performance & consistency	Performance & precision
Size	Mega/Gigabytes	Giga/Tera/Petabytes

Figure 1.3: Operational Vs. decisional systems

Most issues pointed out in table 1.3 have been already pointed out, but some others may still remain unclear. Operational (mainly transactional -OLTP- systems) focus on giving solution to the daily business necessities, whereas decisional systems, such as data warehousing, focus on providing a single, detailed view of the organization to help us on decision making. Thus, data warehouses tend to be massive, and nowadays we can find the first petabyte data warehouses (i.e., at least, one order of magnitude larger than OLTP systems). The reason is that data warehouses not only load data (including historical data) integrated from several sources but also pre-computed data aggregations derived from it. In decision making, aggregated (i.e., summarized) queries are common (for example, revenue obtained per year, or average number of pieces provided per month, etc.) since they provide new interesting perspectives of view. This is better explained in the following section.

1.1.1.1 Kinds of Data

Data extracted from the data sources (owned databases, shared with other partners, external data coming from the Web, etc.) can be classified in many ways, but we emphasized the following three categories, which helps to understand the operational - decisional systems duality:

- **Operational Vs. decisional data:** Our data sources contain heaps of data. An operational system stores any detail related to our daily processes. However, not all data is relevant for decision making. For example, many retailers are interested in our postal code when shopping but they show no interest in our address, since their decision making processes deal with city regions (represented by the postal code). Another clear example is the name and surname, which are not very interesting in the general case. Oppositely, the age and sex tend to be considered as first-class citizens in decision making. Indeed, one of the main

challenges to populate a data warehouse is to decide which data subset from our operational sources is needed for our decision making processes. Thus, the decisional data can be seen as a strategic window over the whole operational data. Furthermore, operational data quality is poor, as update transactions happen constantly and data consistency can be compromised by them. For this reason, the cleaning stage (see section 1.1.2) when loading data in the data warehouse is crucial to guarantee reliable and exact data.

- **Historical Vs. current data:** Operational sources typically store *daily* data; i.e., current data produced and handled by the organization processes. Old data (from now on, historical data), however, was typically moved away from the operational sources and stored in secondary storage systems such as tapes in form of backups. What is current or old data depends, in the end, on each organization and its processes. For example, consider an operational source keeping track of each sale in a supermarket. For performance reasons, the database only stores a one year window. In the past, older data was dumped into backup tapes and left there but nowadays, it is kept in the decisional systems. Note some features about this dichotomy. A decisional system obviously needs both. Posing queries regarding large time spans is common in decision making (such as what was the global revenue in the last 5 years). Both data, though, come from the same sources and current data will, eventually, become old. Thus, it is critical to load data periodically so that the decisional system can keep track of both.

- **Atomic, derived and aggregated data:** These concepts are related to the *data granularity* at which data is delivered. We refer to atomic data to the granularity stored by the operational sources. For example, I can store my sales as follows: the user who bought it, the shop it was sold, the exact time (up to milliseconds), the price paid and the discount applied. However, decisional systems often allow to compute derived and aggregated data from atomic data to give answer to interesting business questions. On the one hand, derived data results from computing a certain function over atomic data to produce non-evident knowledge. For example, we may be interested in computing the revenue obtained from a user, which can be obtained by applying the discount over the initial price and summing up all his sales. Another example would be data produced by data mining algorithms. Normally, different attributes or values are needed to compute derived data. On the other hand, aggregates results from applying an aggregation function for a certain value. For example, the average item price paid per user, or the global sales in 2011. Thus, it can be seen as a specific kind of derived data. Derived and aggregated data frequently queried are often pre-computed in decisional systems (although they also allow to compute them on-the-fly). The reason is that previous experiences suggest to pre-compute the most frequent aggregates and derived data in order to boost performance.

1.1.2 ETL Tools: Extraction Transformation and Load

The ETL processes extract, integrate, and clean data from operational sources to feed the data warehouse. For this reason, the ETL process operations as a whole are often defined as reconciliation. These are also the most complex and technically challenging among all the data

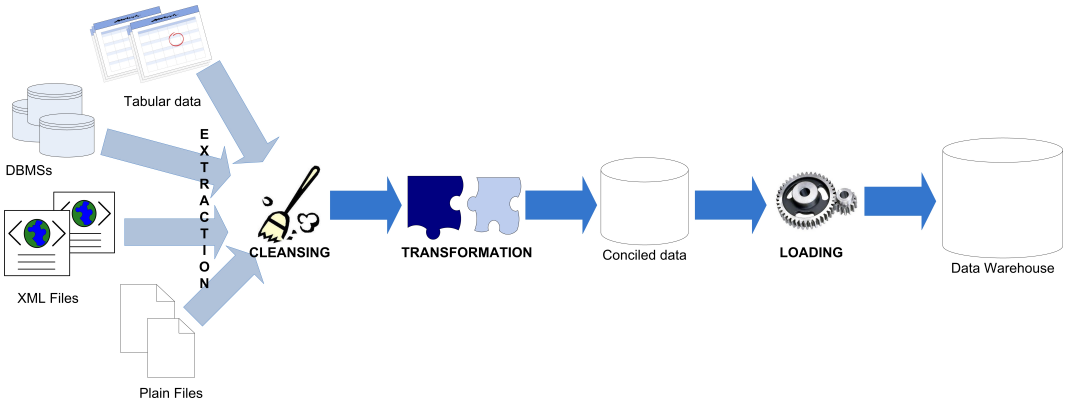


Figure 1.4: Extraction, transformation and loading

warehouse process phases (ETL processes are responsible to disregard data heterogeneities of any kind). ETL takes place once when a data warehouse is populated for the first time, then it occurs every time the data warehouse is regularly updated. Figure 1.4 shows that ETL consists of four separate phases: extraction (or capture), cleansing (or cleaning or scrubbing), transformation, and loading. In the following sections, we offer brief descriptions of these phases.

The scientific literature shows that the boundaries between cleansing and transforming are often blurred from the terminological viewpoint. For this reason, a specific operation is not always clearly assigned to one of these phases. This is obviously a formal problem, but not a substantial one. Here, cleansing is essentially aimed at rectifying data values, and transformation more specifically manages data formats.

1.1.2.1 Extraction

Relevant data is obtained from sources in the extraction phase. You can use static extraction when a data warehouse needs populating for the first time. Conceptually speaking, this looks like a snapshot of operational data. Incremental extraction, used to update data warehouses regularly, seizes the changes applied to source data since the latest extraction. Incremental extraction is often based on the log maintained by the operational DBMS. If a timestamp is associated with operational data to record exactly when the data is changed or added, it can be used to streamline the extraction process. Extraction can also be source-driven if you can rewrite operational applications to asynchronously notify of the changes being applied, or if your operational database can implement triggers associated with change transactions for relevant data. The data to be extracted is mainly selected on the basis of its quality. In particular, this depends on how comprehensive and accurate the constraints implemented in sources are, how suitable the data formats are, and how clear the schemas are.

1.1.2.2 Cleansing

The cleansing phase is crucial in a data warehouse system because it is supposed to improve data quality, normally quite poor in sources. The following list includes the most frequent mistakes and inconsistencies that make data "dirty":

- Duplicate data: For example, a patient is recorded many times in a hospital patient management system
- Inconsistent values that are logically associated: Such as addresses and ZIP codes
- Missing data: Such as a customers job
- Unexpected use of fields: For example, a SSN (social Security Number) field could be used improperly to store office phone numbers
- Impossible or wrong values: Such as 30/2/2009
- Inconsistent values for a single entity because different practices were used: For example, to specify a country, you can use an international country abbreviation (I) or a full country name (Italy); similar problems arise with addresses (Hamlet Rd. and Hamlet Road)
- Inconsistent values for one individual entity because of typing mistakes: Such as Hamet Road instead of Hamlet Road

In particular, note that the last two types of mistakes are very frequent when you are managing multiple sources and are entering data manually. The main data cleansing features found in ETL tools are rectification and homogenization. They use specific dictionaries to rectify typing mistakes and to recognize synonyms, as well as rule-based cleansing to enforce domain-specific rules and define appropriate associations between values.

1.1.2.3 Transformation

Transformation is the core of the reconciliation phase. It converts data from its operational source format into a specific data warehouse format. Establishing a mapping between the data sources and the data warehouse is generally made difficult by the presence of many different, heterogeneous sources. If this is the case, a complex integration phase is required when designing your data warehouse. The following points must be rectified in this phase:

- Loose texts may hide valuable information. For example, BigDeal LtD does not explicitly show that this is a Limited Partnership company.
- Different formats can be used for individual data. For example, a date can be saved as a string or as three integers.

Following are the main transformation processes at this stage:

- Conversion and normalization that operate on both storage formats and units of measure to make data uniform.

- Matching that associates equivalent fields in different sources.
- Selection that reduces the number of source fields and records.

When populating a data warehouse, you most surely may need to sum up data properly and pre-compute interesting data aggregations. So that, pre-aggregated data may be needed to be computed at this point.

1.1.2.4 Loading

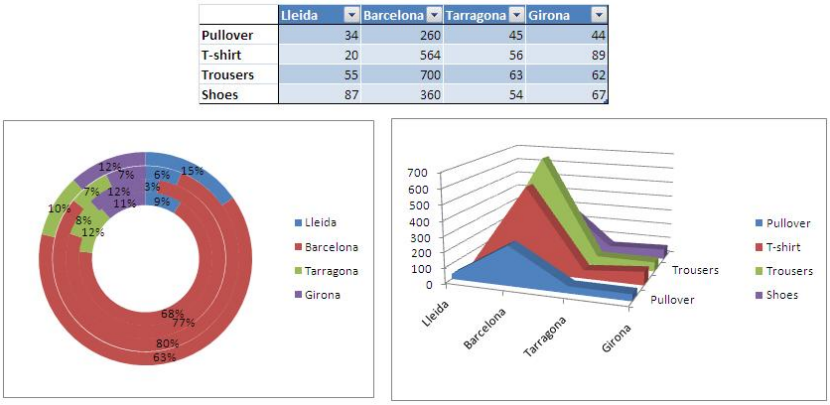
Loading into a data warehouse is the last step to take. Loading can be carried out in two ways:

- Refresh: Data warehouse data is completely rewritten. This means that older data is replaced. Refresh is normally used in combination with static extraction to initially populate a data warehouse.
- Update: Only those changes applied to source data are added to the data warehouse. Update is typically carried out without deleting or modifying preexisting data. This technique is used in combination with incremental extraction to update data warehouses regularly.

1.1.3 Exploitation Tools

The final aim of every data warehousing system is to exploit the data warehouse. The data warehouse is a huge repository of data that does not tell much by itself; like in the operational databases field, we need auxiliary tools to query and analyze data stored. In this field, those tools aimed at extracting relevant information from the repository of data are known as the exploitation tools. Without the appropriate exploitation tools, we will not be able to extract valuable knowledge of the organization from the data warehouse, and the whole system will fail in its aim of providing information for giving support to decision making. Most used exploitation tools can be classified in three categories:

- Query & Reporting: This category embraces the evolution and optimization of the traditional query & reporting techniques. This concept refers to an exploitation technique consisting of querying data and generating detailed pre-defined reports to be interpreted by the end-user. Mainly, this approach is oriented to those users who need to have regular access to the information in an almost static way. A report is defined by a query and a layout. A query generally implies a restriction and an aggregation of multidimensional data. For example, you can look for the monthly receipts during the last quarter for every product category. A layout can look like a table or a chart (diagrams, histograms, pies, and so on). Figure 1.5 shows a few examples of layouts for a query.
- Data Mining: Data mining is the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules. The data mining field is a research area per se, but as the reader may note, this kind of techniques and tools suit perfectly to the final goal of data warehousing systems. Typically, data from the data warehouse is dumped



	Lleida	Barcelona	Tarragona	Girona
Pullover	34	260	45	44
T-shirt	20	564	56	89
Trousers	55	700	63	62
Shoes	87	360	54	67

Figure 1.5: Query & reporting

into plain files that feed the data mining algorithms. Thus, both techniques are traditionally used sequentially: populate the data warehouse and then, select relevant data to be analyzed by data mining algorithms. Recently, though, new techniques to tight the relationship between both worlds have been addressed, such as performing the data mining algorithms *inside* the data warehouse and avoiding the bottleneck produced by moving data out. However, this kind of approaches stay out of this course contents.

- OLAP Tools: OLAP stands for *On-Line Analytical Processing*, which was carefully chosen to confront the OLTP acronym (*On-Line Transactional Processing*). Its main objective is to analyze business data from its dimensional or components perspective; unlike traditional operational systems such as OLTP systems. For a deep insight on OLAP, see chapter 3.

Chapter 2

Architecture

The following architecture properties are essential for a data warehouse system[1]:

- **Separation:** Analytical and transactional processing should be kept apart as much as possible.
- **Scalability:** Hardware and software architectures should be easy to upgrade as the data volume, which has to be managed and processed, and the number of users' requirements, which have to be met, progressively increase.
- **Extensibility:** The architecture should be able to host new applications and technologies without redesigning the whole system.
- **Security:** Monitoring accesses is essential because of the strategic data stored in data warehouses.
- **Administerability:** Data warehouse management should not be overly difficult.

In the following, sections 2.1, 2.2, and 2.3 present a structure-oriented classification that depends on the number of layers used by the architecture.

2.1 Single-layer Architecture

A single-layer architecture is not frequently used in practice. Its goal is to minimize the amount of data stored; to reach this goal, it removes data redundancies. Figure 2.1 shows the only layer physically available: the source layer. In this case, data warehouses are virtual. This means that a data warehouse is implemented as a multidimensional view of operational data created by specific middleware, or an intermediate processing layer.

[1]This chapter is mainly based on the book "Data Warehouse Design: Modern Principles and Methodologies", published by McGraw Hill and authored by Matteo Golfarelli and Stefano Rizzi, 2009 [GR09].

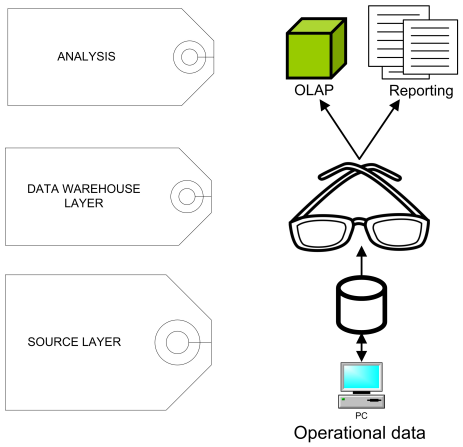


Figure 2.1: Single-layer architecture for a data warehouse system

The weakness of this architecture lies in its failure to meet the requirement for separation between analytical and transactional processing. Analysis queries are submitted to operational data after the middleware interprets them. It this way, the queries affect regular transactional workloads. In addition, although this architecture can meet the requirement for integration and correctness of data, it cannot log more data than sources do. For these reasons, a virtual approach to data warehouses can be successful only if analysis needs are particularly restricted and the data volume to analyze is not huge.

2.2 Two-layer Architecture

The requirement for separation plays a fundamental role in defining the typical architecture for a data warehouse system, as shown in Figure 2.2. Although it is typically called a two-layer architecture to highlight a separation between physically available sources and data warehouses, it actually consists of four subsequent data flow stages:

- **Source layer:** A data warehouse system uses heterogeneous sources of data. That means that data feeding the data warehouse might come from inside the organization or from external sources (such as the Cloud, the Web, or from partners). Furthermore, the technologies and formats used to store this data may also be heterogeneous (legacy[2] systems,

[2]The term legacy system denotes corporate applications, typically running on mainframes or minicomputers, that are currently used for operational tasks but do not meet modern architectural principles and current standards. For this

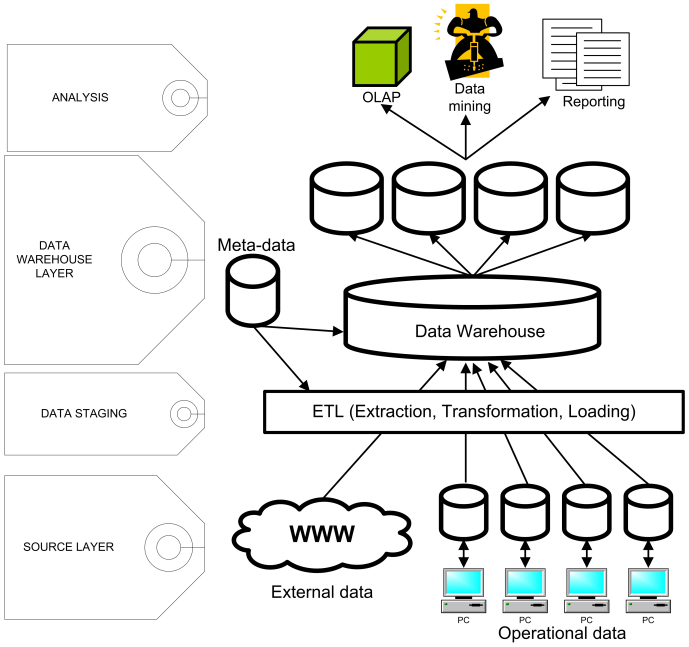


Figure 2.2: Two-layer architecture for a data warehouse system

relational databases, XML files, plain text files, e-mails, pdf files, Excel and tabular tables, OCR files, etc.). Furthermore, some areas have their specificities; for example, in medicine other data inputs such as images or medical tests must be treated as first-class citizens.

- **Data staging:** The data stored to sources should be extracted, cleansed to remove inconsistencies and fill gaps, and integrated to merge heterogeneous sources into one common schema. The so-called Extraction, Transformation, and Loading tools (ETL), can merge heterogeneous schemas, extract, transform, cleanse, validate, filter, and load source data into a data warehouse. Technologically speaking, this stage deals with problems that are typical for distributed information systems, such as inconsistent data management and incompatible data structures. Section 1.1.2 deals with a few points that are relevant to data staging.

- **Data warehouse layer**: Information is stored to one logically centralized single repository: a data warehouse. The data warehouse can be directly accessed, but it can also be used as a source for creating *data marts* (in short, local data warehouses), which partially replicate data warehouse contents and are designed for specific enterprise departments.

reason, accessing legacy systems and integrating them with more recent applications is a complex task. All applications that use a pre-relational database are examples of legacy systems.

Meta-data repositories (see section 2.4) store information on sources, access procedures, data staging, users, data mart schemas, and so on.

- **Analysis:** In this layer, integrated data is efficiently and flexibly accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios. Technologically speaking, it should feature aggregate data navigators, complex query optimizers, and user-friendly GUIs. Section 1.1.3 deals with different types of decision-making support analyses.

The architectural difference between data warehouses and data marts needs to be studied closer. The component marked as a data warehouse in Figure 2.2 is also often called the primary data warehouse or corporate data warehouse. It acts as a centralized storage system for all the data being summed up. Data marts can be viewed as small, local data warehouses replicating (and summing up as much as possible) the part of a primary data warehouse required for a specific application domain. More formally, A *data mart* is a subset or an aggregation of the data stored to a primary data warehouse. It includes a set of information pieces relevant to a specific business area, corporate department, or category of users. The data marts populated from a primary data warehouse are often called dependent. Although data marts are not strictly necessary, they are very useful for data warehouse systems in midsize to large enterprises because:

- they are used as building blocks while incrementally developing data warehouses;
- they mark out the information required by a specific group of users to solve queries;
- they can deliver better performance because they are smaller than primary data warehouses.

Sometimes, mainly for organization and policy purposes, you should use a different architecture in which sources are used to directly populate data marts. These data marts are called independent. If there is no primary data warehouse, this streamlines the design process, but it leads to the risk of inconsistencies between data marts. To avoid these problems, you can create a primary data warehouse and still have independent data marts. In comparison with the standard two-layer architecture of Figure 2.2, the roles of data marts and data warehouses are actually inverted. In this case, the data warehouse is populated from its data marts, and it can be directly queried to make access patterns as easy as possible. The following list sums up all the benefits of a two-layer architecture, in which a data warehouse separates sources from analysis applications:

- In data warehouse systems, good quality information is always available, even when access to sources is denied temporarily for technical or organizational reasons.
- Data warehouse analysis queries do not affect the management of transactions, the reliability of which is vital for enterprises to work properly at an operational level.
- Data warehouses are logically structured according to the multidimensional model (see chapter 3), while operational sources are generally based on relational or semi-structured models.

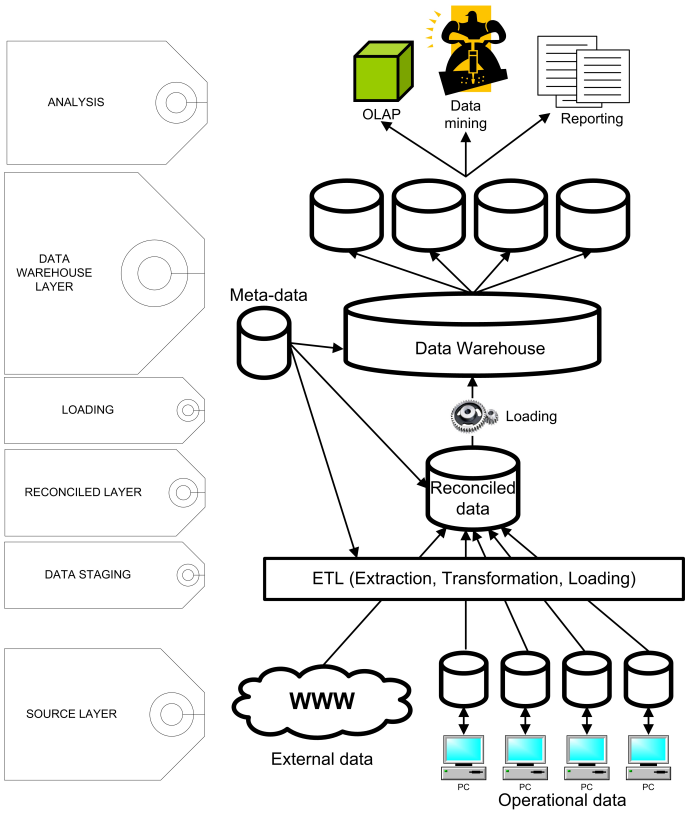


Figure 2.3: Three-layer architecture for a data warehouse system

- A mismatch in terms of time and granularity occurs between OLTP systems, which manage current data at a maximum level of detail, and OLAP systems, which manage historical and aggregated data.

- Data warehouses can use specific design solutions aimed at performance optimization of analysis and report applications.

Finally, it is worth to pay attention to the fact that a few authors use the same terminology to define different concepts. In particular, those authors consider a data warehouse as a repository of integrated and consistent, yet operational, data, while they use a multidimensional representation of data only in data marts. According to our terminology, this "operational view" of data warehouses essentially corresponds to the reconciled data layer in three-layer architectures.

2.3 Three-layer Architecture

In this architecture, the third layer is the reconciled data layer or operational data store. This layer materializes operational data obtained after integrating and cleansing source data. As a result, those data are integrated, consistent, correct, current, and detailed. Figure 2.3 shows a data warehouse that is not populated from its sources directly, but from reconciled data. The main advantage of the reconciled data layer is that it creates a common reference data model for a whole enterprise. At the same time, it sharply separates the problems of source data extraction and integration from those of data warehouse population. Remarkably, in some cases, the reconciled layer is also directly used to better accomplish some operational tasks, such as producing daily reports that cannot be satisfactorily prepared using the corporate applications, or generating data flows to feed external processes periodically so as to benefit from cleaning and integration. However, reconciled data leads to more redundancy of operational source data. Note that we may assume that even two-layer architectures can have a reconciled layer that is not specifically materialized, but only virtual, because it is defined as a consistent integrated view of operational source data.

2.4 Meta-data

In addition to those kinds of data already discussed in section 1.1.1.1, meta-data represents a key aspect for the discussed architectures. Prior to come up with a formal definition for meta-data, let us formally introduce the data and information concepts. According to the ISO definition, *data* is a representation of facts, concepts and instructions, done in a formalized manner, useful for communication, interpretation and process, by human beings as well as automated means. Information, however, is something more. According to the ISO definition, *information*, in the processing of data and office machines, is the meaning given to data from the conventional rules used in their representation. The difference can be seen crystal clear with an example. Consider the following data extracted from a database: 1100. So what information represents this data? to answer this, it will help to know that this data is in *binary* format, its type is an *integer*, represents the *age* attribute (in *months*) from a table named *dogs* and this data is updated every *year* (and last update was last *December*). All this data about the 1100 data needs to be stored in order to process it as information. This is what we know as meta-data, which is applied to the data used to define other data. In short, it is data that allows to interpret data as information. A typical meta-data repository is the relational databases catalog.

In the scope of data warehousing, meta-data plays an essential role because it specifies source, values, usage, and features of data warehouse data and defines how data can be changed and processed at every architecture layer. Figures 2.2 and 2.3 show that the meta-data repository is closely connected to the data warehouse. Applications use it intensively to carry out data-staging and analysis tasks. One can classify meta-data into two partially overlapping categories. This classification is based on the ways system administrators and end users exploit meta-data. System administrators are interested in internal (technical) meta-data because it defines data sources, transformation processes, population policies, logical and physical schemas, constraints, and user profiles. External (business) meta-data is relevant to end users. For example, it is about

definitions, quality standards, units of measure, relevant aggregations, etc.

Meta-data is stored in a meta-data repository which all the other architecture components can access. A tool for meta-data management should:

- allow administrators to perform system administration operations, and in particular manage security;
- allow end users to navigate and query meta-data;
- use a GUI;
- allow end users to extend meta-data;
- allow meta-data to be imported/exported into/from other standard tools and formats.

Chapter 3

OLAP and the Multidimensional Model

OLAP tools are intended to ease information analysis and navigation all through the data warehouse, for extracting relevant knowledge of the organization. This term was first introduced by E.F. Codd in 1993 [CCS93], and it was carefully chosen to confront OLTP. Consider Fig. 3.1, which depicts a conceptual transactional model. These models were defined to:

- Reduce the amount of redundant data
- Eliminate the need to modify many records because of one modification, very efficient if data change very often

However, it is also well-known that:

- Degrades response time in front of queries (mainly due to the presence of join operators)
- It is easy to make mistakes if the user is not an expert in computer science

These pros and cons suit perfectly operational (typically transactional) systems, but it does not suit decisional systems like data warehouses, which aim at fast answering and easy navigation of data. OLAP tools are precisely defined by means of the FASMI (*Fast Analysis of Shared Multidimensional Information*) test [Pen08]. According to it, an OLAP tool must provide *Fast* query answering to not frustrate the end-user reasoning; offer *Analysis* tools, implement security and concurrent mechanisms to *Share* the business *Information* from a *Multidimensional* point of view. This last feature is the most important one since OLAP tools are conceived to exploit the data warehouse for analysis tasks based on *multidimensionality*.

Right now, we can talk about the multidimensional model, which plays the same role as the relational model for relational databases. As a model, we can talk about the multidimensional data structure, multidimensional integrity constraints (what constraints must be preserved) and multidimensional algebra (aimed at manipulating data). Unfortunately, unlike relational databases, there is not yet consensus about a standard multidimensional model (among other

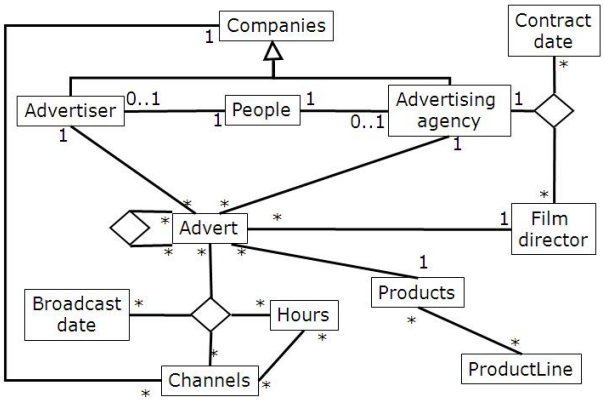


Figure 3.1: A transactional model

reasons because major software vendors are not interested to reach such agreement). However, we can nowadays talk about a de facto multidimensional data structure (or multidimensionality), and to some extent, about a de facto multidimensional algebra, which we next present at three different levels: what reality multidimensionality models, and how this reality can be represented at the conceptual and logical (and physical) level.

3.1 The Real World: Events and Multidimensionality

The multidimensional point of view is based on the cube metaphor. Specifically, the multidimensional conceptual view of data is distinguished by the *fact* / *dimension* dichotomy, and it is characterized by representing data as if placed in an n-dimensional space, allowing us to easily understand and analyze data in terms of *facts* (the subjects of analysis) and *dimensions* showing the different points of view from where a subject can be analyzed. For instance, Fig. 3.2 depicts `sales` (subject of analysis) of an organization from three different dimensions or perspectives of view (`time`, `product` and `place`). In each cube cell the sales data would be placed for the corresponding place, product and time. One fact and several dimensions to analyze it give rise to what is known as the *data cube*[1].

This paradigm, unlike SQL and relational data structure, provides a friendly, easy-to-understand and intuitive visualization of data for non-expert end-users. Importantly, most real-world events recorded nowadays are likely to be analyzed from a multidimensional point of view. An event (a potential fact) is recorded altogether with a set of relevant attributes (potential dimensions). For example, consider a `sales` event. We may record the `shop`, `city` and `country` where it was purchased, the `item`, `color`, `size` and `add-ons` selected, the `time` (`hour`, `minute`,

[1]The data cube refers to the placement of factual data in the multidimensional space. And thus, it can be thought as a mathematical function. Nevertheless, nowadays it is rather common also refer to the multidimensional space as the data cube. However, note that, in both cases, it is a language abuse, since the multidimensional space (or the placement of data in the multidimensional space) only gives rise to a cube if three analysis dimensions are considered.

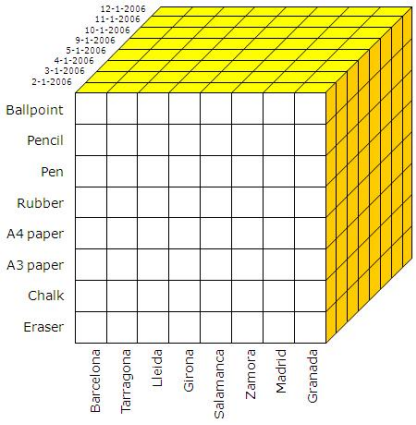


Figure 3.2: Multidimensional view of data

`second` and even `millisecond`) and `date` (`day`, `month`, `year`), `payment method`, `price`, `discount applied`, the `customer`, etc. Interestingly, every attribute opens a new perspective of analysis for the `sales` event. In short, multidimensionality is used to model any real-world event consisting of metrics (facts of interest) and a set of descriptive attributes (dimensions) related to these metrics.

More precisely, OLAP functionality is characterized by dynamic multidimensional analysis of consolidated data supporting end-user analytical and navigational activities. Thus, OLAP users must be able to navigate (i.e., query and analyze) data in real-time. The user provides a *navigation path* in which each node (resulting in a data cube) is derived from the previous node in the path (and thus we say that the user *navigates* the data). Each node is transformed into the next one in the path by applying specific multidimensional operators. Most popular multidimensional operators[2] are "roll-up" (increase the aggregation level), "drill-down" (decrease the aggregation level), "slicing and dicing" (specify a single value for one or more members of a dimension) and "pivot" (reorient the multidimensional view). Some works, add "drill-across" (combine data from cubes sharing one or more dimensions) to these basic operations.

For example, consider the data cube in Fig. 3.2. Now, the user could decide to slice it by setting constraints (e.g., product == 'eraser', date == '2-1-2006', city != 'Barcelona', etc.) over any of the three dimensional axis (see Fig. 3.3) or apply several constraints to more than one axis (see Fig. 3.4). In general, after applying a multidimensional operator, we obtain another cube that we can further navigate with other operations. As a whole, the set of operators applied over the initial cube is what we call the navigation path.

As a result, multidimensionality enables analysts, managers, executives and in general those people involved in decision making, to gain insight into data through fast queries and analytical tasks, allowing them to make better decisions.

[2]Right now, these operators are intended to be seen as an example. Later in this chapter we will discuss a multidimensional algebra in depth.

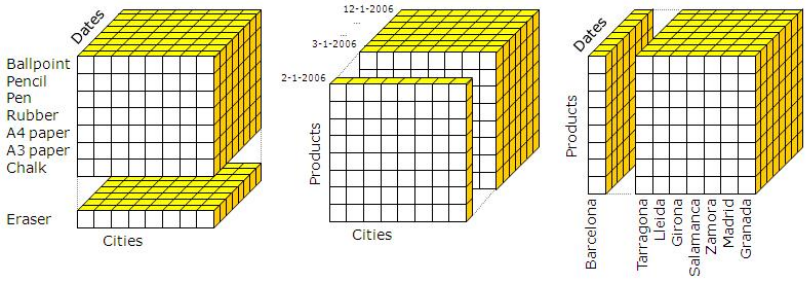


Figure 3.3: Different examples of slice

3.1.1 Presentation Issues

Although multidimensional events are based on the cube metaphor, any real commercial product hardly shows data cube in 3D. Instead, they flatten these cubes into tabular tables or alternative graphical representations, in what is usually known as BI dashboards (like the one shown in Fig. 1.5). For example, consider Fig. 3.5. There, a *cube* showing sales from January to April 2006 for any kind of products considered as paper or writing tools is shown in tabular form. This cube could be manipulated by the user by means of drill-down (i.e., produce a finer data granularity) and thus, data about specific paper types (such as A3 and A4) and writing tools (such as ballpoints and pencils) are shown (again in tabular form). As a side note, note, we may get back to the previous data granularity by means of the roll-up operator.

3.2 Conceptual Design: Star Schemas

Lots of efforts have been devoted to multidimensional modeling, and several models and approaches have been developed and presented in the literature to support the conceptual design of the data warehouse. Multidimensionality, as it is known today, was first introduced by Kimball in [Kim96], where the author argued about the necessity of an ad hoc modeling technique for data warehouses. Multidimensional modeling optimizes the system query performance in

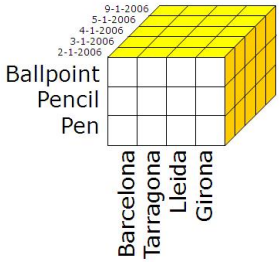
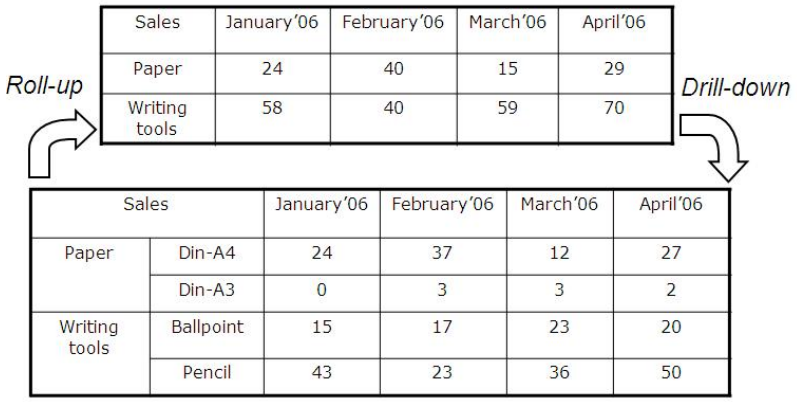


Figure 3.4: Example of dice



Sales	January'06	February'06	March'06	April'06
Paper	24	40	15	29
Writing tools	58	40	59	70

Sales		January'06	February'06	March'06	April'06
Paper	Din-A4	24	37	12	27
	Din-A3	0	3	3	2
Writing tools	Ballpoint	15	17	23	20
	Pencil	43	23	36	50

Figure 3.5: An example of roll-up / drill-down over a cube represented in tabular form

contrast to conventional *Entity-Relationship* (ER) models [Che76] (widely used for modeling relational databases) that are constituted to remove redundancy in the data model and optimize OLTP performance.

As discussed, multidimensionality is based on the fact / dimension dichotomy. **Dimensional concepts** produce the multidimensional space in which the **fact** is placed. **Dimensional concepts** are those concepts likely to be used as a new analytical perspective, which have traditionally been classified as **dimensions**, **levels** and **descriptors**. Thus, we consider that a **dimension** consists of a hierarchy of **levels** representing different granularities (or levels of detail) for studying data, and a **level** containing **descriptors** (i.e., **level** attributes). We denote by **atomic level** the **level** at the bottom of the **dimension** hierarchy (i.e., that of the finest level of detail) and by **All level** the **level** at the top of the hierarchy containing just one instance representing the whole set of instances in the **dimension**. In contrast, a **fact** contains **measures** of analysis. Importantly, note that a **fact** may produce not just one but several different levels of data granularity. Therefore, we say that a certain **granularity** contains individual cells of the same granularity from the same **fact**. A specific **granularity** of data is related to one **level** for each of its associated **dimensions** of analysis. Finally, one **fact** and several **dimensions** for its analysis produce what Kimball called a star schema.

For example, consider Figure 3.6, which depicts a star-schema. There, one **fact** (`sales`) containing two **measures** (`price` and `discount`) is depicted. This fact has three different **dimensions** of analysis (`place`, `time` and `product` sold). Each **dimension** has its own aggregation hierarchy. For instance, the `time` **dimension** has three **levels** of detail (i.e., `year`, `month` and `day`) that, in turn, contain some **descriptors** (for example, the `holiday` or `leap year` attributes). Furthermore, the `sales` **fact** may produce several **granularities**. Each **level** available for a given **fact** gives rise to a **granularity** within that **fact**. Thus, we may analyze this **fact** from a wide range of data granularities: from its finest data granularity (i.e, {`product` × `day` × `city`}, which is the one

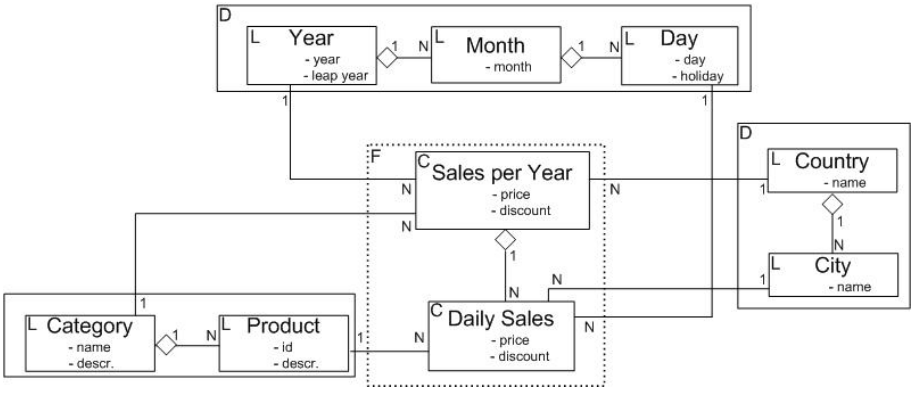


Figure 3.6: The multidimensional concepts

shown in the figure as `daily sales`) up to its coarsest data granularity[3] (i.e., {`category` × `year` × `country`}, which is shown as `sales per year`). Any of the factual instances within `daily sales` or `sales per year` is called a cell.

Finally, note that we consider {`product` × `day` × `city`} to be the multidimensional **base** of the the finest fact granularity level (i.e., that related to the **atomic levels** of each **dimension**, which is also known as the **atomic granularity**) and {`category` × `year` × `country`} to be the **base** of the **coarsest data granularity**. Thus it means that one value of each one of these **levels** determines one cell (i.e., a `sale` with its `price` and `discount`). Importantly, this is a relevant feature of multidimensionality. Levels determine factual data or, in other words, they can be depicted as **functional dependencies** (the set of levels determine the fact, and each fact cell has associated a single value from each level). That is the reason why level - fact relationships have 1:N multiplicities. In the multidimensional model, N:N relationships are meaningless as they do not preserve the model constraints.

Recall now the transactional schema used as example in Fig. 3.1. Now, you should be able to distinguish why it cannot be considered a multidimensional schema. Oppositely, check the multidimensional schemas in Figs. 3.7 and 3.8 and see the differences:

- There is a single fact (with some measures or metrics)
- There is a set of dimensions (with descriptive values)
- Only 1:N relationships are considered

Furthermore, they only include data relevant for decision making. Thus, they are simpler and do not contain as much data as transactional schemas. Consequently, they are easier to use and understand, and queries are fast and efficient over such schemas. Note another important property of these schemas: dimensions are represented as a single object. For example, in a relational implementation (see next section for further details), it would imply there is only one table for

[3]Note that we do not consider the **All levels** in this example.

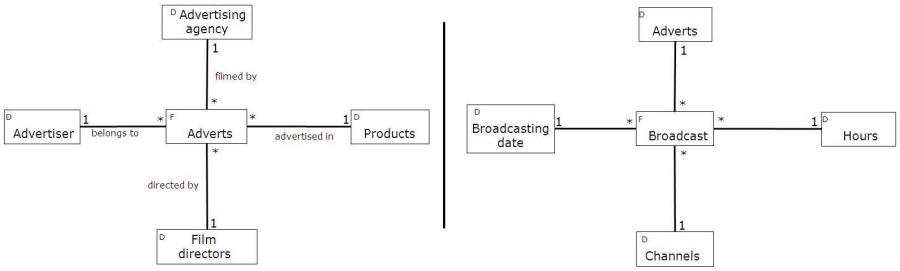


Figure 3.7: Two examples of star-schemas

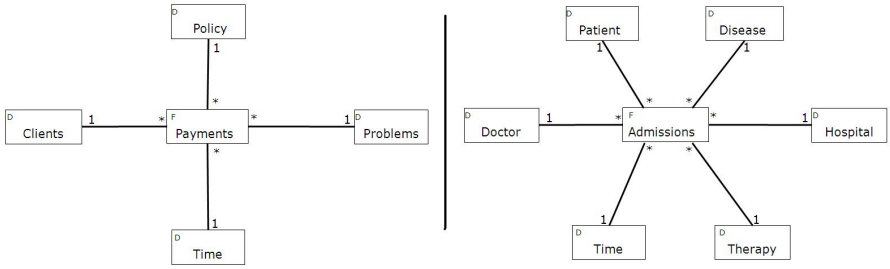


Figure 3.8: Two (more) examples of star-schemas

all the dimension. Clearly, this is against the second and third normal forms, but denormalization of data is usual in data warehousing as we aim to boost querying performance and, in this way, we avoid joins, the most expensive relational operator. Denormalization is acceptable in such scenarios since the users are not allowed to insert data, and they only query the data warehouse. Thus, since the ETL process is the only responsible to insert data is sound and acceptable to denormalize such schemas.

Although the star-schema is the most popular conceptual representation, there are some others such as the snowflake schema and the constellation schema. The first one (see the left-hand side schema on Fig. 3.9) corresponds to a normalized star-schema; i.e., without denormalizing dimensions and hence each concept is represented as an object. Finally, when a schema contains more than one fact, which share dimensions is called a constellation (see the right-hand side schema on Fig. 3.9).

3.3 Logical Design: ROLAP Vs. MOLAP

When implementing our data warehouse, there are two main trends: using the relational technology or an ad hoc one, giving rise, respectively, to what are known as ROLAP (*Relational On-line Analytical Processing*) and MOLAP (*Multidimensional On-line Analytical Processing*) architectures. ROLAP maps the multidimensional model over the relational one (a multidimensional middleware on the top of the relational database makes this fact transparent for the users),

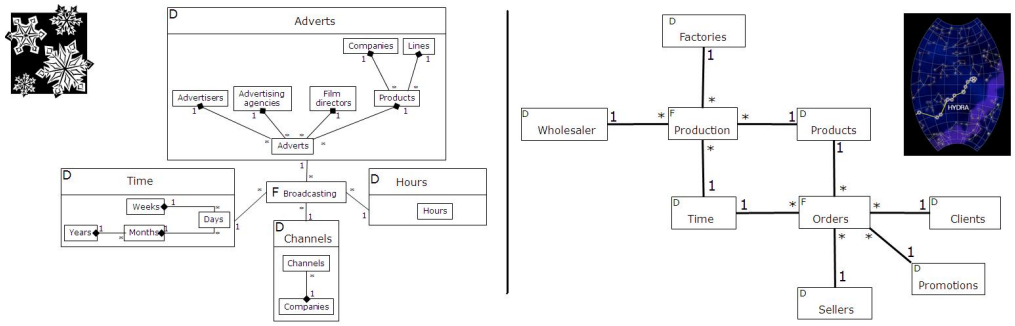


Figure 3.9: Examples of snowflake and constellation schemas

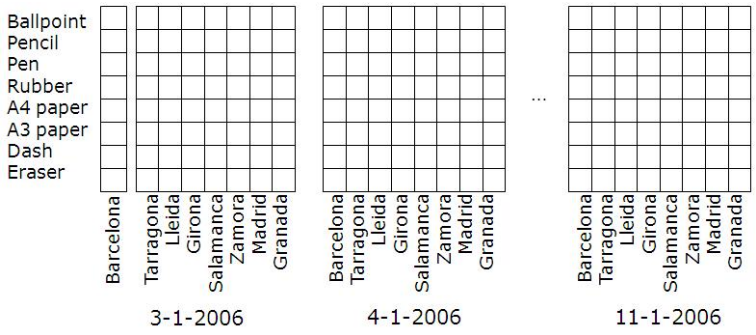


Figure 3.10: Example of a MOLAP tool storing data as arrays

allowing them to take advantage of a well-known and established technology, whereas MOLAP systems are based on an ad hoc logical model that can be used to represent multidimensional data and operations directly. The underlying multidimensional database physically stores data as arrays (and the access to it is positional), Grid-files, R*-trees or UB-trees, which are among the techniques used for this purpose (see Fig. 3.10).

As consequence, ROLAP tools used to deal (nowadays, this statement is starting to crumble) with larger volumes of data than MOLAP tools (i.e., ad hoc multidimensional solutions), but their performance for query answering and cube browsing is not as good (mainly because relational technology was conceived for OLTP systems and they tend to generate too many joins when dealing with multidimensionality). Thus, new HOLAP (*Hybrid On-line Analytical Processing*) tools were proposed. HOLAP architecture combines both ROLAP and MOLAP ones trying to obtain the strengths of both approaches, and they usually allow to change from ROLAP to MOLAP and viceversa. Specifically, HOLAP takes advantage of the standardization level and the ability to manage large amounts of data from ROLAP implementations, and the query speed typical of MOLAP systems. HOLAP implies that the largest amount of data should be stored in a relational DBMS to avoid the problems caused by sparsity, and that a multidimensional

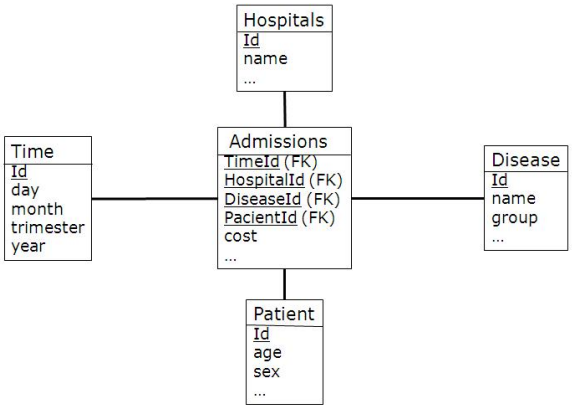


Figure 3.11: Example of a star-join schema corresponding to the right-hand star-schema in Fig. 3.8

system stores only the information users most frequently need to access. If that information is not enough to solve queries, the system will transparently access the part of the data managed by the relational system.

Although ROLAP tools have failed to dominate the OLAP market due to its severe limitations (mainly slow query answering) [Pen05], at the beginning, they were the reference architecture. Indeed, Kimball's reference book [KRTR98] presented how a data warehouse should be implemented over a relational DBMS (*Relational Database Management System*) and how to retrieve data from it. To do so, he introduced for first time the *star-join* (to implement star-schemas) and *snowflake* schemas (to implement the conceptual schemas with the same name). At relational level, the star schema consists of one table for the fact and one denormalized table for every dimension, with the latter being pointed by *foreign keys* (FK) from the *fact table*, which compose its *primary key* (PK) (see Fig. 3.11). The normalized version of a star schema is a snowflake schema; getting a table for each level with a FK pointing to each of its parents in the dimension hierarchy. Nevertheless, both approaches can be conceptually generalized into a more generic one consisting in partially normalizing the dimension tables according to our needs: completely normalizing each dimension we get a snowflake schema, and not normalizing them at all results in a star schema.

To retrieve data from a star-join schema, Kimball presented the SQL cube-query pattern:

```
SELECT l1.ID, ..., ln.ID, [ F( )c.Measure1[ ) ], ...
FROM Fact c, Level1 l1, ..., Leveln ln
WHERE c.key1=l1.ID AND ...  AND c.keyn=ln.ID [ AND li.attr Op.  K ]
[ GROUP BY l1.ID, ..., ln.ID ]
[ ORDER BY l1.ID, ..., ln.ID ]
```

The FROM clause contains the "Fact table" and the "Dimension (or level in case of a snowflake schema) tables". These tables are properly linked in the WHERE clause by "joins" (if a star-schema, only between the fact and dimension tables. In case of snowflake schema, also between

Hospital	Month	Average Cost
Duran i Reinals	January'06	3300
Duran i Reinals	February'06	4500
Duran i Reinals	...	...
Duran i Reinals	All	4300
Bellvitge	January'06	180
Bellvitge	February'06	300
Bellvitge	...	...
Bellvitge	All	200

Figure 3.12: Example of the output produced by a cube-query

dimensional tables) that represent *concept associations*. The WHERE clause also contains logical clauses restricting a specific **level** attribute (i.e., a **descriptor**) to a constant using a comparison operator (used to slice the produced cube). The GROUP BY clause shows the identifiers of the **levels** at which we want to aggregate data. Those columns in the grouping must also be in the SELECT clause to identify the values in the result. Finally, the ORDER BY clause is designed to sort the output of the query. As output, a cube-query will produce a single data cube (i.e., a specific level of data granularity). For example, think of the cube-query necessary to produce the cube (shown in tabular form) in Fig. 3.12, from the star-join schema in Fig. 3.11.

3.3.1 Implementing a Multidimensional Algebra

In a ROLAP implementation, the multidimensional operators are implemented as modifications in the cube-query clauses. To show how it would work, we first introduce a multidimensional algebra and discuss how each operator is implemented over the cube-query.

For the sake of understandability, we present the algebra by means of an example. Consider a snowflake implementation of the conceptual schema depicted in Fig. 3.6 (see page 30). The cube-query that would retrieve the `daily sales` cube is the following:

```
SELECT d.day, p.id, c.name, s.price, s.discount
FROM sales s, day d, product p, city c
WHERE s.product_id = p.id AND s.day = d.day
AND s.city_name = c.name
```

Note that no grouping is needed as we are just retrieving atomic data. Next, we show how this cube-query would be modified by each multidimensional operator next introduced (we suggest the reader to follow Fig. 3.13 to grasp the idea behind each operator):

- **Selection or Dice**: By means of a logic predicate over the dimension attributes, this operation allows users to choose the subset of points of interest out of the whole n-dimensional space (remember to check Fig. 3.13).

  In SQL, it means to *and* the corresponding comparison clause to the cube-query WHERE clause. For example, consider the atomic cube-query presented as example. If we want to analyze the `sales` data regarding to the `city` of `Barcelona`, we must perform a **selection** over the `city` dimension (see Fig. 3.14).

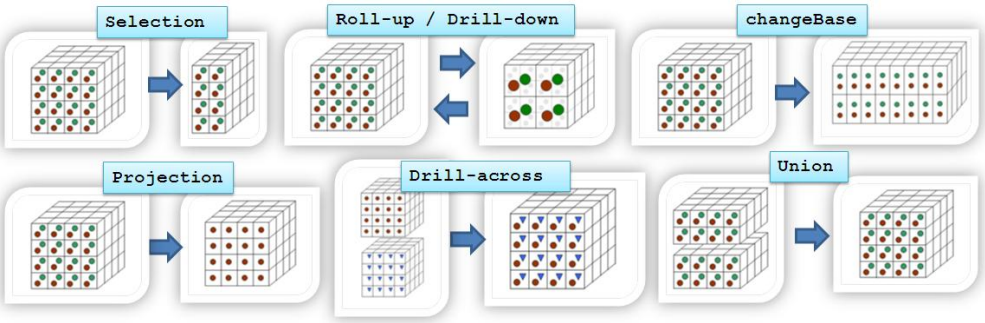


Figure 3.13: Conceptual representation of the multidimensional operators

- **Roll-up**: Also called "Drill-up", it groups cells in a Cube based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relationship which relates instances of two levels in the same dimension. For example, it is possible to roll-up monthly sales into yearly sales moving from "Month" to "Year" level along the temporal dimension.

  In SQL, it entails to replace the identifiers of the level from where we **roll-up** with those of the level that we **roll-up** to. Thus, the SELECT, GROUP BY and ORDER BY clauses must be modified accordingly. Measures in the SELECT clause must also be summarized using an aggregation function. In our example (see Figure 3.14), we perform two different **roll-ups**: on the one hand, we **roll-up** from `product_id` to the `All` **level**. On the other hand, we **roll-up** from `city` to `country`. Note that the `country` table is added to the FROM clause, and we replace the `city` identifier with that of the `country` level in the SELECT, GROUP BY and ORDER BY clauses. Finally, we add the proper links in the WHERE clause. About **rolling-up** from `product` to the `All` level, note that it is equivalent to remove both the `product` identifiers and its links.

- **Drill-down:** This is the counterpart of Roll-up. Thus, it removes the effect of that operation by going down through an aggregation hierarchy, and showing more detailed data.

  In SQL, Drill-down can only be performed by undoing (i.e, changes introduced in the cube-query) the Roll-up operation.

- **ChangeBase**: This operation reallocates exactly the same instances of a cube into a new n-dimensional space with exactly the same number of points. Actually, it allows two different kinds of changes in the space: rearranging the multidimensional space by reordering the dimensions, interchanging rows and columns in the tabular representation (this is also known as pivoting), or adding/removing dimensions to/from the space.

  In SQL it can be performed in two different ways. If we reorder the base (i.e., when "pivoting"), we just need to reorder the identifiers in the ORDER BY and SELECT clauses. But if **changing the base**, we need to add the new level tables to the FROM and the

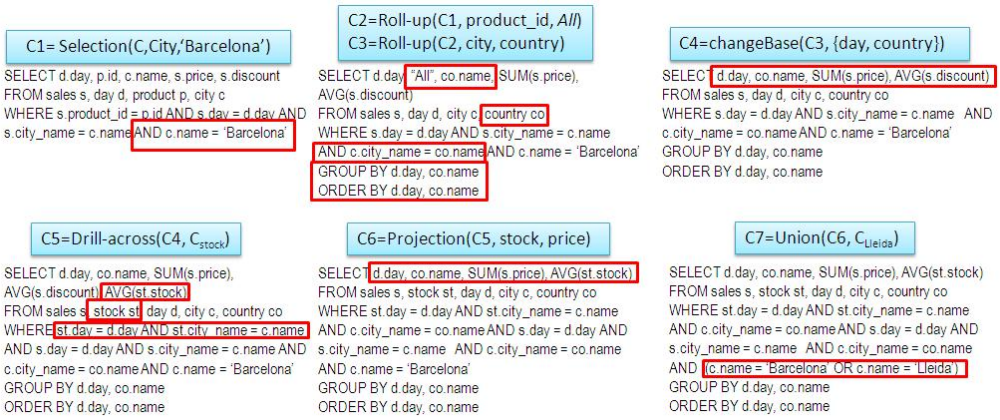


Figure 3.14: Exemplification of an OLAP navigation path translation into SQL queries

corresponding links to the WHERE clause. Moreover, identifiers in the SELECT, ORDER BY and GROUP BY clauses must be replaced appropriately. Following with the same example shown in Figure 3.14, we can change from $\{\texttt{day} \times \texttt{country} \times \texttt{All}\}$ to $\{\texttt{day} \times \texttt{country}\}$. Note that both bases are conceptually related by means of a one-to-one relationship. Specifically, this case typically applies when dropping a dimension (i.e., **rolling-up** to its `All` level and then **changing the base**). We **roll-up** to the `All` for representing the whole dimensions instances as a single one and therefore, producing the following base: $\{\texttt{day} \times \texttt{country} \times \texttt{1}\}$. Now, we can **changeBase** to $\{\texttt{day} \times \texttt{country}\}$ without introducing aggregation problems (since we **changeBase** through a one-to-one relationship).

- **Drill-across**: This operation changes the subject of analysis of the cube, by showing measures regarding a new fact. The n-dimensional space remains exactly the same, only the data placed in it change so that new measures can be analyzed. For example, if the cube contains data about sales, this operation can be used to analyze data regarding stock using the same dimensions.

  In SQL, we must add a new fact table to the FROM clause, its measures to the SELECT, and the corresponding links to the WHERE clause. In general, if we are not using any semantic relationship, a new fact table can always be added to the FROM clause if fact tables share the same base. In our example, suppose that we have a `stock` cube sharing the same dimensions as the `sales` cube. Then, we could **drill-across** to the `stock` cube and show both the `stock` and `sales` measures (see Figure 3.14).

- **Projection**: It selects a subset of measures from those available in the cube.

  In SQL it entails to remove measures from the SELECT clause. Following our example, we can remove the `discount` measure by projecting the `stock` and `price` measures.

- **Set operations**: These operations allow users to operate two cubes defined over the same n-dimensional space. Usually, Union, Difference and Intersection are considered.

  In this document, we will focus on Union. Thus, in SQL, we unite the FROM and WHERE clauses of both SQL queries and finally, we *or* the **selection** conditions in the WHERE clauses. Importantly, note that we can only **union** queries over the same fact table. Intuitively, it means that, in the multidimensional model, the **union** is used to undo **selections**. We can unite our example query to one identical but querying for data concerning `Lleida` instead of `Barcelona`.

3.4 Final Discussion

All in all, the design steps to undertake when designing a OLTP or a OLAP system show some inherent differences, since they must consider different premises. Fig. 3.15 sums up all the discussion introduced in this chapter:

- A transactional conceptual schema has no further restrictions than those of the application domain. For example, it can be modeled using any UML feature. However, multidimensional schemas are a simplified version, denoted by the star-shaped schemas.
- OLTP systems are traditionally implemented using the relational technology, which has been proven to suit their necessities. However, a multidimensional schema can be either implemented via ROLAP or MOLAP approaches.

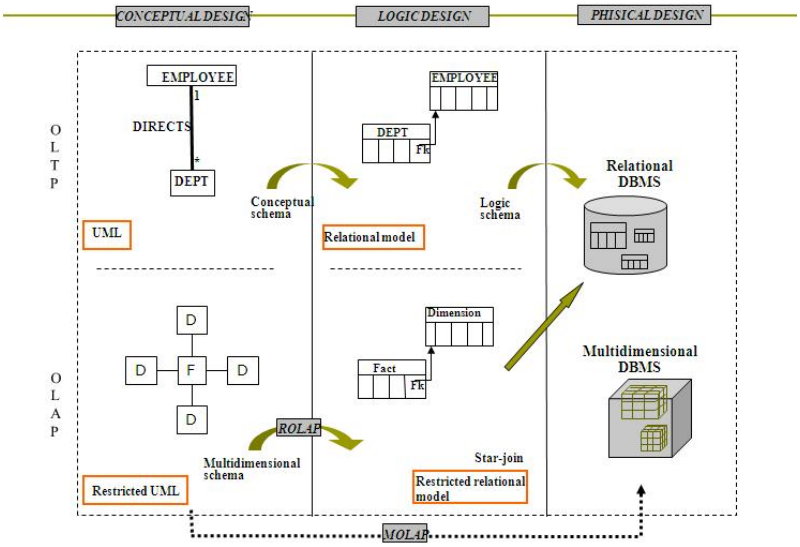


Figure 3.15: Sketch summarizing the main design differences between a transactional and a decisional system

Chapter 4

A (Brief) Introduction to Data Warehousing 2.0

Data warehousing has been around for about 2 decades now and has become an essential part of the information technology infrastructure[1]. Data warehousing originally grew in response to the corporate need for information. Thus, a data warehouse is a construct that supplies integrated, granular, and historical data to the corporation. This simple definition has rendered extremely hard to match. Nowadays, after all the experiences gathered in data warehousing, authors start to distinguish between first-generation and next-generation data warehouses.

Basically, there are two main concerns behind this new paradigm: the inclusion of ALL the relevant data (from inside the organization and even from external sources such as the Web) and the REAL use of meta-data. However, the second one is somehow a direct cause of the first one. After the success of data warehousing, the organizations widened their look towards data and wanted to include in their decision making processes alternative data such as unstructured data (of any kind, from plain text and e-mails to voice over IP), which makes even more difficult the data consolidation process. At this point is when meta-data emerges as a keystone to keep and maintain additional semantics to data. Specifically, consider the following shaping factors:

- In first-generation data warehouses, there was an emphasis on getting the data warehouse built and on adding business value. In the days of first-generation data warehouses, deriving value meant taking predominantly numeric-based, transaction data and integrating that data. Today, deriving maximum value from corporate data means taking ALL corporate data and deriving value from it. This means including textual, unstructured data as well as numeric, transaction data.

- In first-generation data warehouses, there was not a great deal of concern given to the medium on which data was stored or the volume of data. But time has shown that the medium on which data is stored and the volume of data are, indeed, very large issues. In

[1] This chapter is mainly based on the book "DW 2.0. The Architecture for the Next Generation of Data Warehousing", published by Morgan Kaufmann and authored by W.H. Inmon, Derek Strauss and Genia Neushloss, 2008 [ISN08].

2008, the first petabyte data warehouses have been announced by Yahoo and Facebook. Such amount of data is even beyond the limits of current relational DBMS, and new data storage mechanisms have been proposed[2].

- In first-generation data warehouses, it was recognized that integrating data was an issue. In today's world it is recognized that integrating old data is an even larger issue than what it was once thought to be.
- In first-generation data warehouses, cost was almost a non-issue. In today's world, the cost of data warehousing is a primary concern.
- In first-generation data warehousing, metadata was neglected. In today's world metadata and master data management are large burning issues. Concepts such as data provenance, ETL and data legacy are built on top of metadata.
- In the early days of first-generation data warehouses, data warehouses were thought of as a novelty. In today's world, data warehouses are thought to be the foundation on which the competitive use of information is based. Data warehouses have become essential.
- In the early days of data warehousing, the emphasis was on merely constructing the data warehouse. In today's world, it is recognized that the data warehouse needs to be malleable over time so that it can keep up with changing business requirements (which, indeed, change frequently).

All these issues represent challenges by themselves and still the community and major vendors are working on them. However, it is not part of this course objectives to gain insight, but the interested reader is addressed to [ISN08].

[2]Namely, the NoSQL -Not Only SQL- wave has focused on the storage problem for huge data stores.

Bibliography

[CCS93] E. F Codd, S.B. Codd, and C.T. Salley. Providing OLAP (On Line Analytical Processing) to Users-Analysts: an IT Mandate. *E. F. Codd and Associates*, 1993.

[Che76] P. P. S. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[GR09] M. Golfarelli and S. Rizzi. *Data Warehouse Design. Modern Principles and Methodologies.* McGraw-Hill, 2009.

[Inm92] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 1992.

[ISN08] W.H. Inmon, Derek Strauss, and Genia Neushloss. *DW 2.0. The Architecture for the Next Generation of Data Warehousing*. Morgan Kaufmann, 2008.

[Kim96] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., 1996.

[KRTR98] R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., 1998.

[Pen05] Nigel Pendse. Market Segment Analysis. *OLAP Report*, Last updated on February 11, 2005. http://www.olapreport.com/Segments.htm (Last access: July 2009).

[Pen08] Nigel Pendse. What is OLAP? *OLAP Report*, Last updated on March 3, 2008. http://www.olapreport.com/fasmi.htm (Last access: July 2009).