Bases de Dades Avançades - Data Warehousing and OLAP

Alberto Abelló Petar Jovanovic

Database Technologies and Information Management (DTIM) group
Universitat Politècnica de Catalunya (BarcelonaTech), Barcelona
September 10, 2021

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 (starting in the 90s) and from the urgent need to use those 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 SQL query with aggregates) 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 and 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.

In the following, we present an introduction to such systems and some insights to the techniques, technologies 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 give answer to extract, store (conciliate) and analyze data (what is also know as the BI cycle). Fig. 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 metadata repository, the ETL tools and the most relevant exploitation tool related to these systems: OLAP tools.

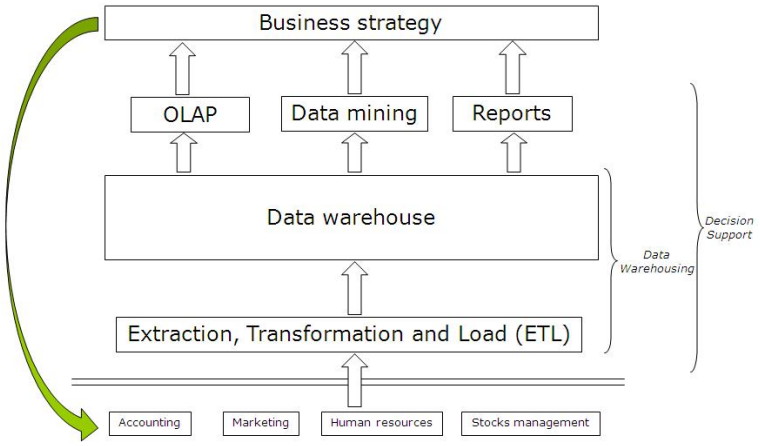


Figure 1.1: Business Intelligence cycle

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, 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

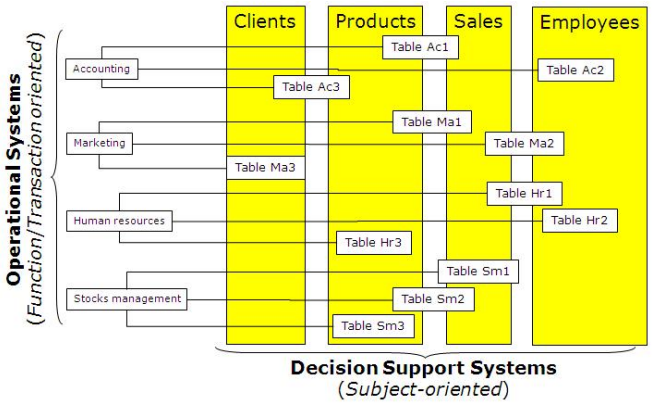


Figure 1.2: Subject oriented Vs. transaction oriented

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. These data are 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 broken into small pieces of data before loading the data warehouse, which is interested 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 spread on different 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 (usually called *Valid Time*, representing when data are valid in the real world) and for example, historical data (which is of no interest for OLTP systems) is essential; and finally,
- *Non-volatile* means that data are stable in the data warehouse. Thus, more data are added but data are never removed (usually a *transaction time* attribute is added to represent when things are recorded). This enables management to gain a consistent picture of the business.

Despite this definition was introduced almost 30 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 discussed, 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. Usually, a decision is not made on the isolated view of current data, but putting this in the context of historical evolution, and data are consequently never updated or deleted, but only added. This addition of data is done in bulk loads performed by batch processes. Users never do it manually, so the data warehouse is considered "read-only" from this perspective. Thus, data warehouses tend to be massive (incrementally integrate historical information from different sources, which is never deleted), 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 (e.g., revenue obtained per year, or average number of pieces provided per month, etc.) since they provide new interesting perspectives. 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 emphasize the following three categorizations, 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 are 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 gender 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. Some data are clearly useless for decision making (e.g., customer names), some are clearly useful (e.g., postal codes), but others are arguable or not that clear (e.g., telephone numbers per se can be useless to make decisions, but result useful as identifiers of customers). 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 them.
- **Atomic, derived and aggregated data:** These concepts are related to the *data granularity* at which data are 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 where 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 (e.g., the customer profile defined by a clustering algorithm). 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 can also be computed 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 (the trade-off between update and query frequencies needs to be considered).

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 referred as reconciliation. These are also the most complex and technically challenging among all the data 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. Fig. 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 are 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

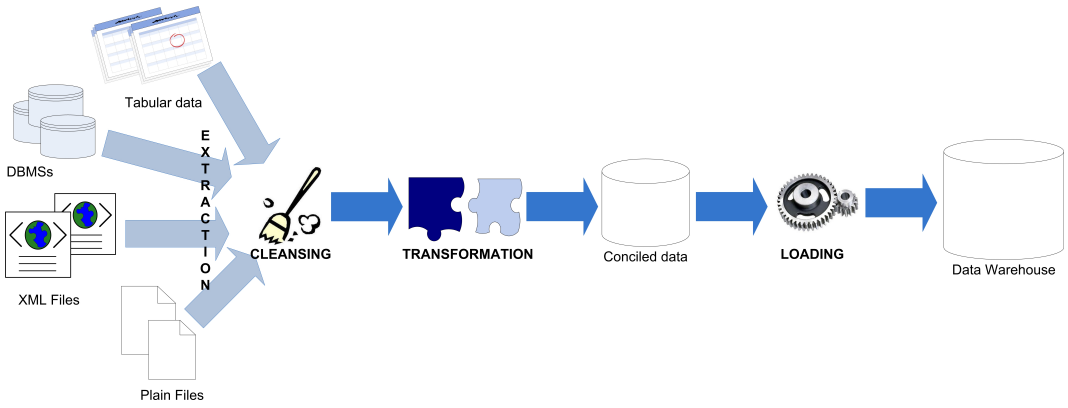


Figure 1.4: Extraction, transformation and loading

when the data are 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 customer's 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

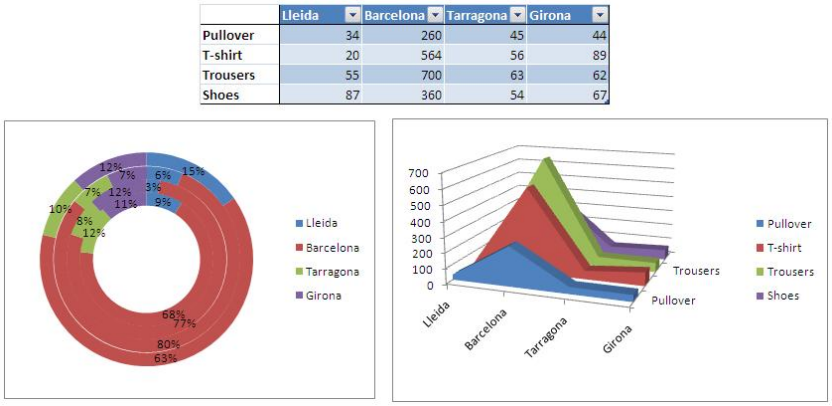
Loading into a data warehouse is the last step to take. Loading can be carried out in two ways:

- Refresh: Data warehouse data are completely rewritten. This means that older data are 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). Fig. 1.5 shows a few examples of layouts for a query.



	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

- 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 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.

Multimedia Materials

Data Warehouse definition (en)

Data Warehouse definition (ca)

Chapter 2

Data Warehousing Architectures

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. Fig. 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.

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. Thus, a relevant consideration about this approach is the nature of the sources, because depending on it, the architecture becomes impractical or even impossible (this also impacts to some extent the other architectures).

- Non computerized sources (i.e., manual extraction) is more and more rare, but if required, makes the single-layer architecture impossible, because implies to repeat the manual effort once and again, since extracted information is not kept for further use.
- World Wide Web, which is not under our control and frequently changes, also brings problems to the single-layer architecture, because information that disappears from the Internet will also be lost for use, if we didn't make any copy.

[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].

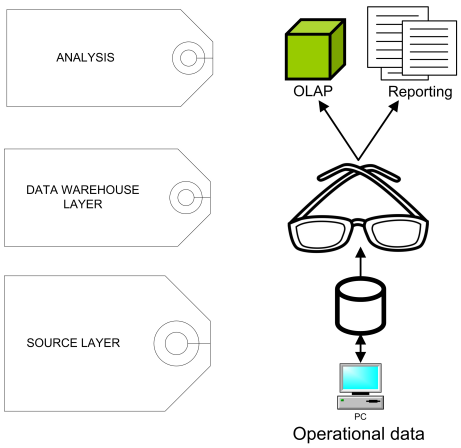


Figure 2.1: Single-layer architecture for a data warehouse system

- Partners' systems can also be available under some kind of agreement, which can include what happens in case of changes in the availability or the format of data.
- Own operational systems always result the easiest data source, because they are well known and we have control over their changes.

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 Fig. 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 these data may also be heterogeneous (legacy[2] systems, 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

[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 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.

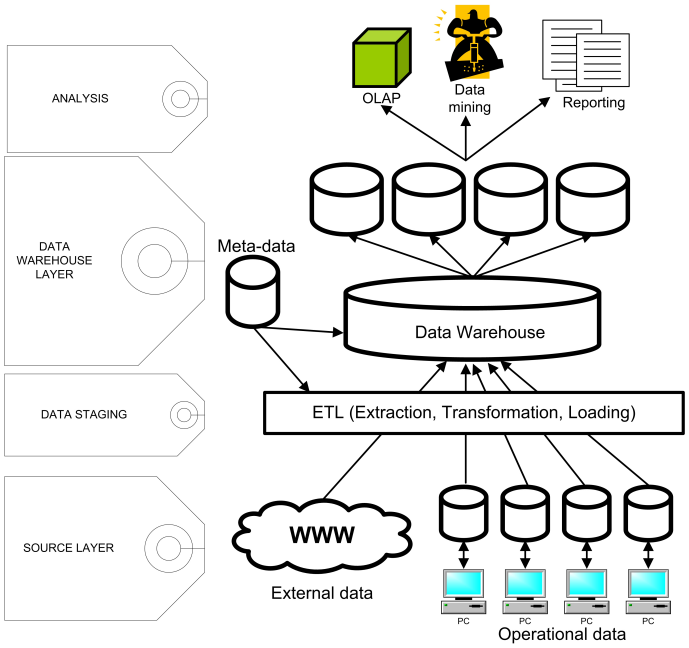


Figure 2.2: Two-layer architecture for a data warehouse system

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/smaller data warehouses), which partially replicate data warehouse contents and are designed for specific enterprise departments. Metadata repositories (see section 2.4) store information on sources, access methods available, data staging, users, data mart schemas, and so on.

- **Analysis:** In this layer, integrated data are 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 Fig. 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 stored together. Data marts can be viewed as small, local data warehouses replicating (and pre-computing 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 (i.e., only partial history, not all sources and not necessarily the most detailed data) 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 Fig. 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.
- 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. Fig. 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 (a.k.a. Operational Data Store) 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 Metadata

In addition to those kinds of data already discussed in section 1.1.1.1, metadata represents a key aspect for the discussed architectures. The prefix "meta-" means "more abstract" (e.g., metarule, metaheuristic, metalanguage, metaknowledge, metamodel, etc.). Prior to come up with a formal definition for metadata, 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

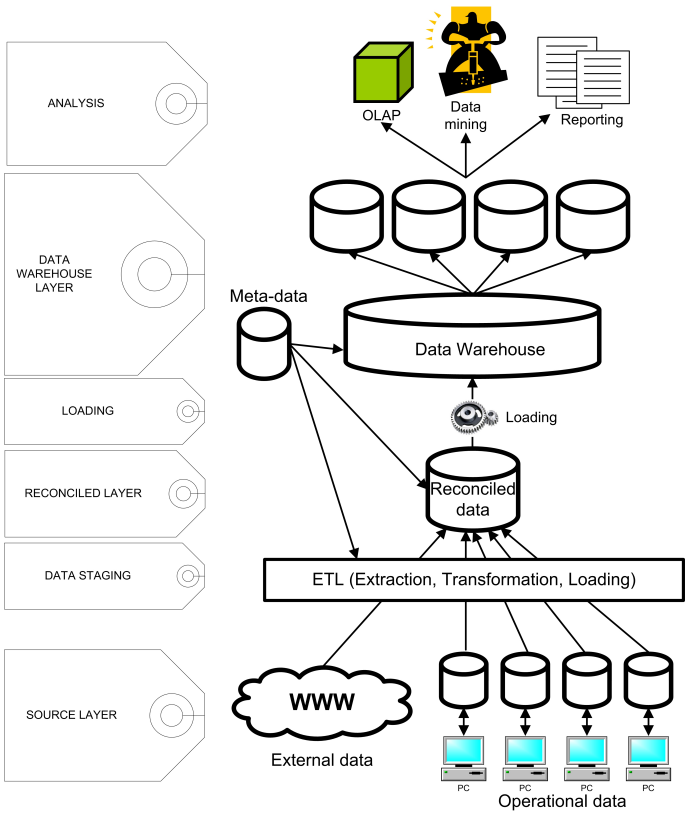


Figure 2.3: Three-layer architecture for a data warehouse system

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 datum extracted from a database: 1100. So what information represents this datum? to answer this, it will help to know that this datum is in *binary* format, its type is an *integer*, represents the *age* attribute (in *months*) from a table named *dogs* and this datum is updated every *year* (and last update was in *December*). All these data about the 1100 datum needs to be stored in order to process it as information. This is what we know as metadata, 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 metadata repository is the Relational databases catalog.

In the scope of data warehousing, metadata play an essential role because it specifies source, values, usage, and features of data warehouse data, which is fundamental to come up and justify innovative analysis, but also because they define how data can be changed and processed at every architecture layer (hence, fostering automation). Fig. 2.2 and 2.3 show that the metadata repository is closely connected to the data warehouse. Applications use it intensively to carry out data-staging and analysis tasks. One can classify metadata into two partially overlapping categories. This classification is based on the ways system administrators and end users exploit metadata. System administrators are interested in internal (technical) metadata because it defines data sources, transformation processes, population policies, logical and physical schemas, constraints, and user profiles and permissions. External (business) metadata are relevant to end users. For example, it is about definitions, actualization information, quality standards, units of measure, relevant aggregations, derivation rules and algorithms, etc.

Metadata are stored in a metadata repository which all the other architecture components can access. A tool for metadata management should:

- allow administrators to perform system administration operations, and in particular manage security;
- allow end users to navigate and query metadata;
- use a GUI;
- allow end users to extend metadata;
- allow metadata 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. In the context of data warehousing, as depicted in Fig. 3.1, OLAP tools are placed in between the data warehouse and the front-end presentation tools.

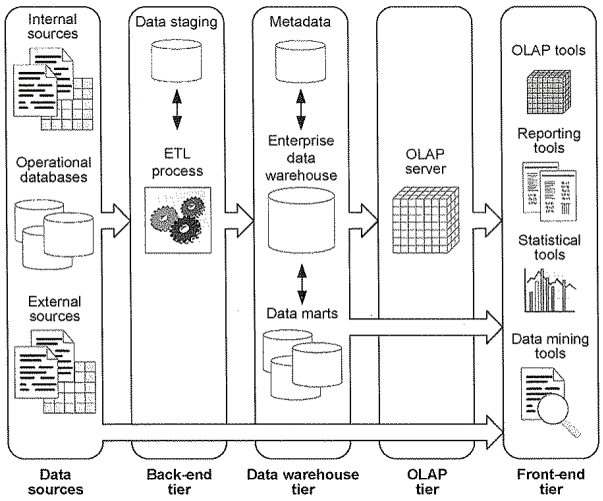


Figure 3.1: OLAP Reference Architecture [VZ14]

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*.

We can say that the multidimensionality plays for OLAP the same role as the Relational model for Relational databases. Unfortunately, unlike Relational databases, there is not yet consensus about a standard multidimensional model (among other 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: (1) what reality multidimensionality models, (2) how this reality can be represented at the conceptual level, and (3) which are the logical (also physical) alternative representations level.

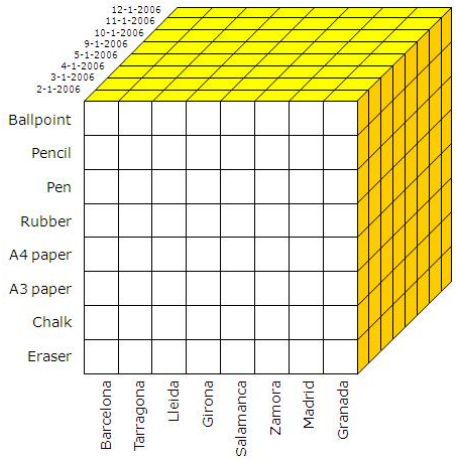


Figure 3.2: Multidimensional view of data

3.1 The Real World: Events and Multidimensionality

The multidimensional point of view is based on the cube metaphor (which is not actually that "real"). Specifically, the multidimensional 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 determined by 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 or features (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`, `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

[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 (in general, we can have many more).

[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.

basic operations.

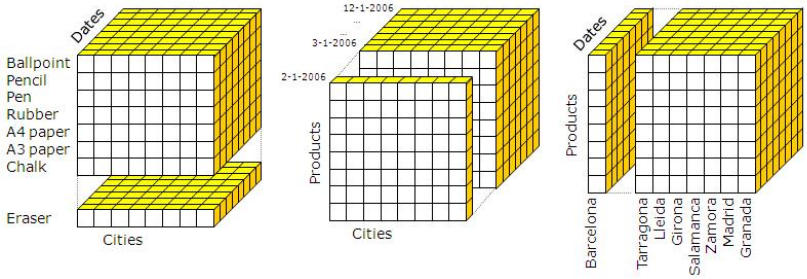


Figure 3.3: Different examples of slice

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.

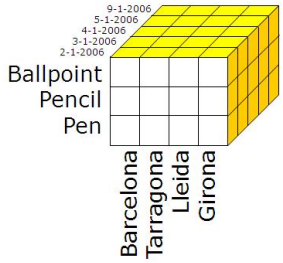
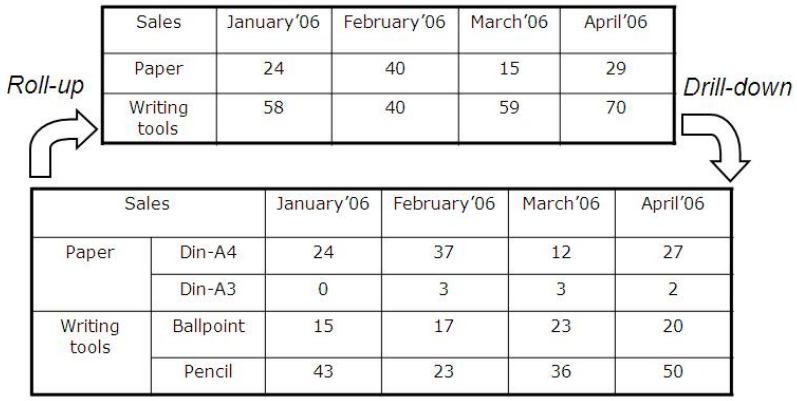


Figure 3.4: Example of dice

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.



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

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, 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 design, and several methods 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 designing technique for data warehouses. Multidimensional modeling optimizes the system query performance in contrast to conventional *Entity-Relationship* (ER) models [Che76] (widely used for modeling Relational databases) that are constituted to remove redundancy in the data and optimize OLTP performance.

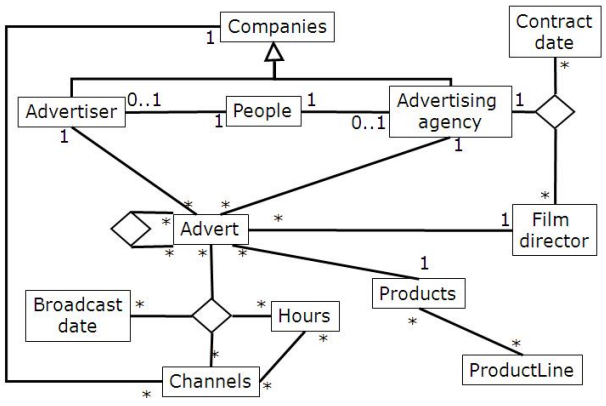


Figure 3.6: A transactional model

Consider Fig. 3.6, which depicts a conceptual transactional UML model (similar to ER). 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.

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.

Finally, note that we consider {`product` × `day` × `city`} in Fig. 3.2 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**). Thus, it means that one value of each one of these **levels** determines one cell (i.e., a `sale` with its `price`, `discount`, etc.). 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-* (one-to-many) multiplicities. In the multidimensional model, *-* (many-to-many) relationships are meaningless as they do not preserve the model constraints.

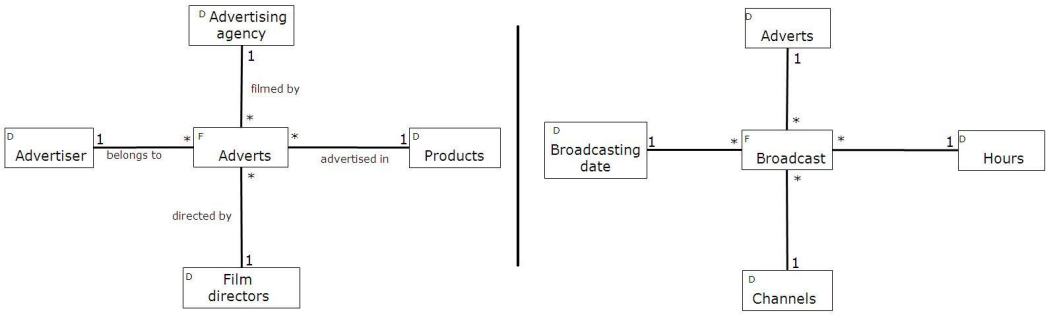


Figure 3.7: Two examples of star-schemas

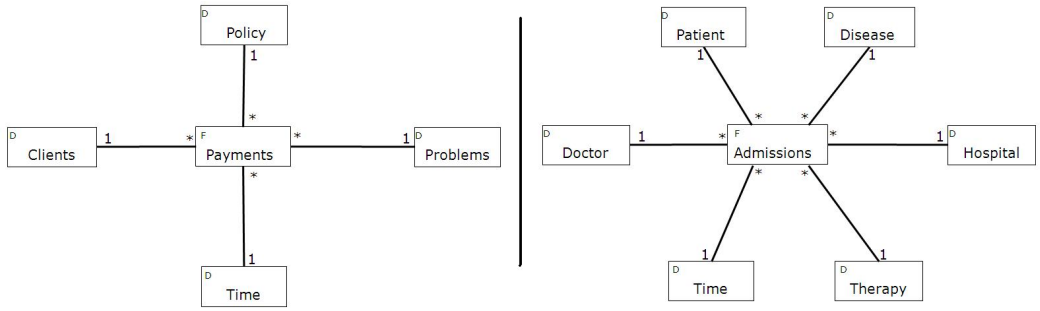


Figure 3.8: Two (more) examples of star-schemas

Recall now the transactional schema used as example in Fig. 3.6. 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 numeric measures or metrics)
- There is a set of dimensions (with descriptive discrete 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 all the dimension. Clearly, this is against the second and third Relational normal forms, but denormalization of data is common 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, denormalize such schemas is sound and acceptable (even encouraged).

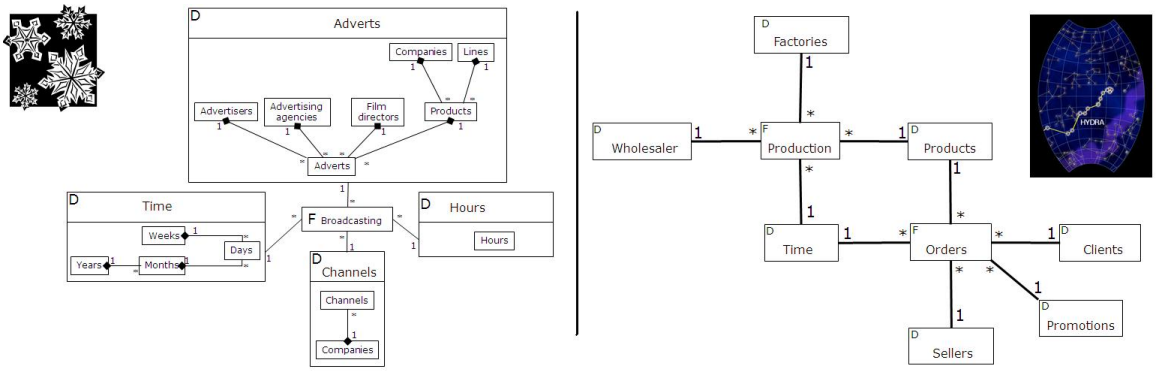


Figure 3.9: Examples of snowflake and constellation schemas

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 explicited separately. Finally, when a schema contains more than one fact, which share dimensions is called a galaxy or constellation (see the right-hand side schema on Fig. 3.9), and facilitates drilling-across (see algebraic operations bellow).

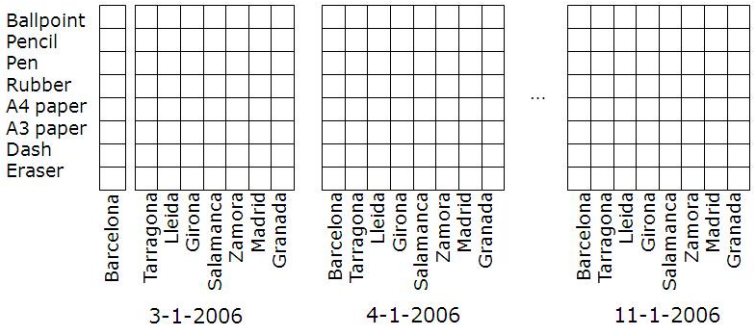


Figure 3.10: Example of a MOLAP tool storing data as arrays

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), 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 flatten/serialized 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 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.

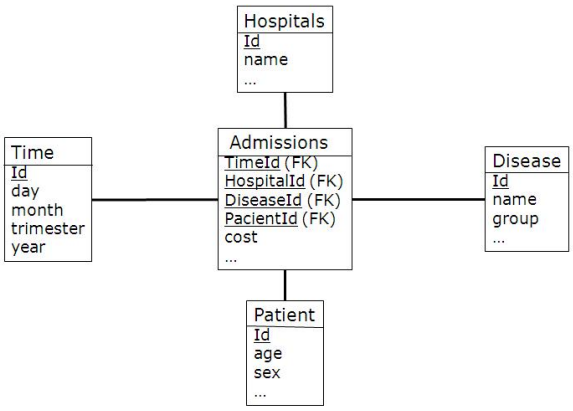


Figure 3.11: Example of a star-join schema corresponding to the right-hand star-schema in Fig. 3.8 (ignoring dimensions "Doctor" and "Therapy" for the sake of simplicity)

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[3] 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. In general, normalizing the dimensions requires a very good reason, since it produces a very little gain in the size of dimension and a big loss in the performance of queries (since it generates more joins). Normalization was conceived to minimize redundancies that hinder performance in the presence of updates. However, the DW is considered read-only and dimensional data is specially

[3] https://youtu.be/SSio_jhAmzg

static, since the dynamicity of business is reflected in the fact tables.

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 ]
```

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

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 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.

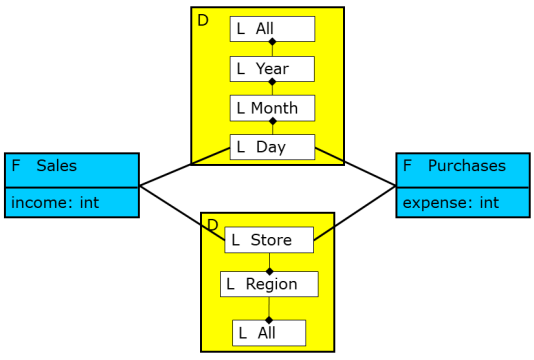


Figure 3.13: Example of conceptual snowflake schema

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. 3.13. 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. 3.14 to grasp the idea behind each operator):

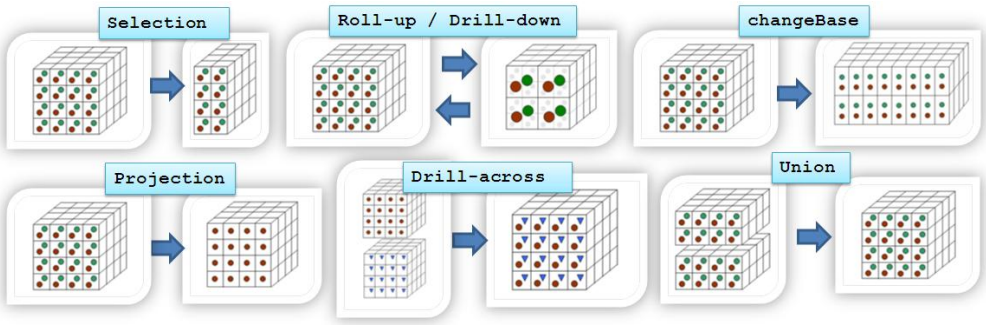


Figure 3.14: Conceptual representation of the multidimensional operators

- **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. 3.14).

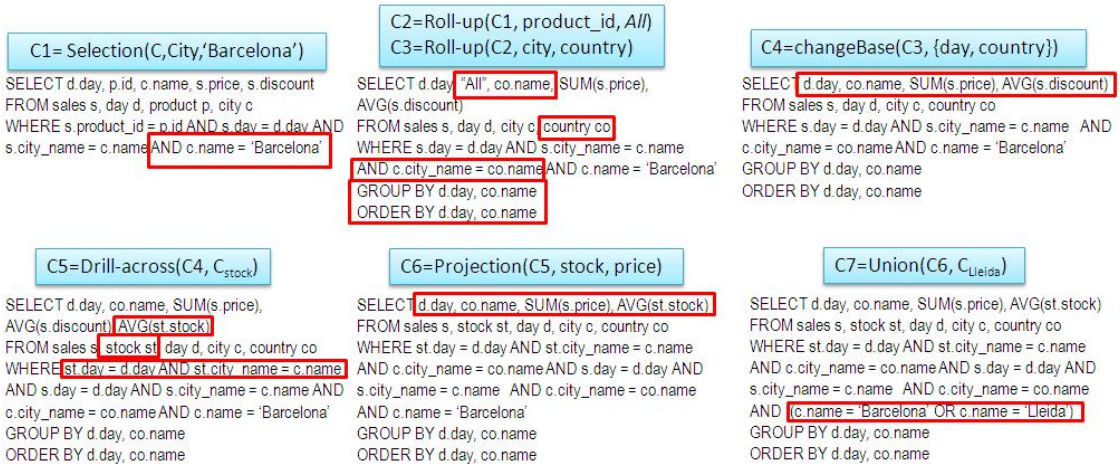


Figure 3.15: Exemplification of an OLAP navigation path translation into SQL queries

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. 3.15).

- **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 Fig. 3.15), 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 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 Fig. 3.15, we can change from {`day` × `country` × `All`} to {`day` × `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: {`day` × `country` × 1}. Now, we can **changeBase** to {`day` × `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 Fig. 3.15).

- **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`.

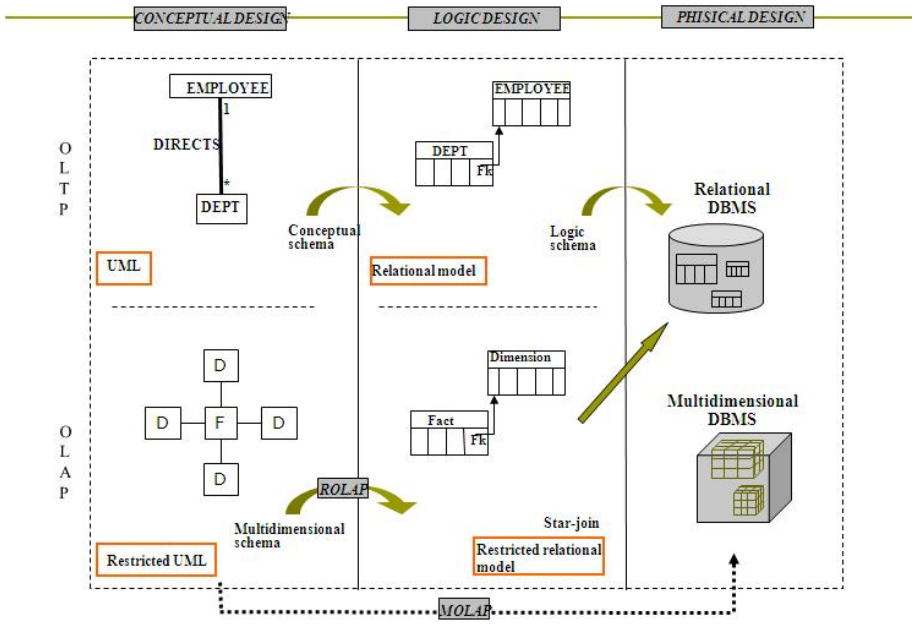


Figure 3.16: Sketch summarizing the main design differences between a transactional and a decisional system

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. 3.16 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.

Multimedia Materials

Multidimensionality (en)

Multidimensionality (ca)

Benefits of the Star Schema (en)

Benefits of the Star Schema (ca)

SQL Transformations with Multidimensional Algebra (en)

SQL Transformations with Multidimensional Algebra (ca)

Advanced Multidimensional Design (en)

Advanced Multidimensional Design (ca)

Summarizability Conditions (en)

Summarizability Conditions (ca)

Chapter 4

Extraction, Transformation and Load

There are many situations in which we have to move data from one database to another, which implies Extracting the data, then potentially performing some Transformation to finally Load them in the destination. Some situations where we can find this are:

- Restructure data to be used by other tools
- Transactional data safekeeping
- Use multiple sources together
- Improve data quality by removing mistakes and complete missing data
- Provide measures of confidence in data (e.g., filtering out erroneous records)

This kind of data intensive flows is so common that has created the need of specialised tools like Microsoft Data Transformation Services[1], Informatica Power Center[2], Pentaho Data Integration[3], and many others.

4.1 Preliminary legal and ethical considerations

Before creating any data migration flow, we should ask ourselves if we are really allowed to do it, and even if we are, whether this is ethically acceptable. That the technology allows to do something, does not mean that it should be done. In the end, it is the people (i.e., data engineers in this case) not the tools alone that perform the tasks. Thus, this job comes with a big responsibility[4].

This is specially true when we are talking about personal data. Due to that, there is a specific EU directive to regulate what can and cannot be done, as well as different agencies to safeguard the rights of protection of personal data at different administrative levels, like the Catalan one[5].

Thus, before doing anything, you should ask yourself:

1. Who is the owner of the data?
2. Are we allowed to process them?
3. Are we allowed to use them for this purpose?
4. Will we keep confidentiality of data?
5. Did we implement the required anonymization and inference control mechanisms?
6. How will the results of my work be used?

[1] https://en.wikipedia.org/wiki/Data_Transformation_Services
[2] https://www.informatica.com/gb
[3] https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho-platform.html
[4] http://wp.sigmod.org/?p=1900
[5] https://apdcat.gencat.cat/en/inici/index.html

4.2 Definition

As we said, we can find ETL flows in many situations and systems, but we should focus now on DW. If we pay attention to the different architectures in Chapter 2, it is easy to see that the ETL layer is explicit in case of two- and three-layers. However, it is also implicit in some other places including the single-layer architecture. In Fig. 2.2, we see it between the sources and the DW, but there is also data movement between the DW and the DMs. In Fig. 2.3, it is again explicit between the sources and the Reconciled data, but there is also data movement from this to the DW and then to the DM. Finally, it is not that obvious, but some data movement is implicit in the virtual DW layer of Fig. 2.1, too.

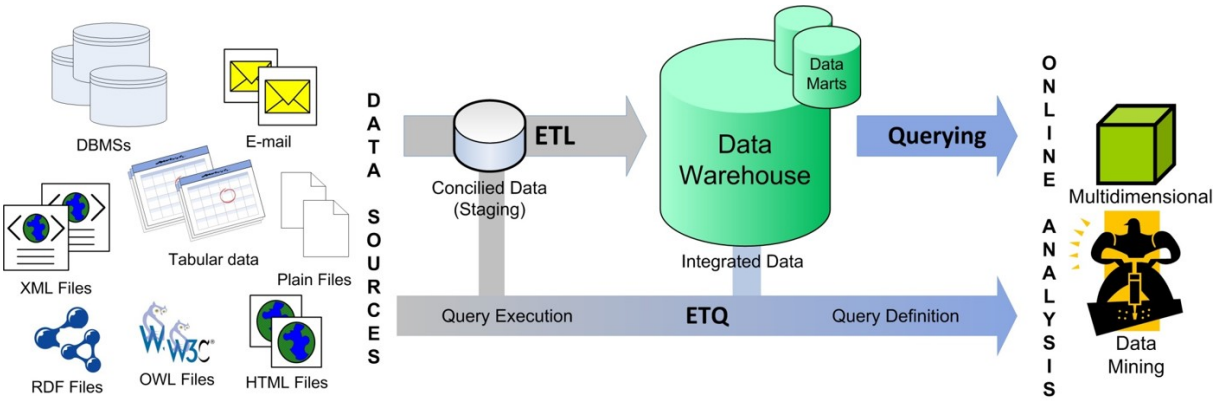


Figure 4.1: ETL flows

These different possibilities are generalized in Fig. 4.1, where we distinguish between ETL (corresponding to two- and three-layers) and ETQ where the data are not loaded anywhere, but directly Queried by the final user without any materialization (corresponding somehow to single-layer architecture). Even more generally, we can also find some authors and software providers proposing and advocating for an ELT variant, where extracted data are firstly loaded into a powerful DBMS to then transform them inside using ad-hoc SQL statements.

4.2.1 Extraction

Some data tasks require multiple and heterogeneous sources, that can be in different supports (e.g., hard disk, Cloud), different formats (e.g., Relational, JSON, XML, CSV), from different origins (e.g., transactional systems, social networks), and with different gathering mechanisms (e.g., digital, manual). So, it will be the first task of the ETL to solve all these differences as data moves through the pipe.

Besides purely formatting, since our DW must be historic, it is specially important to pay attention too to the differences in the temporal characteristics of sources:

Transient: The source is simply non-temporal, so if we do not poll it frequently, we'll miss the changes (any change between two extractions will be lost).

Semi-periodic: The source keeps a limited number of historical values (typically the current and the previous one), which gives us some margin to space extractions, so reducing the disturbance to the source.

Temporal: The source keeps an unlimited number of historical values (but may still be purged periodically), which gives complete freedom to decide the frequency of extraction.

Fig. 4.2 sketches the different mechanisms we have to extract data from a DBMS:

a) Application-assisted: Implies that we modify the application and intercept any change of the data before it goes to the DBMS.

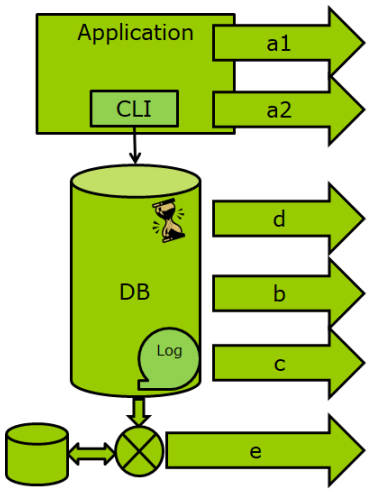


Figure 4.2: Data extraction mechanisms

1) This can obviously be done by modifying the code (e.g., PhP), and injecting the required instruction to push any change in the data to the ETL pipe of the DW. The obvious problem of this approach is that it will be expensive to maintain (any application accessing the DBMS must be modified) and hardly sustainable.
2) A more elegant alternative is to modify the Call Level Interface (i.e., JDBC or ODBC driver) so that every time a modification call is executed, this is replicated in the ETL pipe of the DW. Obviously, this is more sustainable than the previous, but can be harder to implement and some times expensive to execute (interactions to the DBMS are more expensive and this would still impact the application performance).

b) Trigger-based: If the DBMS provides triggers, we can use them to intercept any change in the database and propagate it to the DW. The limitation is obviously that not all DBMS provides triggers, and they may also result too expensive.

c) Log-based: Any DBMS provides durability mechanisms in the form of either logs or incremental backups. This can easily be used to efficiently extract the data. The main problem is that depending on the software provider, it may be using proprietary mechanisms and formats, which are hard to deal with.

d) Timestamp-based: Some databases are temporal (or even bi-temporal) and already attach timestamps to any stored value (or can be easily modified to do so). If allowed and does not generate much overhead, this would be the best option, because offers the maximum flexibility to the extraction. However, it requires having the control over the source and the power to impose the modification.

e) File comparison: A simple option is to perform periodic bulk extractions of the database (a.k.a. backups). The first time, we would push all of it to the DW and keep a copy in some staging area. From there on, every new extraction would compare against the previous one in the staging area to detect the changes and only push those to the DW.

Besides these technical issues, in case of existing alternatives, we should also consider the interest and the quality of the sources in the choice. Firstly, a source should always be relevant to the decision making (i.e., it should provide some data to either a dimension or fact table). In general, having more data does not mean making better decisions (it can be simply harder or even confusing if data are irrelevant or wrong)[6]. On the other hand, it should not be redundant, because it would simply be a waste of resources. If there are some alternative sources for the same data, we should chose the one

[6] https://dangerousminds.net/comments/spurious_correlations_between_nicolas_cage_movies_and_swimming_pool

of highest quality in terms of completeness, accuracy, consistency and timeliness. A good reason to extract the same data from more than one source would be that we can obtain higher quality from them together.

4.2.2 Transformation

The second part of the ETL consists of standardizing data representation and eliminating errors in data. It can be seen in terms of the following activities, not necessarily in this order (actually, due to its difficulty, its development is an iterative process estimated to take 80% of the knowledge discovery process):

Selection has the objective of having the same analytical power (either describing or predicting our indicators), but with much less data. Obviously, we can decide not to use some source, but even on taking one source, we can use it partially by reducing:

a) Length (i.e., remove tuples) by sampling (might be interesting not to remove outliers), aggregating (using predefined hierarchies) or finding a representative for sets of rows (i.e., cluster).

b) Width (i.e., remove attributes) by eliminating those that are correlated, performing an analysis of significance, or studying the information gain (w.r.t. a classification).

Integration has the purpose of crossing independent data sources, which potentially have semantic and syntactic heterogeneities. The former requires reshaping the schema of data, while the latter is solved tranforming their character set and format.

Cleaning includes generating data profiles, splitting the data in some columns and standardize values (e.g., people or street names). However, the main purpose is to improve data quality in terms of:

- Completeness by imputing a default value (or even manually assigning different ones if not many are missing), which can be either a constant, some coming from a complementary source/lookup table, the average/median/mode of all the existing ones, the average/median/mode of the corresponding class, or the one maximizing the information gain.
- Accuracy by detecting outliers performing some variance analysis, evaluating the distance to a regression function, or identifying instances far from any cluster.
- Correctness, by checking constraints and business rules, or matching dictionaries/lookup tables.

Feature engineering derives new characteristics of the data that are expected to have more predictive power.

Preparation sets the data ready for a given algorithm or tool. In some cases, this requires to transform categorical attributes into numerical ones (just creating some encoding), but others it is the other way round and numerical attributes need to be discretized (e.g., by intervals of the same size, intervals of the same probability, clustering, or analysis of entropy). Some algorithms are also affected by differences in scales, so numerical attributes need to be normalized (e.g., dividing by the maximum $\frac{x}{max}$, dividing by the domain size $\frac{|x-min|}{|max-min|}$, dividing by the standard deviation $\frac{x-\mu}{\sigma}$, or simply dividing by some power of ten $\frac{x}{10^j}$). Some other algorithms require the transformation of data into metadata by pivoting and converting rows into columns.

4.2.3 Load

The last phase is loading the data in the target of the flow[7]. A typical technique to facilitate this in a DW environment is using the concept of *update window*. This is a period of time (typically at night or non-working days) during which analysist cannot access the DW (so they cannot interfere in the loading). Separating user queries from ETL insertions, we firstly save the overhead of concurrency

[7] As previously said, we could also send it directly to the user ready to be consumed in an ETQ

control mechanisms, which can be a significant gain by itself, but also allows to disable materialized view updates and indexes (since users are not using them). Once the load is over, we can rebuild all indexes and update all the materialized views in batch, which is much more efficient than doing it incrementally and intertwined with insertions.

4.3 ETL Process Design

Many organizations are opting for an ad-hoc, hand-coded ETL processes, mostly due to no established standards and tools for conceptual modeling of the ETLs. However, such settings soon become unmanageable with hundreds of scripts with usually poor documentation and little or no reusability of the code.

Conceptual ETL process design: Many research attempts propose conceptual modeling for ETL to provide better understandability and sharing of ETL processes as well as their maintenance and reusability. Some are proposing either novel, ad-hoc notation [SV03], or based on well-known modeling languages, like UML [TL03] or BPMN [VZ14]. Besides better documentation of the ETL processes in an organization, such approaches using a strictly defined conceptual model for designing an ETL process allow for automation of code generation. One of the main tasks of the ETL process conceptual design is the identification of schema mappings between data source schemata and target DW schema [SV18]. Furthermore, the conceptual design also specify a primary data flow that describes the route of data from the sources toward the data warehouse, as they pass through the transformations of the workflow.

Logical ETL process design: At the logical level, we must specify the detailed workflow for executing our ETL process. On the one hand, we specify a **data flow** in terms of what transformation each ETL operation does over the data. On the other hand, we specify a **control flow**, which takes care of scheduling, executing, and monitoring of the ETL process. In addition, we can as well specify recovery plans in case of failure occurrences. Most ETL tools provide the designer with user friendly (drag&drop) functionalities to construct both the data and the control flow for an ETL process (e.g., Pentaho Data Integration - Kettle).

Physical ETL process design: Finally, we need to provide an executable version of the previously designed ETL process. While ETL tools conveniently integrate logical ETL design and its execution, hand-coded solutions require the implementation of procedures and script that fulfill the data flow and control flow tasks.

- **Hand-coded ETL.** As previously discussed, to implement an ETL, we do not really need an ETL tool. It can be done programmatically with a skillful programming team, as well. The advantage of this is that we are not limited by the features of the tool, we can reuse legacy routines as well as know-how already available. A good example of this approach lately is MapReduce, which facilitates processing schemaless raw data in read-once datasets, cooking them before being loaded in a DBMS for further processing. However, in general, such programmatical approach only works for relatively small projects. If the project has a certain volume and requires some sophisticated processing, manual encoding of data transformations is not a good idea, specially from the point of view of the maintenance and sustainability in the long term.

- **ETL tools.** Quoting Pentaho, "The goal of a valuable tool is not to make trivial problems mundane, but to make impossible problems possible". An ETL tool, like Pentaho Data Integration, cloverETL, JasperETL, or Talend, firstly offers a GUI that facilitates the encoding of the flows. Moreover, they provide some metadata management functionalities and allow to easily handle complex data type conversions as well as complex dependencies, exceptions, data lineage and dependency analysis. Also, they facilitate common cumbersome tasks like scheduling processes, failure recovery and restart, and quality handling.

4.4 Architectural setting

Data flow vs. control flow. As previously explained a proper ETL processes design assumes two levels of design, a data flow and a control flow.

- **Data flow** is in charge of performing operations over data themselves in order to prepare them for loading into a DW (or directly for exploiting them by end users). That is, data extraction (reading from the data sources), various data transformation tasks (data cleaning, integration, format conversions, etc.) and finally loading of the data to previously created target data stores of the DW. Data flows are typically executed as a pipeline of operations (rather than a set of strictly sequential steps).
- **Control flow** on the other side is responsible of orchestrating the execution of one or more data flows. It does not work directly with data, but rather on managing the execution of data processing (i.e., scheduling, starting, checking for possible errors occurred during the execution, etc.). Unlike data flow, in control flow the order of execution is strictly defined by the sequence of activities, meaning that one activity does not start its execution until all its input activities have finished. This is especially important in the case of dependent data processing, where the results of one data flow are needed before starting the execution of another.

Staging area. Typically, as a complement to the ETL tool, we need to devise a staging area where to place temporal files. This facilitates *recoverability*, backup of the processes and *auditing*. The purpose of this area is only to support the ETL processing and can contain from plain files to more complex Relational tables, through XML or JSON.

4.5 ETL operations

As summarized in Fig. 4.3, we can find many different operations in ETL tools. Even if they take many different names depending on the tool, they offer simmilar functionalities. Just to hightlight some in Pentaho Data Integration (PDI) terminology:

Calculator allows to derive some new columns from existing ones.

Unique rows assumes the input is ordered and removes duplicates.

Sort rows orders the rows according to some criteria.

Group by assumes the input is sorted and generates groups of rows to be aggregated.

Filter rows obviously filters out those rows that are not interesting.

Join is a very expensive operation, which consequently has different implementations to chose. For example, *Merge Join* assumes inputs are sorted by the join key and simply goes through the inputs sequentially. Alternatively, *Stream lookup* goes sequentially through one of the inputs and keeps the other in memory to be used as a lookup.

Sorted Merge Append is actually a union eliminating duplicates.

Concat fields merges two columns into a single one.

Select values is a powerful operator that includes datatype conversion, field renaming and projecting out some columns.

Extraction operations allow to connect to different sources, which range from CSV files or Relational tables.

Loading operations allow to store the result of the flow in some repository. Available types use to coincide with those in the extraction (from CSV files to Relational tables).

We usually refer to the ETL flow as a pipe, in the sense that rows go through it one after another non-stop. Nevertheless, this is not actually true for all operations. Only some of them (known as *non-blocking*) really allow that behaviour (e.g., Field Value Alteration, Single Value Alteration, Sampling, Dataset Copy, Duplicate Row, Router, Union, Field Addition, Datatype Conversion, Field Renaming, Projection, Pivot, Extraction, Loading), while the others actually need to hold all the rows back (i.e., they *block* the flow) until the get the last one (e.g., Duplicate Removal, Sort, Aggregation, Join, Intersect, Difference, Unpivot).

Multimedia Materials

ETL Operations (en)

ETL Operations (ca)

Operation Level	Operation Type	Pentaho Data Integration	Talend Data Integration	SSIS	Oracle Warehouse Builder
Attribute	Attribute Value Alteration	Add constant Formula Number ranges Add sequence Calculator Add a checksum	tMap tConvertType tReplaceList	Character Map Derived Column Copy Column Data Conversion	Constant Operator Expression Operator Data Generator Transformation Mapping Sequence
Dataset	Duplicate Removal	Unique Rows Unique Rows (HashSet)	tUniqRow	Fuzzy Grouping	Deduplicator
	Sort	Sort Rows	tSortRow	Sort	Sorter
	Sampling	Reservoir Sampling Sample Rows	tSampleRow	Percentage Sampling Row Sampling	
	Aggregation	Group by Memory Group by	tAggregateRow tAggregateSortedRow	Aggregate	Aggregator
	Dataset Copy		tReplicate	Multicast	
Entry	Duplicate Row	Clone Row	tRowGenerator		
	Filter	Filter Rows Data Validator	tFilterRow tMap tSchemaComplianceCheck	Conditional Split	Filter
	Join	Merge Join Stream Lookup Database lookup Merge Rows Multiway Merge Join Fuzzy Match	tJoin tFuzzyMatch	Merge Join Fuzzy Lookup	Joiner Key Lookup Operator
	Router	Switch/Case	tMap	Conditional Split	Splitter
	Set Operation - Intersect	Merge Rows (diff)	tMap	Merge Join	Set Operation
	Set Operation - Difference	Merge Rows (diff)	tMap		Set Operation
	Set Operation - Union	Sorted MergeAppend streams	tUnite	Merge Union All	Set Operation
Schema	Attribute Addition	Set field value Set field value to a constant String operations Strings cut Replace in string Formula Split Fields Concat Fields Add value fields changing sequence Sample rows	tMap tExtractRegexFields tAddCRCRow	Derived Column Character Map Row Count Audit Transformation	Constant Operator Expression Operator Data Generator Mapping Input/Output parameter
	Datatype Conversion	Select Values	tConvertType	Data Conversion	Anydata Cast Operator
	Attribute Renaming	Select Values	tMap	Derived Column	
	Projection	Select Values	tFilterColumns		
Relation	Pivoting	Row Denormalizer	tDenormalize tDenormalizeSortedRow	Pivot	Unpivot
	Unpivoting	Row Normalizer Split field to rows	tNormalize tSplitRow	Unpivot	Pivot
Value	Single Value Alteration	If field value is null Null if Modified Java Script Value SQL Execute	tMap tReplace	Derived Column	Constant Operator Expression Operator Match-Merge Operator Mapping Input/Output parameter
Source Operation	Extraction	CSV file input Microsoft Excel Input Table input Text file input XML Input	tFileInputDelimited tDBInput tFileInputExcel	ADO .NET / DataReader Source Excel Source Flat File Source OLE DB Source XML Source	Table Operator Flat File Operator Dimension Operator Cube Operator
Target Operation	Loading	Text file output Microsoft Excel Output Table output Text file output XML Output	tFileOutpu tDelimited tDBOutput tFileOutputExcel	Dimension Processing Excel Destination Flat File Destination OLE DB Destination SQL Server Destination	Table Operator Flat File Operator Dimension Operator Cube Operator

Figure 4.3: ETL operations [The17]

Chapter 5

Data Quality

Following from the general definition of quality, the data is considered of high quality if it is *fit for its intended use* [BS16], meaning that the level of quality to be considered as acceptable, depends on the real needs of end user, and the purpose of the underlying data.

Data quality is sometimes wrongly reduced just to *accuracy* (e.g., name misspelling, wrong birth dates). However, other dimensions such as *completeness*, *consistency*, or *timeliness* are often necessary in order to fully characterize the quality of information.

5.1 Sources of problems in data

Different data quality problems may be introduced throughout the entire data pipeline.

- **Data ingestion.** While data being initially retrieved, accuracy issues can be introduced due to faulty *data conversions* or simply by making typos in the *manual data entry*. Similarly, *data timeliness* can also be affected if the data are being fetched periodically in the form of *batch feeds*, while being consumed through *real-time interfaces* implies expecting high *freshness* of the data. If data are being fetched from *consolidating various systems*, *consistency* errors may occur due to the heterogeneity of these source systems.

- **Data processing.** New errors can also be introduced while data are being processed for their final use. For instance, *process automation* may lead to overlooking some special cases and thus applying a default processing that result in inaccurate outputs (e.g., a missing case in a conditional statement in the code can lead to a default option). Paradoxically, new accuracy issues and other data imperfections can also be introduced during *data cleansing* actions (e.g., replacing missing values with default constants or precalculated aggregates like means or mediums). Finally, we can also affect the *completeness* and *consistency* of our data due to incautious *data purging* actions, by mistakenly deleting some of data values.

- **Inaction.** Lastly, data quality can also be largely affected by not taking necessary actions to prevent their occurrence. For instance, our data can become *inconsistent* if the *changes* that occur at one place are not properly propagated to the rest of the data that relates to it. Also, various *inaccuracies* can occur if the new *system upgrades* are not properly controlled, especially those that lack backward compatibility (e.g., data types or operations being changed or removed from DBMS). Generally, the same data that previously were considered of high quality can become flawed if the *new use of data* is considered (see the definition of quality above as data fitting their intended use).

5.2 Data Conflicts

Data conflicts refer to the deviations between data capturing the same real-world entity [Do]. Cleaning of data with conflicts is required in order to avoid misleading and faulty data analysis.

5.2.1 Classification of Data Conflicts

Data conflict can be classified from two orthogonal perspectives: (a) based on the level at which they occur (i.e., *schema* vs. *instances*), and (b) if they occur within a *single source* or among *multiple sources* (see Figure 5.1).

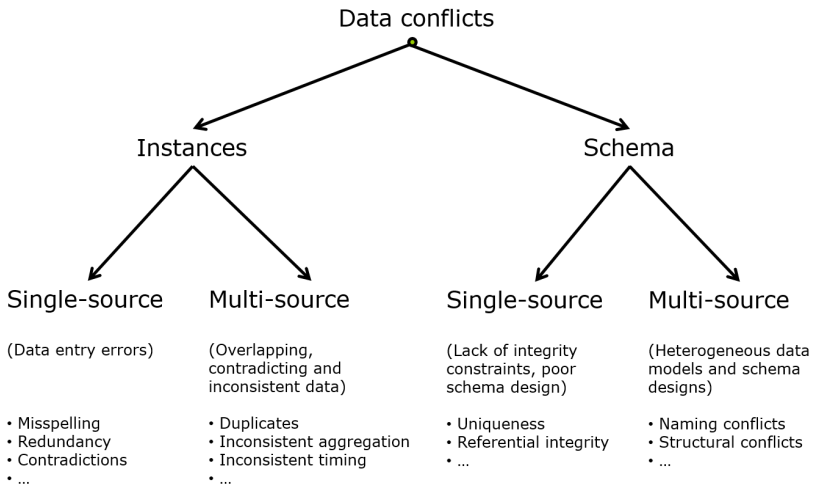


Figure 5.1: Data conflicts classification

Schema vs. instances. *Schema-level conflicts* are caused by the errors in the design of the data schemata (e.g., duplicates caused by the lack of unique constraint, inconsistency due to mistake in the referential integrity, structural schema conflicts among various data sources). *Instance-level conflicts* refer to errors in the actual data contents, which are typically not visible nor preventable at the schema design time (e.g., misspelling, redundancy or data duplicates and inconsistencies among various data sources). Both schema- and instance-level conflicts can be further differentiated based on the scope at which they occur: within an individual *attribute*, between attributes of a single *record*, between different records of certain *record type*, and between different records of different record type. **Single- vs. Multi-source.** The number of *single-source data* conflicts obviously largely (but not only) depends on the degree to which it is governed by schema constraints that control permissible data values. Therefore, the data sources that do not have native mechanisms or are more flexible to enforce a specific schema (e.g., CSV files, JSON documents) are more prone to errors and inconsistencies. All the errors potentially present in a single source are aggravated when *multiple sources* need to be integrated. Data may already come with errors from single sources, but they may additionally be represented differently, may overlap or even contradictory among various data sources. Problems that are typically caused by the fact that these source systems are independently developed.

5.2.2 Dealing with data conflicts

Conflicts in data can be either *prevented* from occurring or *corrected* once they occur. Obviously, preventing data conflicts is preferable due to the typically high costs of data cleaning. This requires proper schema design with well-defined data constraints or strict enforcement of constraints (e.g., by limiting user errors in the graphical user interface). However, in some cases, and almost always for the case of multiple sources, a posteriori data cleaning actions are necessary to guarantee a required level of data quality for a particular analysis. Such actions may involve very complex set of data transformation and are typically part of extract-transform-load (ETL) processes (see Chapter 4).

5.3 Data quality dimensions and measures

As already discussed before, data quality is a multidimensional or multifacet problem. The framework proposed in [BS16] classifies data quality in clusters of dimensions, where dimensions are grouped based on their similarity, and the representative (first) dimension gives a name to the cluster.

1. **Completeness**, pertinence, and relevance refer to the capability of representing all and only the relevant aspects of the reality of interest.
2. **Accuracy**, correctness, validity, and precision focus on the adherence to a given reality of interest.
3. **Redundancy**, minimality, compactness, and conciseness refer to the capability of representing the aspects of the reality of interest with the minimal use of informative resources.
4. **Readability**, comprehensibility, clarity, and simplicity refer to ease of understanding and fruition of information by users.
5. **Accessibility** and availability are related to the ability of the user to access information from his or her culture, physical status/functions, and technologies available.
6. **Consistency**, cohesion, and coherence refer to the capability of the information to comply without contradictions to all properties of the reality of interest, as specified in terms of integrity constraints, data edits, business rules, and other formalisms.
7. **Usefulness**, related to the advantage the user gains from the use of information.
8. **Trust**, including believability, reliability, and reputation, catching how much information derives from an authoritative source. The trust cluster encompasses also issues related to security.

In what follows, we present in more details the four most relevant data quality dimensions, i.e., *completeness*, *accuracy*, *timeliness* (a special case of temporal accuracy), and *consistency*, and for each dimension we present as well a quality measure formula used to estimate the achievement of such data quality dimension within the dataset [Sat18].

5.3.1 Completeness

Data completeness is most typically related to the issue of missing or incomplete data, and as such it is considered one of the most important data quality problems.

Most often interpretation of the data completeness is the absence of *null* values, or more exactly, the ratio of non-null values and the total number of values. Such interpretation can be easily represented as a measure. First, at the level of a single attribute (A_i) we have:

$$Q_{Cm}(A_i) = \frac{|R(NotNull(A_i))|}{|R|} \tag{5.1}$$

Following from there, the completeness of the entire Relation (R) can be measured by counting the number of tuples containing no null values. That is:

$$Q_{Cm}(R) = \frac{|R(\wedge_{A_i \in R} NotNull(A_i))|}{|R|} \tag{5.2}$$

However, the problem of measuring the completeness in this way is that *null* can have different semantics in our dataset. It could really mean a missing value or simply a not-applicable case (e.g., a customer without a special delivery address).

Moreover, the absence of values for different attributes may have higher or lower importance. For instance, while we typically always need an identifier attribute (e.g., customer ID), having the city of birth missing may be optional. To address such a case, we can assign corresponding weights to the calculation of the completeness of each attribute in the Relation.

Note these measures compute the completeness of the data that are inside the database, i.e., following the Closed World Assumption (CWA). However, the original definition expresses the completeness

as the degree to which a given dataset describes the corresponding set of real-world objects. In this case, measuring the completeness can be more difficult, as it may require additional metadata or cross checking with real world (e.g., possibly by sampling). Lastly, another aspect of completeness refers to the fact if a real-world property is represented as an attribute in a dataset or not. However, assessing this aspect as well requires (often manual) inspection with the real world.

5.3.2 Accuracy

The quality of data can as well be affected by measurement errors, observation biases, or improper representations. This aspect is covered by the accuracy dimension, which can be defined "as the extent to which data are correct, reliable, and certified free of errors". The meaning of accuracy is application-dependent, and it can be expressed as the distance from the actual real-world value or the degree of detail of an attribute value. For instance, a product sales volume variable in our data warehouse can have a value 10.000$, while the value in reality (e.g., in accounting system) is 10.500$. While in general this can be interpreted as inaccurate, in another use case, where the user is only interested in binary sales categories (low: $\leq$ 20.000$, high: $>$ 20.000$), the value in the data warehousing system can be considered as accurate enough.

To assess the accuracy in a general case, we need to calculate the distance of the value stored in a our database from the actual real value.

$$e_A = |V_a - V_{RealWorld}| \tag{5.3}$$

In the case of numerical attribute such distance is simply an arithmetic difference, while in the case of other data types it may require more complex way of calculation. For example, *Hamming distance* calculates the number of positions at which two strings of the same length differ, or its more complex version, *Levenshtein (or Edit) distance*, that calculates the minimal number of character-level operations (insertions, deletion or substitutions) required to change one string into another one.

Regardless of the way e_A is calculated, we further assess the accuracy of an attribute as follows.

$$Q_A(A_i) = \frac{|R(e_{A_i} \leq \varepsilon)|}{|R|} \tag{5.4}$$

Notice that the threshold ε is application specific and it determines the level of tolerance for the accuracy of a certain attribute.

As before, we can further apply this to the whole Relation as follows:

$$Q_A(R) = \frac{|R(\wedge_{A_i \in R} e_{A_i} \leq \varepsilon)|}{|R|} \tag{5.5}$$

However, in some cases, we may need to adopt more complex means for assessing the accuracy. For instance, for textual attributes we may need to consider *semantic distance* of their values (e.g., both Munich and München represent the same city although they are syntactically "far from each other"). Resolving such a problem, would require the existence of a dictionary or ontology mapping.

5.3.3 Timeliness

Another problem that data may suffer is that they are outdated, meaning that they are captured/extracted from the original source potentially before the new changes in the source may occurred. This dimension is also referred to as *freshness*. The level of timeliness as well depends on the specific application that the data is used for. For example, the last year's accounting data may be considered as "old", but if the users are actually auditing company's last year's operations, these data is just what they need.

On the other hand, the real "age" of the data is also determined by the *frequency* of updates that may occur over the data since the last capturing/extraction. For example, the system that uses UN's population and migration indicators may have data that are months old, but they may be considered as "fresh" having that the UN publishes these indicators annually.

We thus need to consider both value's *age* (i.e., calculated from the time when the datum is committed in the database, a.k.a. Transaction Time[1], as $age(v) = now - transactionTime$) and its *frequency of updates* per time unit ($f_u(v)$) in order to assess its timeliness.

$$Q_T(v) = \frac{1}{1 + f_u(v)\dot{a}ge(v)} \tag{5.6}$$

We can further extend this to an attribute (A_i), as:

$$Q_T(A_i) = Avg_{v \in R[A_i]} Q_T(v) \tag{5.7}$$

, and to the entire Relation (R), as:

$$Q_T(R) = Avg_{A_i \in R} Q_T(A_i) \tag{5.8}$$

The consequence of this is that data with higher update frequency "ages" faster while those that are never updated are always considered as "fresh".

5.3.4 Consistency

Consistency refers to the degree to which data satisfies defined semantic rules. These rules can be integrity constraints defined in the DBMS (i.e., *entity* - "a primary key of a customer cannot be null", *domain* - "customer age column must be of type integer", *referential* - "for each order a customer record must exists"), or more complex business rules describing the relationship among attributes (e.g., "a patient must be female in order to have attribute pregnant set to true"), typically defined in terms of *check constraints* or *data edits*[2]. Recent research approaches also propose automatic derivation of such rules from the training data by applying rule induction.

Having that B is a set of such rules for Relation R, we can calculate the ratio of tuples that satisfy all the rules as:

$$Q_Cn(R, B) = \frac{|R(\wedge_{rule \in B} rule(A_1, .., A_n))|}{|R|} \tag{5.9}$$

5.3.5 Trade-offs between data quality dimensions

Data quality dimensions often cannot be considered in an independent manner, given that there is a correlation between some of them [BS16]. Therefore, favoring one data quality dimension may have negative consequence for the other ones. We distinguish two important cases.

1. *Timeliness* $\Longleftrightarrow$ (*Accuracy*, *Completeness*, and *Consistency*). Having accurate (or complete or consistent) data may require either checks or data cleaning activities that take time and thus can affect negatively the timeliness dimension. Conversely, timely data may suffer from accuracy (or completeness or consistency) issues as their freshness was favored. Such trade-offs must always be made depending on the application. For instance, in some Web applications, timeliness is often preferred to accuracy, completeness and consistency, meaning that it is more important that some information arrives faster to the end user, although it initially may have some errors or not all the fields be completed.

2. *Completeness* $\Longleftrightarrow$ (*Accuracy* and *Consistency*). In this case, the choice to be made is if it is "better" (i.e., more appropriate for a given domain), to have less but accurate and consistent data, or to have more data but possibly with errors and inconsistencies. For instance, for social statistical analysis it is often required to have a significant and representative input data, thus we would favor completeness over accuracy and consistency in such case. Conversely, when publishing scientific experiment results, it is more important to guarantee their accuracy and consistency than their completeness.

[1] https://en.wikipedia.org/wiki/Transaction_time
[2] https://en.wikipedia.org/wiki/Data_editing

5.4 Data quality rules

As discussed before, data quality conflicts can be either prevented a priori or corrected a posteriori. Relational DBMSs offer various mechanisms to prevent such conflicts by defining data quality rules in terms of *integrity constraints* or *dependencies*. However, such rules need to satisfy a set of *logic properties* in order to guarantee their minimality and that they can accommodate the data (i.e., that there is an instance of a dataset satisfying all the rules).

5.4.1 Integrity constraints and dependencies

Integrity constraints can be further classified as follows:

- **Intra-Attribute.** Constraints that affect a single database column.
  - *Domain.* The set of values permitted for a given column. This is typically defined by the column data type, but can as well be more complex (e.g., to detect column's outliers).
  - *Not null.* To guarantee the existence of the value for such a column, and thus improve data *completeness* dimension.
- **Intra-Tuple.** Constraints that affect single tuples (i.e., among different columns of a single tuple).
  - *Checks* are the most common way of defining the intra-tuple constraints in a Relational database. For instance, "`basicSalary > 2000`" in the *department* table or "`originAirport <> destinationAirport`" in the *flight* table.
- **Intra-Relation.** Constraints that have as a scope the entire Relation (i.e., entire database table).
  - *Primary key* or *Unique* are the most common intra-Relation constraints defined over a database table that guarantee uniqueness (i.e., no repetition of values) of a column or a set of columns in a table.
  - *Temporal* intra-Relation constraints are typically defined by means of database *triggers* and assume the existence of temporal attributes. They may be used to express several rules, such as (being X and Y potentially different for each object): (1) *Currency:* each object must have at least one value for the temporal attribute in the last X days; (2) *Retention:* no object can have a value before data X; (3) *Granularities:* the time distance between two consecutive values of the temporal attribute must be between X and Y; (4) *Continuity:* if the temporal attribute has begin and end timestamps, these intervals cannot overlap and must be adjacent. More complex rules can also be defined. For example, we could check the distance of each point to the linear regression of the temporal attribute.
- **Inter-Relation.** Constraints that define the relationship between the columns of two different database tables.
  - *Foreign key* is the most typical way used to define the referential integrity constraint among two tables. That is, the values of a foreign key column(s) of one table must match some value in the primary key column(s) in another table.
  - *Assertions* can be used to define more complex inter-Relation constraints, not limited to foreign-primary key relationships. Such rules are defined in terms of database triggers that are fired (with the corresponding action execution) when a rule is violated.

Using these integrity constraints we can maintain the following dependencies among the attributes of our database.

- **Multivalued dependency** (whose special case is *functional dependency*) defines that a certain set of attributes X unequivocally determines a value (for functional dependency) or a set of values (in a general case) of another set of attributes Y, of the same Relation. This intuitively means that two tuples sharing the same values for attributes in X must have the same values for the attributes in Y (i.e., $t1.X = t2.X \Rightarrow t1.Y = t2.Y$).

- **Inclusion dependency** is maintained by defining the referential integrity constraint (*foreign key*) between the attributes of two database tables. Intuitively, if Y is a set of attributes that defines a primary key of table S and X is a set of attributes that defines a foreign key in table R that references $S.Y$, it is gurauranteed that a set of tuples over attributes $R.X$ is subsumed by a set of tuples over attributes $S.Y$ (i.e., $R.X \subseteq S.Y$).

5.4.2 Logic properties of data quality rules

However, if not defined with care, the constraints that express the desired data quality rules may yield certain imperfections, subotimalities or even contradictions. For instance, two checks that are defined over the same salary attribute may be contradictory (e.g., $CHECK\ (salary < 1000\ AND\ salary > 2000)$) and hence generate empty empty results (i.e., prevent generating any tuple). Similarly, a rule may also be redundant to another previously defined rule (e.g., $CHECK(salary > 2000)$ and $CHECK(salary > 1000)$), meaning that the latter rule will be impossible to violate, hence its checking introduces an unnecessary overhead.

To guarantee that a set of data quality rules defined over a database is minimal and allows tuple generation, we can rely on the following set of *logic properties of rules*. Notice that these logic properties are defined over different database elements (schema, tables, views, constraints, tuples, queries), depending on their scope and semantics.

- **Schema-satisfiability**: A *schema* is satisfiable if there is at least one consistent DB state containing tuples (i.e. each and every constraint is fulfilled).
- **Liveliness**: A *table/view* is lively if there is at least one consistent DB state, so that the table/view contains tuples.
- **Redundancy**: An *integrity constraint* is redundant if the consistency of the DB does not depend on it (none of the tuples it tries to avoid can never exist).
- **State-reachability**: A given *set of tuples* is reachable if there is at least one consistent DB state containing those tuples (and maybe others).
- **Query containment (subsumption)**: A *query* Q1 is contained in another Q2, if the set of tuples of Q1 is always contained in the set of tuples of Q2 in every consistent DB state.

5.4.3 Fine-tuning data quality rules

As we discussed earlier, data quality rules, may yield certain imperfections when it comes to data quality assessment. Indeed, we can find both situations where data quality rules fall short, these being *false positives* and *false negatives* [May07]. The former refers to the well known cases when a rule detects as erroneous tuples that in reality are correct, while in the latter case, the rule fall short in detecting an erroneous tuple. These rule imperfections are very important for accurately assessing the quality of data and thus require further fixing and fine-tuning in order to enhance the rules as much as possible. A process of rule fine-tuning proposed in [May07] is depicted in Figure 5.2.

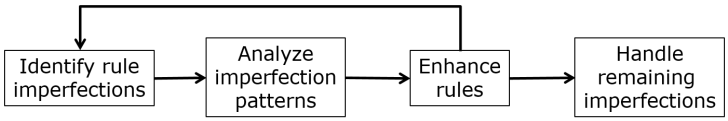


Figure 5.2: Steps in rule fine-tuning process [May07]

The process starts by identifying rule imperfections, followed by the analysis of such findings and searching for the patterns in the data. In the third step, the rules are enhanced to eliminate as many imperfections as possible. These three steps are executed iteratively until we are satisfied with the enhancement or we simply run out of the resources. Such process largely relies on manually verifying

samples of real-world data and the comparison with the errors identified by applying rules. In the final step, we handle the remaining imperfections, i.e., those that could not be improved but we are at least aware of and are accounted for in the data quality assessment.

5.5 Data quality improvement

As we said, data quality rules can be used to prevent data conflicts from occurring in our datasets. However, and especially in the case when data are coming from external and heterogeneous data sources, it is impossible to prevent errors in data a priori. In such cases, when data conflicts are detected, we need to proceed (a posteriori) with their corrections in order to improve the data quality in the resulting dataset. Such process of restoring correct values is commonly known as *imputation*. Imputations should be done following the two main goals [FH76]: (1) *minimum change principle*, stating that the corrections on data must be done by changing the fewest values possible, and (2) maintain the marginal and joint frequency distribution of values in the different attributes. However, these two goals may be in conflict. For example, a tuple <6, married, retired> of table <age, maritalStatus, typeOfWork> is detected as erroneous given the previously defined data quality rule ($maritalStatus = married \Rightarrow age \geq 15$). To correct this and similar tuples with minimum change, we can impute a value 15 to any value violating the given rule. However, by doing so, we alter the *marginal distribution* of values of age. Even if we perform imputation respecting the distribution of values of age, we may affect the *joint distribution* with attributes maritalStatus and typeOfWork. Following from this, we can see that data quality improvement requires more complex and wider set of changes.

Generally, the methodologies proposed for improving the quality of data adopt two general types of strategies, i.e., *data-driven* and *process-driven* [BS16]. *Data-driven* strategies improve the quality of data by directly modifying the value of data. For example, obsolete data values are updated by refreshing a database with data from a more current database. *Process-driven* strategies improve quality by redesigning the processes that create or modify data. As an example, a process can be redesigned by including an activity that controls the format of data before storage.

Both data-driven and process-driven strategies apply a variety of techniques: algorithms, heuristics, and knowledge-based activities, whose goal is to improve data quality. In what follows, we present a (non complete) list of data-driven data quality improvement techniques:

1. **Acquisition of new data**, which improves data by acquiring higher-quality data to replace the values that raise quality problems;

2. **Standardization** (or **normalization**), which replaces or complements nonstandard data values with corresponding values that comply with the standard. For example, nicknames are replaced with corresponding names, (e.g., Bob with Robert), and abbreviations are replaced with corresponding full names, (e.g., Channel Str. with Channel Street);

3. **Entity resolution** (or **record linkage**), which identifies if the data records in two (or multiple) tables refer to the same real-world object;

4. **Data and schema integration**, which define a unified view of the data provided by heterogeneous data sources. Integration has the main purpose of allowing a user to access the data stored by heterogeneous data sources through a unified view of these data. Data integration deals with quality issues mainly with respect to two specific activities:

   - *Quality-driven query processing* is the task of providing query results on the basis of a quality characterization of data at sources.
   - *Instance-level conflict resolution* is the task of identifying and solving conflicts of values referring to the same real-world object.

5. **Source trustworthiness**, which selects data sources on the basis of the quality of their data;

6. **Error localization** and **correction**, which identify and eliminate data quality errors by detecting the records that do not satisfy a given set of quality rules. These techniques are mainly studied in the statistical domain. Compared to elementary data, aggregate statistical data, such as average, sum, max, and so forth are less sensitive to possibly erroneous probabilistic localization and correction of values. Techniques for error localization and correction have been proposed for inconsistencies, incomplete data, and outliers.

7. **Cost optimization**, defines quality improvement actions along a set of dimensions by minimizing costs.

5.6 Object identification

Object identification is the most important and the most extensively investigated information quality activity [BS16], and it refers to the process of identifying whether the data in the same or in different data sources represent the same object in the real world.

Table 5.1: Example of three national registries (agencies) that represent the same business [BS16]

Agency	Identifier	Name	Type of activity	Address	City
Agency 1	CNCBTB765SDV	Meat production of John Ngombo	Retail of bovine and ovine meats	35 Niagara Street	New York
Agency 2	0111232223	John Ngombo canned meat production	Grocer's shop, beverages	9 Rome Street	Albany
Agency 3	CND8TB76SSDV	Meat production in New York state of John Ngombo	Butcher	4, Garibaldi Square	Long Island

Example in Table 5.1 shows that the same object in the real world (i.e., the same business) can be represented in different manners in three national registries (i.e., *Agency 1*, *Agency 2*, and *Agency 3*). Some differences like different identifiers may be simply due to the different information systems from where these tuples are coming from. Other attributes like name, type of activity, address, and city also differ (although with some similarities), and this can be due to several reasons, like typos, deliberately false declarations, or data updated at different times.

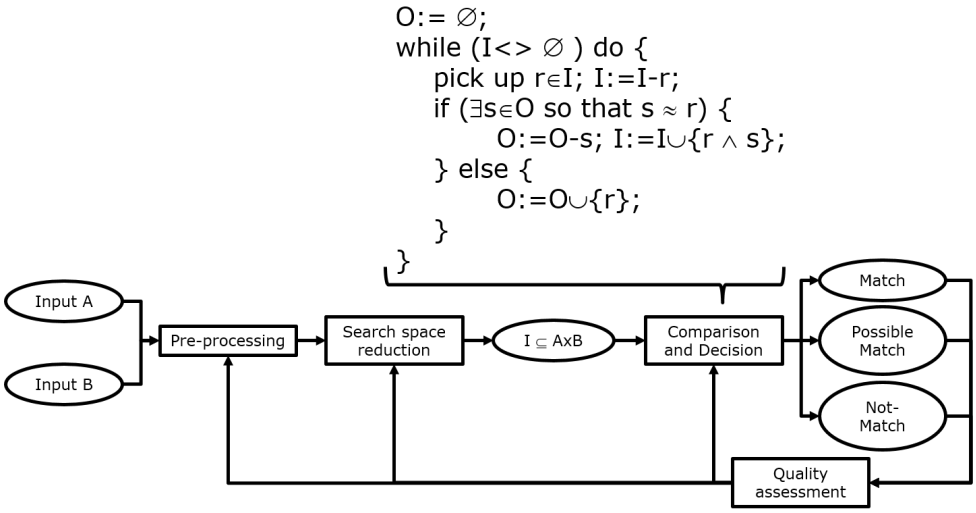


Figure 5.3: Relevant steps of object identification techniques [BS16]

The high-level overview of the *object identification process* is depicted in Figure 5.3. Assuming for simplicity two input data sources (A and B), the process includes the following activities:

1. **Pre-processing** has as a goal to normalize/standardize the data and correct evident errors (e.g., conversion of upper/lower cases) and reconcile different schemata.

2. **Search space reduction.** Performing entity resolution on the entire search space (i.e., Cartesian product of tuples in inputs) would result in complexity $O(n^2)$, n being the cardinality of input Relations. To make this process tractable we first need to reduce the given search space. This is typically done by means of three different methods: (a) *blocking*, which implies partitioning a file into mutually exclusive blocks and limiting comparisons to records within the same block, (b) *Sorted neighborhood* consists of sorting a file and then moving a window of a fixed size on the file, comparing only records within the window, and (c) *Pruning* (or *filtering*) implied removing from the search space all tuples that cannot match each other, without actually comparing them.

3. **Comparison and decision.** This step includes an entity resolution (or record linkage) technique that first selects a *comparison function* used to compare if two tuples represent the same object in the real world, by calculating the "distance" between them (see Section 6.3.4.1 for more details), and then *decides* if the compared tuples *match*, *do not match*, or *potentially match* based on the analysis of the results of the comparison function. An example of the entity resolution algorithm (R-Swoosh [GUW09]) is presented in Figure 5.3, while more details are provided in Section 6.3.4.3. It may also happen that no decision can be made automatically and a domain expert has to be involved to make the decision.

4. **Quality assessment** is finally performed based on the result of the previous comparison and data quality indicators are measured to assess if the results are satisfactory. Minimizing *possible matches* is a typical goal to avoid as much as possible the involvement of the domain expert. In addition, minimizing *false positives* and *false negatives* are as well common goals in the quality assessment step.

Multimedia Materials

Data Quality Measures (en) 

Data Quality Measures (ca) 

Data Quality Rules (en) 

Data Quality Rules (ca) 

Chapter 6

Schema and data integration

With contributions of Fernando Mendes Stefanini and Oscar Romero

The integration problem appears when a user needs to be able to pose one query, and get one single answer, so that in the preparation of the answer data coming from several DBs is processed. It is important to notice that the users of the sources coexist (and should be affected as little as possible) with this new global user. In order to solve this, we can take three different approaches:

a) Manually query the different databases separately (a.k.a. *Superman* approach), which is not realistic, since the user needs to know the available databases, their data models, their query languages, the way to decompose the queries and how to merge back the different results.

b) Create a new database (a.k.a. DW or ERP) containing all necessary data.

c) Build a software layer on top of the data sources that automatically splits the queries and integrates the answers (a.k.a. *federation* or *mediation* approach).

6.1 Distributed Databases

In the end, independently of the approach we take, we are talking about a distributed database but with a variable degree of autonomy in the components. This autonomy impacts the design (components are designed independently), the execution (components can execute in different ways and offering different guarantees), and the association (components can easily and freely leave the system).

6.2 Semantic heterogeneities

All this autonomy generates heterogeneities in the end. In what follows, we first discuss the semantic heterogeneities that may appear among the autonomously built data sources.

6.2.1 Definitions

Some terms like *classes*, *entities*, *attributes* can have different meanings depending on the context they are applied. In order to avoid ambiguities, we first define how some common terms are going to be employed. With this, we hope to aid explanations in further sections.

6.2.1.1 Concepts

In the context of this chapter, the term *class* and *entity* is used interchangeably to define a concept of the domain, analogously to a Relational table in Relational databases or a class in object oriented designs, such class is composed by attributes and might have relationship with other classes. Thus, we define class C as a concept containing a set of attributes. Any concept can be represented by a class in a

UML Class diagram. For example, the class `Playlist` containing attributes `name, lastModifiedData, Description, #Followers`; and the class `Track` containing attributes `trackName, artistName, note`; can be represented as:

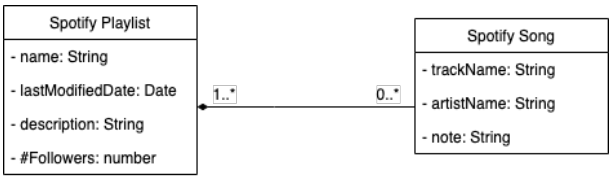


Figure 6.1: Sample of Spotify classes.

This representation is conceptual and agnostic to the way the data are physically stored. To illustrate that, we display an snippet of the file `Playlists.json` from Spotify, which contains the data from which the diagram in Figure 6.1 was generated. The full conceptual model for a given data source is then called the *schema*.

Listing 6.1: Snippet of File `Playlist.json`

```
{
"playlists": [
{
"name": "Bjork Pitchfork",
"lastModifiedDate": "2018-10-16",
"tracks": [
{
"trackName": "Cosmogony",
"artistName": "Bjork",
"note": "Usually the first song of the concert"
},
{
"trackName": "Hunter",
"artistName": "Bjork",
"note": null
},
...],
"description": null,
"numberOfFollowers": 0
},
...]
}
```

Lastly, an *instance* will be an instantiation of a concept of the schema; i.e.: the playlist "Bjork Pitchfork" is an instance of the class playlist.

6.2.1.2 Equivalence and Hierarchy

Furthermore, we elaborate in the notation used to express relationship between classes, and attributes, that will be further used in the rest of the work. Given a set of classes $\{C_1, C_2, ..., C_n\}$ we say that:

1. $C_1 \equiv C_2$ iff C_1 represents the same concept as C_2 (equivalent).
2. $C_1 \sqsubset C_2$ iff C_2 represents a broader concept that subsumes C_1 (C_1 subclass of C_2 / C_2 superclass of C_1).

6.2.2 Classification of semantic heterogeneities

In order to integrate the data from various data sources, we face the problem of dealing with the heterogeneity between the two schemas, their concepts and their instances. Thus, we formalize the possible heterogeneities between data sources by splitting them in two main groups, *Intra-Class heterogeneity* and *Inter-Class heterogeneity*; some naming and definitions were inspired on the work of [GSSC95] . We use

C_B to represent the set of classes $\{C_{B1}, C_{B2}, ..., C_{Bn}\}$ from one domain schema (represented in blue), and C_G to represent the set of classes $\{C_{G1}, C_{G2}, ..., C_{Gm}\}$ of another domain schema (represented in green).

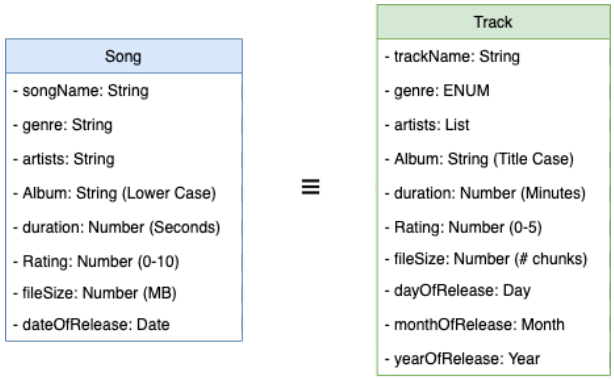


Figure 6.2: $C_{B1} \equiv C_{G1}$ where C_{B1} = Song and C_{G1} = Track

6.2.2.1 Intra-Class Heterogeneity

We classify as intra-class heterogeneity the cases where there is a class in one schema that is equivalent to a class in the other, i.e.: $C_{B1} \equiv C_{G1}$. This means that both classes represent the same concept, but because of the characterization of the concepts in each schema, some discrepancies can be found both within the classes and within their attributes. We take for example the case in Figure 6.2, where song from the C_B and track from the C_G are equivalent concepts. We consider the intra-class heterogeneities in Table 6.1.

Class	Naming		When the classes have different names (e.g., Song in C_B and `Track` in C_G).
Attribute	Naming		When the attributes have different names (e.g., `songName` and `trackName`).
	Type		When two equivalent attributes have different type representation between domains (e.g., `genre` in C_B and in C_G).
	Single/Multi-valuation		When the attribute is single-valued in one domain but multi-valued in another (e.g., `artists` in C_B and C_G).
	Representation	Format	When the attributes have the same type, but are represented in different formats (e.g., `Album` in C_B and C_G).
		Measure	When two attributes are numerical, but the unit to represent them differs (e.g., `duration` in C_B and C_G).
		Scale	When two attributes are numerical, but the scale to represent them differs (e.g., `rating` in C_B and C_G).
		Dimension	When two attributes are numerical, but the dimension to represent them differs (e.g., `fileSize` in C_B and C_G).
	Composition		When one attribute is the result of the composition of many attributes (e.g., `dateOfRelease` in C_B and `dayOfRelease`, `monthOfRelease` and `yearOfRelease` in C_G).

Table 6.1: Intra-Class heterogeneities

6.2.2.2 Inter-class Heterogeneity

For the cases when we can not do a direct equivalence between two classes, we might need to express it as a composition of one or many classes, or super/sub classes. Thus, we give an example in Figure 6.3 of the classes belonging to C_B and C_G, with the following arrangements between two schemata:

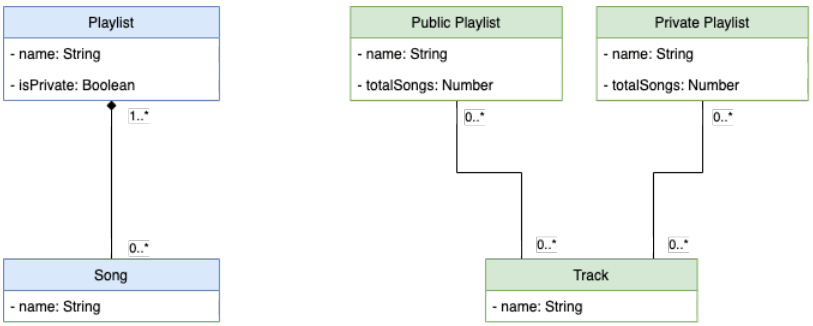


Figure 6.3: C_B and C_G classes.

- Public Playlist ⊏ Playlist
- Private Playlist ⊏ Playlist
- Song ≡ Track

We identify the inter-class heterogeneities that we consider between the two sets of classes in Table 6.2.

Super/Subclass of		When a Class in a schema is a generalization or specification of a class in another (e.g., `Playlist` in C_B and `Public/Private Playlist` in C_G).
Aggregation	Composition	When a class is composed by an aggregation of another class in one domain, but is an independent class in another (e.g., Song is an aggregation inside `Playlist` in C_B while `Track` has simple associations with `Public/Private Playlist` in C_G).
	Derivation	When the characterization of a class is derived from the aggregation of another (e.g., `Public/Private Playlist` in C_G have `totalSongs`, an attribute defined by the aggregation of `Track` in C_G. While in C_B such dependency is absent).

Table 6.2: Inter-Class heterogeneities

6.3 Overcoming heterogeneity

Next, we will see how some of these heterogeneities can be resolved by schema and data integration solutions.

6.3.1 Wrappers and mediators

One solution (that belongs to the class of federation/mediation approaches) for overcoming the heterogeneity and resolving data integration problem is a *mediator-based system* [GUW09]. Differently from the DW systems, *mediators* do not store any data, but rather offer an interface for posing a single query, decomposing such query so that several (typically more than 2) data sources are queried and then integrates the results of individual queries in order to return a single unified answer to the user (see Figure 6.4).

Mediators can be seen as virtual views but instead of accessing various database tables, they access various (typically independent) data sources. To access data of each data source, mediators work in combination with another set of components - *wrappers*. A wrapper represent a piece of software that understands the underlying language and format of a data source system, and it can query it in order to retrieve the data back to the mediator which combines the results and return the answer to the user.

For instance, consider that one data source is an RDBMS and that a wrapper has access to table `vehicles`:

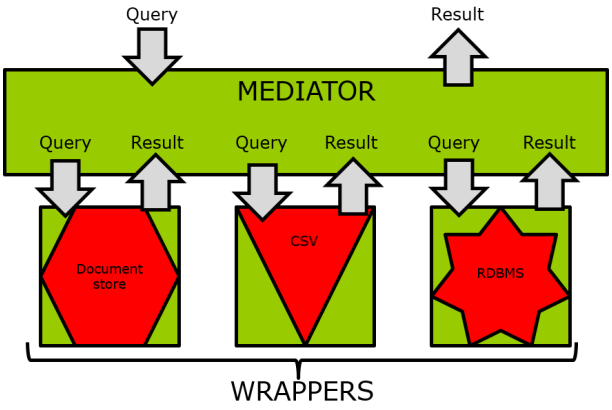


Figure 6.4: Mediator-based system

```
vehicle (serial_n,make,model,auto_transmission,air_conditioning,color)
```

We can implement such wrappers in the form of an SQL query template that includes parameters representing constants. For example, a wrapper template to access the vehicles of certain color would be:

```
SELECT serial_n,model,make,auto_transmission, color
FROM vehicle
WHERE color = '$c';
```

Then the mediator, for example implemented in Java, after receiving a global query (over the med_car table) needs to forward the query (using the corresponding JDBC driver) to a corresponding wrapper template and generate a wrapper needed to retrieve the data.

```
SELECT *                    SELECT serial_n,make,model,auto_transmission,color
FROM med_car          =>    FROM vehicle
WHERE color = '$c';         WHERE color = '$c';
```

In addition, consider that there exists another data source based on CSV files, and with schemata:

```
car(serial_n, make, model, auto_transmission, color)
optional_equipment(serial_num, option, price)
```

To answer the previous query, our mediator needs also to access the CSV files of the second data source (e.g., using Java file API) and retrieve data from the car file.

Finally, the mediator combines the results of the two queries and returns the final result to the user.

6.3.2 Major steps to overcome semantic heterogeneity

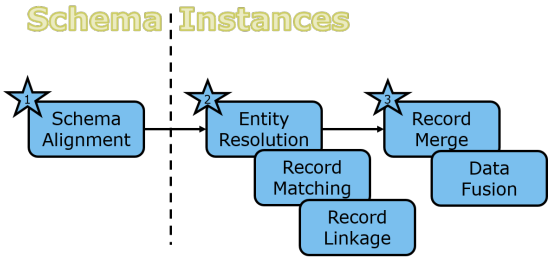


Figure 6.5: Major steps to overcome semantic heterogeneities

For enabling the mediator-based system to automatically handle the heterogeneity among various data sources, there are three major steps to be considered (see Figure 6.5):

1. *Schema alignment* works at the schema level, and includes establishing the correspondences (a.k.a. mappings) among the concepts (i.e., attributes or variables) of different data sources.

2. *Entity resolution*, also know as *record matching* or *record linkage*, is the process focused on finding the instances (i.e., records, tuples or values) that represent the same entities in reality.

3. *Record merge*, also known as *data fusion*, is the final step of data integration, in which, after the equivalent records are found, they are merged, either by combining them or keeping the more "accurate" or more "trustworthy" one.

Schema alignment corresponds to what is know as *schema integration*, while the latter two (*entity resolution* and *record merge*) correspond to the *data integration* process. In what follows, we present in more detail these two processes.

6.3.3 Schema integration

The process of aligning different data source schemata results with three main outcomes [DS15]:

1. *Mediated/Integrated/Global schema*, which represents a unified and reconciled view of the underlying data sources [Len02]. It captures the semantics of the considered user domain, including the complete set of concepts of the data sources and the relationships among them.

2. *Attribute matching*, which specifies which attributes of the data sources' schemata correspond to which concept of the global schema. This matching can be 1-1, but sometimes one concept of the global schema may correspond to the combination of several attributes in the data sources' schemata.

3. *Schema mappings* are then built based on the previously found attribute matchings and they specify the semantic relationships (i.e., correspondence) between the schema of a data source (i.e., *local schema*) and the previously established *global schema*, as well as the transformations needed to convert the data from local to global schema.

6.3.3.1 Schema mappings

Schema mappings are typically expressed as views (or queries) either over the data sources' schemata or over the global schema.

First, we distinguish 3 main kinds of schema mappings, i.e., *sound*, *complete*, and *exact* [FM18], depending on the data they intent to map ($q_{Desired}$) and the data returned by the query of the mapping ($q_{Obtained}$).

- *Sound mappings* ($q_{Obtained} \subseteq q_{Desired}$) define the correspondence in which the data returned by the mapping query is a *subset* of those required, but it may happen that some of the required data is not returned by the query.

- *Complete mappings* ($q_{Obtained} \supseteq q_{Desired}$) on the other hand, define a correspondence in which the data returned by the mapping query include all those data that are required, but it may happen that some other data are returned as well (i.e., the returned data are a *superset* of the required data).

- *Exact mappings* ($q_{Obtained} = q_{Desired}$) finally are combination of both *sound* and *complete* mappings, meaning that the mapping query returns exactly the data required by the mapping, no more and no less.

Furthermore, we can implement schema mappings by means of two main techniques, i.e., *Global As View* (GAV) and *Local As View* (LAV).

In *GAV*, we characterize each element of global schema in terms of a view (or query) over the local schemata. Such approach enables easier query answering over the global schema, by simply unfolding the global schema concepts in terms of the mapped data sources, which is absolutely equivalent to

processing views in a centralized RDBMS. However, this technique can result too rigid, having that a change over a single data source schema may require the change of multiple (and in the extreme case all) schema mappings.

In the case of *LAV*, we characterize elements of the source schemata as views over the global schema. LAV mappings are intended to be used in the approaches where changes in data source schemata are more common having that changes in the local schema only require updating the subset of mappings for the affected data source. However, the higher flexibility of LAV mappings comes with the higher cost of answering the queries over the global schema, which implies the same logic as answering queries using materialized views, already known as a computationally complex task [Hal01].

There are also some research attempts to generalize the previous two techniques, like *Global/Local As View*, which maps a query over the global schema in terms of a query over the local schemata, thus benefiting from the expressive power of both GAV and LAV techniques [FLM+99], and *Peer To Peer (P2PDBMS)*, in which case each data source acts as an autonomous "peer", while separate (*peer-to-peer*) mappings are defined among each pair of data sources [DGLLR07].

Despite their rigidness, GAV is the most widely used schema mapping approach in production data integration systems because of its simplicity.

6.3.4 Data integration

Once the schemata of different data sources are aligned, we move to the instance level. Here, as we mentioned before, there are two main steps. First, we look for the matching of records of different data sources (i.e., *entity resolution*) and then, we perform merging of the previously matched records such that at the output we have a single record that represents the object in the real world.

6.3.4.1 Entity resolution

Entity resolution tackles the problem of finding whether two (or more) records (typically from different data sources) represent the same object in the real world [GUW09].

For instance, if the records are representing individuals with their `name`, `address`, and `phone number`, to find the records that represent the same individual, it is not sufficient to only look for exact/identical values of these records, due to various factors that may affect how these values are represented: (1) *misspellings*, which may occur due to similar pronunciation ("Smythe" vs "Smith") or adjacency of letters on the keyboard ("Jones" vs "Jomes"); (2) *variant names* that may differ depending on the form in which the name is provided (e.g., with full first name and middle initial "Susan B. Williams", with initial of the first name "S. Williams", or with nickname "Sue Williams"); (3) *misunderstanding of foreign names*, that may occur in the cases in which one form may be assumed as default (e.g., Name Surname), while in other countries the typical order may be reverse, so "Chen Li" may (or may not) refer to the same individual as "Li Chen"; (4) *evolution of values* that occur when two records that represent the same object are created at different point in times, so some fields (e.g., phone number, address, marital status) may differ; and (5) *abbreviations* that may cause that the same objects (e.g., address) are spelled differently in different data source (e.g., "Sesame Street" vs "Sesame St.").

Therefore, in the entity resolution process, we need to look carefully at the kinds of discrepancies that occur and devise a scoring system or other tests that measure the similarity of records. Ultimately, we must turn the score into a yes/no decision in order to answer the question whether the records represent the same entity or not.

Some quick simplifications may be used to tackle the problem of value discrepancy and make the further matching process easier, like *normalization* of the strings (e.g., using first name initial without middle initials, or using abbreviation dictionary to always replace abbreviations with their full names). Also, we can decompose otherwise complex problem of entity resolution of the entire records, and do the comparisons field by field.

Similarity function ($\approx$). In a general case, in order to find how "close" are two values, we need to employ an effective *similarity function*. Depending on the expected level of discrepancy, and the type of values, we can use different methods to measure their similarity:

- A simple *Hamming distance*, which for two strings of the same length, measures the number of positions at which the corresponding symbols are different (e.g., $Hamming(\text{“}Jones\text{”},\text{“}Jomes\text{”}) = 1$, $Hamming(\text{“}Indiana\ Jones\text{”},\text{“}Indyana\ Jomes\text{”}) = 2$), or
- its more generalized version that works for any two strings, called *Edit distance*, which measures the minimum number of operations (*additions*, *deletions*, *substitutions*) needed to convert one string value to the another (a.k.a. *Levenshtein distance*).

However, these measures return an absolute “distance” between two strings, which may not always be reliable to decide whether two string represent the same object or not (e.g., $Levenshtein(\text{“}UK\text{'},\text{“}US\text{”}) = 1$ may be much more significant distance than $Levenshtein(\text{“}Unyted\ Kinqdon\text{'},\text{“}United\ Kingdom\text{”}) = 3$). To address this issue,

- *Jaccard similarity* represents another option for the similarity function, and it measures the similarity of two sets (where strings can be seen as sets of characters) relative to the their total size (i.e., $\frac{A \cap B}{A \cup B}$). Going back the example above, we can see that *Jaccard*(“UK’,“US”)=0.33 is comparatively smaller than *Jaccard*(“Unyted Kinqdon’,“United Kingdom”)=0.67, hence in this case Jaccard similarity is more effective as the similarity function.

Finally, regardless of the method used, the resulting similarity value must be converted to a binary (yes/no) decision, which is typically done by establishing a case-specific threshold (e.g., we can consider that if $Jaccard(A,B) \geq 0.5$, then $A \approx B$).

6.3.4.2 Record merge

Once the two records are found to be similar, the next step is to merge and replace them by a single record, that would further represent the real world object. Therefore, we need to employ an effective **merge function** ($\wedge$).

For instance, we may take the *union* of all the values, or we may somehow *combine the values* to make a single one, in which case the approach would depend on the type of the values (e.g., we may establish a rule that a full name should replace a nickname or initials, and a middle initial should be used in place of no middle initial - hence “Susan Williams” and “S. B. Williams” would be combined into “Susan B. Williams”). However, the rule to be used may not always be obvious, especially in the case of misspellings.

The problem becomes even more complex when we consider the complete records, where different fields may use different similarity and merge functions. For instance, if two “similar” records disagree in one field, to merge these records we could choose to *generate null* values for that field, to take the *union of all the values*, or to choose the value depending on the *trustworthiness* of the data source from where the record is coming. Alternatively, we can also use users’ feedbacks to establish the more accurate value (i.e., by means of *crowdsourcing*) [DS15].

6.3.4.3 R-Swoosh algorithm

Lastly, once the *similarity* and *merge* functions are decided and they satisfy certain set of properties[1], we can execute a simple algorithm (*R-Swoosh*, see Listing 6.2) to iteratively merge all the possible records [GUW09].

```
O:= ∅;
while (I<>∅) do {
        pick up r∈I;
        if (∃s∈O so that s≈r){
                I:=I-r; O:=O-s; I:=I∪{r∧s};
        } else {
                I:=I-r; O:=O∪{r};
        }
```

[1] Idempotence, Commutativity, Associativity, Representability (ICAR) properties. See [GUW09](21.7.3) for more details.

}

Listing 6.2: R-Swoosh algorithm

The R-Swoosh algorithm considers at the input a set of records from the underlying data sources (I) and starts with an empty output set O. In each iteration, it takes out one record r from I and it looks for the "similar" record s in the output. If matching is found, r is removed from the I and s from O, and the "merging" of these two records is put back to I so that the matching and merging with other records can continue. Otherwise, the record does not have matching with any record so far and thus it is removed from I and added to the output. Notice that in such way it is guaranteed that the previously merged records will be potentially matched and merged with all the other records from the input. Finally, the algorithm returns a set of all "uniquely" merged records (i.e., those that cannot be further merged with any other records in the output).

Multimedia Materials

Overcoming Heterogeneity (en)

Overcoming Heterogeneity (ca)

Schema Integration (en)

Schema Integration (ca)

Data Integration (en)

Data Integration (ca)

Chapter 7

A (Brief) Introduction to Data Warehousing 2.0

With contributions of Oscar Romero

Data warehousing has been around for about three 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 metadata. 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 metadata 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, transactional data and integrating those 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, transactional 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 are stored and the volume of data are, indeed, very large issues. In 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.

[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].

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

- 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].

Bibliography

[BS16] Carlo Batini and Monica Scannapieco. *Data and Information Quality: Dimensions, Principles and Techniques*. Springer, 2016.

[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.

[DGLLR07] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 133–142, 2007.

[Do] Hong Hai Do. Data conflicts. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*, pages 565–569. Springer.

[DS15] Xin Luna Dong and Divesh Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1):1–198, 2015.

[FH76] Ivan P Fellegi and David Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical association*, 71(353):17–35, 1976.

[FLM+99] Marc Friedman, Alon Y Levy, Todd D Millstein, et al. Navigational plans for data integration. *AAAI/IAAI*, 1999:67–73, 1999.

[FM18] Ariel Fuxman and Renée J. Miller. Schema mapping. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.

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

[GSSC95] Manuel García-Solaco, Fèlix Saltor, and Malú Castellanos. *Semantic Heterogeneity in Multidatabase Systems*, pages 129–202. Prentice Hall International (UK) Ltd., GBR, 1995.

[GUW09] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[Hal01] Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[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.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM, 2002.

[May07] Arkady Maydanchik. *Data quality assessment*. Technics publications, 2007.

[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).

[Sat18] Kai-Uwe Sattler. Data quality dimensions. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*, pages 612–615. Springer, 2018.

[SV03] Alkis Simitsis and Panos Vassiliadis. A methodology for the conceptual modeling of ETL processes. In Johann Eder, Roland T. Mittermeir, and Barbara Pernici, editors, *The 15th Conference on Advanced Information Systems Engineering (CAiSE '03), Klagenfurt/Velden, Austria, 16-20 June, 2003, Workshops Proceedings, Information Systems for a Connected Society*, volume 75 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[SV18] Alkis Simitsis and Panos Vassiliadis. Extraction, Transformation, and Loading. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*, pages 1432–1440. Springer, 2018.

[The17] Vasileios Theodorous. *Automating User-Centered Design of Data-Intensive Processes*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, January 2017.

[TL03] Juan Trujillo and Sergio Luján-Mora. A UML based approach for modeling ETL processes in data warehouses. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *Conceptual Modeling - ER 2003, 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003, Proceedings*, volume 2813 of *Lecture Notes in Computer Science*, pages 307–320. Springer, 2003.

[VZ14] Alejandro A. Vaisman and Esteban Zimányi. *Data Warehouse Systems - Design and Implementation*. Data-Centric Systems and Applications. Springer, 2014.

Appendix A

Acronyms

BI Business Intelligence
CPU Central Process Unit
CSV Comma-separated Values
DAG Directed Acyclic Graph
DB Database
DBMS Database Management System
DDL Data Definition Language (subset of SQL)
DM Data Mart
DML Data Management Language (subset of SQL)
DW Data Warehouse
ELT Extraction, Load and Transformation
ER Entity-Relationship
ERP Enterprise Resource Planning
ETL Extraction, Transformation and Load
ETQ Extraction, Transformation and Query
FASMI Fast Analysis of Shared Multidimensional Information
FK Foreign Key
GUI Graphical User Interface
HOLAP Hybrid OLAP
I/O Input/Output
ISO International Organization for Standardization
JDBC Java Database Connectivity
LHS Left Hand Side
MBR Minimum Bounding Rectangle
MOLAP Multidimensional OLAP
NUMA Non-Uniform Memory Access
ODBC Object Database Connectivity
OLAP On-Line Analytical Processing
OLTP On-Line Transactional Processing
PDI Pentaho Data Integration
PK Primary Key
RDBMS Relational DBMS
RHS Right Hand Side
ROLAP Relational OLAP
SQL Structured Query Language
UML Unified Modelling Language

Appendix B

Glossary of terms

ACID: Transaction model adopted by all RDBMS and some NOSQL. It stands for:

- **A**tomicity: Either all operations inside the transaction are committed, or none of them is executed.
- **C**onsistency: If the database was consistent before the transaction, so it is after it.
- **I**solation: The execution of a transaction does not interfere with the execution of others, and vice-versa.
- **D**urability: If a transaction is committed, its changes in the database cannot be lost.

Access Path/Method: Concrete data structure (a.k.a. index) and algorithm used to retrieve the data required by one of the leaves of the process tree.

Access Plan: Procedural description of the execution of a query. This is the result of query optimization and is typically summarised/depicted in terms of a process tree.

ANSI/SPARC: The Standards Planning And Requirements Committee of the American National Standards Institute defined a three-level architecture in 1975 to abstract users from the physical storage, which is still in use in the current RDBMSs.

Architecture: A system (either hardware or software) architecture is the blueprint of the system that corresponds to its internal structure, typically in terms of independent modules or components and their interactions. It is said to be "functional" if it explicitly shows the functionalities of the system. An architecture can be centralized if the system is expected to run in a single location/machine, or distributed if it includes the components needed to orchestrate the execution in multiple and independent locations/machines. The architecture is said to be parallel if different components can work at the same time (i.e., not necessarily one after another). Distributed and parallel architectures are typically organized in terms of one coordinator component and several workers orchestrated by it.

Block: The transfer unit between the disk and the memory. This is also the smallest piece of disk assigned to a file.

Catalog: Part of the database corresponding to the metadata.

Data (or Database) Model: Set of data structures, constraints and query language used to describe the Database (e.g., Relational, Object-oriented, Semi-structured, Multidimensional).

Data Type: Kind of content stored in an attribute or column (e.g., integer, real, string, date, timestamp). Besides basic data types, we can also find complex (or user-defined) ones (e.g., an structured address which is composed by the string of the street name, the house number, zip code, etc.).

Database: Set of interrelated files or datasets (a.k.a. tables or Relations in the Relational model). According to its use, a database if called "operational" or "transactional" if it is used in the operation of the company (e.g., to register and contact customers, or manage the accounting), or "decisional" if it is used to make decisions (e.g., to create dashboards or predictive models). According to the location of the files, a database can be centralized if they reside in a single machine, or distributed if they reside in many different machines.

Database Management System (DBMS): Software system that manages data. The most popular one are those following the Relational model (a.k.a. RDBMS), or the more modern mixture or Relational and object features (a.k.a. Object-Relational DBMS). A Distributed DBMS (a.k.a. DDBMS) is that able to manage distributed databases. Otherwise, it is said to be centralized.

Entry: Smallest component of indexes, composed by a key and the information associated to it. In B-trees, we find entries in the leaves, while in hash indexes, the entries are in the buckets.

ETL: Data flow in charge of extracting the data from the sources, transforming them (i.e., formatting, normalizing, checking constraints and quality, cleaning, integrating, etc.), and finally loading the result in the DW.

Metadata: Data that describes the user data (e.g., their schema, their data types, their location, their integrity constraints, their creation date, their description, their owner). There are usually stored in the catalog of the DBMS, or in an external, dedicated repository.

Process tree: Representation at the physical level of the access plan of a query.

Relation: Set of tuples representing objects or facts in the real world. They are usually represented as tables and stored into files, but they can also be derived in the form of views. Each tuple then corresponds to a row in the table and a record in the file. Tuples, in turn, are composed by attributes, which correspond to columns in the table and fields in the records of the file. In statistical lingua, a relation represents a dataset containing instances described by features.

Relational Model: Theoretical model of traditional databases consisting of tables with rows (a.k.a. tuples) and columns (a.k.a. attributes), defined by Edgar Codd in 1970. Its main structure is the Relation.

Schema: Description of the data in a database following the corresponding data model (e.g., in the Relational model, we describe the data by giving the relation name, attribute names, data types, primary keys, foreign keys, etc.). We can find schemas representing the same data at the conceptual (i.e., close to human concepts and far from performance issues; e.g., UML), logical (i.e., theoretico-mathematical abstraction; e.g., Relational model) or physical (i.e., concrete data structures used in a DBMS; e.g., indexes and partitions in PostgreSQL) levels.

Selectivity [**Factor**]**):** Percentage of data returned by a query or algebraic operator, with regard to the maximum possible number of elements returned. We say that selectivity is "low" when the query or operator returns most of the elements in the input (i.e., the percentage is high).

Syntactic tree: Abstraction of the process tree in terms or Relational algebra operators.

Transaction: Set of operations over a database that are executed as a whole (even if they affect different files/-datasets/tables). Transactions are a key concept in operational databases.

Transactional: Refers to databases or systems (a.k.a. OLTP) based on transactions, and mainly used in the operation of the business (i.e., not decisional/analytical).