TIS Notes

Original Author: Simone Staffa

Re-edited by: Francesco Sammarco

Disclaimer!!

This document is not intended to be a source of truth to prepare the exam. The materials included here are taken from the official course slides, I'm not responsible for wrong answers or errors reported here (even though I've done my best to provide valid material that regards all the topics that were spoken in the course).

This notes are an extension of this document: Appunti Riassuntivi del Corso

January 16, 2022

Contents

1 Data Integration

1.1 Introduction

Data Integration is the problem of combining data coming from different data sources, providing the user with a unified vision of the data, detecting correspondences between similar concepts that come from different sources, and solving possible conflicts.

The aim of Data Integration is to set up a system where it is possible to query different data sources as if they were a unique one (through a global schema).

1.1.1 Interoperability

Data integration is needed because of a need for interoperability among SW applications, services and information managed by different organizations:

- find information and processing tools, when they are needed, independently of physical location;
- understand and employ the discovered information and tools, no matter what platform supports them, whether local or remote;
- evolve a processing environment for commercial use without being constrained to a single vendor's offerings.

1.1.2 4 Vs of Big Data in data integration

In recent years, the term *4 Vs of Big Data* is used to refer to the four dimensions that summarises the main characteristics of Big Data:

- **Volume** (data at scale): not only can each data source contain a huge volume of data, but also the number of data sources has grown to be in the millions.
- **Velocity** (data in motion): as a direct consequence of the rate at which data is being collected and continuously made available, many of the data sources are very dynamic.
- **Variety** (data in many form): data sources (even in the same domain) are extremely *heterogeneous* both at:
  - the schema level, regarding how they structure their data,
  - and at the instance level, regarding how they describe the same real world entity

  exhibiting considerable variety even for substantially similar entities
- **Veracity** (data uncertainty): data sources (even in the same domain) are of widely differing qualities, with significant differences in the coverage, accuracy and timeliness of data provided. This is consistent with the observation that "1 in 3 business leaders do not trust the information they use to make decisions".

In particular we are interested in:

- **The Variety Dimension**: people and enterprises need to integrate data and the systems that handle those data (relational DBMSs and their extensions containing heterogeneous content)
- **The Veracity Dimension**: data quality is the most general and used term, and represents a number of quality aspects besides veracity:
  - Completeness (essential fields are present)
  - Validity (soundness)
  - Consistency (no contradiction)
  - Timeliness (up-to-date)
  - Accuracy (registered in an accurate way)

These dimensions, together, brought to the rise of a new, difficult problem related to data management: *information overload* represents the difficulty related to decision making and understanding when too much information for such purpose is present in a data storage system.

1.1.3 Heterogeneity

The problem of **heterogeneity** mainly derives from various forms of **autonomy** of those who manage business systems, i.e., designers, decision makers, et similia. We can frame the problem with respect to three points of view:

- **Design** (representation) autonomy: design a dataset in different ways w.r.t. to other systems;
- **Communication** (querying) autonomy: way in which a system query the data, to seek specific information;
- **Execution** (algorithmic) autonomy: each DBMS has its way of extracting data.

On the other hand, there is a **need for interoperability** among software applications, services and information (databases, and others) managed by different organizations that need to reuse legacy applications, existing data repositories (e.g., deep web), and reconcile the different points of view adopted by the various players using the information. This, obviously, is the main reason why such autonomy is a problem: *heterogenity makes interoperability hard to implement.* When heterogeneity is reflected on the integration problem, we can clearly describe the effect on the dimensions discussed before.

VARIETY (HETEROGENEITY)

Variety among several data collections to be used together:

- **Different platforms**: *technological heterogeneity*;
- **Different data models at the participating DBMS**: *model heterogeneity*;
- **Different query languages**: *language heterogeneity*;
- **Different data schemas and different conceptual representations in DBs previously developed**: *schema heterogeneity*;
- **Different values for the same info** (due to errors or to different knowledge): *instance (semantic) heterogeneity.*

VERACITY

Main Data Quality dimensions:

- Completeness
- Validity
- Consistency
- Timeliness
- Accuracy

1.2 The Steps of Data Integration

Data Integration task requires various steps in order to be accomplished:

1. **Schema reconciliation**: mapping the data structure
2. **Record linkage (aka Entity resolution)**: data matching based on the same content, that is understanding which record corresponds to the others in different data sources
3. **Data fusion**: reconciliation of non-identical content, that is deciding which of the two linked record to maintain

This will be further discussed later on. There are two relevant ways of integrating database systems:

1. Use a **materialized data base**, by merging data in a new database.

- Data warehouses are materialized integrated data sources

2. Use a **virtual non-materialized data base**, where data remain at sources, and it may be accessed by producing an interface to expose the unified view of data sources.
    - Enterprise Information Integration Systems (common front-end to the various data sources)
    - Data Exchange (source-to-target)

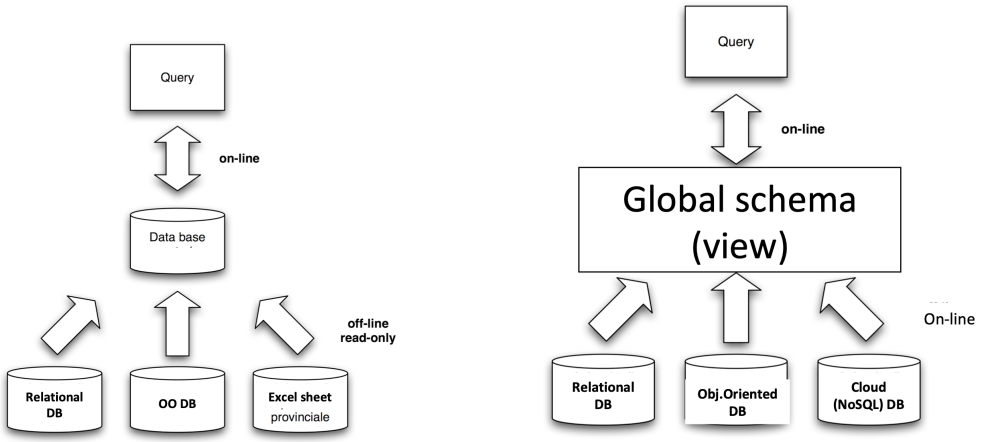


Figure 1: **Materialized Integration** (left): a physical view aggregation different sources over one single structure. Every X days we need to synchronize/update the materialized DB, that is a different physical DB in the system (offline approach).
Virtual Integration (right): virtual view over different sources providing one single structure. No need to install a new physical DB. The view is always up to date (online approach).

MATERIALIZED INTEGRATION

Materialised Integration is an integration technique which is based on the idea of creating a *new, intermediate database* whose role is to store data coming from different sources. This is commonly called a *materialised view*, while the process of creating such view is called *materialisation*. The materialised view is also meant to act as a bridge of a business system, which usually demands query to the data sources, and the data sources themselves: in practice, it caches the query results from the data sources, and arrange them in order to execute the query generated by the main system. One of the most common application of materialised integration occurs in **data warehouses**. They often rely on ad-hoc softwares to access, scrape, transform, and load data into their materialised view, became known as extract, transform, and load (ETL) systems. These softwares are periodically used in data warehouses to modify data based on business needs. Since a materialised view is also an *historical set of records*, these operations are usually performed periodically.
The main reason which may justify a materialised integration approach is the need of analysing and managing data, by developing a progressive, non-volatile historical database.

VIRTUAL INTEGRATION

The **virtual integration** approach leaves the idea of a new database, by instead constructing a *virtual view* which has the only role of transforming the main system query in terms of the data sources. The main difference between virtual and materialised view is that virtual integration *leaves the information in the local sources*. This allows the approach to always return a fresh answer to the main system query. The query posted to the global schema is reformulated into the formats of the local information system. However, the information retrieved needs to be combined to answer the query.
The main reason which may justify a virtual integration approach lies in its ability to guarantee up-to-date query results. Since a virtual view always asks for data at the sources, it allows the creation of an integrated, operational up-to-date database.

1.2.1 Design Steps for Data Integration

Data integration problems arise even in the simplest situation, that is with a unique, centralized database. This particular situation is called *unique db*.
In this simple setting, the ideal situation would be that each datum, whatever application uses it, must only appear once. Adopting an integration approach may eliminate useless redundancies which would cause inconsistencies and useless memory occupation. It would potentially allows to reach a situation in which, starting from a single, centralised DB, each possible branch of a company integrating the informative system will be able to access the data via *procedure views*, i.e., views adapted for a specific purpose.

We study data integration for:

- **Federated DB**: organizations that have been merged together. Typically, their data may be:
  - Homogeneous data: same data model;
  - Heterogeneous data: different data models (structured, unstructured or semi-structured).

They may also be characterised by:

- **Transient, initially unknown** data sources

The techniques used for data integration in this course are applied to the *Multidatabase case*, which consists in the application of the virtual integration technique by creating a *global schema* for unifying the data sources information set and allowing the translation of the main system query to the data sources. The typical design steps for a data integration process in the Multidatabase are:

1. Source schema identification (when present)
2. Source schema reverse engineering (data source conceptual schemata)
3. Conceptual schemata integration and restructuring (related concept identification, conflict analysis and resolution, conceptual schema integration)
4. Conceptual to logical translation of the obtained global schema
5. Mapping between the global logical schema and the single schemata (logical view definition)
6. After integration: query answering through data views

1.3 Conflict Analysis

The first phase of Data Integration is the **conflict analysis**, whose aim is to identify differences between the way data is represented in the data sources.

1.3.1 Conflict Types

- **Name conflicts** (homonymies or synonymies e.g., customer vs client)
- **Type conflicts**: in a single attribute (e.g the attribute gender: Male/Female, M/F, 0/1) or in an entity type (different abstractions of the same real world concept produce different sets of attributes).
- **Data semantics**: different currencies (euros, dollars, etc), different measure systems (kilos vs pounds, centigrades vs Fahrenheit), different granularities (grams, kilos, etc).
- **Structure conflicts**: e.g. the entity Person is extended by entities Male and Female, or entity Person has an attribute "gender" (a concept is an attribute in a source and an entity in another one).
- **Cardinality conflicts** (e.g. a movie may have just one director in a source, while another source allows more directors).
- **Key conflicts**: two same entities have two different keys (e.g. in a source the person is identified by the SSN and in another source it is identified through the email address).

1.4 Mapping between the Global Logical Schema and the Single Source Schemata (logical view definition)

We properly define a data integration system as a triple (G, S, M), where G is the schema, S are the data sources and M are the mappings. **Schema mapping** is the process of creating a link between global schema and data sources, in order to allow the retrieval of data from the data sources according to the query. The scope is *minimising the amount of data lost*. There are two approaches for schema mapping:

- **GAV** (Global As View): a set of assertion over all the elements of the global schema is created, in the form $g \rightarrow q_s$, where q_s is a query to the data source $s \in S$. In other words, the global schema is derived from the integration process of the data source schemata, thus the global schema is expressed in terms of the data source schemata. This approach is appropriate for stable data sources, it is in fact difficult to extend with a new data source. Mapping quality depends on how well we have compiled the sources into the global schema through the mapping. Whenever a source changes or a new one is added, the global schema needs to be reconsidered. The main advantage is that query processing is extremely effective, since each element of the global schema has a direct translation in terms of the data source.
- **LAV** (Local As View): a set of assertion over all the elements of the data sources is created, in the form $s \rightarrow q_g$, where q_g is a view over the global schema G. The global schema is designed independently of the data source schemata. The relationship (mapping) between sources and global schema is obtained by defining each data source as a view over the global schema. This approach is appropriate if the global schema is unstable, i.e., if data sources may be added in the future, as it favors extensibility. By the way, query processing is much more complex, involving reasoning (we have no explicit connection between elements of the global schema and elements of the data sources). In fact, mapping, in this case, specifies the opposite transformation of the one needed. Mapping quality depends on how well we have characterized the sources.

A third approach is the union of the previous two:

- **GLAV** (Global and Local As View): the relationship (mapping) between sources and global schema is obtained by defining a set of views, some over the global schema and some over the data sources.

The most useful integration operators to write relational GAV views are:

- **union**
- **outerunion**: used with different schemas, putting null all the information we don't have. There are all the attributes of the tow sources (without repetitions)
- **outerjoin**: it doesn't admit different values for 1 attribute
- **generalization**: we keep only common attributes

NOT IN THE 2021/22 COURSE A mapping defined over some data source is **sound** when it provides a subset of the data that is available in the data source that corresponds to the definition.
A mapping is **complete** if it provides a superset of the available data in the data source that corresponds to the definition.
A mapping is **exact** if it provides all and only data corresponding to the definition: it is both sound and complete.
With the GAV approach, the mapping can be exact or only sound.
With the LAV approach, the mapping can be exact or only complete, due to the incompleteness of one or more sources (they do not cover the data "expected" from the global schema, which has been defined independently of the source contents). END

1.5 Inconsistencies in the Data

At query processing time, when a real world object is represented by instances in different databases, they may have different values. This problem is dealt via two phases of the Integration process:

- **Record Linkage (aka Entity Resolution)**: finding the info that refer to same real-world entities;
- **Data Fusion**: once recognized that two items refer to the same entity, the act of reconciling information which are thought to be common.

1.5.1 Record Linkage (aka Entity Resolution)

Whatever the data model, we have to recognize when two datasets contain the same information. Considering the case of a relational database (the most common), several techniques can be used:

- **String matching**: concatenate multiple columns into a string, then compare the two strings. Not suggested!
- **Tuple (or structured-data) matching**: compare records field by field. It is much easier to spot similarities, since we can also associate different meanings to the columns.
- **String similarity and similarity measures**: given two sets of strings, find all pairs from the two sets that refer to the same real-world entity. Each of these pairs is called a match.
  Types of similarity measures:
  - Sequence-based:
    - *edit-distance*: based on the minimal number of operations that are needed to transform string a to string b

  - Set-based:
    * *Jaccard*: divide the strings into tokens, and compute the measure on the two sets of tokens (intersection divided by union of tokens)
  - Phonetic:
    * *Soundex*: calculates a four-character code from a word based on the pronunciation and considers two words as similar if their codes are equal. Similar sounding letters are assigned the same soundex code. Note that such techinque is strongly language-based and may fail in some cases (for example, same pronunciation words...).

- **Record Matching:**
  - *Rule-based matching*: manually written rules that specify when two tuples match (e.g., two tuples refer to the same person if they have the same SSN)
  - *Learning Matching Rules*:
    * Supervised: learn how to match from training data, then apply it to match new tuple pairs (requires a lot of training data)
    * Unsupervised: clusterizing records based on similar values
  - *Probabilistic Matching*: model the matching domain using a probability distribution. Provides a principled framework that can naturally incorporate a variety of domain knowledge. By the way, it is computationally expensive and hard to understand fully.

1.5.2 Data Fusion

Once you have understood that some data clearly represent the same entity, you still have to cope with the problem of what to do when other parts of the info do not match.

Inconsistency may depend on different reasons:

- One (or both) the sources are incorrect;
- Each source has a correct but partial view, e.g., databases from different workplaces (salary is the sum of the two);
- Often the correct value may be obtained as a function of the original ones.

2 Data Integration in case of Data Model Heterogeneity

Data Model Heterogeneity refers to the situation in which we are dealing with an organisation of data based on different models. This difference arises when the scope of such organisation is to preserve some characteristics of the data, or to accommodate specific business purposes.

Design steps for data integration in case of data model heterogeneity:

1. Reverse engineering (production of the conceptual schema) of the different model sources;
2. Conceptual schemata integration;
3. Choice of the target logical data model and translation of the global conceptual schema;
4. Definition of the language translation (wrapping);
5. Definition of the data views.

Among this different steps, one of particular interest is language translation. Regarding such task, **wrappers** are the elements that allows its full applicability. Wrappers role is to convert queries into queries/commands which are understandable for the specific data source. They convert query results from the source format to a format which is understandable for the application. Wrappers are useful (and enough) if the data represented by the different data models is structured.

2.1 Semi-structured Data Integration

For **semi-structured data** we intend data which does not obey to a fixed structure. This does not mean that there is not any form of structure, but it is not as prescriptive, regular and complete as in traditional DBMSs. Typical examples of semi-structured data are:

- texts;
- trees;
- graphs.

They are all different and do no lend themselves to easy integration.

We would like to:

- integrate
- query
- compare

data with different structures also with semi-structured data, as if they were all structured. An overall data representation should be progressively built, as we discover and explore new information sources.

2.2 Mediators

A **mediator** has the same purpose as the integration systems. Mediators are interfaces specialized in a certain domain which stand between application and wrappers. They accept queries written in the application's language, decompose them and send them to each specific wrapper. They also send the responses back to the application, providing a unified vision of data. In this sense, a mediator acts as an intermediate element between the application and the wrappers, in case of semi-structured data.
The term mediation includes:

- the **processing** needed to make the interface work
- the **knowledge structures** that drive the transformations needed to transform data to information
- any **intermediate storage** that is needed

2.2.1 Tsimmis

TSIMMIS is the first system based on the mediator/wrapper paradigm, proposed in the 90's at Stanford.
In this system:

- unique, graph-based data model
- data model managed by the mediator
- wrappers for the model-to-model translations
- query posed to the mediator
- mediator knows the semantics of the application domain

In TSIMMIS, queries are posed to the mediators in an object-oriented language **LOREL** (Lightweight Object REpository Language). The data model adopted is OEM (Object Exchange Model), a graph-based and self-descriptive model, and it represents directly data with no schema at all. In OEM, each object appears in a *nested structure*, which allows to express subsumption relationships or properties of the objects.

2.3 Complications in Data Integration task with structured or unstructured data

Even though methods for data integration have been described, we may still encounter difficulties when dealing with semi-structured or un-structured data. When a mediator is used, it is typically created by specializing it into a certain domain. Thus, each mediator must know domain metadata which convey the data semantics. If data source changes a little, the wrapper has to be modified. This is mostly because wrappers act based on *extraction rules*, which may be inappropriate in case their input changes.

This has led, in recent year, toward the research of the so-called **automatic wrappers**. Automatic wrappers are wrappers that may act in dynamic environments which, although changing, always present some regularities in the data. This situation typically occurs in *data intensive* web sites.

The **Road Runner Project** is one of the most famous example of automatic wrapper generation based data integration process. It is based on the *page class*, a collection of pages generated by some script from a common dataset. It acts by finding the underlying dataset structure from a pool of sample HTML pages.

2.4 Ontologies

The complications previously described pose a complex problem: the only way of automatically integrating different model data sources, when semi-structured or unstructured data occurs, is through automatic wrapper generators. However, they are not always applicable, as analysed.

One of the concept that gained a lot of popularity, also in the integration context, is the **ontology**.

An **ontology**:

- is a formal specification of a conceptualization of a shared knowledge domain.
- is a formal and shared definition of a vocabulary of terms and their inter-relationships.
- is a controlled vocabulary that describes objects and the relationships between them in a formal way.
- it has a grammar for using the terms to express something meaningful within a specific domain of interest

Predefined relations:

- synonimy
- homonymy
- hyponymy

N.B. An ER diagram, a class diagram, any conceptual schema is an ontology.

The notable properties of an ontology are also the basis of the scope it should serve:

- It should enforce a **formal specification**, which allows for use of a common vocabulary for automatic knowledge sharing;
- It is **shared**, so captures knowledge which is common (there should be a consensus);
- It ensures **conceptualization**: gives a unique meaning to the terms that define the knowledge about a given domain.

We distinguish between different **Ontology Types**:

- **Taxonomic ontologies**: definition of concepts through terms, their hierarchical organization, and additional (predefined) relationships;
- **Descriptive ontologies**: definition of concepts for alignment of existing data structures or to design new specialized ontologies (domain ontologies). Descriptive ontologies require rich models to enable representations close to human perception.

An ontology consists of:

- **concepts**: generic concepts (express general world categories), specific concepts (describe a particular application domain, domain ontologies)
- **concept definition** (in natural language or via a formal language)
- **relationships between concepts**: taxonomies, user-defined associations, synonymies,...

A formal definition of an ontology identifies it as a quadruple (C, R, I, A), where:

- C is the set of concepts;
- R is the set of relations between concepts;
- A is the set of axioms;
- I is the instance collection, i.e., the set of objects that are stored in the information source and belongs to a specific class or concept.

An ontology is composed by:

- a **T-Box**: contains all the concept and role definitions, and also contain all the axioms of our logical theory (e.g., "A father is a Man with a Child").
- an **A-Box**: contains all the basic assertions of the logical theory (e.g., "Tom is a father" is represented as Father(Tom)).

2.4.1 OpenCyc

OpenCyc is one of the most notable example of general-purpose ontology, i.e., an ontology whose aim is to the describe how the world works. It is the realisation of the human consensus property of an ontology.
It starts from top level concepts, which progressively specialises in subcategories, expressed through relations between each possible level.

2.5 Semantic Web

Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. It is built on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data.

Linked Data consists of connecting datasets across the Web. It describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP, RDF and URIs, but extends them to share information in a way that can be read automatically by computers, enabling data from different sources to be connectd and queried.

Open data is data uploaded to the Web and accessible to all.
Linked Open data extend the web with a data commons by publishing various open datasets as RDF on the Web and by setting RDF links among them.

2.5.1 RDF Language

RDF may be defined as a instrument for encoding, exchanging and using metadata between applications, ensuring interoperability when information are shared in the web.
At the core of RDF is the subtle notion of a triple **subject-predicate-object**, which may also be represented graphically (the first one connected to the last one through the middle one). The connection between a subject and an object is expressed in term of a predicate, which enforce a link with a specific meaning. For example, if we want to represent a person, we may use as a subject the SSN and link it to the object it want to represent, i.e. a person, and to the name associated to the SSN, for example "John James". The two predicates are then one which express the meaning of the subject and the association of the subject.

2.5.2 OWL

OWL is the first level above RDF. It is an ontology language that can formally describe the meaning of terminology used in Web documents, going beyond the basic semantics of RDF schema.

- **XML**: provides a syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- **XML Schema**: is a language for restricting the structure of XML documents and also extends XML with data types.
- **RDF**: is a data model for objects (resources) and relations between them, provides a simple semantic for this data model, and can be represented in an XML syntax.
- **RDF Schema**: is a vocabulary for describing properties and classes of RDF resources

OWL adds more vocabulary for describing properties and classes. The OWL (Web Ontology Language) is designed for use by applications that need to process the content of information instead of just presenting information to humans. It facilitates greater machine interpretability of Web content than that supported by XML, RDF and RDF Schema by providing additional vocabulary along with a formal semantics.
It has three increasingly-expressive sublanguages:

- OWL Lite: supports users primarily needing a classification hierarchy and simple constraints. It has a lower formal complexity than OWL DL.
- OWL DL: supports users who want maximum expressiveness while all conclusions are guaranteed to be computed (computational completeness) and all computations will finish in a finite time (decidability).
- OWL Full: meant for users who want maximum expressiveness and the syntactic freedom of RDF: no computational guarantees.

2.6 Reasoning Services

Services for the T-Box:

- **Subsumption**: verifies if a concept C subsumes (is a subconcept of) another concept D
- **Consistency**: verifies that there exists at least one interpretation which satisfies the given Tbox
- **Local Satisfiability**: verifies for a given concept that there exists at least one interpretation in which it is true.

Services for the A-Box:

- **Consistency**: verifies that an A-Box is consistent w.r.t a given Tbox
- **Instance Checking**: verifies if a given individual belongs to a particular concept
- **Instance retrieval**: returns the extension of a given concept C, that is the set of individuals belonging to C.

2.7 Ontologies and Integration Problems

A database and an ontology are not distant concepts: they serve, instead, different purposes. It is possible to see that an ontology may be translated to an ER and viceversa, but this implies also a *degradation* in terms of the properties of the other. For example, an ontology cannot represent complex data structure, but can surely define concept, a task that a database may not accomplish. For this reason, ontology and database may be seen as *complementary* concept. How should we improve database conceptual models to fulfill ontology requirements?

- Supporting defined concepts (views) and adding the necessary reasoning mechanisms
- Managing missing and incomplete information: explicit semantic differences between the two (referring to facts)
  - Closed World Assumption → whatever is not in the database is FALSE
  - Open World Assumption → if an information is missing it is possible that is TRUE or FALSE

Problems:

- Discovery of equivalent concepts (mapping): what does equivalent mean? (we look for some kind of similarity)
- Formal representation of these mappings: how are these mapping represented?
- Reasoning of these mappings: how do we use the mappings within our reasoning and query-answering process?

Ontology matching is the process of finding pairs of resources coming from different ontologies which can be considered equal in meaning. We need some kind of *similarity measure*, this time taking into account also semantics (i.e., not only the structure of words).
As already seen, similarity is strictly related to distance. Three main categories:

- Metric (distance)-based measures
- Set-theoretic based measures
- Implicators based measures

Ontology mapping is the process of relating similar concepts or relations of two or more information sources using equivalence relations or order relations.

Reasons for ontology mismatches:

- Scope: two classes seem to represent the same concept but do not have exactly the same instances

- Model coverage and granularity: a mismatch in the part of the domain that is covered by the ontology, or the level of detail to which that domain is modeled (e.g., ontology for all the animals vs. ontology for birds)
- Paradigm: different paradigms can be used to represent concepts such as time (e.g., temporal representations → continuous interval vs. discrete sets of time points
- Encoding
- Concept description: a distinction between two classes can be modeled using a qualifying attribute or by introducing a separate class, or the way in which is-a hierarchy is built.
- Homonyms and Synonyms

2.7.1 Ontologies support to integration

An ontology may, in principle, serve as an integration support tool. The type of aid an ontology may represent is divided in four classes:

- **An ontology can be a schema integration support tool**. Ontologies are used to represent the semantics of schema elements (if the schema exists);
- **An ontology can be used instead of a global schema**:
  - schema-level representation only in terms of ontologies
  - ontology mapping, merging etc. instead of schema integration
  - integrated ontology used as a schema for querying

  Data source heterogeneity is solved by extracting the semantics in an ontological format. Data source ontologies are mapped to the Domain Ontology (Global Schema). This allows to solve problem like *impedance mismatch* and *unstructured data source management*.

- **An ontology as a support for content interpretation and wrapping** (e.g., HTML pages).
- **An ontology as a mediation support tool for content inconsistency detection and resolution** (record linkage and data fusion).

When we use ontologies to interact with databases we have to take care of:

- Transformation of ontological query into the language of the datasource, and the other way round
- Different semantics (CWA vs. OWA)
- What has to be processed where (e.g., push of the relational operators to the relational engine)

3 New trends in Data Integration Research and Development

3.1 Uncertainty in Data Integration

Databases are assumed to represent certain data (a tuple in the database is true). But real life is not as certain. Uncertain databases attempt to model uncertain data and to answer queries in an uncertain world.
Moreover:

- Data itself may be uncertain (e.g. extracted from an unreliable source);
- Mappings might be approximate
- Reconciliation is approximate
- Imprecise queries are approximate

Whatever the semantics of uncertainty (e.g, fuzzy or probabilistic..) an uncertain database describes a set of possible worlds.
Example: assign each tuple a probability. Then the probability of a possible world is the product of the probabilities for the tuples.

3.1.1 Entity Resolution

Entity Resolution (ER) is the problem of accurately identifying multiple, differing and possibly contradicting representations of unique real-world entities in data.
The causes of multiple representations of the same real-world entity are:

- Noise (small errors)
- Inconsistencies (conflicting or different values for properties/relations of an object The record matching rules are used to detect duplicates in data.

A common approach is to use **record-matching rules**. Record-mathing rules are constraints that state: *"if any two records are similar on properties P1,...Pn, then they refer to the same entity".*

Problems of this approach:

- it is difficult to generate automatically record-matching rules
- one-to-one record matching is too expensive (quadratic)

3.1.2 Data Provenance

Sometimes knowing where the data have come from and how they were produced is critical (e.g., an information extractor might be unreliable, or one data source is more authoritative than others).
Provenance of a data item records "where it comes from":

- Who created it
- When it was created
- How it was created (as value in a database, as the result of a computation, coming from sensor, etc ...)

Two viewpoints on Provenance (even though they are perfectly equivalent):

- **Provenance as annotations on data**: models provenance as a series of annotations describing how each data item was produced. These annotations can be associated with tuples or values
- **Provenance as a graph of data relationships**: models provenance as a graph, with tuples as vertices. Each possible direct derivation of a tuple from a set of source tuples is a edge-connecting the source and derived tuples.

Provenance is achieved via various techniques. In principle, determining provenance of data requires: giving a suitable explanation of the nature of the data, scoring the source and the data for assessing its quality and evaluating the influence of a source with respect to another.

These steps may be complex to perform manually; at the same time, they may not be automated. **Crowdsourcing** provide a cheap and scalable way of annotating provenance of data: a set of people is rewarded to annotate provenance of data.

3.2 Building Large-Scale Structured Web Databases

Building a large-scale structured web database is a subtle task. It poses new problems with respect to the one discussed, which may be summarised in this objective: develop methodologies for designing a fully integrated database coming from heterogeneous data sources. However, depending on the type of system we aim to integrate, we may also simplify the steps required for dealing with these problems.

3.2.1 Lightweight Integration

We may need to integrate data from multiple sources to answer a question asked once or twice. The integration must be done quickly and by people without technical expertise. This problem is defined ad **lightweight integration**, as it is a small-scale integration process with prioritise speed over "stability" (a solid and long-lasting implementation).
Problems typical of lightweight data integration:

- Locating relevant data sources
- Assessing source quality
- Helping the user understand the semantics
- Support the process of integration

3.2.2 Mashup

Mashup is a paradigm for lightweight integration.
It is an application that integrates two or more mashup components at any of the application layers (data, application logic, presentation layer) possibly putting them in communication with each other.
Key elements:

- **Mashup component**: is any piece of data, application logic and/or user interface that can be reused and that is accessible either locally or remotely (e.g., Craiglist and GoogleMaps)
- **Mashup logic**: is the internal logic of operation of a mashup; it specifies the invocation of components, the control flow, the data flow, the data transformations and the UI of the mashup.

Mashups introduce integration at the presentation layer and typically focus on non-mission-critical applications.

BENEFITS AND PROBLEMS

Benefits:

- Easy development of situational applications for power users
- Fast prototyping for developers
- Increased ROI for Service-Oriented-Application investments

Problems:

- Mashup development is complex, due to the need of integrating different elements from different sources.

Luckily, mashups typically work on the ”surface”.

- Reuse of existing components
- Composition of the outputs of software systems

The work of developers can be facilitated by suitable abstractions, component technologies, development paradigms and enabling tools.

A mashup generally differs from the other integration practices as it also lies on the logic and presentation layer (it apply integration also with respect on *how* the user perceive the informative contribute of different sources), focusing on non-mission-critical application. Consider, in fact, that a mashup *may not be data related*!

TYPES OF MASHUPS

- **Data mashups**: retrieve data from different resources, process them and return an integrated result set. Data mashups are a Web-based, lightweight form of data integration, intended to solve different problems.
- **Logic mashups**: integrate functionality published by logic or data components. The output is a process that manages the components, in turn published as a logical component.
- **User interface mashups**: combine the component's native UIs into an integrated UI. The output is a Web application the user can interact with. It is mostly client-side, generally short-lived.
- **Hybrid mashups**: span multiple layers of the application stack, bringing together different types of components inside one and a same application.

Data mashups are the lightweight form of data integration, which serve a different purpose with respect to normal data integration techniques: it is specific for ad-hoc data analyses, it's simple and not suitable for critical tasks.

3.2.3 Pay-as-you-go management

In contrast the other kind of data integration, **pay-as-you-go systems** try to avoid the need for the initial setup phase that includes creating the mediated schema and source descriptions. The goal is to offer a system, that provides useful services on a collection of heterogeneous data with very little initial effort.

3.3 Data Spaces

A Database Management System (DBMS) is a generic repository for the storage and querying of structured data. Unfortunately in data management scenarios today it is rarely the case that all the data can be fit nicely into a conventional relational DBMS, or into any other single data model or system. Moreover, they often require a *semantic integration* phase beforehand, thus demanding additional work.
One solution is using a **dataspace**: it does not represent a data integration approach, rather, it is more of a data coexistence approach. The goal of dataspace support is to provide base functionality over all data sources, regardless of how integrated they are. Moreover, this is simplified as a dataspace tries to leverage pre-existing mapping and matching generation techniques.

Note that, although Dataspace represents a valid solution to the problem stated before, it does not guarantee the same benefits of a classical integration system. In fact, it does not fully enjoy of durability and consistency of data.

NOT IN THE 2022 COURSE

The required techniques for actually deploying Dataspaces are and integrates:

- Schema mapping
- Uniform search over multiple types of data
- Combining structured, semi-structured and unstructured data
- Approximate query processing
- Managing and querying uncertain data and its lineage
- Stream and sensor data management and processing.

A Dataspace is modeled as a set of **Participants** and **Relationships**. The participants in a data space are the individual data sources (relational databases, XML repositories, text databases..). Some participants may support expressive query languages, while others are opaque and offer only limited interfaces for posing queries. A dataspace should be able to model any kind of relationship between two or more participants. Relationships may be stored as query transformations, dependency graphs, or sometimes even textual descriptions. The catalog contains information about all the participants in the data space and relationships among them. Users should be able to query any data item regardless of its format or data model. END

3.3.1 Data Lakes

The **Data Lake** represents a practical application of the concept of dataspace. A data lake is a big collection of (mostly unstructured) data which is stored exclusively in a raw format. In contrast to a database, a data lake does not rely on a schema, and accepts a great variety of data types. Moreover, it lacks of ETL processes.

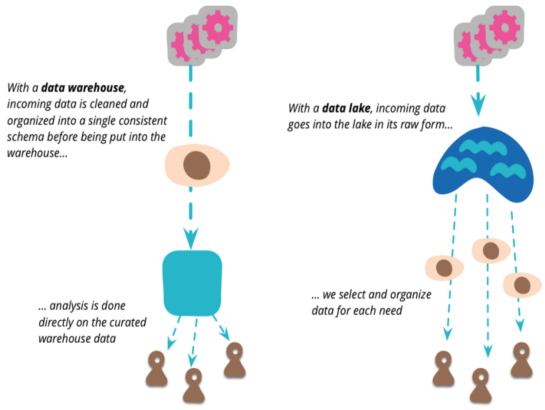
A data lake presents a series of advantages over a traditional database:

- It allows to deal with big-scale data volumes, at a relative small cost;
- Preserving the data "as it is" allow to fully preserve its analysis potential, favouring machine learning purposes (profiling, analytics...);
- It allows wider opportunities of "specialisation", as data in its original format may furtherly be processed for further needs.

However, one should also note the great disavantage of data lakes:

- Lack of organisation in terms of data annotation and processing;
- Redundancies in data;
- No consistency guarantees;
- Stricter requirements in terms of computational power when searching for a specific information (scan required);
- Requires processing method when data should be used.

Although presenting problems, data lakes are, in principle, dataspaces: therefore, the disavantages presented may be mitigated by on demand data integration techniques.



Data Warehouse	Dataspace
Usually Relational DBMS	Often HDFS/Hadoop
Data of recognized high value	Candidate data of potential value
Aggregated data	Detailed source data
Data conforms to enterprise standards	Fidelity to original format and condition (transformed/cleaned later, on demand, when needed)
Known entities	Raw material (metadata) to discover entities
Data integration up front	Data integration on demand
Schema on write	Schema on read (when doing analytics)

4 Data Analysis and Exploration

4.1 Analysis of Data

Data Analysis is a process of inspecting, cleaning, transforming and modeling data with the goal of highlighting useful information, suggesting conclusions, and supporting decision making.

Data exploration is a preliminary exploration of the data to better understand its characteristics.

Data mining is a particular data analysis technique that focuses on modeling and knowledge discovery for predictive rather than purely descriptive purposes.

Machine Learning is a field of study that gives computers the ability to learn without being explicit programmed. A computer program is said to learn from experience E w.r.t some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Why analyze data?

- Often information is hidden in the data but not readily evident
- Human analysts may take weeks to discover useful information
- Lots of commercial data is being collected and warehouse
- Computers have become cheaper and more powerful
- Data is collected and stored at enormous speeds (GB/hour)
- Data mining may help scientists in classifying and segmenting data

4.2 Data Exploration

Data Exploration is a preliminary exploration of the data to better understand its characteristics.
The key motivations of data exploration include:

- Helping to select right tool for preprocessing or analysis
- Making use of humans abilities to recognize patterns

Basic Traditional techniques of data exploration:

- **Summary statistics**: are numbers that summarize properties of the data, such as frequency (the frequency of an attribute value is the percentage of times the value occurs in the data set). Most summary statistics can be calculated in a single pass through the data.
  - Mean: the most common measure of the location of an ordered set of points. However, it is very sensitive to outliers.

  - Median: also commonly used (p-50 percentile)
  - Range: the difference between the max and the min
  - Variance and Standard Deviation: most common measures of the spread of a set of points. This is also sensitive to outliers.
  - For continuous data, is useful to know the notion of *percentile*.
    Given a continuous attribute x and a number p between 0 and 100, the p-th percentile is a value x_p of x such that p% of the observed values of x are less than x_p

- **Visualization**: is the conversion of data into a visual or tabular format so that the characteristics of the data and the relationships among data items or attributes can be analyzed or reported.

  - Human have a well developed ability to analyze large amounts of information that is presented visually
  - Can detect general patterns and trends
  - Can detect outliers and unusual patterns

  Selection is the elimination of certain objects and attributes. It may involve choosing a subset of attributes.

  - *Dimensionality Reduction* is often used to reduce the number of dimensions to two or three

  Visualization Techniques:

  - Histograms:
    * usually shows the distribution of values of a single variable
    * divide the values into bins and show a bar plot of the number of objects in each bin
    * the height of each bar indicates the number of objects
  - Box Plots: it allows to visualise the locality, the spread and the skewness of the data
    * displays the distribution of data (over percentiles)
    * can be used to compare attributes
    * allows to identify outliers

4.3 What is Data Mining

Data Mining is the non-trivial extraction of implicit, previously unknown and potentially useful information from data. It is the automatic (or semi-automatic) exploration and analysis of a large quantities of data in order to discover meaningful patterns.
The Data Mining Tasks are:

- **Predictive methods**: use some variables to predict unknown or future values of other variables. Related to *supervised learning*, which consists in inferring a function from labeled training data. The most used method is classification.
- **Descriptive methods**: find human-interpretable patterns that describe the data. Related to *unsupervised learning*, where a machine learning algorithm is used to draw inferences from datasets consisting of input data without labeled responses. The most used method is cluster analysis, often used for exploratory data analysis to find hidden patterns of grouping in data.

4.3.1 Methods

- **Classification** (predictive): given a collection of records, each record contains a set of attributes, one of which is the class. The goal is to find a model for the "class" attribute as a function of the values of other attributes, in order to assign a class to a previously unseen record as accurately as possible. The accuracy of the model is then evaluated over a set of unseen records called test set.
  (Examples: predict fraudolent cases in credit card transactions)

- **Clustering** (descriptive): tries to divide data points into cluster such that data points in one cluster are more similar to one another and data points in separate clusters are less similar to one another.
  (Examples: market segmentation into distinct subsets of customers, find group of documents that are similar to each other based on the important terms appearing in them).

- **Association Rule Discovery**: given a set of records each of which contains some number of items from a given collection, called itemsets, produce dependency rules which will predict occurrence of an item based on occurrences of other item. (e.g., If a customer buys diaper and milk, then she is very likely to buy beer). (Examples: marketing and sales promotions).
  In the itemset theory, we distinguish between these concepts:

  - the support count is the number of occurrences of an item set in a list of transactions (e.g., 2)
  - the support is a fraction of transactions that contain an item set (e.g., 2/5)
  - the frequent itemset is an item whose support is greater than or equal to a minsup threshold.
  - an association rule is an implication expression of the form X $\rightarrow$ Y, where X and Y are itemsets.

  The association rule is the final product of an association rule discovery task.

- **Sequential pattern discovery** (descriptive): it seeks a pattern in a set of data, which in practice is described in terms of subsequences. In the sequence theory, we distinguish:

  - a sequence, which is an ordered list of elements (transactions)
  - an element, which is part of a sequence and contains a collection of events (items)
  - Example:
    * Sequence: purchase history of a given customer
    * Element: a set of items bought by a customer at time t
    * Event: Books, diary products, CDs, etc.
  - a subsequence, which is a sequence which is contained in another sequence, i.e., a sequence such that each element contained in it is also contained in the first one, assuming the order of appearance is preserved in the subsequence.
  - the support of a subsequence w, defined as the *fraction of data sequences* that contain w at least once
  - a sequential pattern, which is a frequent subsequence
    (i.e., a subsequence whose support is $>= minsup$)
  - Sequential Pattern Mining:
    * given a database of sequences and a user-specific minimum support threshold (minsup)
    * find all subsequences with support $>= minsup$

- **Regression** (predictive): predict a value of a given continuous valued variable based on the values of other variables, assuming a linear or nonlinear model of dependency.
  (Examples: Predicting sales amounts of a new product based on advertising expenditure; predicting wind velocities as a function of temperature, humidity, etc.)

- **Anomaly/Outlier Detection** (predictive):

  - Anomalies/outliers are the set of data points that are considerably different from the remainder of the data.
  - The first step is to build a profile of the normal behavior, then use the normal profile to detect anomalies.
  - Applications:
    * credit card fraud detection
    * fault detection
    * network intrusion detection

4.3.2 Challenges of Data Mining

- Scalability
- Dimensionality (reduction)
- Complex and Heterogeneous Data
- Data Quality
- Data Distribution
- Privacy Detection
- Streaming Data

5 Data Warehouse

5.1 What is a Data Warehouse

- **As a dataset**: decision support database maintained separately from the organization's operational database. A Data Warehouse is a single, complete and constant store of data obtained from a variety of different sources made available to end users, so that they can understand and use it in a business context.
- **As a process**: technique for assembling data from various sources with the purpose of answering business questions. A Data Warehouse is a process for transforming data into information and for making it available to users in a timely enough manner to make a difference.

A data warehouse is a

- subject-oriented,
- integrated,
- time-varying,
- non-volatile

collection of data that is used primarily in **organizational decision making**.
Data Warehouses (DWs) are very large databases (from Terabytes: 10^{12} bytes, to Zottabytes: 10^{24} bytes).

A DW is a data collection built to support decision-making processes. It must allow analytical queries useful for the management of the company. It is usually built starting from several data sources and we update it only periodically. It contains synthetic and aggregated data. In a dynamic environment, one must perform periodically, building up a history of the enterprise. The main purpose of a data warehouse is to allow systematic or ad-hoc data analysis and mining.

DW is a specialized DB (relational), with different kind of operations:

- ***Warehouse (OLAP)*** ↔ *Standard Transactional DB (OLTP)*
- **Mostly reads** ↔ Mostly updates
- **Queries are long and complex** ↔ Many small transactions
- **Lots of scan** ↔ Index/hash on primary key
- **Summarized, reconciled data** ↔ Raw data
- **Hundreds of users** ↔ Thousands of users
- **Relational Redundancy-preserving** ↔ **Normalised Relational Database**

Where is a DW useful?

- Commerce: sales and complaints analysis, client fidelization, shipping and stock control
- Financial services: risk and credit card analysis, fraud detection
- Telecommunications: call flow analysis
- Healthcare structures: patients' ingoing and outgoing flows, cost analysis.

A Data Warehouse-based system may be divided in different parts:

- the Data Sources;
- the Data Warehouse, which may be further divided into several Data Marts, a structure used to retrieve data from the DW. It is a subset of the data warehouse and is usually oriented to a specific business area;
- the Analysis Tool System, an abstraction of the end-system which will use the data stored in the DW.

It is important to note that, although a Data Warehouse is most of the time realised as a relational database, it is not replaceable in terms of cloud-based data stores. In fact, a Data Warehouse is, in principle, a *database organisation paradigm*, in the sense that it is a way of storing pre-processed data beforehand, for a specific business purpose.

5.1.1 Evolution of Data Warehouses

- **Offline DW**: periodically updated from data in the operational systems and the DW data are stored in a data structure designed to facilitate reporting
- **Online DW**: data in the warehouse is updated for every transactions performed on the source data (e.g. by triggers)
- **Integrated DW**: data assembled from different data sources, so users can look up the information they need across other systems.

5.2 Data Model for OLAP

Data Warehouses require a new type of representation for data, in order to accommodate business needs. The most appropriate data model is the **data cube**, a data model whose main characteristics is its ability to represent a piece of information with respect to various **dimension**. A dimension may be defined as a notable piece of information which is able to describe an event with respect to a specific point of view (example: an event like a sale may be described with respect to the type of purchased product).

In the cube, dimensions are the search keys and cube cells contain metric values (measures of the business). The data cube is a multidimensional array of values, used to represent data (facts) based on some measure of interest. It is useful to analyze data from different perspectives.

The dimensional fact model allows one to describe a set of **fact schemata**. A fact schemata is the formal representation of elements that contributes to describe a specific business event.
The components of a fact schema(ta) are:

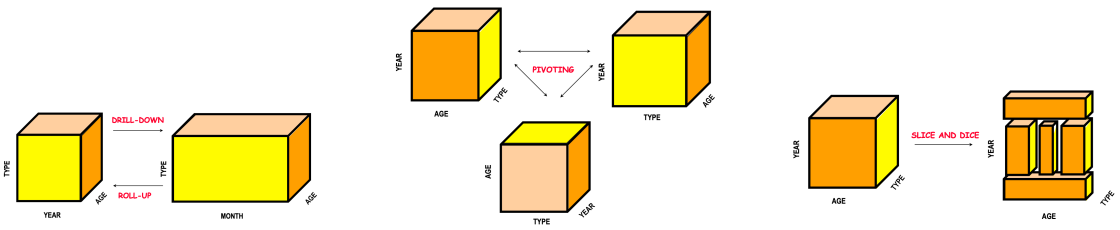
- **Facts**: concepts that are relevant for the decisional process. Typically they model a set of events of the organization;
- **Measures**: a numerical property of a fact, which contributes to describe one of its properties;
- **Dimensions**: a fact property defined w.r.t a finite domain. It describes an analysis coordinate for the fact.
- **Dimension Hierarchy**: is a directional tree whose
  - nodes are dimensional attributes
  - edges describe n:1 associations between pairs of dimensional attributes

  - root is the considered dimension

- Example:
  - Store chain:
    * Fact: sales
    * Measures: sold quantity, gross income
    * Dimensions: product, time, zone

5.2.1 OLAP Operations

- **Roll-up**: aggregates data at a higher level (e.g., last year's sales volume per product category and per region)
- **Drill-down**: de-aggregates data at the lower level (e.g., for a given product category and a given region, show daily sales)
- **Slice-and-dice**: applies selections and projections, which reduce data dimensionality
- **Pivoting**: selects two dimensions to re-aggregate data
- **Ranking**: sorts data according to predefined criteria
- **Traditional operations** (select, project, join, derived attributes, etc)



5.2.2 OLAP Logical Models

In order to properly operate, OLAP techniques requires a logical model. We may distinguish between two different models:

- **MOLAP** (Multidimensional On-Line Analytical Processing):
  - stores data by using a multidimensional data structure (the cube);
  - the storage is not in the relational database, but in proprietary formats (in this sense, it is naturally stored based on the data structure);
  - Advantages:
    * Excellent performance, fast data retrieval, optimal for slicing and dicing operations
    * Can perform complex calculations (pre-generated when the cube is created)
    * It naturally follows the way the data is organised.
  - Disadvantages:
    * Limited in the amount of data it can handle (calculations are performed when the cube is built);
    * Requires additional investment: cube technology are often proprietary and do not already exist in the organization (human and capital resources are needed).
- **ROLAP** (Relational On-Line Analytical Processing):
  - Uses the relational data model to represent multidimensional data.
  - Advantages:
    * Can handle large amount of data

    - Can leverage functionalities inherent in the relational databases
  - Disadvantages:
    - Performance can be slow because ROLAP report is essentially a SQL query (or multiple queries) in the relational database (query time is long if data size is large)
    - Limited by SQL functionalities

- **HOLAP** (Hybrid On-Line Analytical Processing):
  - combines the advantages of MOLAP and ROLAP.
  - for summary-type information it leverages cube technology for faster performance
  - when detail information is needed it can "drill through" from the cube into the underlying relational data

THE DATA CUBE IN SQL

In order to integrate similar aggregation capabilities in SQL (for supporting ROLAP operations), two instructions were defined:

- **WITH CUBE**:
  - generates a result set that shows aggregates for all combinations of values in the selected columns
  - evaluates aggregate expression with all possible combinations of columns specified in group by clause
- **WITH ROLLUP**:
  - generates a result set that shows aggregates for a hierarchy of values in the selected columns
  - evaluates aggregate expressions only relative to the order of columns specified in group by clause
  - it eliminates the results that contain ALL only in one column

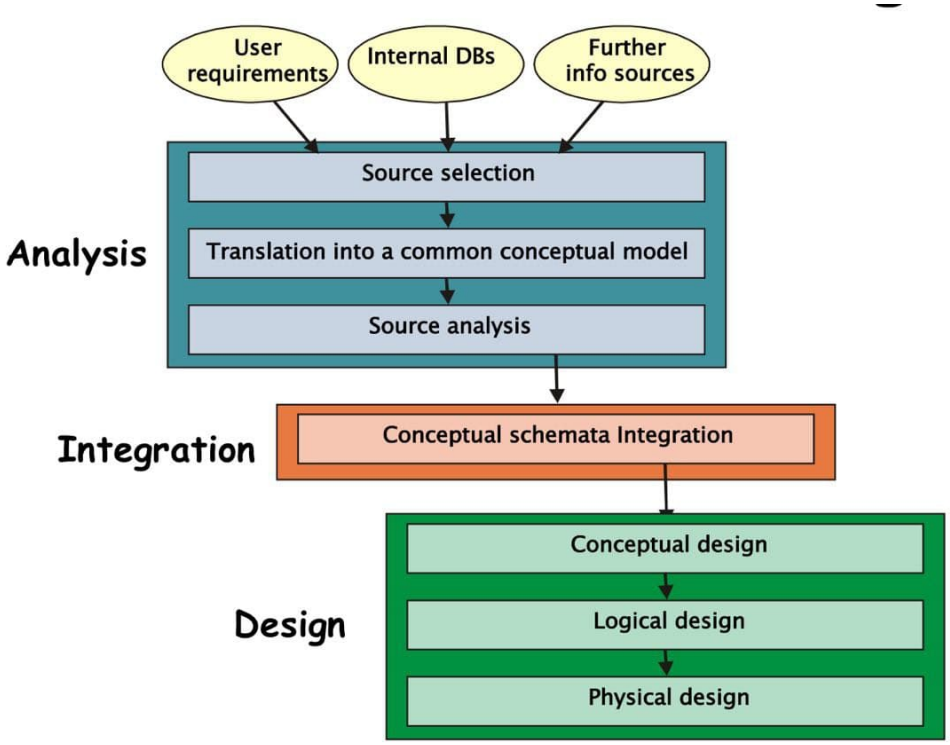
5.3 Data Warehouse Design

Data Warehouse Design is defined as the task of designing a data warehouse from scratch. In general, we distinguish between one initial premise and three operational steps:

- **Pre-Analysis**: it is a phase used to gather business needs, translated as user requirements, internal dbs information and other useful information necessary for the implementation;
- **Analysis**: during this phase, sources are managed in order to discard useless data, create a common conceptual model and optimise the way in which data are dispensed;
- **Integration**: it is the integration of the data sources;
- **Design**: the face in which the data warehouse is conceived conceptually, logically and physically.

In the design phase, the first element which is created is the **conceptual schema**. The conceptual schema is the formal definition of requirements and data in terms of data warehouse elements (or cube elements).

We distinguish between various elements which characterise a conceptual schema.
A **primary event** is an occurrence of a fact. It is represented by means of tuple of values (e.g., On 10/10/2001, ten "Brillo" detergent packets were sold at the BigShop for a total amount of 25 euros).
A **hierarchy** describes how it is possible to group and select primary events. The root of a hierarchy corresponds to the primary event, and represents the *finest aggregation granularity*. Given a set of dimensional attributes, each tuple of their values identifies a **secondary event** that aggregates (all) the corresponding primary events.



For example the sales can be grouped by Product and Month: "in October 2001, 230 "Brillo" detergent packets were sold at the BigShop for a total amount of 575 euros".

A **descriptive attribute** contains additional information about a dimensional attribute. They are uniquely determined by the corresponding dimensional attribute (1:1 relation).
A **cross-dimensional attribute** is a dimensional or a descriptive attribute whose value is obtained by combining values of some dimensional attributes.
In a fact schema, some portions of a hierarchy might be duplicated. As a shorthand we allow **hierarchy sharing**. If the sharing starts witha dimension attribute, it is necessary to indicate the roles on the incoming edge.
Aggregation requires to specify an operator to combine values related to primary events into a unique value related to a secondary event. (e.g., sum of sold quantity aggregated by month).
A measure is **additive w.r.t a given dimension** iff the SUM operator is freely applicable to that measure along that dimension without the need of any additional information (e.g. amount sold is additive w.r.t. time and space while total number of products sold is non-additive w.r.t. to time).

It is possible to identify three different measure categories:

- **Flow measures**: related to a time period
  - e.g., N. of sales per day, N. of births in a year.
  - At the end of the period the measures are evaluated in a cumulative way (SUM, AVG, MIN, MAX).
- **Level measures**: evaluated in particular time instants
  - e.g., N. products in stock, N. inhabitants in a city
  - SUM only on non temporal hierarchies, AVG, MIN, MAX.
  - N. products in stock can be aggregated by SUM over the category/type or the shop/city hierarchies, but NOT over time hierarchies
- **Unitary measures**: relative measures
  - e.g., unitary price at a given instant, money change rate, interest rate

- They are evaluated in particular time instants but they are relative measures (AVG, MIN, MAX, but relative measures)
- unitary price at a given instant cannot be aggregated by sum over the category/type or the shop/city, nor over the time hierarchy

	Over temporal hierarchies	Over non-temporal hierarchies
Flow measures	SUM, AVG, MIN, MAX	SUM, AVG, MIN, MAX
Level measures	AVG, MIN, MAX	SUM, AVG, MIN, MAX
Unitary measures	AVG, MIN, MAX	AVG, MIN, MAX

5.3.1 Aggregate Operators

- **Distributive operator**: allows to aggregate data starting from partially aggregated data (e.g., sum, max, min)
- **Algebraic operator**: requires further information to aggregate data (e.g., avg)
- **Holistic operator**: it is not possible to obtain aggregate data starting from partially aggrega data (e.g., mode, median)

5.3.2 Star Schema vs. Snowflake Schema

STAR SCHEMA

A **Star Schema** is:

- A set of relations $DT_1, DT_2, ..., DT_n$ (**dimension tables**) each corresponding to a dimension
  - Dimension tables are de-normalized, and this introduces redundancy, but fewer joins to do
- Each DT_i is characterized by a primary key d_i and by a set of attributes describing the analysis dimensions with different aggregation levels
- A relation FT, **fact table**, that imports the primary keys of dimensions tables. The primary key of FT is d_1, d_2, d_n. FT contains also an attribute for each measure.

SNOWFLAKE SCHEMA

The **Snowflake Schema** reduces the de-normalization of the dimensional tables DT_i of a Star Schema (removal of some transitive dependencies). This allows to avoid space wasting.
Dimension tables of a Snowflake schema are composed by:

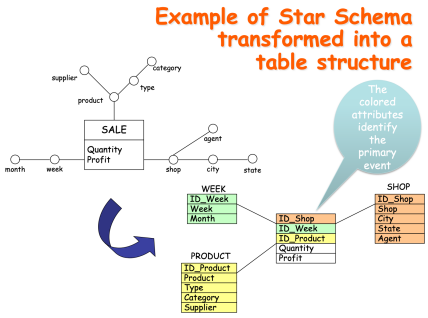
- A primary key $d_{i,j}$
- A subset of DT_i attributes that directly depend on $d_{i,j}$
- Zero or more external keys that allow to obtain the entire information

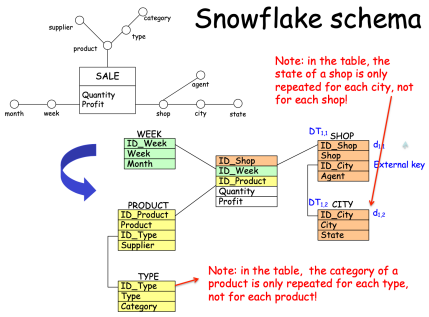
In a Snowflake Schema:

- **Primary dimension tables**: their keys are imported in the fact table
- **Secondary dimension table**

Benefits:

- Reduction of memory space
- New surrogate keys
- Advantages in the execution of queries related to attributes contained into fact and primary dimension tables





5.3.3 Conceptual Design

Conceptual design takes into account the documentation related to the reconciled database.

- Conceptual Schema (E/R, UML class diagram, ...)
- Logical Schema (e.g., relational, XML, ...)

Top-down methodology:

1. **Fact definition**: facts correspond to events that dynamically happen in the organization
2. For each fact:
   (a) **Design of the attribute tree**
   (b) **Attribute tree editing**: allows to remove irrelevant attributes
   - *pruning*: the subtree rooted in v is deleted. Delete a leaf node or a leaf subtree.
   - *grafting*: the children of v are directly connected to the father of v. Delete an internal node, and attach its children to the parent of the internal node.

   (c) **Dimension definition**: dimensions can be chosen among the attributes that are children of the root of the tree (time should always be a dimension)
   (d) **Measure definition**: if the fact identifier is included in the set of dimensions, then numerical attributes that are children of the root (fact) are measures. More measures are defined by applying aggregate functions to numerical attributes of the tree.
   (e) **Fact schema creation**: the attribute tree is translated into a fact schema including dimensions and measures. The fact name corresponds to the name of the selected entity, dimension hierarchies correspond to subtrees having as roots the different dimensions (with the least granularity)

In the **glossary**, an expression is associated with each measure. The expression describes how we obtain the measure at the different levels of aggregation starting from the attributes of the source schema.

6 Big Data Architectures and Data Ethics

NOT PART OF THE 2022 COURSE

6.1 NoSQL Databases

6.1.1 Transactional Systems

ACID properties are a consistency model over data in transactions, that is common in traditional relational databases. They guarantee safe operations on data at anytime.
The ACID acronym stands for:

- **Atomicity**: a transaction is an indivisible unit of execution
- **Consistency**: the execution of a transaction must not violate the integrity constraints defined on the database
- **Isolation**: the execution of a transaction is not affected by the execution of other concurrent transactions
- **Durability** (Persistence): the effects of a successful transaction must be permanent.

The classical DBMSs (also distributed) are transactional systems: they provide a mechanism for the definition and execution of transactions. In the execution of a transaction the ACID properties must be guaranteed. A transaction represents the typical elementary unit of work of a Database Server, performed by an application.
Because ACID properties are not really required in certain domains, new DBMS have been proposed that are not transactional systems.

6.1.2 Big Data and the Cloud

Data Clouds: on demand storage services, reliable, offered on the internet with easy access to a virtually infinite number of storage resources, computing and network.

Classification of possible methods of storage:

- **Centralized or distributed**: transactional, based on a traditional model (e.g., relational), most widely used for traditional business applications
- **Federated and multi-databases**: used for companies and organization that are associated and share their data on the Internet
- **Cloud databases**: to support Big Data by means of load sharing and data partitioning (usually NoSQL)

6.1.3 NoSQL databases

It has been realized that is not always necessary that a system for data management guarantees all transactional characteristics. The non-transactional DBMS are commonly called NoSQL DBMS. This is really not correct because the fact that a system is relational (and uses the SQL language) and that it has a transactional characteristics are independent.
NoSQL databases:

- Provide **flexible schemas**
- **The updates are performed asynchronously** (no explicit support for concurrency)
- Potential inconsistencies in the data must be solved directly by users
- **Scalability**: no joins, no 2PhaseCommit
- Object-oriented friendly
- Caching easier
- Easily evolved to live replicas, made possible by the simplicity of re-partitioning of data
- **Do not support all the ACID properties**

6.1.4 NoSQL for Data Warehouses?

OLAP (On Line Analytical Processing)	OLTP (On Line Transaction Processing)
Processing of decision support operations	Processing of transactions, which implement the operational processes of the company
Complex and random operations	Often predefined and relatively simple operations
Each operation involves a lot of data	Each operation involves few data
Aggregate data, historical, even not necessarily current	Detailed data, updated
The operations are read-only (the ACID properties are not relevant)	The ACID properties of transactions are essential
The dimensions of the databases may reach the TERA or even the PETA bytes	The dimensions of the databases are of the order of the GIGA-TERA bytes

Figure 2: OLAP is typical of Data Warehouses

NoSQL main strengths are **scalability** and **flexibility**.

- Advantages:
  - No updates in DWs → no need of ACID properties
  - NoSQL databases provide very high read (and write) throughput on large objects
- Disadvantages:
  - NoSQL databases aren't intended for use in atomic level data
  - they also don't provide any kind of additional basic data transformation within the database

6.1.5 Data Model

- **Key-Value**
  - single key
  - Value: opaque
  - Querying: find by key
  - No schema
  - Standard APIs: get/put/delete
  - No relationships in the database → easier to scale!
  - Decoupled and denormalized entities are "self-contained"
  - Sharding (horizontal partitioning)
- **Document-based**
  - a unique key represent a document
  - Value: collection of documents whose structure is not fixed
  - Ability to query: very rich (filter on the internal fields of the document)
  - Free and less sensitive to design than a column-based
- **Column-family data model**
  - key is a triple row/column/timestamp
  - Value "opaque"
  - Strongly oriented to BigData
    - maximum scalability
    - data is partitioned horizontally and vertically based on the keys of the row and column (sharding)

  - Ability to query: get or filter only on row and column keys
  - Semi-structured diagram:
    * Row: columns indexed within each row by a row-key
    * Column-family: a set of columns, normally similar in structure to optimize compaction
    * Columns in the same column family will be "close" (stored in the same bloc on disk)
    * Columns: have a name and may contain a value for each row

- **Graph-based**
  - data relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them.

FAMOUS IMPLEMENTATIONS

- **Amazon DynamoDB**:
  - Key-value
  - CAP: AP, guarantees Availability and Partition tolerance, relaxing consistency
- **Google BigTable**:
  - Column-oriented
  - CAP: CP, if there is a network partition, Availability is lost, but strict consistency may be required
- **Cassandra**:
  - Column-oriented
  - CAP: AP, consistency is configurable
- **MongoDB**:
  - Document-based
  - CAP: CP

6.1.6 CAP Theorem

A data management system shared over the network can guarantee at most two of the following properties:

- **Consistency (C)**: all nodes see the same data at the same time
- **Availability (A)**: every request receives a response on success or failure (regardless of the state of any individual node in the system)
- **Partition Tolerance (P)**: the system continues to operate despite the number of message about loss or failure of part of the system (i.e., network partitions)

Note that all three properties are more "fuzzy" than binary. In particular there's a trade-off between Consistency and Availability. Increasing the support for one of them, will result in decreasing the support for the other.
N.B. In ACID the C means that a transaction preserves all the database constraints; in CAP the C refers only to copy consistency, a strict subset of ACID consistency.

6.2 Data Ethics

As data have an impact on almost every aspect of our lives, it is more and more important to understand the nature of this effect. With search and recommendation engines, the web can influences our lives.
Big Data processing is based on algorithmic, thus it must be objective. Unfortunately:

- algorithms are based on data, and data may contain errors
- often the algorithm produce opaque processes that cannot be explained
- technology must be accompanied by ethical and legal considerations

It is up to the data scientists to:

- identify which datasets can genuinely help answering some given question
- understand their contents
- choose the most appropriate knowledge extraction technique (search, query, or data analysis methods) to obtain a fair result

This sequence of choices may strongly influence the process, and biased results might be obtained.

6.2.1 Ethical Dimensions

- **Data protection**: of human rights and their consequences in normative ethics.
- **Fairness**: fairness of data is defined as the absence of bias.
- **Transparency**: is the ability to interpret the information extraction process in order to verify which aspects of data determine its results. In this context, transparency metrics can use the notions of:
  - *data provenance*: describing where the original data come from
  - *explanation*: describing how a result has been obtained

  Transparency may conflict with Data Protection.
- **Diversity**: is the degree to which different kinds of objects are represented in a dataset. The diversity dimension may conflict with data quality needs, that prioritize the use of few, high-reputation sources.

Data quality is a typical ethical requirement: we could never trust a piece of information if it did not have the typical data quality properties.
Data should conform to a high ethical standard, for it to be considered of good quality.
Hence, the satisfaction of the ethical requirements is actually necessary to assert the quality of a result. It is the responsibility of the system designer and of the person/company that ordered the job, to ensure that the necessary ethical properties are satisfied. END

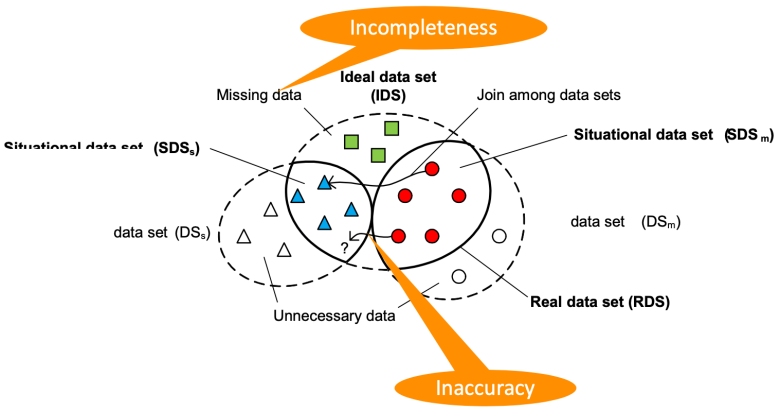
7 Data Quality

7.1 Introduction

Collection of data is the most important activity in a business. The analysis potential intrinsic to data has brought many companies to pursue a new type of business management approach, called **Data Driven Management**. Data Driven Management is the management of business process based on insight derived by the analysis of data. In fact, a proper analysis on data may highlight interesting phenomena and facts about a particular business aspect, which may be used to ensure the development and improvement of the business strategy within a specific market.

Data Driven Management (DDM), however, is not an easy approach to fully implement. One of the most important problem in DDM is probably the **Garbage In - Garbage Out** problem (GIGO). The GIGO problem refers to the characteristics of processing systems to elaborate data without a proper way of recognising inconsistencies and errors in it. In practice, the GIGO problem brings a computer to draw incorrect conclusions from incorrect data.

GIGO problem underlines a basic necessity of DD systems: data has to be managed in the best way possible. In practice, every system requires proper data preparation steps. This task is vital as real-world data is often incomplete, inconsistent, and contain many errors. Data preparation, cleaning, and transformation comprises the majority of the work in a data mining application (90%).



Data Quality is the ability of a data collection to meet user requirements. Causes of poor quality:

- *Historical changes*: the importance of data might change over time
- *Data usage*: data relevance depends on the process in which data are used
- *Corporate mergers*: data integration might cause some difficulties
- *Privacy*: data are protected by privacy rules and thus it is difficult to find data to correct and its own db
- *Data enrichment*: it might be dangerous to enrich internal data with external sources.

Poor data quality effects:

- the customer satisfaction decreases
- is expensive
- impacts on decisional processes
- affects long-term strategies

7.2 Data Quality Management

Data quality management is performed in four phases:

1. Quality dimensions definition;
2. Quality dimensions assessment;
3. Quality issues analysis;
4. Quality improvement.

7.2.1 Quality Dimensions

Quality dimensions are defined as the properties that allow to measure how much data management reflects user requirements. The most used objective dimensions are:

- **accuracy**: the extent to which data are correct, reliable and certified.
- **completeness**: the degree to which a given data collection includes the data describing the corresponding set of real-world objects.
- **consistency**: the satisfaction of semantic rules defined over a set of data items. It refers to the violation of semantic rules defined over a set of data items.
- **timeliness**: the extent to which data are sufficiently updated for a task. It is the average age of the data in a source.

7.2.2 Assessment Techniques

- Objective measuring:
  - completeness → absence of null values
  - accuracy → distance between the current value and the correct value
  - consistency → ratio of correct values w.r.t. integrity and business rules
  - timeliness → max(0; 1 - currency/violatility)
- Questionnaires
  - definition of 12-20 items for data quality dimensions
  - Item: "this information is (attribute or phrase)"
  - e.g., this information is presented consistently, this information is relevant to our work

Data Quality Rules are the requirements that business set to their data and they are associated with the data quality dimensions. They are also designated to check the validity of data.

7.2.3 Analysis Techniques

- Benchmarking Gap Analysis
- Role Gap Analysis
- Plotting Data

7.2.4 Data Quality Improvement

The data quality improvement step aims to identify and solve errors in the data. It may happen in two different ways:

- Data-oriented improvement methods (e.g. Data Cleaning), focusing on data values independently of the context;
- Process-oriented improvements methods, activated when an error occurs in the utilisation context.

DATA PROFILING

Data Profiling is the process of analysing data in order to assess the basic structure of it, by summarising its main characteristics. Among the main purposes of a data profiling task, we distinguish:

- Understanding the main characteristics of a data pool;
- Annotating data, to simplify further processing;
- Discover relevant information, like distribution of data, functional dependencies;
- Identify duplicates;
- Categorise missing values.

DATA CLEANING

Data Cleaning is the process of identifying and eliminating inconsistencies, discrepancies and errors in data in order to improve quality.

We distinguish between various cleaning tasks

- **Standardization/normalization**:
  - Datatype conversion
  - Discretization
  - Domain Specific
- **Missing Values**:
  - Detection
  - Imputing
- **Outlier Detection**:
  - Model
  - Distance
- **Duplicate detection**:
  - discovery of multiple representations of the same real-world object and merging.

DUPLICATE DETECTION ISSUES

- Representations are not identical → we require similarity measures
- Datasets are large → we require scalable algorithms.

NOT PART OF THE 2022 COURSE

Similarity measures:

- String-Based Distance Functions
  - Edit-distance (minimum number of edits from one word to the other)
  - Soundex (phonetical similarity measure)

- Item-Based Distance Functions
  - Jaccard distances (intersection of two sets of words divided by the union of them)
  - Cosine similarity

In order to check the existence of duplicates we should compare all pairs of instances. **Too many comparisons** $\rightarrow (n^2 - n)/2$.
Partitioning is the solution for this problem. Partition records through a selection and comparisons are performed among pairs of records inside the partition.
Example: Sorted neighborhood

1. Creation of key: compute a key for each record by extracting relevant fields
2. Sort data: using the key
3. Merge data: move a fixed size window through the sequential list of records. This limits the comparisons to the records in the window.
4. Data are compared within a rule and a similarity function

7.2.5 Data Fusion

The data integration process is composed of three steps:

1. Schema alignment
2. Entity reconciliation
3. Data fusion

Data Fusion resolves uncertainties and contradictions. Given duplicate records (previously identified), it creates a single object representation while resolving conflicting data values.

- complementary tuples
- identical tuples
- subsumed tuples
- conflicting tuples

DATA FUSION ANSWERS

The result of a query to an integrated information system is called "**answer**".
Properties of an answer:

- **Complete**: the answer should contain all the objects and attributes that have been present in the sources
- **Concise**: all the object and attributes are described only once
- **Consistent**: all the tuples that are consistent w.r.t. a specified set of integrity constraints are present
- **Complete and consistent**: it additionally fulfill a key constraint on some real world ID (contains all attributes from the sources and combines semantically equivalent ones into only one attribute)

END

8 Modern Data Fusion techniques

8.1 The contribute of Machine Learning towards Data Integration

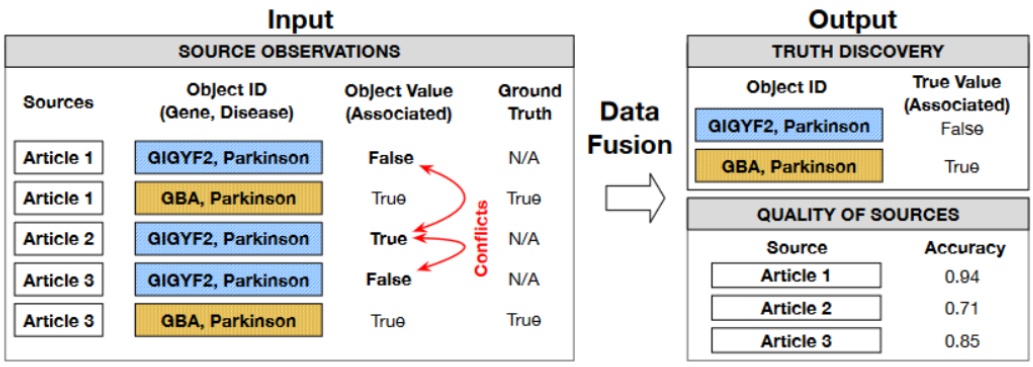
In modern days, one of the most active topic is **Machine Learning**. In the context of data integration, it is clear that paradigms as Big Data Integration and Data Warehouses accommodates Machine Learning data needs. However, the question has recently shifted to: *how it is possible to support Data Integration through Machine Learning*?

Machine Leaning may enable the following benefits in data integration tasks:

- Automation of integration tasks, such as automatic wrapping;
- Understanding of data semantics, as a support tool for data integration steps.

8.2 Discriminative Data Fusion

Discriminative Data Fusion refers to the task of executing a data fusion task by learning a system how to do it automatically. This is mostly leveraging pre-existing data and *domain knowledge*, in order to provide explicit example on how a set of information should be fused to represent a common entity.
A notable example is **SLIMFAST**, a machine learning model which takes as input genetic papers and is able discover the value (true, false) of a particular entity, when conflict occurs between various data sources. Moreover, it is able to assess the quality of a source, in terms of accuracy in providing a truthful value for a piece of information. It leverages two different approaches:



- **ERM (Empirical Risk Minimisation**: it maximise the likelihood of the example labeled as true, creating an easy problem to optimise but which requires a lot of data;
- **EM (Expectation Maximisation**: evaluates an expectation function for the likelihood of the data and maximise this function. It is affecte by source observation and accuracy, but requires less data.

8.3 Source Trustworthiness

Source trustworthiness refers to the property of a data source to provide accurate value. Most of the time, this task is based on majority voting, i.e., based on a truth value the system decrease the trustworthiness of lying sources. However, this method performs badly: it is not always true that a source is poor if it provides a false value.

A way to avoid this problem is useing quality measures, for example via probability assessment. One of the most interesting algorithms in such context is the ACCU algorithm, which allows to express the probability of a source to be trustworthy in terms of source accuracy. Refer to the image below: The EM algorithm allows to compute accuracy and probability values.

ACCU allows also to tackle the problem of **multi-truth data fusion**, which is data fusion in

Iterative Bayesian algorithms

Notation:
s = data source
i = data item
[B. Obama, born-in]
v = value of a data item
(B. Obama, born-in, Hawaii)

Bayesian theorem:

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)} = \frac{P(E|H) * P(H)}{\sum_{H_i} P(E|H_i) * P(H_i)}$$

Bayesian inference for data fusion:

Starting from

$$P(s \text{ provides } v \text{ for } i | \bar{v} \text{ is true for } i)$$

compute

$$P(v \text{ is true for } i \mid \text{values provided for } i) = \frac{P(\text{values provided for } i \mid v \text{ is true for } i) * P(v \text{ is true for } i)}{P(\text{values provided for } i)}$$

$$= \frac{P(\text{values provided for } i \mid v \text{ is true for } i) * P(v \text{ is true for } i)}{\sum_{v^*} P(\text{values provided for } i | v^* \text{is true for } i) * P(v^* \text{is true for } i)}$$

Iterative Bayesian algorithm: ACCU

X. L. Dong et al. "Integrating conflicting data: the role of source dependence." VLDB 2009

TARGET $P(v \text{ is true for } i \mid \text{values provided for } i) = \frac{P(\text{values provided for } i \mid v \text{ is true for } i) * P(v \text{ is true for } i)}{\sum_{v^*} P(\text{values provided for } i | v^* \text{is true for } i) * P(v^* \text{is true for } i)}$

Assuming $P(v' \text{is true for } i)$ is uniform
n number of values for i

$$= \frac{P(\text{values provided for } i \mid v \text{ is true for } i) * \frac{1}{n}}{\frac{1}{n} * \sum_{v^*} P(\text{values provided for } i | v^* \text{is true for } i)}$$

$$= \frac{P(\text{values provided for } i \mid v \text{ is true for } i)}{\sum_{v^*} P(\text{values provided for } i | v^* \text{is true for } i)}$$

Assuming independence of data sources, we can leverage:

$$P(\text{values provided for } i \mid v' \text{ is true for } i) = \prod_{s,v} P(s \text{ provides } v \text{ for } i | v' \text{ is true for } i)$$

$$P(s \text{ provides } v \text{ for } i | \bar{v} \text{ is true for } i) = \begin{cases} A_s & \text{if } v = \bar{v} \\ \frac{1 - A_s}{n - 1} & \text{if } v \neq \bar{v} \end{cases}$$

Notation:
A_s = accuracy of source s

case of multiple true values for a data piece. In this case, we need to consider that a datum may be associated with a set of true values, and therefore incomplete claims which state only a subset of such values is not false.

-