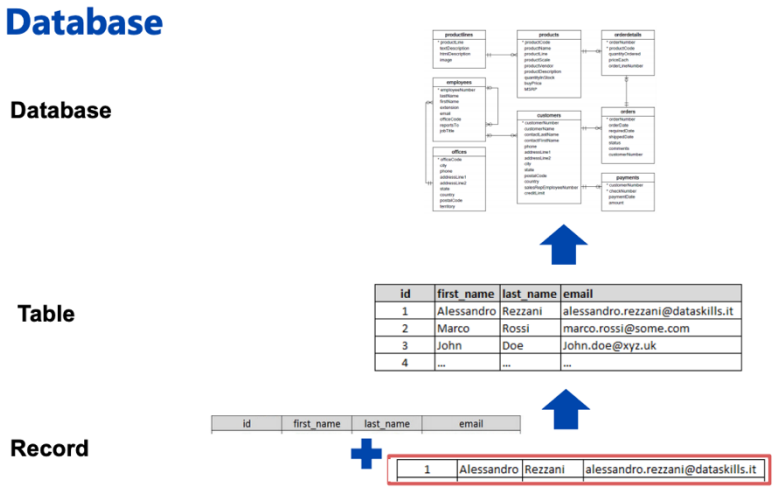
Big Data and Databases – Luca Molteni & Alessandro Rezzani

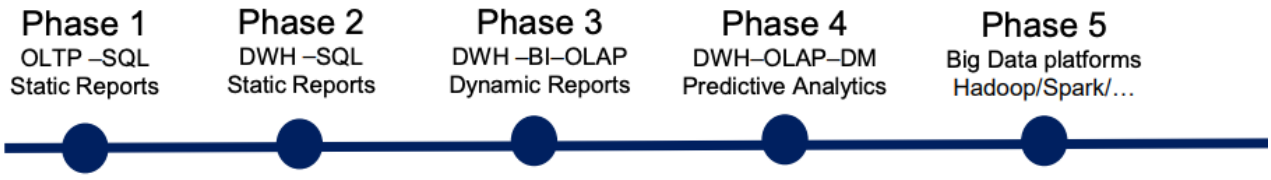
Course Objectives

- The course provides an overview of data management architectures and analytics procedures aimed at organizing, describing and modeling Big Data (structured and unstructured).
- The convents of the course covers both technical aspects of data management/analytics and topics related to analysis managerial evaluation (how to translate the outputs into meaningful business insights).
- Two different approaches are used: theoretical and applicative. A number of data ingestion procedures and machine learning data analysis case histories are shown, on Big and Small Data, using specific data management and machine learning software.
- At the end of the course the students will be able to improve their skills to manage and to take advantages of the huge availability of data nowadays produced by a great variety of sources.

Lesson 1 – The data analytics timeline: from mainframes to predictive analytics



Informal definition
Database = Set of tables (and other objects)
Table (a.k.a relation) = set of records. It's defined by a set of fields (columns) that have a name and a data type.
Record = horizontal group of values within a table. It contains values for multiple fields.



Phase 1 – OLTP-SQL Static Report
Starting from the 80s the RDBMS (Relational DataBase Management System) and the SQL (Structured Query Languange) allow the creation of static reports (mostly printed reports).
All the queries and reports are built on top of the **operational databases**.

What's a report?
A document containing information organized in a narrative, graphic, or tabular form, prepared on ad hoc, periodic, recurring, regular, or as required basis. We can also define a report as the formatted result of database queries that contains useful data for decision-making and analysis. Many types of reports:

- **Strategic Reports**:
  - Aggregated Data
  - It's a summary of the company's situation
  - It's built on a weekly or even monthly basis
- **Tactical Reports**
  - Less aggregated data
  - It's built on a daily or weekly basis
- **Operational Reports**
  - It contains both aggregated and detail data.
  - It's built on a daily basis.

Operational Databases
An operational database is a database that is used to manage and store data in real time. An operational database stores the day-by-day activity of a company. They support the storage,, usage and manipulation of data for a wide range of enterprise application like **ERP** and **CRM**.

- **ERP (Enterprise Resource Planning)**
  - It refers to the systems and software packages used by organizations to manage day-to-day business activities, such as accounting, procurement, project management and manufacturing.
  - ERP systems tie together many business processes and enable the flow of data between them.
  - Some software tools:
    - SAP
    - Microsoft Dynamics AX
    - Oracle JD Edwards
- **CRM: Customer Relationship Management** is a technology for managing all the company's relationships and interactions with customers and potential customers.
  - The goal is improving the business relationships.
  - A CRM system helps companies stay connected to customers, streamline processes, and improve profitability.
  - Some software tools:
    - Salesforce.com CR
    - SAP
    - Microsoft Dynamics

When analyzing data directly on the operational databases we may come across some **problems**.

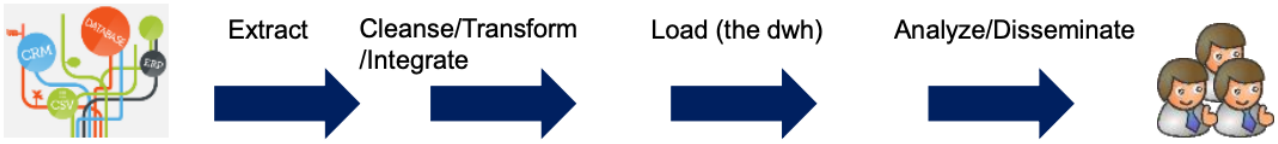
- The operational databases are **not structured for massive reads**. They are designed for fast inserts or fast updates.
- **Data might be replicated** in more than one database (i.e., ERP and CRM).
  If the data is located in different databases:
  - The data format could be different
  - Or we could have different different versions of the same data.
- The **reporting workload might affect performances** of insert/update operations and viceversa.

Note: We'll see a more complete definition of RDBMS in lesson 2. In lesson 2 we'll also see the main differences in database design between Operational Databases and Analytical Databases (Data Warehouse).

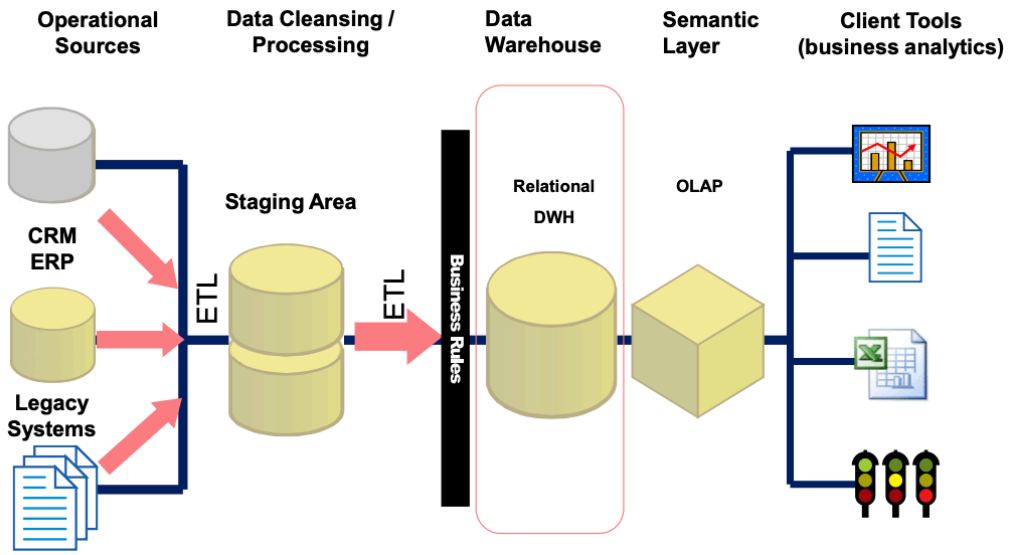
Phase 2 – Data Warehouse
Due to the operational db problems, starting from the '90s, we see the rise of an analytical database, called **data warehouse**. At the beginning it was just a mere copy of the operational databases: it solved the performance issued caused by different workloads (massive reads vs. writes). Later the data coming from different databases were integrated, made consistent, cleansed and organized in a different data model: the dimensional model, which is more appropriate for the analytics tasks. The data warehouse is the starting point of the Business Intelligence process.

The Data Warehouse
A data warehouse (DW) is a collection of corporate information and data derived **from heterogeneous operational systems** and external data sources. A data warehouse is **designed to support business decisions** by allowing data consolidation, analysis and reporting at different aggregate levels. Data is populated into the DW through the processes of **extraction, transformation and loading**. The data warehouse can be considered as the "**single source of truth**": it contains the enterprise data once they've been integrated from different sources and cleansed using business rules and data quality rules.

Business Intelligence
The term **Business Intelligence** (BI) refers to technologies, applications and practices for the collection, integration, analysis, and presentation of **business** information. The purpose of **Business Intelligence** is to support better **business** decision making.



Elements of a Business Intelligence System



ETL (Extract Transform Load)

The ETL acronym identifies both a class software tools and the process of loading the data warehouse.
The ETL process:

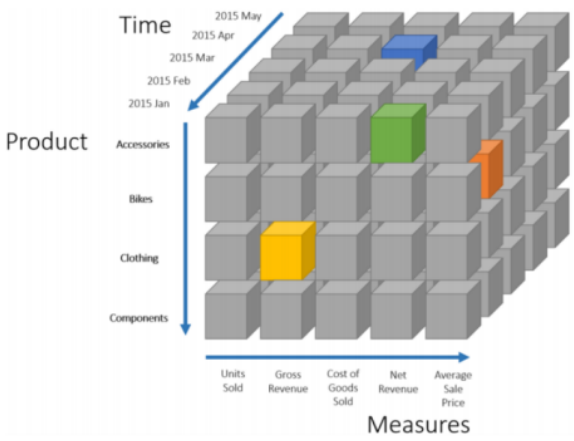
- **E**xtracts data from the operational sources.
- **T**ransforms the data by applying:
  - Business rules
  - Data quality checks
- **L**oads the data into the data warehouse.

Phase 3 – Data Warehouse – BI – OLAP

- The data warehouse requires SQL skills to be queried. The user also needs to know the data structures and the relationships between the tables.
- Starting from the late '90s the OLAP (On Line Analytical Processing) systems come to life.
- OLAP engines are databases where the tables are replaced with multidimensional structures.
- OLAP systems combine data & metadata: the user **doesn't need to know the relationships** when querying the OLAP Database.
- Additionally the OLAP System are much **faster than RDBMS** when querying aggregated data (SUMS, AVERAGES, COUNTS, …)

OLAP

- OLAP (Online Analytical Processing) is the technology behind many Business Intelligence (BI) applications.
- OLAP is a powerful technology for data discovery, including capabilities for limitless report viewing and complex analytical calculations.
- OLAP tools enable users to analyze different dimensions of multidimensional data.
- OLAP can be seen as a Semantic Layer that facilitates the access to the data.



*OLAP clients behave like a huge **pivot table** on all the data warehouse data: they allow "drag&drop" analysis.*

Phase 4 – Predictive Analytics

- Both the DWH and the OLAP systems provide an historical view of the data.
- Starting from year 2000 we see an increasing need to predict future events (such as the next month's sales or the cost of energy).
- A new term is created: **data mining**. It means discovering patterns in large data sets and find hidden patterns that can be exploited in the business.
- The data mining has many applications:
  - Customer segmentation, market basket analysis, churn analysis, campaign targeting, demand forecasting, etc.

Phase 5 – Big Data Platforms

- Starting from **2021** we start to talk about Big Data (& Big Data analytics)
- Big Data can be defined as data with the following features:

    - Volume
    - Velocity
    - Variety

Or, better, we can say that Big Data are "*data that we cannot analyze (or that we have no convenience to analyze) with traditional tools (i.e., relational databases)".*

Big Data Platforms

The main tools that allow us to handle massive amounts of data are:

- **Hadoop**, which is a distributed file system and computation engine (we'll see the definition of both in lesson 6). It is comprised of many tools that cover all the data life-cycle (from the ingestion from the sources to the data presentation to the user). Hadoop can be used both for data storage and for data analysis.
- **Spark**, which is a fast distributed computation engine. It is also made of many components and is more oriented to the data analysis.

Today

In **2019** the trend topics in data analytics are:

- Big Data
    - New versions of Hadoop
    - New versions of Spark
- Predictive Analytics, … a modern way to call the data mining.
- Cloud Computing
    - Platform as Service/Infrastructure as a Service
    - Scalable resources at low prices!

New Data Sources

"Traditional" data sources with a deeper level of detail:

- ERPs
- CRMs
- …

Emerging data sources:

- Scientific/Medical Equipment
- DCS (Distributed Control System)
- High Frequency Trading Systems
- Web 2.0: Blog posts, Tweets, Facebook comments & likes, images & videos.
- IOT: Internet of Things.

Features of Emerging Data Sources

- **Volume**
    - Potentially huge
    - Examples:
        - Detail level credit card transactions
        - Sensors data (from industrial plants, power facilities, etc.)
- **Velocity**
    - Fast data
    - Example: data coming from IOT sensors
- **Variety**
    - **Structured**

    - Data that can be conveniently put into a table
    - Example: data coming from an ERP
  - **Unstructured**
    - Data that can be conveniently put into a table
    - Examples: text data from tweets or blog posts, Image data from medical equipment.
  - **Semi-Structured**
    - Most of the so-called unstructured data, are actually semi-structured.
    - Example: a tweet contains the text, which is unstructured, but it also contains other data as key/value pairs: author, date and time, location, etc.

Machine Learning / Data Mining

- **Machine Learning**
  It's the science of getting computers to act without being explicitly programmed.
- **Data Mining / Predictive Analytics**
  Is the process of discovering patterns in large datasets. It involves methods coming from of artificial intelligence, machine learning, statistics, and database systems. It analyzes the current and historical facts to make predictions about future events.

Two types of analytics

- **Descriptive Analytics**, it summarizes raw data and make it something that is interpretable by humans. It describes the past. The old-style static reporting and the more recent business intelligence techniques can be placed into the descriptive analytics category.
- **Predictive Analytics**, these analytics are about understanding the future. It provides companies with insights based on data, provides estimates about the likelihood of a future outcome. Comprises a variety of statistical, machine learning, and data mining techniques. It analyzes the current and historical facts to make predictions about future events.

Lesson 2 – Relational Databases: An Introduction to Data Structures

Relational Database

A relational database is an entity consisting of logical units known as **tables** (also called relations). A relational database is based on the **relational model** of data proposed by E. F. Codd in 1970. Data in a relational database systems can be accessed using **SQL** (Structured Query Language). An **RDBMS** (**R**elational **D**atabase **M**anagement **S**ystem) is a software system used to run and maintain relational databases.

RDBMS

Key features of RDBMS:

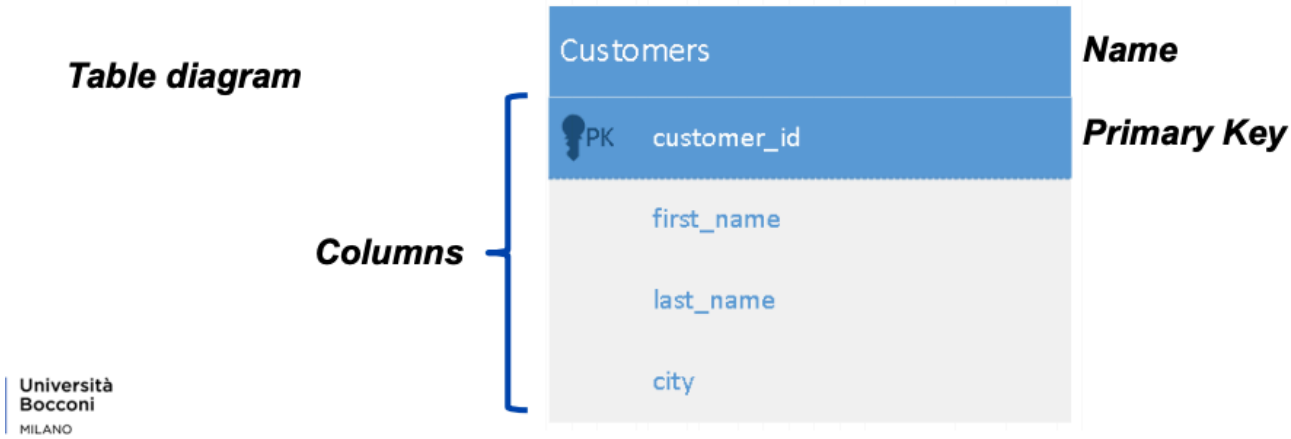
- **Stores** data into tables.
- Allows the **creation of new databases** and their data structures.
- Allows data **query** and **modification** using an appropriate programming language (SQL).
- Allows the **storage of vast amounts of data** over a long period of time.
- Enables database **recovery** in times of failure, error or intentional misuse.
- **Controls data access** from many users at once.

Tables

A table is defined by the following metadata:

- A **Name**

- One or more **columns** (or fields)
- A **primary key**, which uniquely identifies a row (or record)



Fields & Records
Columns or fields

- Fields are the building blocks of a table.
- They define the data structure of the table.
- A field is made of:
  - A **Name**
  - A **Data Type** (string, number, date, boolean, binary)
  - A **NULL constraint**. It's an attribute that specifies if the column can contain NULL values or not.

Records or tuples

- A table row is also called **record** or **tuple**.
- A record holds data for a single "entity" (i.e., a customer, a product, a supplier).
- A record contains a value for each field (for a single entity).

Data Types
String

- CHAR(n) = fixed length n.
- VARCHAR(n) = variable length, maximum n.

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Date

- DATE
- DATETIME
- TIME

Numbers

- INT

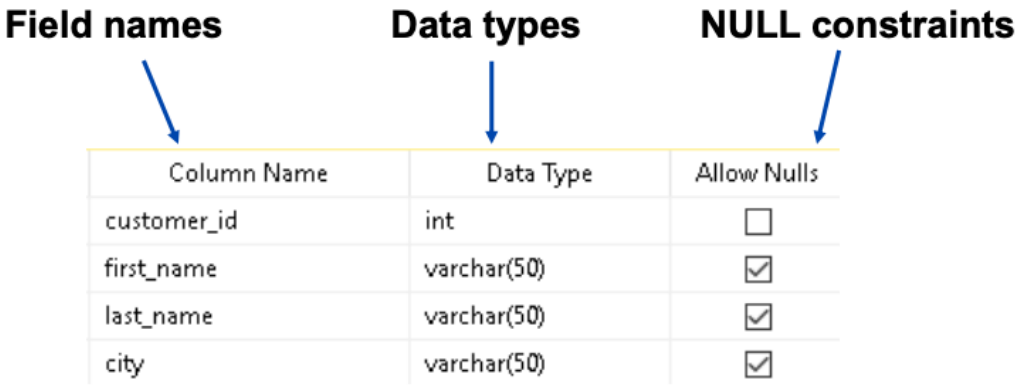
Floating point

- FLOAT

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

Fixed Decimals

- DECIMAL (precision, scale). Ex: DECIMAL(5,2) = 5 digits with 2 decimals.



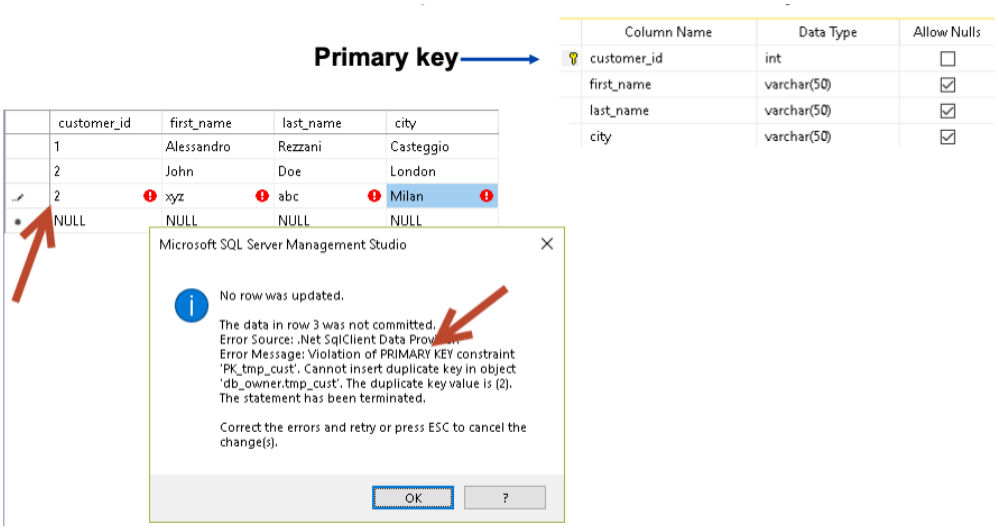
Column Name	Data Type	Allow Nulls
customer_id	int	☐
first_name	varchar(50)	☑
last_name	varchar(50)	☑
city	varchar(50)	☑

Records

customer_id	first_name	last_name	city
1	Alessandro	Rezzani	Casteggio
2	John	Doe	London

Primary Key

The primary key constraint guarantees the record uniqueness. When we put the primary key constraint on a column, its values must be unique. If we try and insert a duplicate value the RDBMS rejects it and returns an error:



Creating a table using SQL

We use the CREATE TABLE command, followed by the columns list (name, data type and NULL constraint).

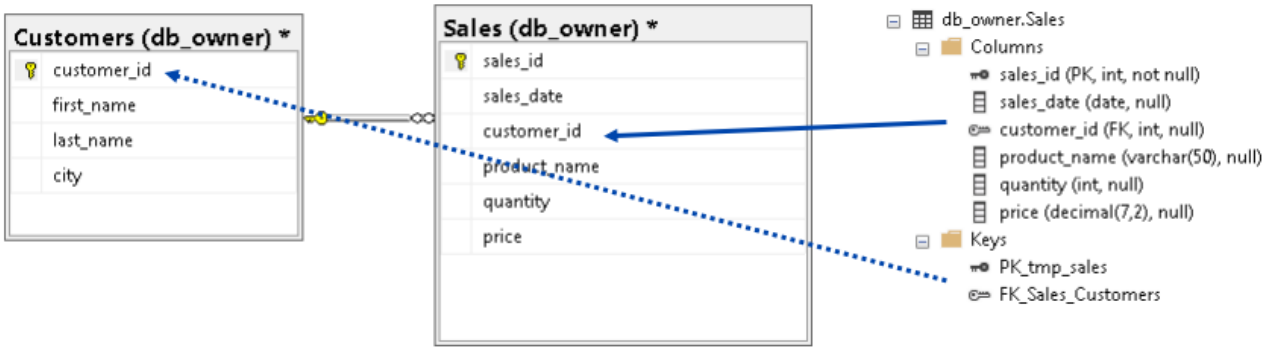
```
CREATE TABLE Customers(
  customer_id int PRIMARY KEY NOT NULL,
  first_name varchar(50) NULL,
  last_name varchar(50) NULL,
  city varchar(50) NULL
)
```

Foreign Key
As the primary key uniquely represents a record in a table, we can use the primary key value to make a reference to that record. For example: let's say that we have the sales table and we want to keep track of each safe, we could create the table like this:

	Column Name	Data Type	Allow Nulls
🔑	sales_id	int	☐
	sales_date	date	☑
	customer_id	int	☑
	product_name	varchar(50)	☑
	quantity	int	☑
	price	decimal(7, 2)	☑

We don't repeat all the data for each customer, but we just use the customer_id field (the primary key) to reference a customer.

But what happens if we put a non existing customer_id into the sales table? We loose consistency, because we would have a reference to a customer that doesn't exist!!
To avoid this situation we can create a **foreign key constraint** on the sales table. The foreign key links the customer_id column of the sales table to the customer_id column in the Customers table and doesn't allow the insertion of a non existing customer_id.



Data Integrity
The primary key constraint guarantees the **entity (table) integrity**, which means no duplicate rows (or no duplicate keys). The foreign key protects the database against the violation of the **referential integrity**. In fact a field with a foreign key constraint:

- Can be NULL
- Or MUST contain a value that matches one value taken from the linked primary key.

The data type of a column and the NULL constraint specify and protect the **domain integrity**: all the values of the same column belong to the same domain. In addition the user can define other constraint, called CHECK CONSTRAINT (for example: quantity > 0).

Full SQL Example

```
CREATE TABLE Sales (
    sales_id     INT  PRIMARY KEY NOT NULL,
    sales_date   DATE,
    customer_id  INT     REFERENCES Customers (customer_id),
    product_name VARCHAR (50),
    quantity     INT     CHECK (quantity > 0),
    price        DECIMAL (7, 2)
);
```

Dropping tables
We can delete a table using the DROP TABLE command:

```
DROP TABLE Sales;
DROP TABLE Customers;
```

Other Objects
A Database contains many object types along with the tables:

- **Indexes**, an index is a data structure that speeds up the data retrieval process. Just like an index in a book, the database index makes it possible to get data in a faster way.

Without indexes the RDBMS must scan the entire table even if we request a single record!

Database Normalization
Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to **eliminate data redundancy** and possible anomalies in insert, update and delete operations.
The normalization rules are divided into the following normal forms:

- **First** normal form
- **Second** normal form
- **Third** normal form
- Other normal forms (we won't see them in these lectures…)
  - Boyce-Codd Normal form
  - Fourth Normal form

Denormalized Tables
Customer Table

	Name	Address
	John Doe	5 th street, New York, USA
	Dave Johns	25 th street, New York, USA
	Dave Johns	25 th street, New York, USA

Employee Table

Employee_id	Name	Dep.code	Dep. Name	Dep. Phone
1	Mary Johns	FIN	Finance	123-456789
2	John White	FIN	Finance	123-456789
3	Lucy Smith	FIN	Finance	123-456789
4	…			

Problems
In the Customer table we have:

- A **duplicate record**.
- The name and address fields that are **not atomic**: many values are store inside those fields:
  - First and Last Name in the Name Column.
  - Street, City and Country in the Address Column.

In this case it's hard to use the address data (for example, getting the customers that live in NY).

In the Employee table we have:

- **Data Repetition** for Dep.Name and Dep.Phone.

In this case there could be **update anomalies**: for example if the department phone changes we must update it for all the employees that belong to that department. If an employee goes to another department, we'd have to update all the dep. Columns (code, name and phone). It's really easy to make a mistake…

First Normal Form
A table should only have single (**atomic**) valued columns. Values stored in a column should be of the **same domain** (same data type). All the columns in a table should have **unique names**. Records should be **unique**. A **primary key** constraint should be set in order to guarantee the record uniqueness.

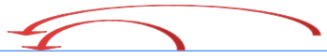
Customers Table

Customer_code	Name	Address
1	John Doe	5 th street, New York, USA
2	Dave Johns	25 th street, New York, USA
3	…	

Customer_code	First Name	Last Name	Street	City	Country
1	John	Doe	5 th Street	New York	USA
2	Dave	Johns	25 th Street	New York	USA
3	…				

Third Normal Form
The table shouldn't have Transitive Dependencies. We have **Transitive Dependency** when a non-primary key attribute depends on other non-primary key attributes rather than depending upon the primary key.

Employee table



Employee_id	Name	Dep.code	Dep. Name	Dep. Phone
1	Mary Johns	FIN	Finance	123-456789
2	John White	FIN	Finance	123-456789
3	Lucy Smith	FIN	Finance	123-456789
4	…			

Department table

Dep_id	Dep.code	Dep. Name	Dep. Phone
1	FIN	Finance	123-456789

Employee table

Employee_id	First Name	Last Name	Dep_id
1	Mary	Johns	1
2	John	White	1
3	Lucy	Smith	1
4	…		

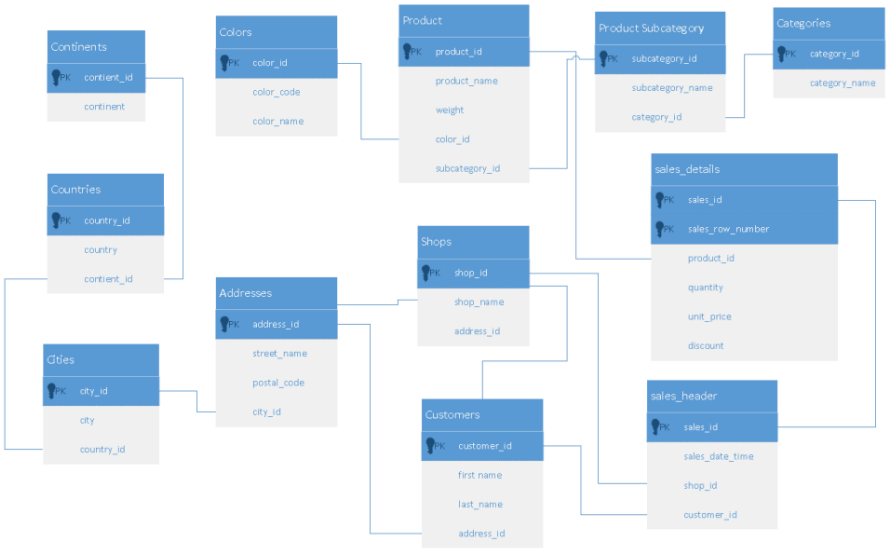
Università Bocconi MILANO

OLTP
On **L**ine **T**ransaction **P**rocessing. **Transaction Oriented** databases.
They are **fully normalized**:

- Many tables
- Each table has few columns
- No data redundancy!

This is the model for operational databases.
PROS: The normalization prevents errors in insert/update/delete operations.
CONS: But… the normalization makes it difficult to query the database: the analyst must put together many tables in order to create a report.

OLTP Example

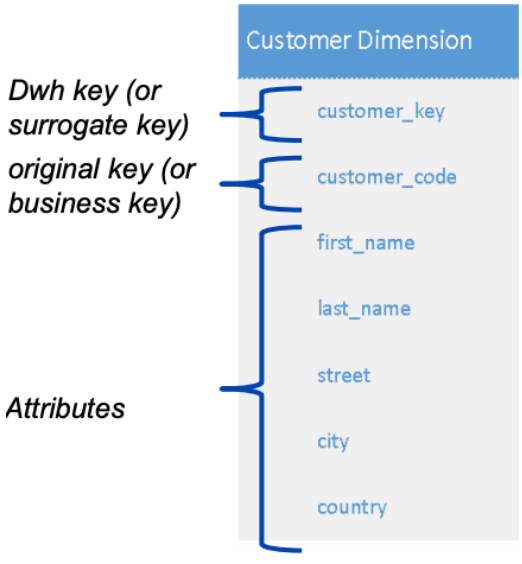


Data Warehouse
The data warehouse is an **analysis oriented** database. The purpose of DWH is to provide a **simple** data model for the analyst. The DWH is made of two types of tables:

- The **dimensions**, which are denormalized (1st Normal form).
- The **fact tables** which are normalized (3rd Normal form).

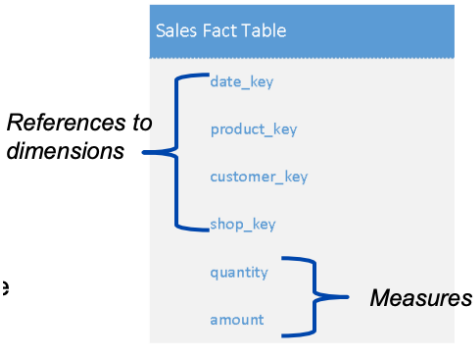
Dimensions
Dimensions contains all the attributes of a **business entity**. The dimension key in the DWH is called **surrogate key**. It's a progressive number created by the ETL process. The original key (coming from the operational source) is called **business key** and it's stored in the dimension table.
Examples of dimensions (or business entities):

- Customer
- Supplier
- Product
- Accounts
- Branches
- Departments



Fact Tables
Fact tables contain:

- The measures (quantities, values, counts)
- The references (surrogate key) to the applicable dimensions.
- Examples:
  - Sales fact table
  - Purchase fact table
  - Bank movements fact table

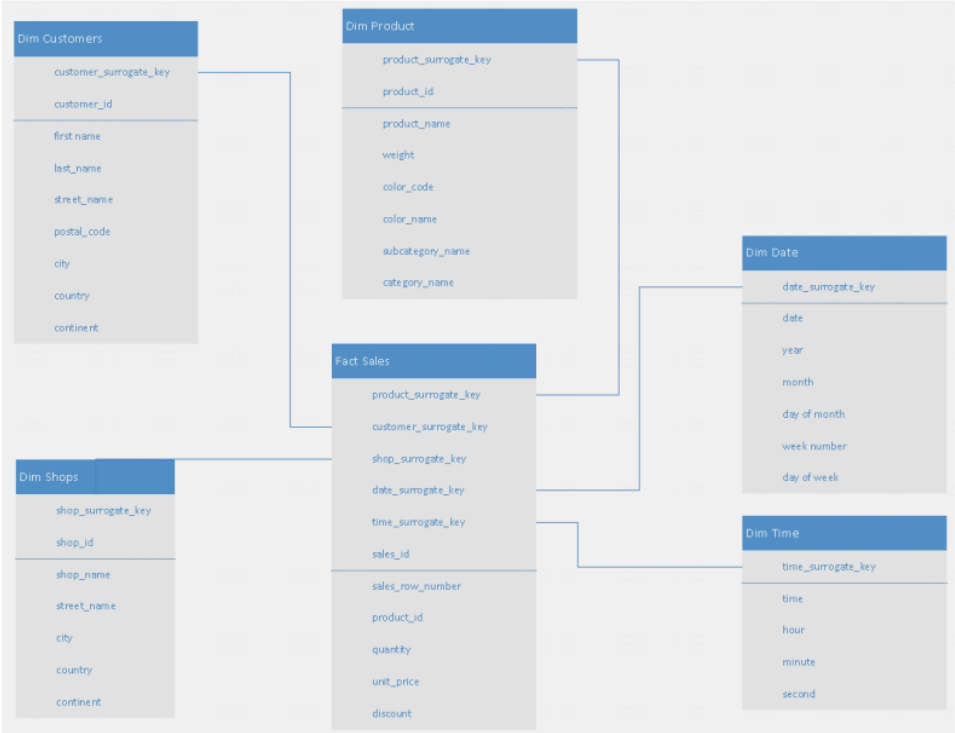


More on DWH

The DWH is fed with data coming from many operational databases. The process that feeds the DWH is called ETL (Extract Transform and Load):

- It extracts data from the data sources.
- It transforms the data according to the business rules.
- It integrates etherogeneous data sources.
- It checks the data quality.
- It generates surrogate keys.
- It takes care of the data integrity in the data warehouse (Generally we have no foreign keys in the DWH and even no primary keys!)

Data Warehouse Example



SQL Language Intro

SQL (Structured Query Language) allows us to interact with databases.
Technically SQL is:

- A **domain-specific** language: it only applies to databases.
- A **declarative language**: it expresses the logic of a computation/data retrieval/data modification without describing its control flow or algorithm. With SQL we write a query, but we don't tell the RDBMS how to actually implement it. RDMBS takes care of that,

using a component called optimizer, which chooses the best way to retrieve (or modify) data.

- A **procedural language**, because all the SQL dialects (implementation inside different RDBMS) contain procedural instructions (like IF, WHILE, …)

Why SQL for big data?
Most of the Big Data and NoSQL tools have an SQL interface:

- Hadoop
  - Hive works with a language called HiveQL which is an ANSI version of SQL + some Hadoop specific commands.
- Spark
  - SparkSQL is a component in the Spark platform.
- NoSQL
  - Native tools
    - Cassandra have the CQL language which is a dialect of SQL.
    - Hbase
  - ODBC Drivers

SQL consists of many types of statements, that can be grouped into 4 sublanguages:

- **Data Query Language (DQL)** used to retrieve data from the database.
- **Data Definition Language (DDL)** used to create tables and other objects (views, procedures, etc.)
- **Data Control Language (DCL)** used to grant access to the database objects. It's used to manage users' permissions on the db objects.
- **Data Manipulation Language (DML)** used to insert, update and delete data.

Some definitions:

- A **Query** retrieves data from one or more tables. It begins with the keyword SELECT.
- A **Statement** modifies data, the table schema (columns and data types) or controls the program flow. Keyword examples:
  - INSERT, UPDATE, DELETE.
  - CREATE TABLE / DROP TABLE
  - BEGIN, END, IF, WHILE
- A **Clause** is a part of a query or statement. We have:
  - The WHERE clause, that filters the records.
  - The SET clause in the UPDATE statement.
- An **Expression** returns scalar values or tabular values (rows and columns).
  - Example: Where city = 'Milan' ('Milan' is the expression)
- A **Predicate** is a logical condition used in the WHERE clause.

Some SQL Dialects

RBDMS	SQL dialect name
IBM DB2	SQL PL (SQL Procedural Language)
Microsoft SQL Server	T-SQL (Transact SQL)
MySQL	SQL/PSM (SQL/Persistent Stored Module)
Oracle	PL/SQL (Procedural Language/SQL)

Lesson 3 – Hadoop: the big data "Ecosystem"

Data Types

- Structured Data
- Unstructured Data
- Semi-Structured Data

Without a structure (fields or columns). We can't conveniently put them into a table:

- Documents (Word, Excel, PowerPoint, PDF, …)
- Images
- E-Mails
- Web 2.0 (blog posts, forum, wikis, …)

Big Data

Data that have one or more of the following features:

- Volume (Potentially huge)
- Velocity (Fast data)
- Variety (Structured, Semi-Structured, Unstructured).

A better definition:

- Data that we can't analyze using a traditional technology (like RDBMS)
- Data that could be analyzed using an RDBMS, but at a huge cost.

For huge amounts of data we need more than one machine: **we need a distributed system** that can split the data among its nodes and can perform distributed computations. For the unstructured data we need a system that works with **non tabular data structures**. For complex calculations we need a system that can handle (for instance) **machine learning tasks in a distributed manner**.
Moreover, we need a tool that is easy to use (at least for the final user). Does such a system exist?

Big Data Tools

We have **two main tools** in the big data panorama:

- **Hadoop** is a complete platform for Big Data.
- **Spark** is an high performance calculation engine.

Hadoop

It is a distributed calculation system based on two components:

- **HDFS**: Hadoop Distributed File System.
- **MapReduce**: a distributed calculation framework.

Key Points:

- **Open Source**
- **Distributed** (Up to thousands of nodes)
- **Fault Tolerant**
  - Hadoop is built to run on commodity hardware.
  - The fault tolerance is guaranteed using replication.
- **Scalable**
  - Up to thousands of nodes.
- Can handle both **structured** and **unstructured** data.

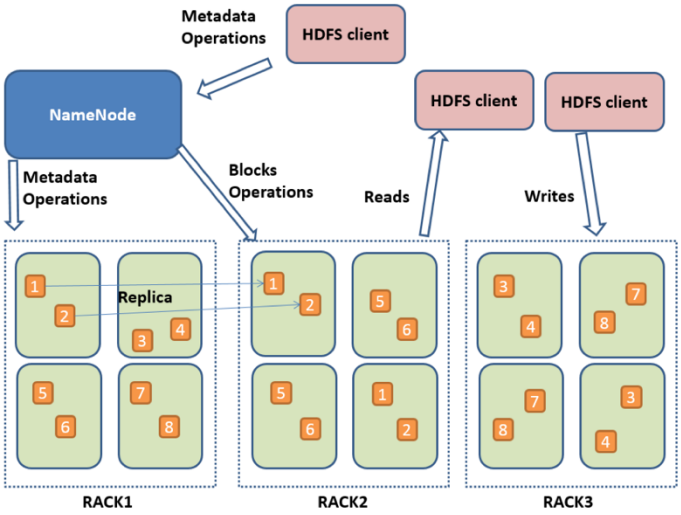
Definitions

- **Node** = Single Server.
- **Rack** = Case with a bunch of Servers.
- **Cluster** = Collection of Servers that behaves as if it was a single machine. Can be made of one or more racks.

How HDFS works

Distribution

- Each file in the distributed file system is made of parts called blocks (generally 64 or 128 MB).
- A server, called **Name Node**, contains all the file system metadata (i.e., the list of all the files, the list if each block and their locations).



Replicas

- Each block is **replicated** (the default replication factor is 3, but it can be changed).
- Replication ensures the **fault tolerance** (at least until all the nodes with the replica of the same block fail).
- HDFS is **rack aware**: the replication ensures that even if an entire rack fails, all the data is still available.
- The **Name Node** is the master node in the Hadoop cluster.
  - It handles **metadata** operations (like creating a new file, renaming file, deleting a file).
  - The name node also takes care of the **replication**.
- The other nodes in the Hadoop cluster are called **Data Nodes**.
  - They are slave nodes.
  - Data nodes take care of the read and write operations.

Replication and Hadoop v3

The replication mechanism ensures a certain level of fault tolerance. But it requires more disk space to store the data (n times the actual file size, where n is the replication factor). In Hadoop 3.x the replication has been replaced by a more space efficient algorithm called "**Erasure Coding**" which reduces to 50% the Storage overhead:

- The EC algorithm adds some data to each block.
- That data is encoded in such a way that the system can use it to rebuild other blocks (in case of their unavailability.

Map Reduce

MR is a **distributed computation engine** that works with data stored in HDFS. Although today we have other (and more efficient) engines in Hadoop, Map Reduce is the one from which all the other system have been derived.

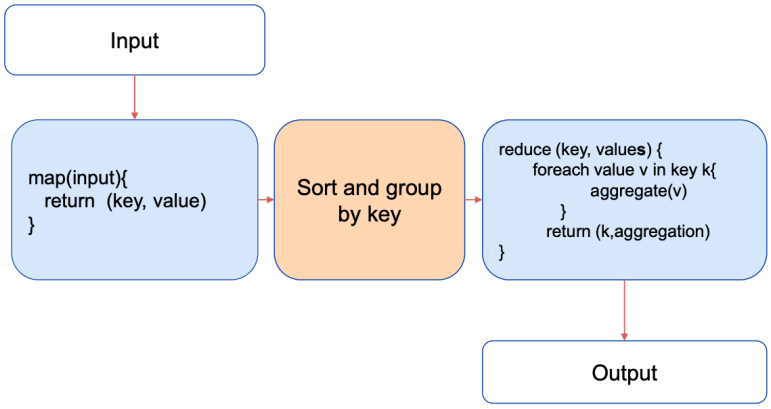
How Map Reduce works

Map Reduce only has a **Java** API: the programmer must split the problem in **two phases**:

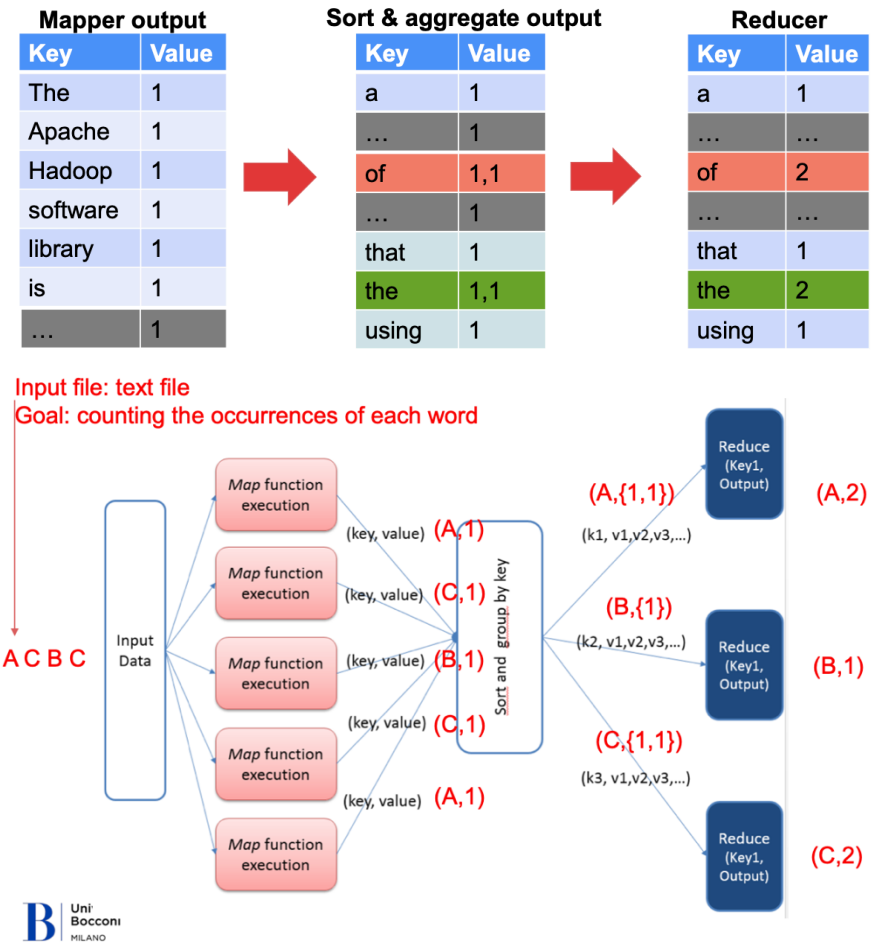
- The **Map** phase, in which the data are prepared and "mapped" to a key-value structure. The "mapper" program output is a key-value data structure.
- The **Reduce** phase, where the actual calculation is performed. The "reducer" output is the final result that is saved to an HDFS file.

The system receives the mapper and the reducer programs and performs the following operations:

- It **executed the mapper program** on each node where the input file parts are stored.
- It **sorts, shuffles** and **aggregates** (puts the values of the same key together) the outputs of the mapper programs.
- It **executes the reducer program** on each node.



Word count is the first task we accomplish when performing a text analysis.
"The Apache Hadoop software library is a framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models."



Map Reduce: Batch Workload

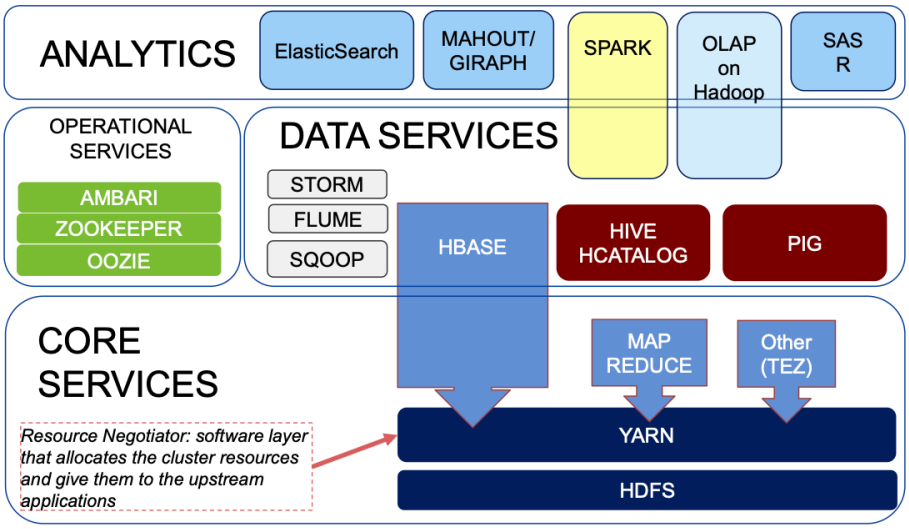
Map Reduce has been designed for **batch workloads**: Computations on a huge amount of data, that complete in minutes or even hours.
Map Reduce isn't suitable for interactive queries (like the ones we issue against an RDBMS, i.e., many queries on a small amount of data): for this type of workload we expect the query to return a result in seconds or even in tenths of a second.
Map reduce is hard to program and most problems are very difficult to design with a Map Reduce logic.

Pig and Hive
Pig and **Hive** are high level tools that simplify the usage of Map Reduce.
They provide an high level language:

- **Pig Latin**: procedural language whose instructions resemble the SQL keywords.
- **HiveQL**: an SQL dialect with implements the ANSI SQL syntax + some built-in functions.

They translate the programs in the high level language into Map Reduce Jobs.
The main Hadoop distributions provide ODBC Drivers that allow the user to connect to hive like if it was an RDBMS: in this way all the Business Intelligence & reporting tools can connect to Hadoop!

The Hadoop Ecosystem (simplified)



Core Service

- **HDFS**: Hadoop Distributed File System
- **YARN**: A resource negotiator, that takes care of allocating the resources for each Hadoop task.
- **Map Reduce, Tez and Spark**: the distributed calculation Engines.

Data Services

- **Hive**: the SQL interface to Hadoop. It translates HiveQL (SQL) programs into Map Reduce, Tez or Spark programs.
- **Pig**: another service that translates Pig Latin programs into Map Reduce, Tez or Spark programs.
- **Hbase**: the NoSQL columnar database that only works inside Hadoop.
- **Storm & Flume**: data ingestion tools used to read data that is produced in streams. (e.g., servers logs, tweets, sensors data from IOT devices, …)

- **Sqoop**: it's a sort of ETL tool that acts as an interface between the RBDMS and Hadoop. By means of Sqoop we can import data from a relational database to Hadoop or export data from Hadoop to an RBDMS.

Operational Services

- **Ambari**: web tool used to manage the Hadoop cluster.
- **Oozie**: workflow tool that the developer can use to orchestrate the execution of different jobs (Map Reduce, Hive, Pig, Sqoop, Flume, Storm, …).
- **Zookeeper**: it shares and keep synchronized the configuration files for each application.

Analytical tools

Tools that only work inside an Hadoop cluster:

- **Giraph**: a graph database tool.
- **Mahout**: a machine learning library.
- **Kylin**: OLAP tool on Hadoop. It exploit the columnar storage provided by HBase for saving aggregations and using them to support Hive queries.

Other distributed tools:

- **Elastic Search**: a very powerful search engine.
- **Spark**: a fast calculation engine.

Other analytic tools that can connect/interact with Hadoop.

- SAS: it provides components for interacting with Hadoop and Spark.
- KNIME: it provides components for interacting with Hadoop and Spark.
- R (The Microsoft R version is full integrated with Hadoop).

The Hadoop Distributions

Installing a cluster using the Apache Hadoop distribution can be very hard. There are other Hadoop distributions that provide a semi-automatic **installer** that make it easy to deploy an Hadoop cluster. The most famous one is **Cloudera**. In 2018 the merged with Hortonworks, another company that provides a Hadoop distribution.

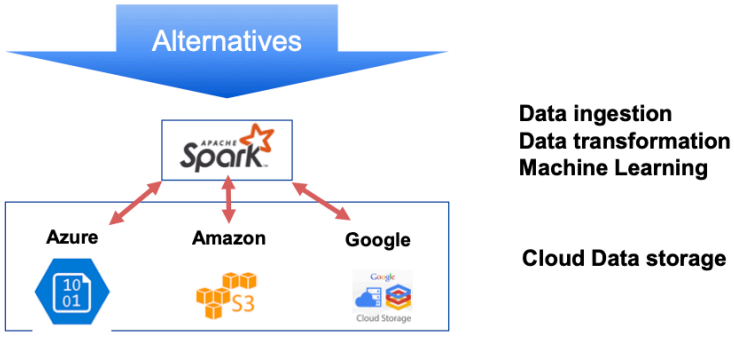
Hadoop pros & Cons

Pros

- Can handle every data type
- Can store and compute huge amounts of data

Cons

- Complex system.
- Difficult to install and maintain.
- Slow for a certain kind of workloads (like interactive queries).



Other tools in the Big Data Scenario

- NoSQL Databases
- Distributed databases: they work on more than 1 server simultaneously
- Mostly open-source

- Horizontally scalable: adding one or more servers we increase the computation power and/or the available storage.
- Schema free: there's no table definition and can host data with variable structures.
- They have a simple programming interface … but most of times they have SQL interfaces.

NoSQL Taxonomy

- Key-Value stores (DynamoDB), data are stored in key-value pairs.

Key	Value
first_name	Mary
last_name	Smith
city	Liverpool
Street Address	123, Main St.

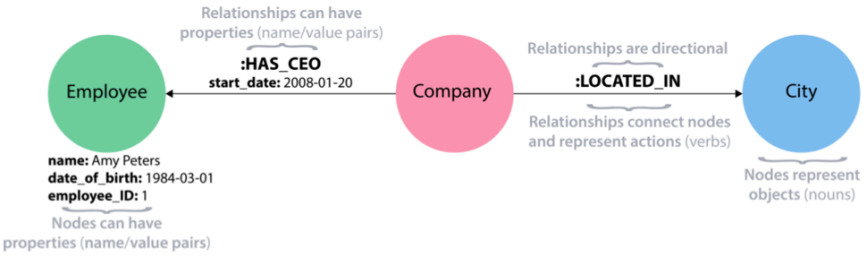
- Document Oriented DB (MongoDB), the typical data structure is a JSON or XML fragment (both contain both metadata (the keys) and the data (the values)).

```
{
    _id:"xyz123456"
    first_name:"John"
    last_name:"Doe"
    address:{
        street:"105, Main Street"
        city:"London"
        Country:"UK"
    }
}
```

- Column Oriented DB (Cassandra), the main data structure is the column.

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
…	…	…
2013-03-31	17.3	100

- Graph oriented DB, based on the graph concept (nodes + relationships).



Lesson 4 – Querying a Relation Database: the SQL language part I

SQL Sub Languages

SQL consists of many types of statements, that can be grouped into 4 sublanguages:

- **Data Definition Language (DDL)**, used to create tables and other objects (views, procedures, etc.)
- **Data Manipulation Language (DML)**, used to insert, update and delete data.
- **Data Query Language (DQL)**, used to retrieve data from the database.
- **Data Control Language (DCL)**, used to grant access to the database objects. It's used to manage users' permissions on the db objects.

Data Definition Language (DDL)

- **CREATE TABLE**, creates a table.

```
CREATE TABLE  table_name (
   column_specification1,
  column_specififcation2,
   ....
)
```

```
CREATE TABLE Sales(
     sales_id int PRIMARY KEY NOT NULL,
     sales_date date NULL,
     customer_id int NULL REFERENCES Customers (customer_id),
     product_name varchar(50) NULL,
     quantity int NULL CHECK (quantity > 0),
     price decimal(7, 2) NULL
);
```

- **DROP TABLE**, deletes a table from the database. It deletes both the data and the table definition.

```
DROP TABLE Sales;
DROP TABLE Customers;
```

Data Manipulation Language (DML)

- **INSERT**, insert a record into a table.

Syntax1: (partial set of columns)

```
INSERT INTO table_name (col1,col2,col3)
VALUES (val1,val2,val3,...)
```

Syntax2 (all the columns with values in the same order as the column specification)

```
INSERT INTO table_name
VALUES (val1,val2,val3,...)
```

Università Bocconi MILANO

```
insert into customers
values(1,'john','doe','london');
```

- **INSERT (Select)**, insert more records into a table.

Syntax1: (partial set of columns)

```
INSERT INTO table_name (col1,col2,col3)
SELECT  val1,val2,val3
FROM source_table
```

Syntax2 (all the columns with values in the same order as the column specification)

```
INSERT INTO table_name
SELECT  val1,val2,val3
FROM source_table
```

Università Bocconi MILANO

```
insert into customers
select id, first_name, last_name, city
from temp_customers;
```

- **DELETE**, deletes one or more records from a table. DELETE supports the WHERE clause (like the SELECT statement).

Syntax:

```
DELETE
FROM table_name
[WHERE  predicate]
```

```
delete from customers
where  customer_id = 10
```

- **UPDATE**, changes one or more values into one or more records. UPDATE supports the WHERE clause (like the SELECT statement).

Syntax:

```
UPDATE table_name
SET column_name1 = value1
 [,column_name2 = value 2
  ,...]
[WHERE  predicate]
```

```
update customers
set city = 'Milan'
where  customer_id = 20
```

Data Query Language (DQL) – The Basics

- **SELECT**, the select statement retrieves data from one or more tables.
- Use SELECT with column list to show columns.
- Use FROM to specify the source table or view
  - Specify both schema and object names.
- End all statements with a semicolon.

Element	Expression	Role
SELECT	<select list>	Defines which columns to return
FROM	<table source>	Defines table(s) to query
WHERE	<search condition>	Filters returned data using a predicate
GROUP BY	<group by list>	Arranges rows by groups
HAVING	<search condition>	Filters groups by a predicate
ORDER BY	<order by list>	Sorts the results

- Displaying only specified columns.

```
SELECT last_name, country
FROM Customers;
```

- Displaying all columns.

```
SELECT *
FROM Customers;
```

SELECT: Calculations & Aliases
Calculations are scalar, returning one value per row.

Operator	Description
+	Add or concatenate
-	Subtract
*	Multiply
/	Divide
%	Modulo

Using scalar expressions in the SELECT clause

```
select
    s.*,
    s.price * s.quantity as amount
from sales s;
```

column alias

table alias

SELECT: Functions
In the select list (but also in the WHERE clause) we can use built-in functions:

- **Date Functions** (not supported in SQL LITE):
  - **YEAR**
  - **MONTH**
  - **DAY**
  - They return the year, the month or the day of a date.
- **String Functions**
  - **TRIM**, removes spaces at the beginning and the end of a string.
  - **UPPER, LOWER**, put a string in upper case or lower case.
- **Math Functions**
  - **ABS**, absolute value
  - **ROUND**, rounds a value to a specified precision.

```
select DISTINCT YEAR(sales_date), customer_id
from Sales
```

```
select TRIM(last_name)
from Customers
```

SELECT DISTINCT & ORDER BY
SQL query results are not truly relational:

- Rows are not guaranteed to be unique
- **DISTINCT** returns unique values.
- No guaranteed order.
- **ORDER BY** forces and **ORDER** (use **ASC** for ascending or **DESC** for descending).

```
select distinct city
from customers;
```

```
select distinct city
from customers
order by city;
```

SELECT & WHERE

The WHERE clause limits the rows returned by a **SELECT** query. The WHERE keyword is followed by a logical predicate. In the predicate we can use:

- Comparison operators like: =, >, <, >=, <=, LIKE.
- Logical operators like: AND, OR, NOT.
- Parenthesis ().

```
select *
from Customers
WHERE city = 'milan'
   or city = 'liverpool'
```

```
select *
from Sales
where product_name ='xyz'
  and quantity > 5
```

use of parenthesis

```
select *
from Sales
where
product_name ='abc'
or (product_name ='xyz'
 and quantity > 5)
```

Università Bocconi MILANO

SELECT and CASE

The CASE statement has the functionality of an IF-THEN-ELSE statement.

```
CASE
  WHEN condition_1 THEN
   {...value when condition_1 is TRUE...}
[ WHEN condition_2 THEN
   {... value when condition_2 is TRUE...} ]
[ WHEN condition_n THEN
   {... value when condition_n is TRUE...} ]

[ ELSE
   {... value when all conditions were FALSE...} ]
END CASE;
```

```
select
  *,
  case when city='milan' then 'Italy'
       when city  IN ('liverpool','london') then 'UK'
       else 'N/A'
  END as Country
from Customers
```

UNION and UNION ALL

The SQL UNION operator is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements. The UNION ALL operator behaves like UNION but it doesn't remove the duplicates.

```
select id, descr
from t1
union
select id, descr
from t2
```

Data Query Language (DQL) – Understanding Joins

Definition: Cartesian product

Characteristics of a Cartesian product

- Output or intermediate result of FROM clause.
- Combine all possible combinations of two sets.
- In SQL queries, usually not desired.

Name
Davis
Funk
King

×

Product
Alice Mutton
Crab Meat
Ipoh Coffee

=

Name	Product
Davis	Alice Mutton
Davis	Crab Meat
Davis	Ipoh Coffee
Funk	Alice Mutton
Funk	Crab Meat
Funk	Ipoh Coffee
King	Alice Mutton
King	Crab Meat
King	Ipoh Coffee

Università Bocconi MILANO

JOINS

A JOIN operation puts together the columns of two tables, by matching the rows based on a predicate.

Join Type	Description
Cross	Combines all rows in both tables (creates Cartesian product)
Inner	Starts with Cartesian product; applies filter to match rows between tables based on predicate
Outer	Starts with Cartesian product; all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders

CROSS JOIN

```
SELECT buyer_name, qty
 FROM buyers
 CROSS JOIN sales
```

buyers

buyer_name	buyer_id
Adam Brun	1
Lucy Thin	2
Eva Unirt	3
Pil Trun	4

sales

buyer_id	prod_id	qty
1	2	25
1	3	5
4	1	65
3	5	11
4	2	1003

Risultato

buyer_name	qty
Adam Brun	25
Adam Brun	5
Adam Brun	65
Adam Brun	11
Adam Brun	1003
Lucy Thin	25
Lucy Thin	5
Lucy Thin	65
Lucy Thin	11
Lucy Thin	1003
Eva Unirt	25
...	...

INNER JOIN

```
SELECT buyer_name, sales.buyer_id, qty
FROM buyers
  INNER JOIN sales
ON buyers.buyer_id = sales.buyer_id
```

buyers

buyer_name	buyer_id
Adam Brun	1
Lucy Thin	2
Eva Unirt	3
Pil Trun	4

sales

buyer_id	prod_id	qty
1	2	25
1	3	5
4	1	65
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Brun	1	25
Adam Brun	1	5
Pil Trun	4	65
Eva Unirt	3	11
Pil Trun	4	1003

LEFT OUTER JOIN

```
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers  LEFT OUTER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
```

buyers

buyer_name	buyer_id
Adam Brun	1
Lucy Thin	2
Eva Unirt	3
Pil Trun	4

sales

buyer_id	prod_id	qty
1	2	25
1	3	5
4	1	65
3	5	11
4	2	1003

Risultato

buyer_name	buyer_id	qty
Adam Brun	1	25
Adam Brun	1	5
Pil Trun	4	65
Eva Unirt	3	11
Pil Trun	4	1003
Lucy Thin	NULL	NULL

OUTER JOINS

- Return all rows from first table, only matches from second:

```
FROM t1 LEFT OUTER JOIN t2 ON
  t1.col = t2.col
```

- Return all rows from second table, only matches from first:

```
FROM t1 RIGHT OUTER JOIN t2 ON
  t1.col = t2.col
```

- Return all matching rows + rows from first table with no match in second + rows from second table with no match in the first:

```
FROM t1 FULL OUTER JOIN t2 ON
  t1.col = t2.col
```

Handling NULLs

- SQL uses NULLs to mark missing values. With no missing values, predicate outputs are TRUE or FALSE only (5 > 2, 1 = 1). With missing values, outputs can be TRUE, FALSE or UNKNOWN (NULL > 99, NULL = NULL).
- Predicates return UNKNOWN when comparing missing value to another value, including another missing value.
- Query filters (ON, WHERE, HAVING) filter out UNKONWNs
- Testing for NULL, use **IS NULL** or **IS NOT NULL** rather than = NULL or <> NULL.
- Replacing NULLS, use the **COALESCE** function.

```
select
    c.last_name,
    COALESCE(s.product_name,'N/A') product_name,
    COALESCE(s.quantity,0) quantity
from Customers c
    left outer join Sales s on c.customer_id = s.customer_id
```

Lesson 5 – Querying a Relation Database: the SQL language part II

Grouping and Summarizing Data

Aggregate Functions

- Return a scalar value (with no column name)
- Ignore NULLs except in COUNT(*)
- Can be used in: SELECT, HAVING, and ORDER BY clauses
- Frequently used with GROUP BY clause.

```
select
 sum(i.quantity) TotalQt,
 count(*) nr_of_sales
from  OrderItem i
```

```
select
p.productname,
sum(i.quantity) TotalQt,
count(*) nr_of_sales
from  OrderItem i
    join Product p on p.id = i.productid
group by  p.productname
```

Aggregate Functions

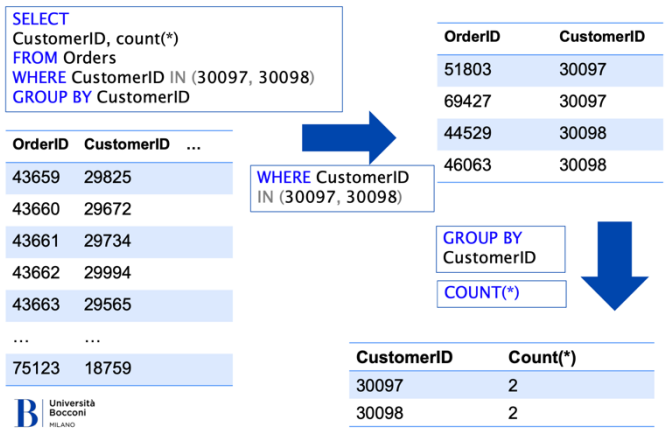
- SUM
- MIN
- MAX
- AVG
- COUNT
- STDEV
- VAR
- VARP

Group By

GROUP BY creates groups for output rows, according to a unique combination of values specified in the GROUP BY clause.
The WHERE clause filters the rows before the aggregation is applied.
We use **GROUP BY + an aggregate function** every time we want to compute an aggregation for each of the groups.
Detail rows are "lost" after the GROUP BY clause is processed.

```
SELECT <select_list>
FROM <table_source>
WHERE <search_condition>
GROUP BY <group_by_list>;
```

How GROUP BY works



Filtering Aggregates

HAVING clause provides a search condition that each group must satisfy.
HAVING clause is processed after GROUP BY.

```
SELECT
 p.productname, sum(i.quantity) TotalQt, count(*) nr_of_sales
FROM  OrderItem i
    join Product p on p.id = i.productid
GROUP BY  p.productname
HAVING sum(i.quantity) > 60
```

COUNT DISTINCT
How many different customers bought placed an order?

```
select
    count(distinct customerid) no_of_distinct_customers
from Orders o
```

SubQueries
Subqueries are nested queries: queries within queries.
Results of inner query passed to outer query.

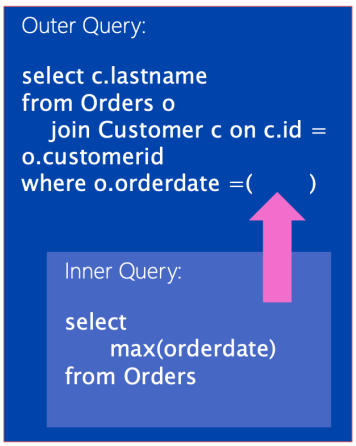
- Inner query acts like an expression from perspective of outer query.

Subqueries can be self-contained or correlated.

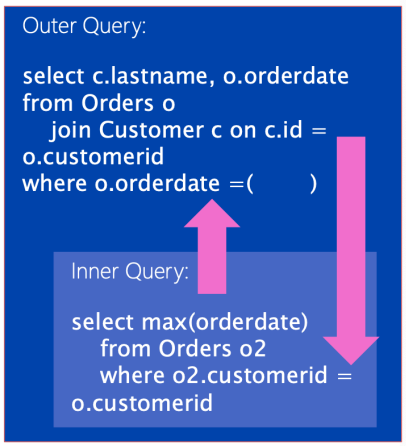
- Self-contained subqueries have no dependency on outer query.
- Correlated subqueries depend on values from outer query.

Subqueries can be scalar, multi-valued, or table-valued.

Self-Contained Subquery:



Correlated Subquery:



EXISTS and NOT EXISTS
When a subquery is used with the keyword EXISTS, it works only as an existence test.

- No rows passed back to outer query.

EXISTS evaluates to TRUE or FALSE

- If any rows are returned by the subquery, EXISTS returns TRUE.
- If no rows are returned, EXISTS returns FALSE.

Syntax: `WHERE [NOT] EXISTS (subquery)`

Customer with at least 1 order:

```
select *
from Customer c
where exists(
    select *
    from Orders o
    where o.customerid = c.id
)
```

Customer with No orders:

```
select *
from Customer c
where not exists(
    select *
    from Orders o
    where o.customerid = c.id
)
```

Analytical Functions
Set of SQL functions for analytical tasks. Analytical functions make many SQL tasks easy.
Some functions:

- ROW_NUMBER()
- LAG/LEAD
- COUNT/MIN/MAX/SUM/AVG with the OVER clause.

There's no loss of details like we have with the GROUP BY operation.

ROW_NUMBER
ROW_NUMBER provides a simple way to get row numbering in a result sets. The row numbers can then be used for many purposes.
Syntax: ROW_NUMBER() OVER(PARTITION BY col1, col2, … ORDER BY col1, col2, …)

Example: find the largest order for each customer.

```
select
*
from (
  select c.LastName, o.*,
  row_number() over(partition by o.customerid order by TotalAmount desc) as rn
  from Orders o
    join Customer c on c.id = o.customerId
 ) t
 where rn=1
```

The OVER clause

The OVER clause consists of 2 parts:

- PARTITION BY, which defines the partitioning criterion.
- ORDER BY which specifies the order we want to apply to the records inside each partition.

TotalAmount	rn	LastName	CustomerId
1813	1	Anders	1
670.8	2	Anders	1
625.2	3	Anders	1
517.8	4	Anders	1
440	5	Anders	1
3730	1	Trujillo	2
2490.5	2	Trujillo	2
1863.4	3	Trujillo	2
1444.8	4	Trujillo	2

LAG & LEAD

LAG returns a value taken from n rows before the current row. (n is a parameter). If no value is found, the function returns the specified default value.

Syntax: LAG(column_name, n, default_value) OVER(PARTITION BY partition_by_cols ORDER BY order_by_cols).

LEAD returns a value taken from n rows past the current row. (n is a parameter). If no value is found, the function returns the specified default value.

Syntax: LEAD(column_name, n, default_value) OVER(PARTITION BY partition_by_cols ORDER BY order_by_cols).

LAG Example

```
select   o.OrderDate,
  c.LastName, o.TotalAmount
  LAG(TotalAmount,1,0) over(partition by o.customerid order by OrderDate asc) as PreviousPurchase
  from Orders o
    join Customer c on c.id = o.customerId
```

OrderDate	customerid	LastName	TotalAmount	PreviousPurchase
2018-07-04T00:00:00Z	1	Anders	440	0
2018-07-05T00:00:00Z	1	Anders	1813	440
2018-07-05T00:00:00Z	1	Anders	670.8	1813
2018-07-08T00:00:00Z	1	Anders	625.2	670.8
2018-07-15T00:00:00Z	1	Anders	517.8	625.2
2018-07-04T00:00:00Z	2	Trujillo	1863.4	0
2018-07-06T00:00:00Z	2	Trujillo	3730	1863.4
2018-07-07T00:00:00Z	2	Trujillo	1444.8	3730
2018-07-09T00:00:00Z	2	Trujillo	2490.5	1444.8

No previous value

No previous value

Aggregation Function with the OVER Clause

The OVER clause allows computing the aggregations for a partition (or for all the rows in the result set). With the OVER clause the aggregations are returned along with each single row.

```
select
  o.customerid,
  c.LastName,
  TotalAmount,
  SUM(TotalAmount ) over() as GrandTotal
from Orders o
    join Customer c on c.id = o.customerId
```

customerid	OrderNumber	LastName	TotalAmount	GrandTotal
1	542378	Anders	440	13595.5
2	542379	Trujillo	1863.4	13595.5
1	542380	Anders	1813	13595.5
1	542381	Anders	670.8	13595.5
2	542382	Trujillo	3730	13595.5
2	542383	Trujillo	1444.8	13595.5
1	542384	Anders	625.2	13595.5

Same value for each row

```
select
  o.customerid,
  c.LastName,
  TotalAmount,
  SUM(TotalAmount) over(partition by customerid) as
GrandTotal
from Orders o
    join Customer c on c.id = o.customerId
```

customerid	LastName	TotalAmount	GrandTotal
1	Anders	1813	4066.8
1	Anders	670.8	4066.8
1	Anders	440	4066.8
1	Anders	625.2	4066.8
1	Anders	517.8	4066.8
2	Trujillo	2490.5	9528.7
2	Trujillo	1863.4	9528.7
2	Trujillo	3730	9528.7
2	Trujillo	1444.8	9528.7

Same value for each row within the same partition

Same value for each row within the same partition

Windowing

In addition to PARTITION BY and ORDER BY the OVER clause allows us to specify a window of rows to which the aggregation function is applied.
Keywords:

- UNBOUNDED PRECEDING, it means that the window starts from the first row (inside a partition and using the order by criteria).
- N PRECEDING (where N is a number), it means that the window starts from n rows before the current row.
- CURRENT ROW, it specifies the current row as start or end of a window.
- UMBONDED FOLLOWING / N FOLLOWING, same as above, but it refers to the following rows.

Windowing Examples

```
select
  o.customerid,
  c.LastName,
  TotalAmount,
  AVG(TotalAmount ) over(partition by customerid
                         order by OrderDate
                         rows between 2 preceding and current row  ) as MovingAverage
from Orders o
    join Customer c on c.id = o.customerId
```

customerid	LastName	TotalAmount	MovingAverage
1	Anders	440	440
1	Anders	1813	1126.5
1	Anders	670.8	974.6
1	Anders	625.2	1036.333333
1	Anders	517.8	604.6
2	Trujillo	1863.4	1863.4
2	Trujillo	3730	2796.7
2	Trujillo	1444.8	2346.066666

1° row only
1° and 2° rows only
1°, 2° and 3° rows
2°, 3° and 4° rows

```
select
  o.customerid,
  c.LastName,
  TotalAmount,
  SUM(TotalAmount ) over(partition by customerid
                         order by OrderDate
                       rows between unbounded preceding and current row  ) as RunningTotal
from Orders o
    join Customer c on c.id = o.customerId
```

customerid	LastName	TotalAmount	RunningTotal
1	Anders	440	440
1	Anders	1813	2253
1	Anders	670.8	2923.8
1	Anders	625.2	3549
1	Anders	517.8	4066.8
2	Trujillo	1863.4	1863.4
2	Trujillo	3730	5593.4
2	Trujillo	1444.8	7038.2
2	Trujillo	2490.5	9528.7

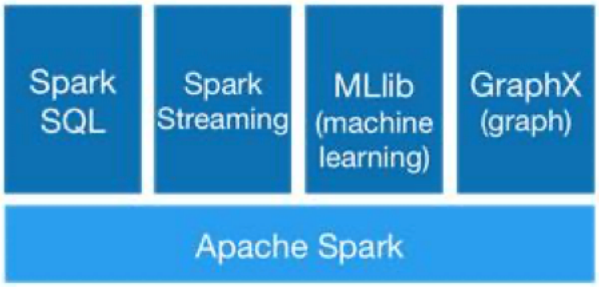
1° row only
1° and 2° row only
1°, 2° and 3° row
1°, 2°, 3° and 4° row
All rows in customerid=1 partition

Lesson 6 – Lighting Fast Computing with Spark I

Spark

Apache Spark is a fast, in-memory data processing engine. APIs in Scala, Java, Python & R.

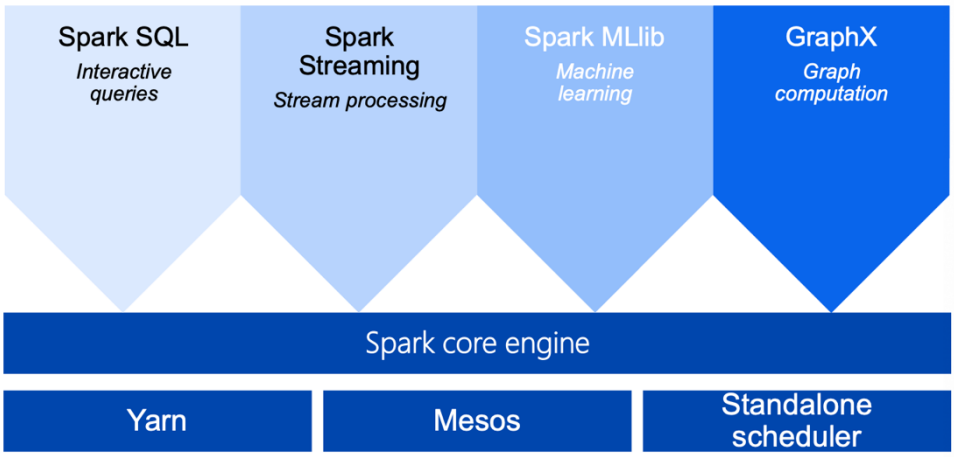
Spark on Apache Hadoop YARN enables deep integration with Hadoop and other YARN enabled workloads in the enterprise. Spark powers a stack of high-level tools including Spark SQL, MLlib for machine learning, GraphX and Spark Streaming.



Spark History

- 2009: Hadoop Subproject, by AMPLab (UCLA)
- 2010: Open Source
- 2013: Apache Project
- 2016: Spark 1.6.2 & 2.0.0
- 2018: Spark 2.3.x
- 2018: Spark 2.4.x

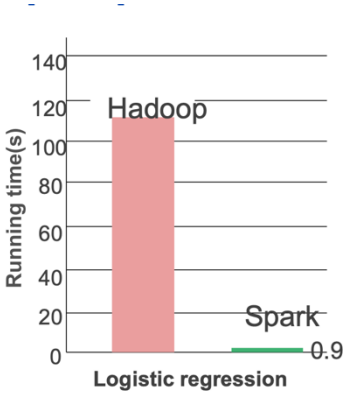
Unified Framework



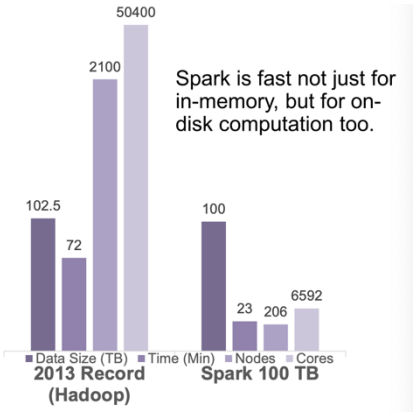
Spark scales well

Some examples of real usage:

Spark Performance



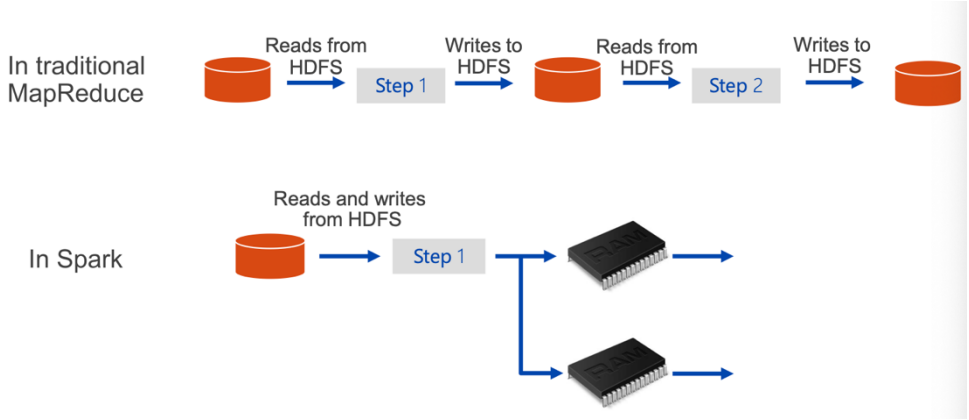
Logistic regression on a 100-node cluster with 100 GB of data.



Spark is the 2014 Sort Benchmark winner.

3x faster than 2013 winner (Hadoop).

What makes Spark fast?



How Spark works

The **Driver** program contains all the instructions that Spark must execute (for example: read a file, aggregate the data, save the aggregated data to another file).
In the Driver program the programmer creates the **SparkSession** object that takes care of:

- Connecting to the Spark cluster (it connects to the cluster manager).
- Sending the program to the cluster and make it execute.

Spark have different **cluster managers**:

- It can use YARN as cluster manager if Spark lives on the same cluster as Hadoop.
- It can use another Apache cluster manager (Mesos).
- Spark itself contains a cluster manager tool (standalone cluster mode).

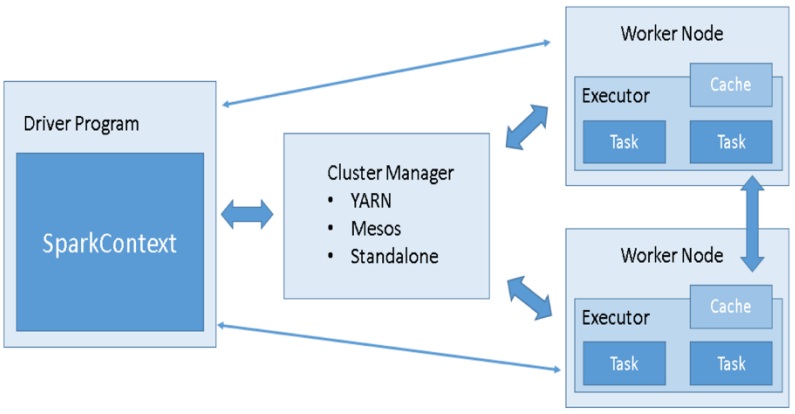
Once the Driver program is connected to the cluster manager it sends lines of code that contain the Spark commands.
The **Worker Nodes** (generally one for each physical node of the cluster) receive the commands and allocate one or more Executors.
The **Executor** is a JVM service that takes in charge:

- The creation of one or more tasks that physically execute the Spark instructions.
- The allocation of enough memory to execute each task.

Executors in different Worker Nodes can exchange data if needed (Data shuffle operations).
Shuffles occur when the Spark program:

- Aggregates data (data with the same aggregation key may be located in different Worker Nodes).
- Sorting data.

Architecture



Spark Distributions

The **Apache Spark** distribution is relatively easy to install. It works on all the operating systems.
There's another Spark distribution: **Databricks** that also provides a cloud version of Spark.
Cloud services:

- Microsoft HDInsight
- Google Cloud
- IBM Blumix

```
from pyspark.sql import SparkSession

spark = SparkSession
    .builder
    .master("spark://myserver:7077")
    .appName("myApp")
    .getOrCreate()
```

Connecting to the Spark cluster

The master parameter for a SparkContext determines which cluster to use:

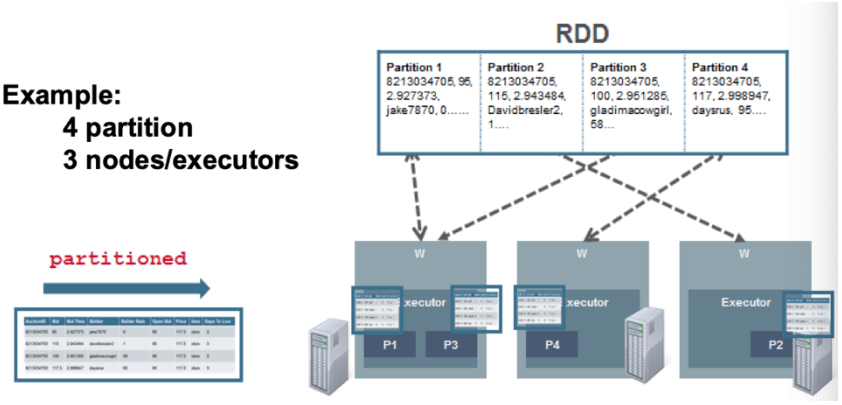
Master	Description
local	Local execution with one thread
local[K]	Local execution with K worker
local[*]	Local execution with a worker number set to the number of cores
spark://HOST:PORT	Connection to a standalone cluster master. Defautport is: 7077
mesos://HOST:PORT	Connection to a Mesos cluster. Defautport is: 5050
yarn	Connection to a YARN cluster. The cluster address is located in some configuration files. The configuration folders are defined in the HADOOP_CONF_DIR and YARN_CONF_DIR system variables.

Resilient Distributed Datasets (RDD)

Resilient **D**istributed **D**atasets (RDD) are the primarily abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel.

- An RDD is comprised of **partitions**.
- Each partition is managed by an **executor** (generally one executor per node).
- Each executor can manage more than one partition of the same RDD.

Example:
4 partition
3 nodes/executors



RDD Examples

There are different types of RDD. For example we can have:

- RDD of strings (created when we read a file).
- RDD of pairs (key/value pairs, programmatically created).

How Spark works

The RDD is an immutable collection of objects that is **partitioned** and **distributed** across multiple physical nodes of a YARN cluster and that can be operated in parallel.

There are currently two types:

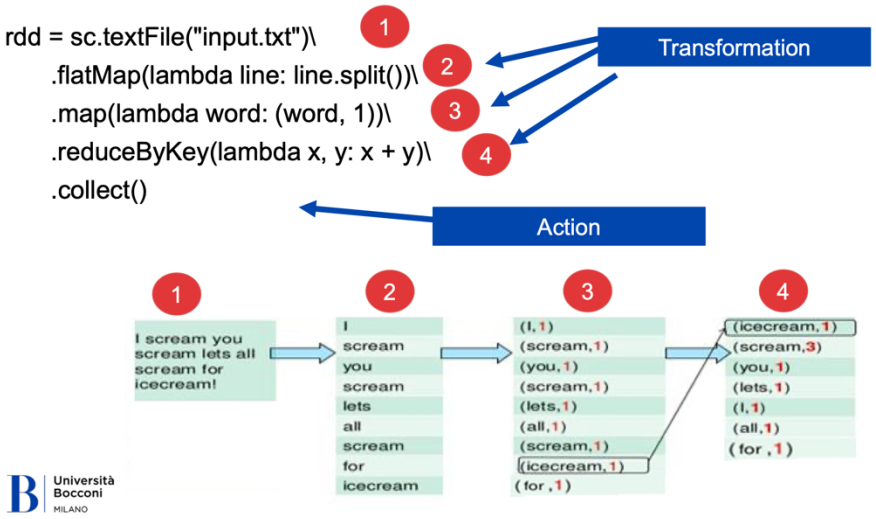
- Hadoop (or other FS) datasets – Typically, RDDs are instantiated by loading data from a shared file system, HDFS, HBase, or any data source offering a Hadoop InputFormat.
- Parallelized collections – take an existing Scala/Python collection and run functions n it in parallel.

Once an RDD is instantiated, you can apply a series of operations. All operations fall into one of two types: transformations or actions.

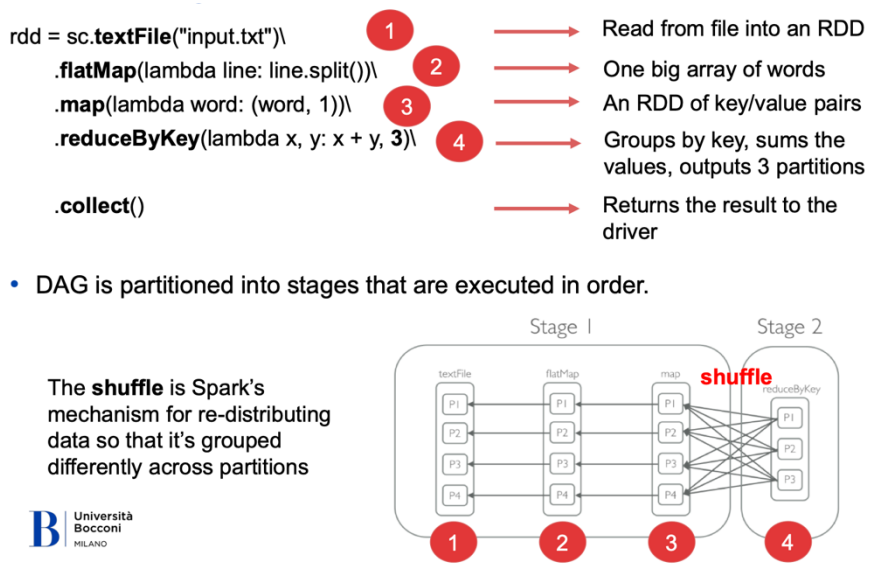
- **Transformation** operations create new datasets from an existing RDD and build out the processing DAG that can then be applied on the partitioned dataset across the YARN cluster.
- An **Action** operation, on the other hand, executes DAG and returns a value.

DAG = Directed Acyclic Graph. It's the **execution workflow** of a Spark program.

Example



DAG, Stages & Shuffle

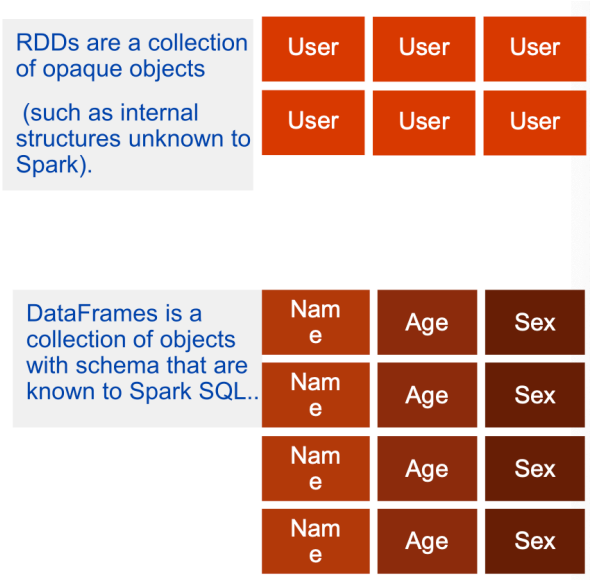


Spark SQL
Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine. A DataFrame is a distributed collection of data organized into named columns.

- It is conceptually **equivalent to a table** in a relational database or a data frame in R/Python, but which richer optimizations under the hood.

DataFrames
A distributed collection of data organized into named columns. Similar to RDD with schema. Conceptually equivalent to tables in relational database, or to DataFrames in R/Python.
With domain-specific functions designed for common tasks:

- Metadata
- Sampling
- Project, filter, aggregation and join.
- UDFs.



Creating DataFrames from data sources
A Spark data source can read in-data to create DataFrames, which has a schema that Spark understands.
Examples include: JSON files, JDBC source, Parquet (*).

```
df = spark.read.json ("somejasonfile.json")

df = spark.read.parquet ("someparquetsource")

df = spark.read.format("jdbc").option("url", jdbcUrl ).option("dbtable",
"TABLE_NAME").option("user", "USER_NAME").option("password", "PASSWORD").load()
```

(*) **Parquet** is an optimized file format. It contains both data and metadata (column names and data types). It stores data in an efficient columnar format.

DataFrame API
DataFrames provide a domain-specific language for structured data manipulation in Scala, Java and Python.

```
df = spark.read.json("examples/src/main/resources/people.json")
df.show()
df.printSchema()
df.select("name").show()
df.select(df['name'], (df['age'] + 1).alias("AGE")).show()
df.filter(df['age'] > 21).show()
df.groupBy("age").count().show()
```

Dataframe registered as a table
A DataFrame can be registered as a table that can then be used in SQL queries.

```
# First create a DataFrame
df = spark.read.json("examples/src/main/resources/people.json")

#Register the DataFrame as a temporary table. Temp tables exist only during
lifetime of this session.
people = df.createOrReplaceTempView("People")

#execute a SQL query on the table. The query returns a DataFrame.
df2 = spark.sql("select Age as Years from People where age > 40 and age <= 50")

spark.catalog.dropTempView("People")
```

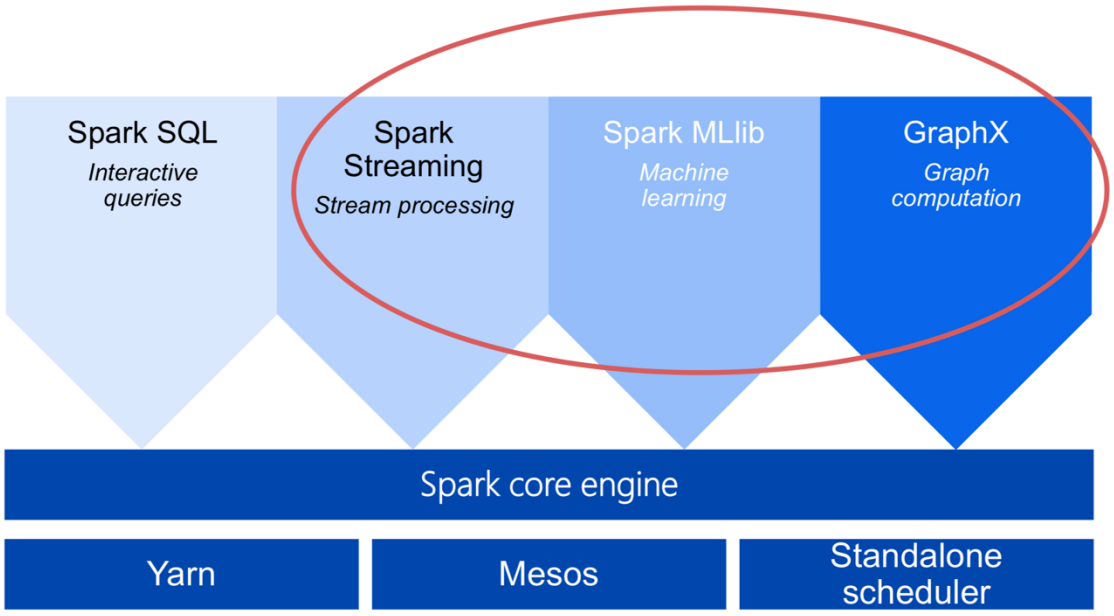
Persistent Tables
DataFrames can also be saved as persistent tables into Hive metastore using the saveAsTable command.

- An existing Hive deployment is not necessary to use this feature. Spark will create a default local Hive metastore (using Derby) for you.

SaveAsTable will materialize the contents of the DataFrame and create a pointer to the data in the Hive metastore. Persistent tables will still exist even after your Spark program has restarted, as long as you maintain your connection to the same metastore.

Lesson 7 – Lighting Fast Computing with Spark II

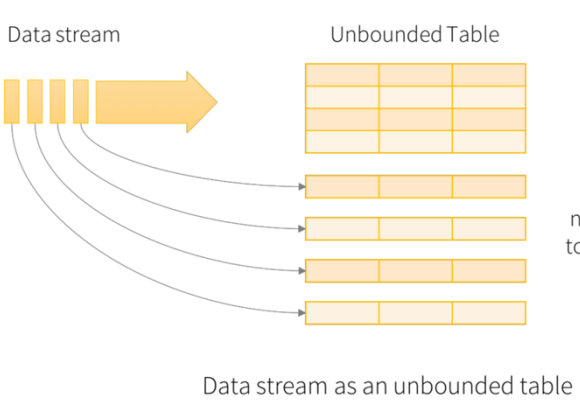
Unified Framework



Spark Streaming

- **Streaming Data** is data that is continuously generated by a data source. Such data should be processed incrementally.
- **Spark Structured Streaming**: enables scalable, high-throughput, fault-tolerant stream processing live data streams.
  - Built on the Spark SQL engine.
  - You can express your streaming computation the same way you would express a batch computation on static data.
  - The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive.

Streaming



new data in the data stream
=
new rows appended to a unbounded table

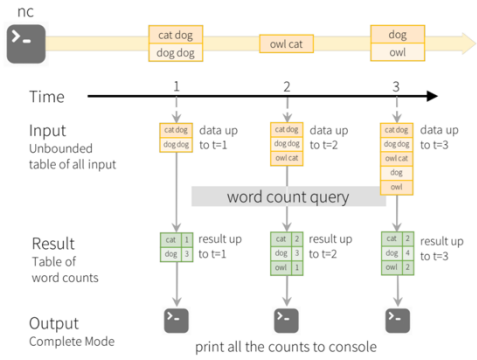
```
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Split the lines into words
words = lines.select(
   explode(
       split(lines.value, " ")
   ).alias("word")
)

# Generate running word count
wordCounts = words.groupBy("word").count()
```

```
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```



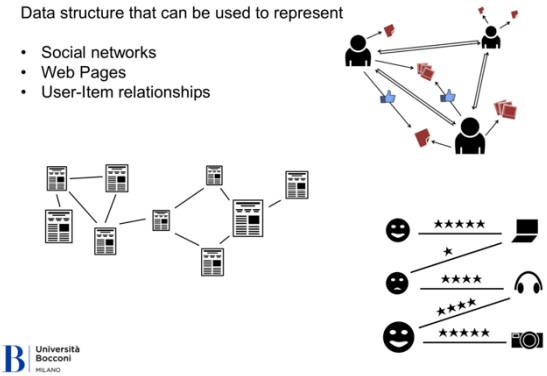
Stream Exercise

- Loading sample data.
- Perform some interactive analysis (on the static data).
- Read the files as a stream.
- Do the same analysis as in point 2.
- See the streaming result.

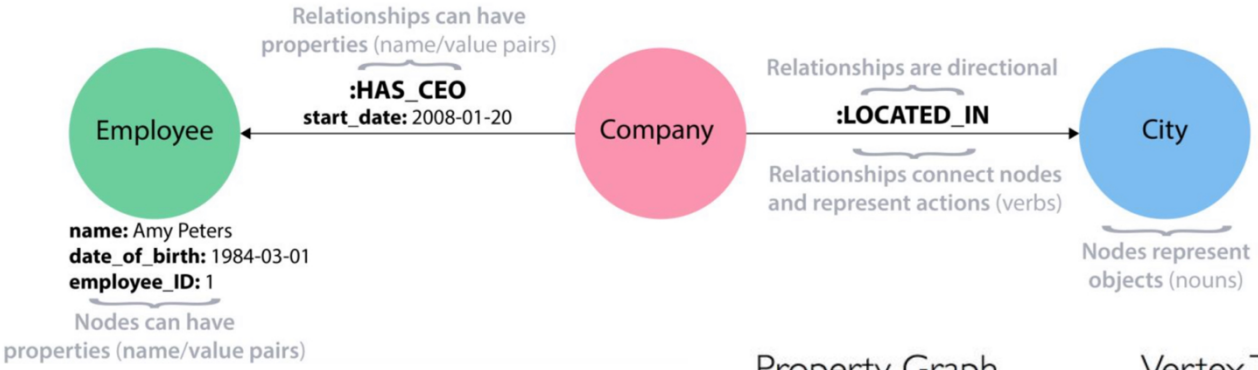
Spark GraphX

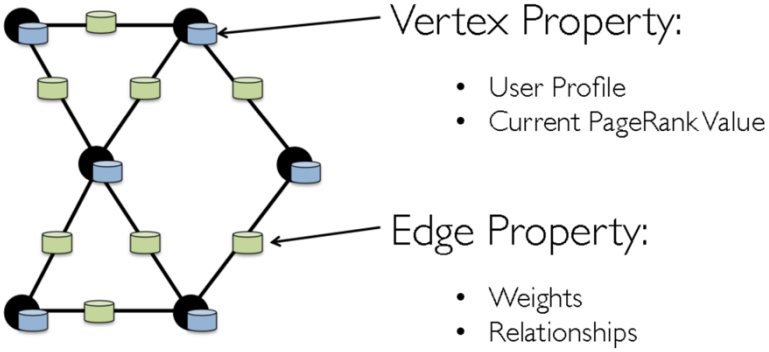
Graphs

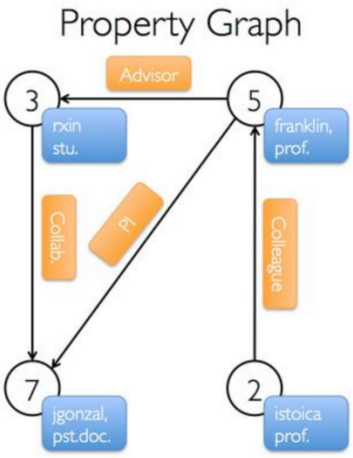
- The graph data structure is comprised of:
  - **Nodes** are the entities in the graph.
    - They can hold any number of attributes (key-value pairs) called *properties*.
    - Nodes can be tagged with *labels*, representing their different roles in your domain.



- **Relationships** provide directed, named, semantically-relevant connections between two nodes.
  - A relationship always has a direction, a type, a start node, and an end node.
- Such a graph is called **Property Graph**.







Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

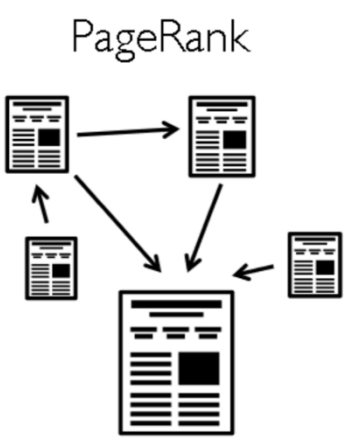
Edge Table

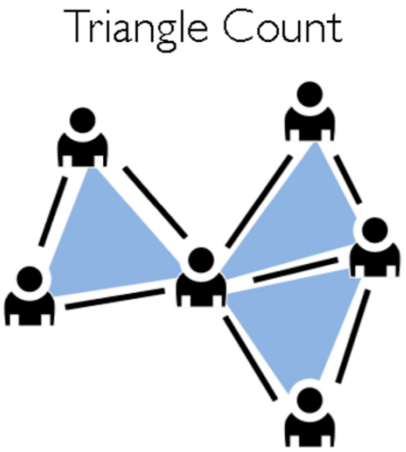
SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

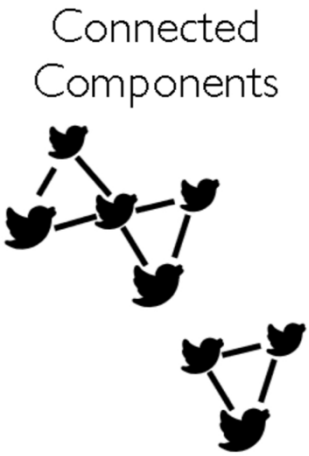
Graph X

- Graph X is a new component in Spark for graphs and graph-parallel computation.
- At a high-level, GraphX extends the Spark RDD by introducing a new Graph abstraction:
  - A directed multi graph with properties attached to each vertex and edge.
- In addition, GraphX includes a collection of graph algorithms and builders to simplify graph analytics tasks.

Built-In Algorithms







GraphX Exercise

- Load the library into the Spark environment.
- Create the graph data structures.
- Basic calculations.
- Built-in Algorithms.

Spark Machine Learning
Typical ML Process

- Understand the data
- Split the data into training set and test set
- Prepare the data
  - Data cleansing
  - Specific data preparation for the algorithms
    - Normalization
    - Conversion of categorical variables to numerical variables
- Train the algorithm using the training set
- Test the model on the test set

Spark MLlib is Spark's machine learning library. Its goal is to make practical machine learning scalable and easy.
MLlib tools:

- **ML Algorithms**: common learning algorithms such as classification, regression, clustering, and collaborative filtering.
- **Featurization**: feature extraction, transformation, dimensionality reduction and selection.
- **Pipelines**: tools for constructing, evaluating and tuning ML workflows.
- **Utilities**: linear algebra, statistics, data handling, etc.

Spark ML Algorithms
Regressions:

- Linear Regression
- Generalized Linear Regression
- Decision Tree Regression
- Random Forest Regression
- Gradient-boosted Tree Regression
- Survival Regression
- Isotonic Regression

Classification

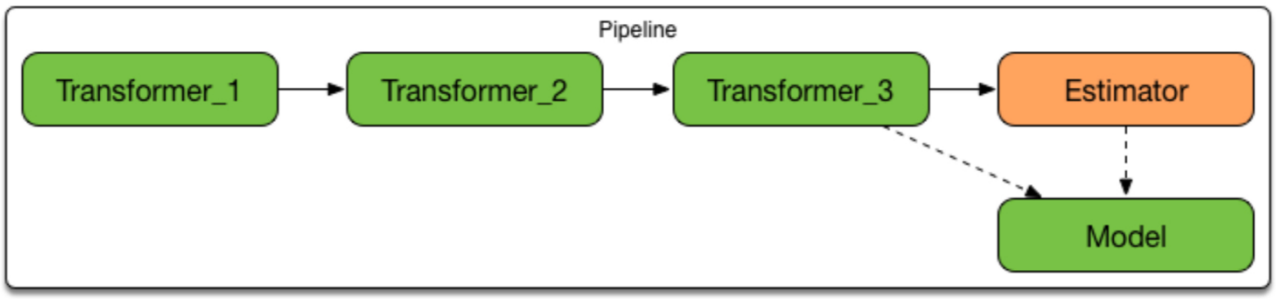
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Gradient-boosted Tree Classifier
- Multilayer Perceptron Classifier
- Linear Support Vector Machine

- One-vs-Rest Classifier (a.k.a One-vs-All)
- Naïve Bayes

Clustering & Other Supervised Algos
- K-Means
- Latent Dirichlet Allocation (LDA)
- Bisecting K-Means
- Gaussian Mixture Model (GMM)

Recommendation System
- Alternating Least Squares (ALS)
- FP-Growth
- PrefixSpan

Spark ML Pipeline
ML Pipeline API lets Spark users quickly and easily assemble and configure practical distributed Machine Learning pipelines by standardizing the APIs for different Machine Learning concepts. The nice thing of using Spark ML is that the **ML Dataset** is simply a DataFrame.



Pipeline Components
A **transformer** is a ML Pipeline component that transforms a DataFrame into another DataFrame:
- OneHotEncoder
- Binarizer
- Tokenizer

An **estimator** is an abstraction of a learning algorithm that fits a model on a dataset.
- An estimator fits a model to the input DataFrame and ParamMap to produce a Transformer (a Model) that can calculate predictions for any DataFrame – based input datasets.

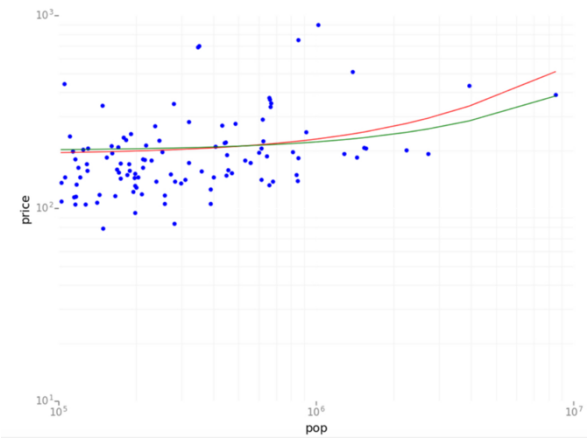
An **evaluator** is a transformer that takes DataFrames and computes metrics indicating how good a model is.
- BinaryClassificationEvaluator
- MulticlassClassificationEvaluator
- RegressionEvaluator

Spark ML Exercise
Example 1 – Logistic Regression
- Creating a toy dataset

- Creating the pipeline
- Training the model
- Evaluating against the test set

Example 2 – Linear Regression

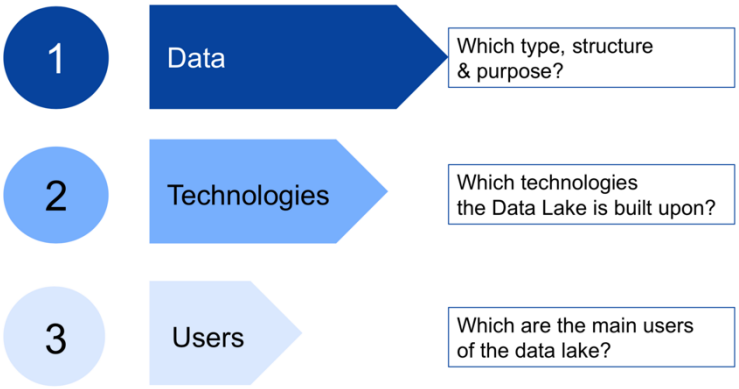
- Loading sample data
- Exploring the data
- Creating the pipeline
- Training the model
- Evaluating against the test set.



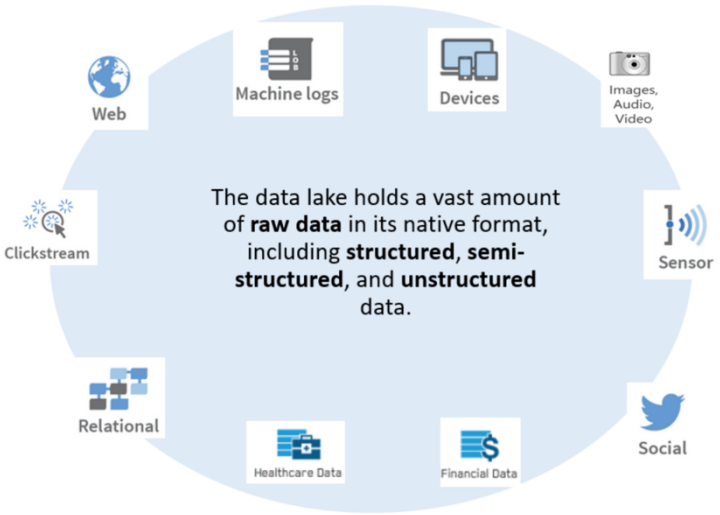
Lesson 8 – Big Data Architectures: Integrated the Big Data tools in the company's information system

Data Lake

"A **data lake** is a collection of storage instances of various data assets additional to the originating data sources. These assets are stored in a near-exact, copy of the source format. The purpose of a **data lake** is to present an unrefined view of data to only the most highly skilled analysts, to help them explore their data refinement and analysis techniques independent of any of the system-of-record compromises that may exist in a traditional analytic data store (such as a data mart or data warehouse)".

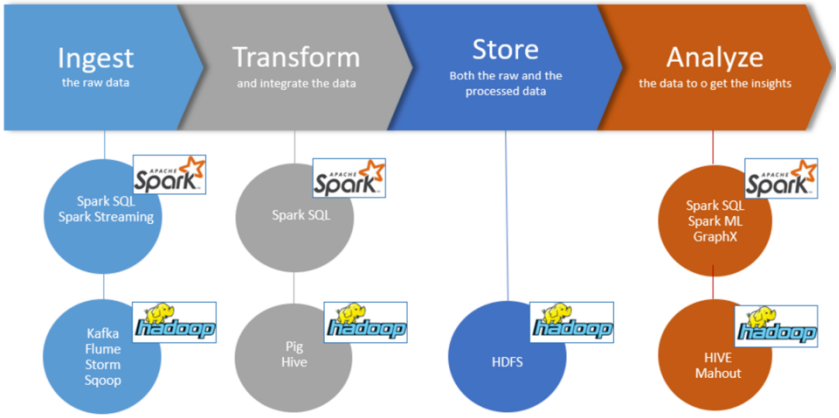


Raw Data

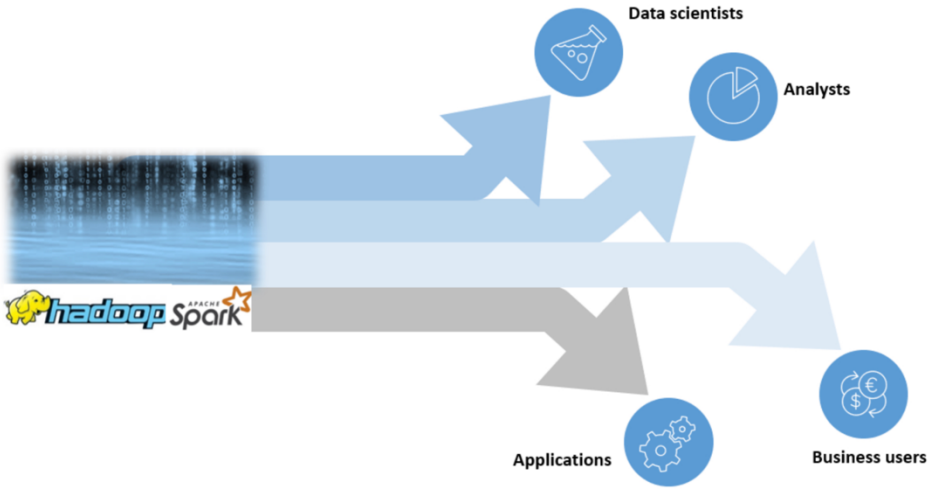


Processed Data
The data lake also contains a vast amount of **processed** (and/or semi-processed) data.

Technologies
The Data Lake is mostly based on Hadoop & Spark.

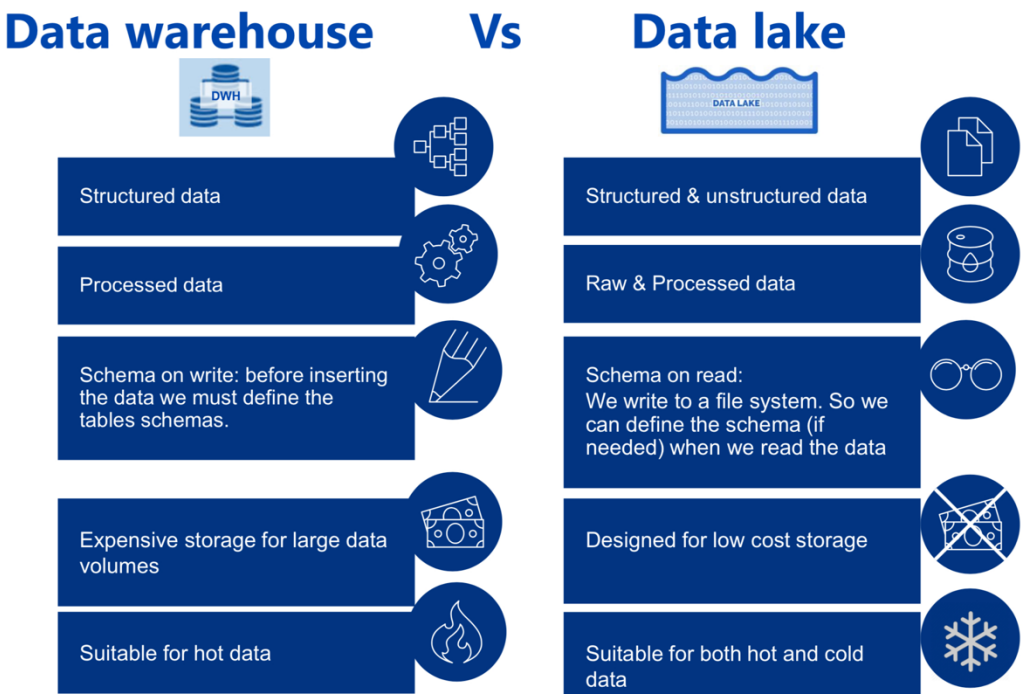


Users

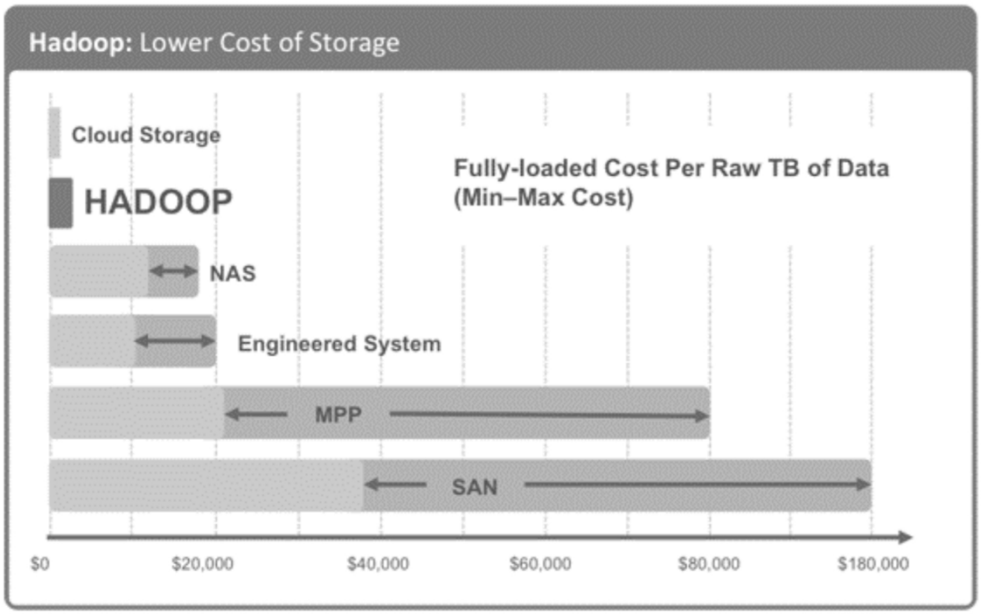


Data Lake
Rezzani's Definition:

- A data lake is a repository that holds:
  - A vast amount of raw data in its native format, including structured, semi-structured and unstructured data.
  - A vast amount of processed (and/or semi-processed) data.
- The data structure and requirements are not defined until the data is needed (schema on read).
- The data lake provides data for as reporting, analytics and machine learning tasks.
- The data lake is mostly based on the Hadoop & Spark platforms.



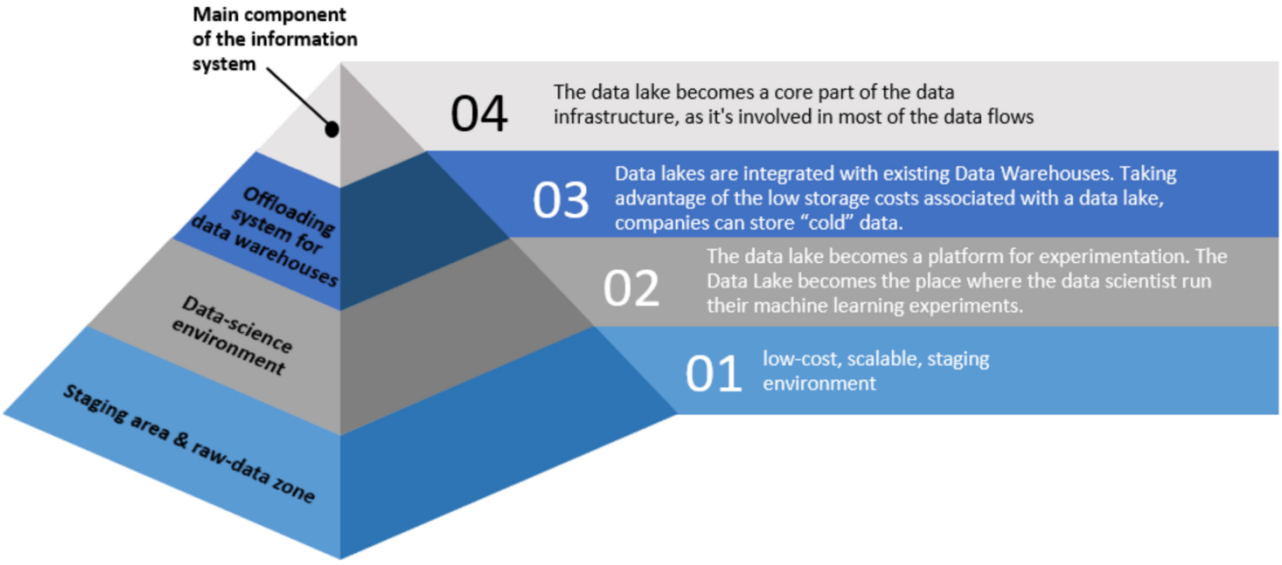
Low Cost Repository



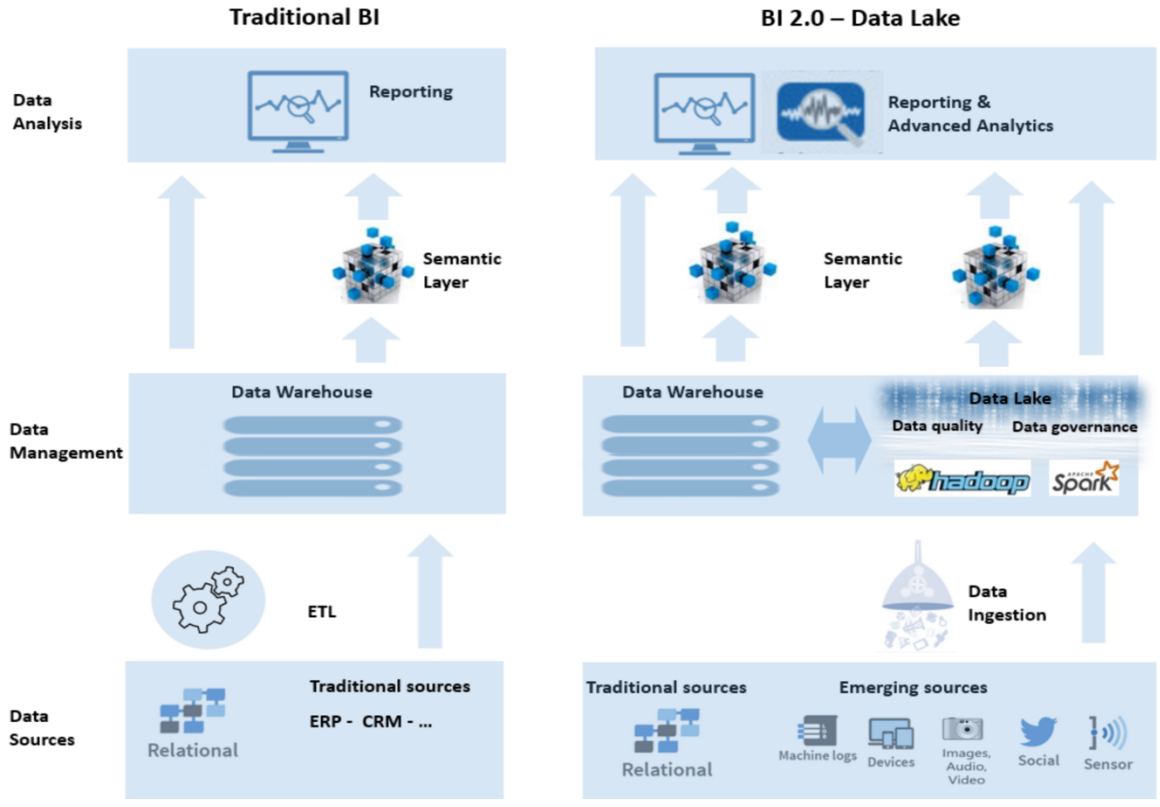
Four steps of the Data Lake adoption

- **Staging area & raw-data zone**. At the first level, the data lake can separate collect data from operational systems and servers as a low-cost, scalable, staging environment.
- **Data Science Environment**. As a second step, companies may start to use the data lake as a platform for experimentation. The Data Lake becomes the place where the data scientist run their machine learning experiments.
- **Offloading system for data warehouses**. As a further step, data lakes are integrated with existing Data Warehouses. Taking advantage of the low storage costs associated with a data lake, companies can store "cold" (rarely used) data.

- **Main component of the information system**. The data lake becomes a core part of the data infrastructure, as it's involved in most of the data flows.



Traditional Business Intelligence vs Data Lake



Data Lake Architecture: Data Flows

Data from the traditional sources can still be processed by the ETL processes, but some (or all) the ETL procedures could moved to the data lake (DL).

- So the DL becomes the repository of raw data and the place where some ETL transformations happens.

Data from the new sources go directly to the DL.

- Those data can be processed or used in complex analytical tasks (like machine learning).
- And, once we get the results, they can be moved to the data warehouse.

Data Lake Architecture: the DWH Role
The data warehouse still exists. It contains:

- The "hot" data coming from the ETL processes (whether they grab the data directly from the sources or from the DL).
- The "distilled" data coming from the analytical processes that run over the data lake.

The data warehouse allows for low-latency (interactive) queries. It's part of the Business Intelligence system that still carries out its job in the data lake era. Keeping the data warehouse means preserving the querying performances and all the reporting systems that are built on top of it.

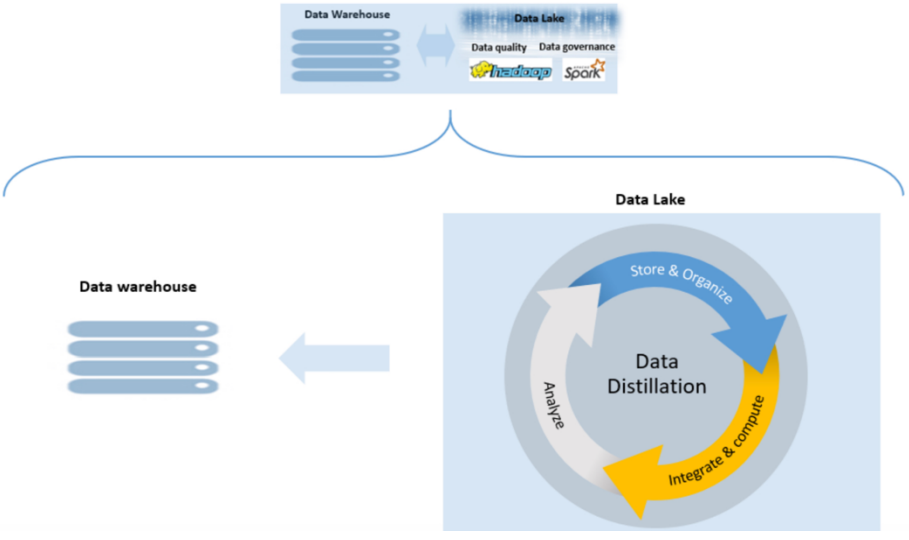
Data Lake Architecture: the DL role
The data lake contains:

- (All) the raw data, both structured and unstructured.
- The final processed data (which are structured)
  - Some of those data is pushed to the data warehouse (the hot data).
  - The "cold" data are stored in the data lake and are available with a longer latency.

It eventually takes care of all the data transformations (ETL + Analytical tasks). In the DL we can store more data: more details + greater historical depth. Supplies data:

- To the data warehouse.
- To the Data Scientists. The data scientists can use both the raw and the processed data to build both descriptive reports and predictive models.
- To the applications. The data lake could provide data services for corporate applications.

Data Lake Architecture: Data Distillation
The data lake allows more detailed and more complex analysis on a larger amount of data. Often the results of the computations are small enough to be transferred to the Data Warehouse. So, the results are injected in the Business Intelligence workflow.



Critical Success Factors for the Data Lake
A big data lake will take whatever data you give it, and that's going to be a problem. How to address it?

- **Data Governance**
- **Data Quality Process**

Many users with different profiles and needs may access the Data Lake. How can we make the usage easy for any user profile?

- **Metadata management (part of the Data Governance)**
- **User skills improvement (i.e., training)**
  - Data Scientists
  - End Users

Data Governance Policy

- Data Sources mapping (applications, input services)
- Data Flows mapping: How data is loaded into the Data Lake?
- Output services mapping:
  - Data Warehouse
  - Applications
  - Output Data Services
- Metadata Management: the Metadata make it easy to search for data and to interpret the content of the data lake.

Data Quality Process

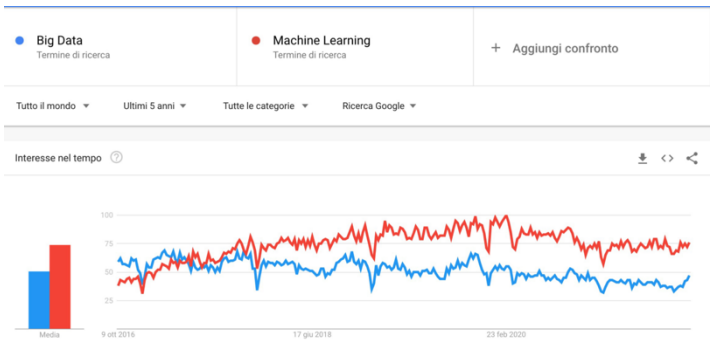
The data quality rules should be applied on the output data. If we apply data quality filters on the input data, we would prevent some data to enter the data lake.

- Data that have a medium quality level could be useful for certain applications.

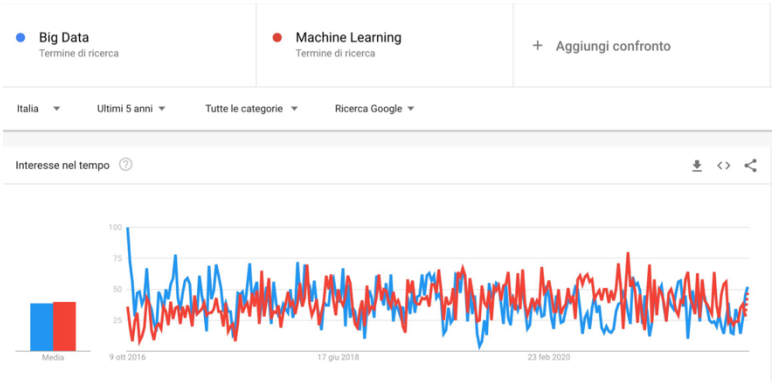
The best way to proceed is putting a "**quality label**" to the data and let the final user decide whether or not to use it.

Lesson 9 – Big Data Business Applications

The Big Data Phenomenon (In the World)



The Big Data Phenomenon (In Italy)



"Traditional" Data Sources

Operational sources referring companies daily operational activity:

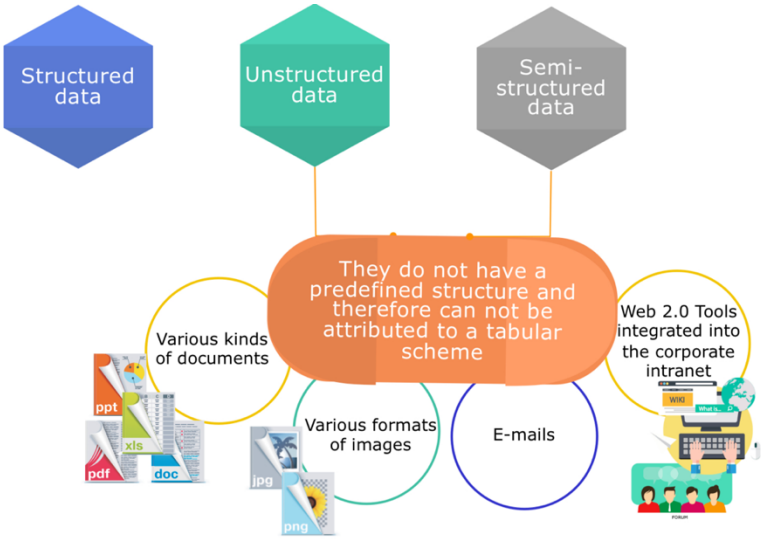
- Software for production management
- Software for purchasing management
- Software for orders and deliveries management
- Software for accounting
- Software for personnel management
- Software for customer management
- Software for back-office management
- Software for financial instruments management and measurement

"Emergent" Data Sources

Next to the data produced by management software's there might be systems closer to the production that generate huge amounts of data:

- DCS (Distributed Control System) computer systems for the control of industrial installations. Generate data on the state of the installations through sensors connected to the components running measurements at very small intervals.
- Data generated by scientific equipment for measurement and analysis
- Data generated by medical and diagnostic equipment
- High Frequency Trading Systems
- Web 2.0: Blog Posts, Tweets, Facebook comments & likes, images & videos
- IOT: Internet Of Things

Type of Data



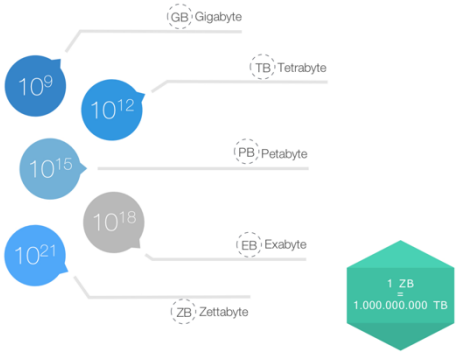
"Classical" definition of Big Data
Data with three characteristics:

- **Volume** (Large amounts of data)
- **Variety** (Variety of structures, data types, sources, Structural Complexity and Unstructured or semi-structured data)
- **Velocity** (The speed with which they are produced

The other 2 V's of the classic definition

- **Value** (It expresses a result of the use of big data: the ability to generate value)
- **Veracity** (It expresses critical issues: not everything that is on the internet or in the corporate data assets is trustworthy!)

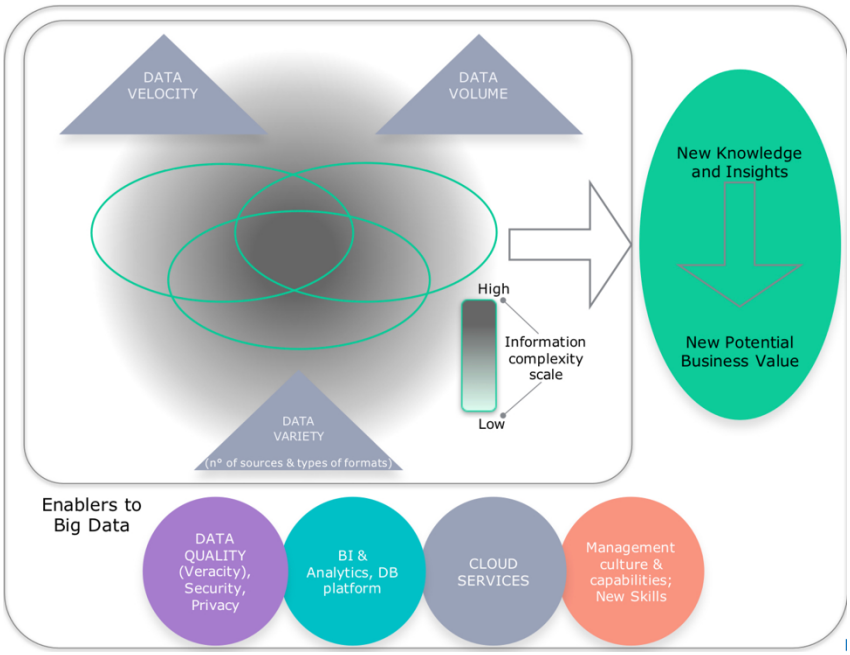
Estimations
There are certain estimates that see by 2025, the set of all the data digitally created and consumed in a year (books, videos, music, etc.), it will be equal to 180 Zettabyte.
Other estimates say 160 Zettabyte for 2025 (IDC).



But do we have to analyze them all?
Let's clarify
The name "Big" Data, the traditional definition and some publications, are misleading.
The Big Data is not only the data with large volumes. They come from the Web, but also from other sources (both new and traditional).
The impressive numbers which are found in the statistics they refer to the data created throughout the world, we hardly need to use them all…

Big Data Definition
Big Data is a new concept of **potential Business knowledge** that leverages on today **variety** of data formats and sources, on improved data **velocity** and, thus, on today increasing data **volume**, to generate new **insights** previously considered beyond our capability and to find new business value by using proper **enablers**.

Big Data Framework

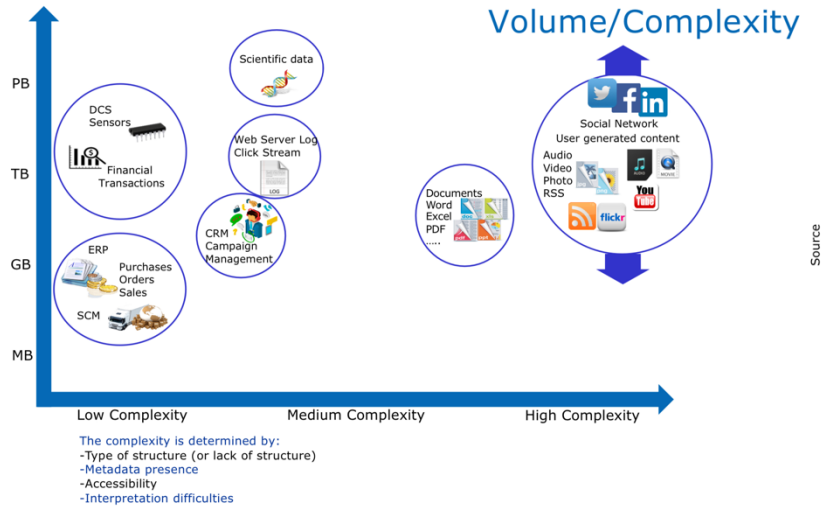


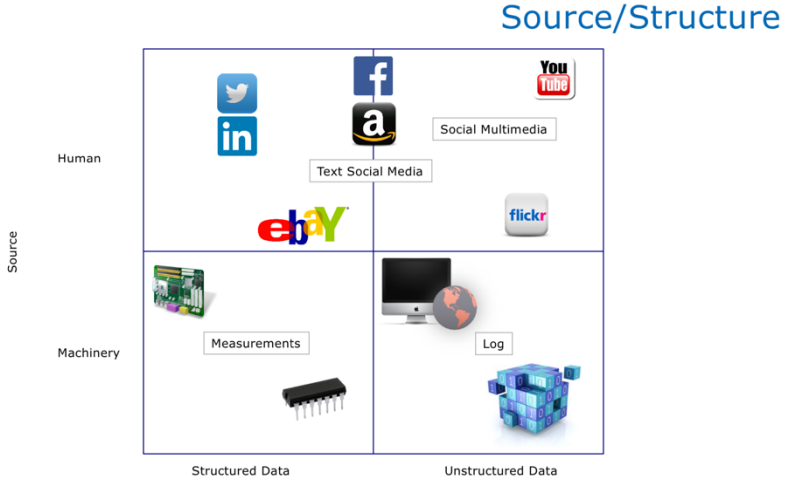
An alternative definition (more technical)

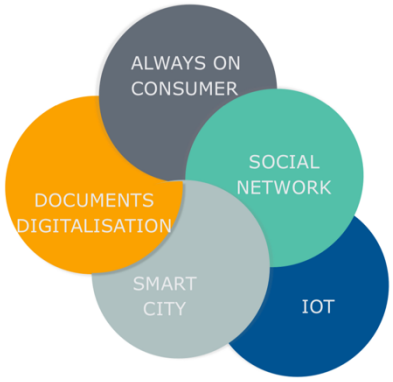
The Big Data are: Data that cannot be analyzed on a single machine or that is not convenient to analyze through traditional technologies (hw or sw).

Can then be: Data with such volumes that the analysis on RDBMS is impossible or very costly Unstructured data difficult to keep in an RDBMS. Data that may need sophisticated analytical techniques to extract value, although this is not always true. Data that arises both structured and unstructured (or semi-structured).

However: unstructured data (or semi-structured) must be processed and placed in the form of structured data, before it can be analyzed.



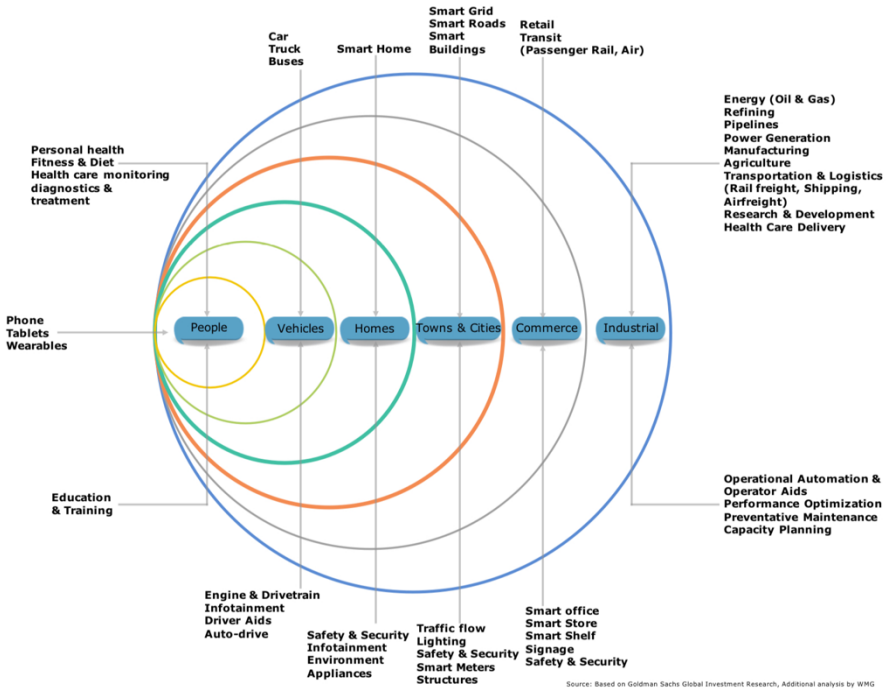




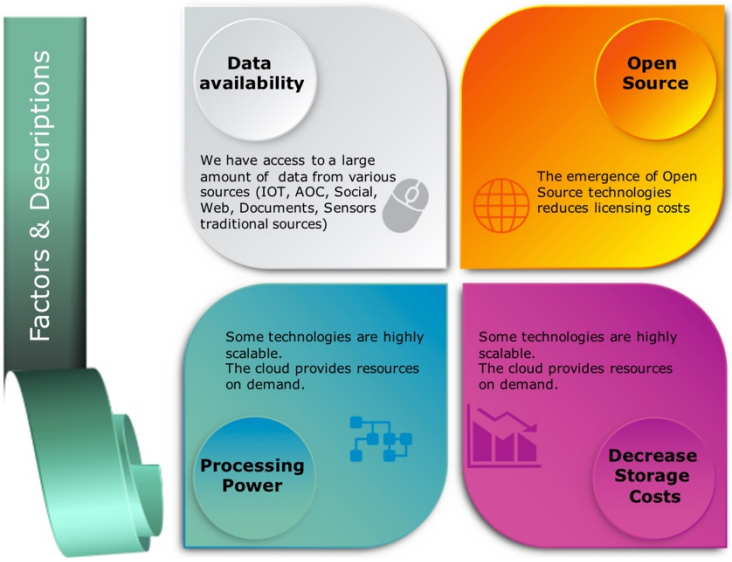
IOT (Internet Of Things)

Network of objects containing electronic parts, software, sensors and connectivity. They add value to the product or service: exchange data with the producer and with other connected devices. Interact through the Internet.

IOT & Smart Cities



Big Data Phenomenon: Triggers



Analysis Process

Example of Big Data Applications in Business

- **Reduce Time To Market**
  - Introducing new products or services involves many life cycle stages, some of which are easier to accelerate than others; for the past couple of decades, drug manufacturers have been using clinical trial simulations to speed learning, reduce costs, and limit unnecessary burdens on patients participating in the trials.
  - Armed with the power of the cloud and big data, the clinical trial simulations can be done faster in ways that benefit the manufacturer and patients.
  - Bristol-Myers Squibb reduced the time it takes to run clinical trial simulations by 98% by extending its internally hosted grid environment into the AWS Cloud; the company has also been able to optimize dosing levels, make drugs safer, and require fewer blood samples from clinical trial patients.
  - As a result of the move, Bristol-Myers Squibb was able to reduce the number of clinical trial subjects in a pediatric study from 60 to 40, while shortening the length of the study by more than a year.
- **Optimize The Workforce**
  - Some HR departments are using talent analytics and big data to reduce costs and effectively manage workforce-related issues.
  - The data allows them to select new hires that are a better fit for the company, reduce employee turnover, understand the skills and output of the existing workforce, and determine the talent life the company needs moving forward.
  - Xerox used big data to reduce the attrition rate in its call centers by 20%: to do that, it had to understand what was causing the turnover, and determine ways to improve employee engagement.
- **Improve Financial Performance**
  - Corporate finance departments are moving beyond periodic reporting and BI, using big data to reduce risks and costs, identify opportunities, and improve the accuracy of forecasts: specifically, they're using data to identify risky customers, monitor suppliers, thwart fraud, pinpoint revenue leaks, and inform new or more efficient business models.
  - A recent partnership between The Weather Company and IBM will allow companies to better manage the impact of half a trillion dollars annually in the US alone.

- The weather data is being collected from more than 100,000 weather sensors and aircraft, as well as millions of smartphones, buildings, and moving vehicles; that data is combined with data from other sources to yield 2.2 billion unique forecast points, and an average of more than 10 billion forecasts on an active weather day.
- Retailers will be able to use the data to adjust staffing and supply chain strategies. Energy companies will be able to improve supply and demand forecasting. Insurance companies will be able to warn policy holders of severe weather conditions, so they can minimize the possibility of car damage in the event of a hail storm, for example.

- **Sell Intelligently**
  - Slight modifications to sales and marketing strategies can have a profound effect on the bottom line, especially when informed by big data.
  - Imagine a direct mail campaign with a coupon return rate of more 70% within six weeks of the mailing: according to the Direct Marketing Association, the average direct mail return rate is 3.7%.
  - How does grocery store chain Kroger do it? For one thing, it personalizes its direct mailer based on the shopping history of the individual customer; Kroger also has a loyalty card program that is rated No. 1 in the Grocery industry.
  - More than 90% of its customers use its loyalty card wen they purchase products; although there are many factors that have collectively enabled Kroger'S financial performance, at least part of its continued worth over 45 consecutive quarters has been attributed to its customer loyalty programs.
- **Minimize Equipment And Asset Failures**
  - Businesses want to avoid unnecessary disruption and customer angst.
  - Now that sensors are being embedded into just about everything, companies are using the data to determine when maintenance is required for planes, trains, automobiles, and even household appliances.
  - Ideally, when an issue has arisen, companies want to understand the issue, what caused it, and how it can be resolved, preferably before a maintenance professional or crew is dispatched.
  - Pratt&Whitney, a unit of United Technologies Corp., is attempting to reduce unplanned aircraft engine maintenance. According to AirInsight.com, today's engines collect about 100 parameters in multiple snapshots while a plane is in flight. By comparison, a new-generation engine is able to collect about 5,000 parameters continuously in flight. The process generates about 2 petabytes of data. Using the data, Pratt&Whitney and its partner IBM are trying to enable proactive maintenance.
- **Leverage Customer Lifetime Value**
  - Today's empowered customers are more demanding and fickle than ever: maintaining or increasing market share requires businesses to understand as much as possible about their customers, continually improve their products and services, and be willing to adapt their business models to reflect the actual needs of their customers.
  - Avis Budget has committed to doing all of this. It implemented an integrated strategy to increase market share, which has yielded hundreds of millions of dollars in additional revenue: the initiative involved determining the value of customers, segmenting them, and offering tiered incentives to improve customer loyalty.
  - To do this, its IT partner CSC applied a model that predicts lifetime value to Avis Budget's customer database, and then validated it using a multichannel marketing campaign and accompanying analytics; the customer valuation data is now combined with other data, including rental history, service issues, demographics, corporate affiliation, and customer feedback.

- Avis Budget is also collecting and analyzing social media data: it has a team of social media specialists who respond to brand mentions.
- The company recently updated its website to further improve customer experiences, and it is using big data to forecast regional demand for fleet placements and pricing.

Lesson 10 – Data Preparation for Predictive Modeling

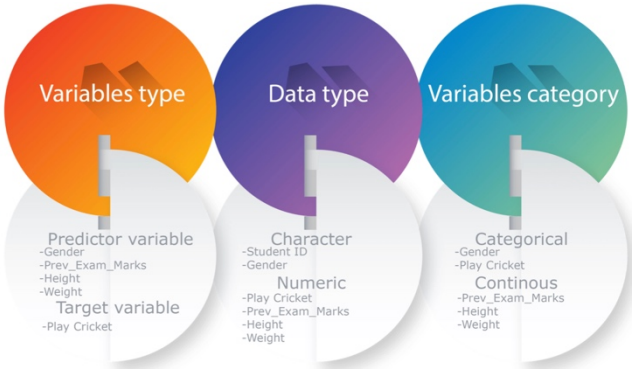
Data Exploration and Data Preparation

The phases of exploration, cleaning and preparation of the headset can cover the 70% of the total time of a data mining project. The steps are summarized as follows:

- **Identification Of Variables**
  - First we need to identify the variables Predictors (Inputs) and Target (Output).
  - Secondly, we must recognize the type and level of data measurement, example:

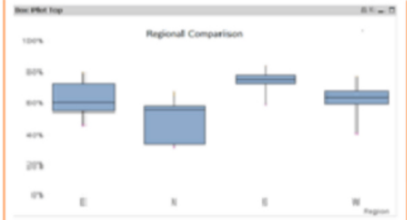
Student_ID	Gender	Prev_Exam_Marks	Height (cm)	Weight Caregory (kgs)	Play Cricket
S001	M	65	178	61	1
S002	F	75	174	56	0
S003	M	45	163	62	1
S004	M	57	175	70	0
S005	F	59	162	67	0

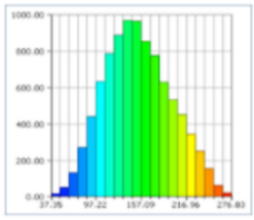
  - The variables in the previous slide table may be reclassified as follows:



- **Univariate Analysis**
  - In this phase the variables are analyzed one at a time.
  - The type of analysis depends on the nature of the variable> categorical or numerical/continuous.
  - For numerical/continuous variables, you must first understand the values of central tendency and variability:

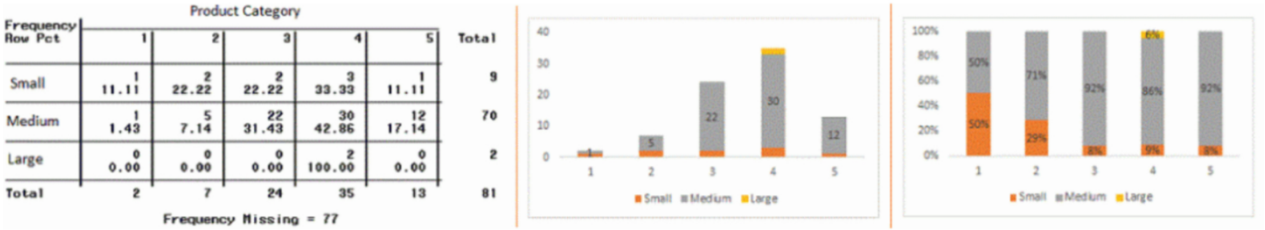
Central Tendency	Measure of Dispersion	Visualization Methods
Mean	Range	Histogram
Median	Quartile	Box Plot
Mode	IQR	
Min	Variance	
Max	Standard Deviation	
	Skewness and Kurtosis	





  - Univariate analysis is also useful to highlight missing and outlier data, which will be discussed specifically later.
  - For categorical variables, frequency tables will be used to understand the distribution of each category (values and absolute or percentage frequencies); the bar chart can be used as a graphic representation.
- **Bivariate Analysis**
  - The bivariate analysis shows the relationship between two variables.
  - In this case we try to evaluate whether there is an association or independence between the variables, considering a specific significance level.

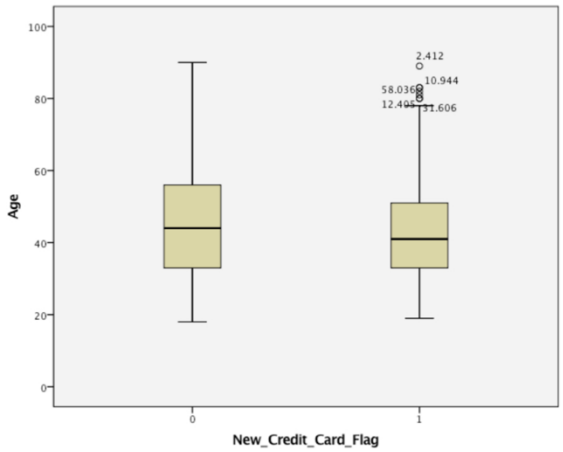
- We can perform bivariate analysis for any combination of categorical and numerical/continuous variables.
- The combinations can be:
    - **Categorical vs categorical**
        - In this case we use a contingency table and stacked bar charts.
        - With contingency tables, we analyze the absolute or percentage joint frequencies of the observed values of the two variables; on the rows we will find the categories of a variable and on the columns the categories of the second variable.
        - The stacked column chart enables a visual representation of the relationship.
        - The significance test used is the chi-square, and to examine the strength of the relationship using indices such as the V Cramer.



Frequency / Row Pct	Product Category 1	2	3	4	5	Total
Small	1 11.11	2 22.22	2 22.22	3 33.33	1 11.11	9
Medium	1 1.43	5 7.14	22 31.43	30 42.86	12 17.14	70
Large	0 0.00	0 0.00	0 0.00	2 100.00	0 0.00	2
Total	2	7	24	35	13	81

Frequency Missing = 77

    - **Categorical vs numerical/continuous**
        - In this case, we typically use box plots of the numeric variable for each level of the categorical variable.
        - To verify the significance of the relation, we can use the t-test, if the number of the qualitative categories is equal to 2, or F-Test if the number of categories is greater than 2.



    - **Continuous vs continuous**
        - In this case you use a scatter diagram and the linear correlation coefficient, to find the strength of the relationship.
        - The correlation varies between -1 and +1.
            - -1: Perfect negative linear correlation.
            - +1: Perfect positive linear correlation.

- 0: No correlation.

X	65	72	78	65	72	70	65	68
Y	72	69	79	69	84	75	60	73

Metrics	Formula	Value
Co-Variance (X,Y)	=COVAR(E6:L6,E7:L7)	18.77
Variance (X)	=VAR.P(E6:L6)	18.48
Variance (Y)	=VAR.P(E7:L7)	45.23
Correlation	=G10/SQRT(G11*G12)	0.65

- Several methods are used to analyze these combinations during the data preparation process.

- **Missing Data Management**
  - The presence of missing data in the training dataset may reduce the fit and predictive ability of the model, possibly leading to incorrect forecasts and mis-classification.

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	2	1	50%
M	4	2	50%
Missing	2	2	100%

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	4	3	75%
M	4	2	50%

  - Missing values **consequences** on predictive models:
    - Considering the precedent tables, in the left-wing scenario, we did not treat missing values.
    - The inference from this dataset is that the ability of males to play cricket is equivalent to that of females.
    - If you look at the second table, showing data after treatment of missing values (based on gender), we can see that females are more likely to play cricket than males.
  - Missing values **Sources**:
    - The origin of the missing data can be essentially linked to two stages of the process:
      - Data Extraction
      - Data Collection
    - Considering the first stage mentioned, we should conduct a double check in the extraction stage, possibly using the "data type guardians software" and/or using transformation of the extracted strings procedures (hashing procedures) that help to eliminate or minimize such extraction errors.
    - As for the second one, the problems are certainly more complex and they can have various configurations.
  - Missing values **Configurations**:
    - **Completely Random Missing**, in this case, the probability of having a missing data is identical for each observation.
    - **Random Missing**, in this case, the presence of missing data in a variable is random, but has a relationship with other input variables (ex. The missing data in the age variable is more common for women than men).

- **Missing that depend on Predictors Not Observed**, in this case, the missing data are not random, but depend on a variable that has not been inserted in the database. Ex. If in a clinical study a diagnostic procedure may cause discomfort in some patients, they may leave the study prematurely, generating a non-random missing profile, unless we introduce the "uncomfortable" missing variable to control themselves.
- **Missing that depend on the Same Value of the Variable**, in this case, the probability of missing depends on the missing value in itself. Ex. Who has a low income tend to not declare their income in a survey.

- Missing value **How To Process**:
  - **Deletion of Missing Data**, there are two ways of deleting missing data: **Listwise** or **Pairwise**
    - In **Listwise** deletion, the entire observation that contains the missing data is deleted. Clearly it is the easiest approach to the database clean, but it has the disadvantage of markedly reduce the sample size.
    - In the **Pairwise** deletion, any statistical analysis is calculated with all the valid cases present in each variable, eliminating the observation only if one of the variables involved containts a giving missing. The advantage is to preserve degrees of freedom, the disadvantage is that the sample size may be different for different variables involved in the analysis.

List wise deletion

Gender	Manpower	Sales
M	25	343
F	.	280
M	33	332
M	.	272
F	25	.
M	29	326
	26	259
M	32	297

Pair wise deletion

Gender	Manpower	Sales
M	25	343
F	.	280
M	33	332
M	.	272
F	25	.
M	29	326
	26	259
M	32	297

This type of deletion is used when missing data are totally random, otherwise we could invalidate the predictive ability of the model.

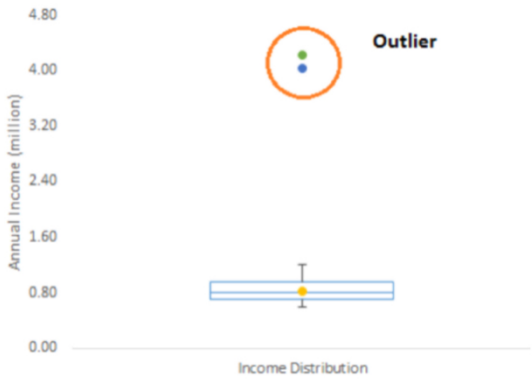
  - **Assignment with mean / mode / median:** this approach replaces the missing data with estimated values. The objective is to use the relationships between variables in the dataset to estimate as accurately as possible the missing data itself. The mean and median are used for numeric data, the mode for categorical data:
    - **Generalized replacement**, in this case we calculate the mean or median for all valid data available and replace all missing data with these values. For example in the table above the average of valid cases for Manpower is 28,33 and then the missing data is replaced with that value.
    - **Similarity replacement**, in this case, we calculate separately the average of Labour for men (29.75) and women (25) and replace missing values taking into account the genre to which the observation with missing data relate.
  - **Building Predictive Models**, the setup of a forecasting model is one of the most sophisticated method for handling missing data: it creates a predictive model to estimate the values that will replace the missing data.
    In this case, the interested data are divided into two groups: a set without missing values for the investigated variable and another with missing values:

the first dataset becomes the set of the model training data, while the second set with values missing is set as test data, and therefore the variable with missing values are treated as target.

- **Predictive Models**, a model is then estimated to predict the target variable based on other attributes of the training dataset and to populate the missing values of the target variable; the techniques used are regression, ANOVA, logistic regression and various other modeling techniques.
  There are two disadvantages in this approach:
  - The model estimated values are usually more regular than the true values.
  - If there are no relationships between the variables in the dataset and the data with missing values, the model will be inaccurate in estimating missing values themselves.
- **Recording with KNN Neural Network**, with this allocation method, the missing attribute values are calculated using the similarity pattern between the observations of several variables, on the basis of an appropriate distance function between those patterns; the missing value of a variable is therefore replaced with the "closer" valid value. The advantages of the method:
  - It can be used interchangeably to qualitative and quantitative variables.
  - It is not necessary to build a predictive model for each attribute.
  - We can treat cases with missing values of a different nature.
  - It is considered the structure of correlation between the variables.

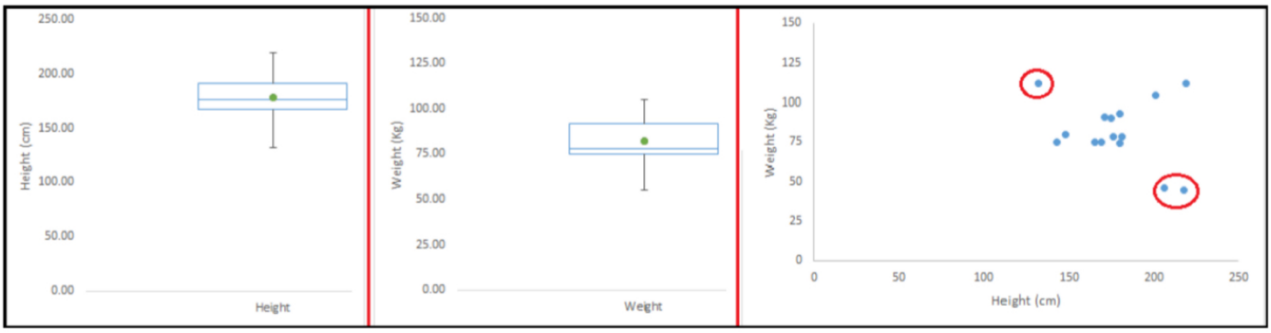
  The disadvantages are related to the computational difficulty of large datasets and the sensitivity of the model parametrization.

- **Outlier Treatment**
  - **Outliers Definition**
    - Outlier detection is commonly used by analysts and data scientists; the problem of outliers needs a lot of attention, otherwise you can get the production of heavily wrong estimates in predictive models.
    - An outlier is an abnormal value, an observation that appears distant and deviates from a typical pattern of a given sample.
    - Example: customer profiling, it turns out that the average annual income of customers is \$0.8 million, but two customers have annual incomes of \$4 and \$4.2 million. These two clients have therefore an annual income much higher than the rest of the population and will be seen as outliers.



  - **Outliers Types**
    - Outliers can be of two types: univariate and multi variate. In the slide before, we discussed the example of univariate outliers. These outliers can be found by observing the distribution of a single variable, and multivariate outliers are outliers in a n-dimensional space. In order to detect them, the multi-dimensional distribution must be examined.
    - Example, here we can observe the univariate and bivariate distribution for height, weight. Looking at the box plots, we have no abnormal value (above and below 1.5 * IQR, the most common method). Instead, looking at the

scatter plot we notice two lower and one higher than the average values in a specific segment of the weight and height.



- **Outliers Origins**
  - **Data Entry Errors**, human errors, such as errors caused during data collection, recording, or inputs can generate atypical values in the same data.
  - **Measurement Error**, is the most common source of outliers. It occurs when the measurement tool used is faulty. Example, 10 weighing machines. 9 of them are correct, one is faulty. The weight measured on a faulty machine will be higher / lower than the rest of the measurements in the gourds. The weights measured on faulty machines can lead to abnormal values.
  - **Experimental Error**, another cause of outliers is the experimental error, caused by an abnormal event that has affected the outcome of the experiment itself.
  - **Intentional Outlier**, in general are related to sensitive data. For example: imagine you interview some young people on alcohol consumption. Only some of them will report the actual value; in this case the actual values may appear as outliers.
  - **Data Processing Error**, usually in data mining activities, we extract data from more sources. It is possible that some errors of manipulation or extraction bring outliers in the set of final data.
  - **Sampling Error**, for example, we have to measure the height of few athletes. For error, we include a pair of basket players in the sample. This inclusion can cause abnormal values in the set of data.
  - **Natural Outlier**, when an outlier is not artificial (caused by one of the previously identified errors), is precisely a natural erratic value.
- **Outlier Consequences**
  - Increase the error variance and reduce the power of the statistical tests.
  - If outliers are not distributed randomly, they can compromise the normality of some distributions.
  - They can heavily influence the estimates of interest.
  - They may have an impact on the basic assumptions of regression, ANOVA and other statistical models.
  - Example:

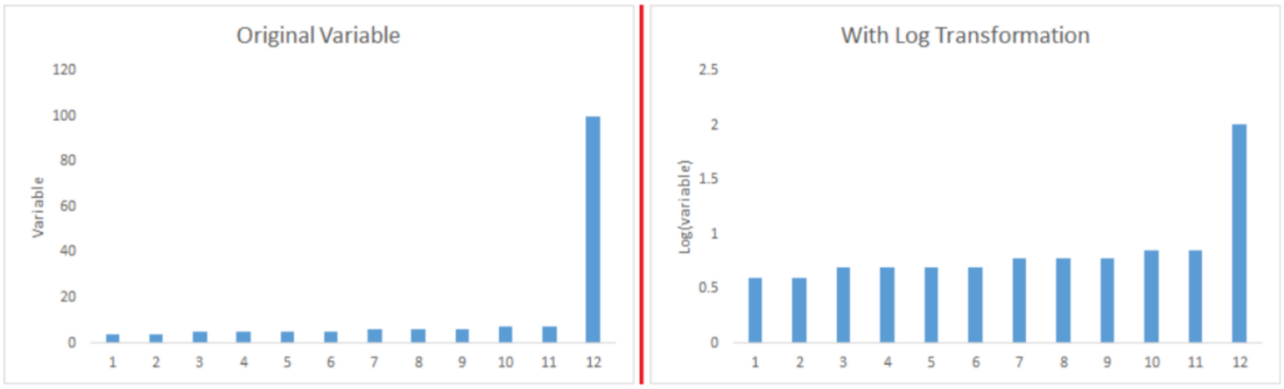
Without Outlier	With Outlier
4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7	4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300
Mean = 5.45	Mean = 30.00
Median = 5.00	Median = 5.50
Mode = 5.00	Mode = 5.00
Standard Deviation = 1.04	Standard Deviation = 85.03

As you can see, the data with outliers have significantly different mean and standard deviation. In the first scenario, the average is 5.45. But with the

outlier, the average increases to 30. This changes completely our estimate of the central distribution location.

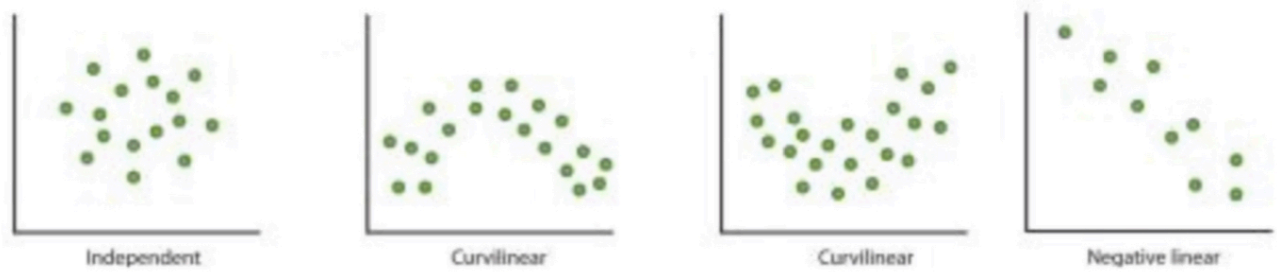
- **Outliers, How To Identify Them**, the most commonly used methods to detect outliers are charts, such as Box Plot, Histogram, Scatter Plot (previously we used box plots and scatter plots to display).
  Some analysts also various rules of thumb to detect outliers:
  - Values that are beyond the limits of -1.5 x 1.5 x IQR.
  - Values outside the range between the 5th and 95th percentile.
  - Values far three or more standard deviations from the mean.
  - Identification of influential points, through appropriate influence or leverage ratios, such as Mahalanobis distance and Cook's D.
- **Outliers, How To Remove**, most of the ways to put a remedy to the anomalies of the data are similar to the methods used to treat the missing values:
  - **Elimination of Observations**, it is appropriate if they are the result of a data entry error or if the outliers are relatively few in number, we can also cut the extreme tails of the distribution.
  - **Data Transformations**, logarithmic transformation, grouping into categories, reducing their impact on the analysis (weighting).
  - **Treating them as a Separate Group**, in the presence of a significant number of abnormal data, it may be appropriate to separate them from the rest of the observations, estimate an ad hoc predictive model and integrate it with the base model.
  - **Replacing values**, as in the case of the missing data, we can replace with mean, mode or specific predictive models.

- **Variables Transformation**



- **Motivation and Methods**
  In modeling, the variable transformation involves the replacement of the variable with a function of the variable itself: for example, the square, the square root, the cube root or the logarithm x is a transformation.
  Motivations to proceed into a variable transformation:
  - When you want to change the scale of a variable or standardize his values. Sometimes this transformation is essential if you are working with data expressed in different scales, in any case do not change the shape of the distribution of the variable.
  - When we can not transform complex linear relationships in linear relationships. The existence of a linear relationship between variables is easier to estimate and manage than a non-linear. The scatter plot can be used to find the relationship between two continuous variables; these

transformations also improve the prediction. The logarithmic transformation is one of the commonly used processing techniques for this purpose.



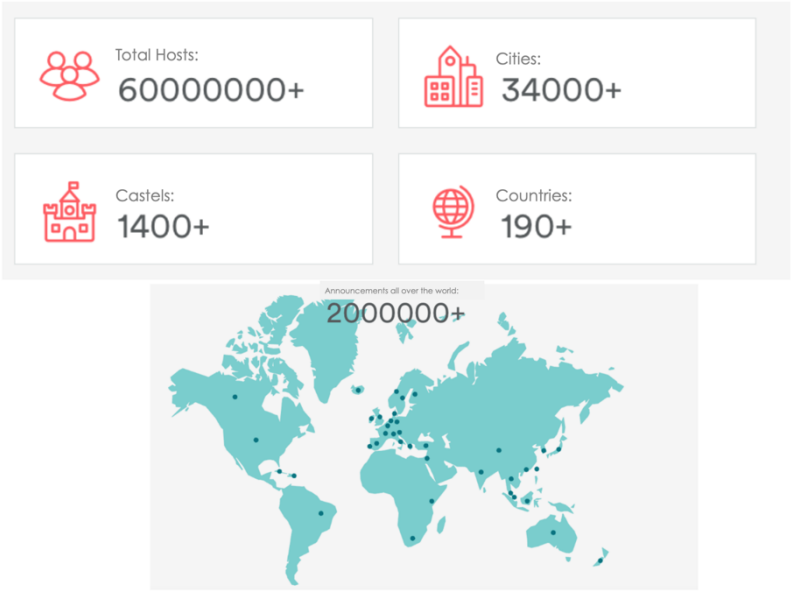
- When a symmetrical distribution is preferred to an asymmetrical: some predictive models require the normality of the variables used and the transformations of variables may at least partially remove the asymmetry; in particular for right asymmetric distribution you can use the square root transformations or cubic or the logarithm, for those left oblique the square, the cube or the exponential.
- When business evaluations lead us to conclude that is significant for the analysis and for the deployment of the model to use appropriate categorizations of the variables (ex. Age group, household income classes, etc.)
- When you want to use categorical variables in models that accept only numerical variables (transformation of categorical variables into dummy variables – 0/1).

- **Creation of New Variables**

Lesson 11 – Data Preparation: Airbnb Case

Some info about Airbnb

Airbnb is an online portal that connects people who look for a home or a room for short periods with people who have an extra space to rent, generally private. The site was opened in October 2007 by Brian Chesky, Joe Gebbia and Nathan Blecharczyk. The idea comes from the fact that in 2007 Brian and Joe moved to San Francisco; at that time the Industrial Design Society of America was organizing the annual conference and the availability of rooms in the hotel was exhausted. They were not able to pay their rent, so offered part of their apartment to other travelers interested in the conference. In January 2009 the company started thanks to the incubator Y Combinator.



The continued its rapid growth, and in November 2010 received 7.2 million dollars from Greylock Partners and Sequoia Capital, reaching the milestone of 700,000 nights booked, the 80% in the last 6 months of 2010. In February 2011, the nights booked arrived at one million and the turnover increased by 65% compared to the previous month. 25 May

2011, the actor and partner of A-Grade Investments, Ashton Kutcher, announced a major investment in the company and its role as strategic brand ambassador. The company is in continuous international expansion: buying Accoleo, a German ambassador. In July 2011, Airbnb receives an additional $112 million dollars and the valuation of $1 billion. Currently, Airbnb has reached a quote share higher than $20 billion dollars, far above the value of many of the world's leading hotel chains.

The Case: Which Users Will Probably Book a First Experience with Airbnb?
The new Airbnb users can reserve a location in more than 34,000 cities and in more than 190 countries. Being able to accurately predict whether a new user will book the first travel experience, Airbnb can:

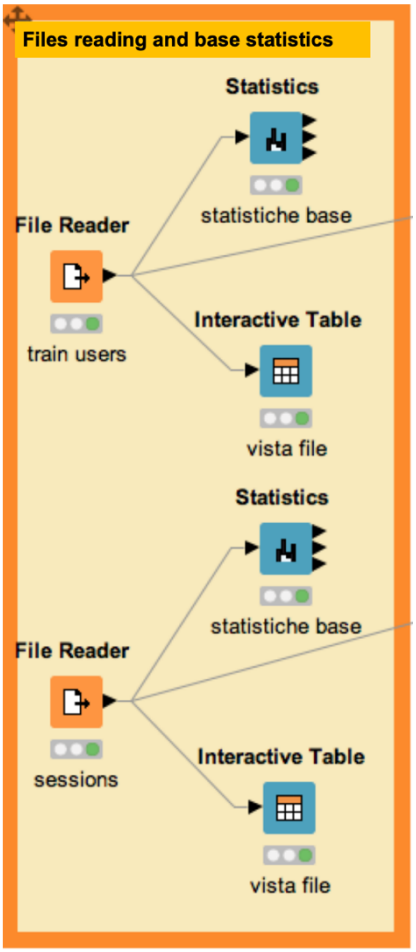
- Share more personalized content to their community, aiming the communication on higher potential users.
- Reduce the average time to the first booking.
- Better predict the demand.

For the analysis, a sample of users has been extracted with their demographics data, some web sessions records, and some summary statistics on users. The objective is indeed to predict how likely will be made a reservation by a new user. All users included in this analysis are in the US.

Available Data
There are 12 possible outcomes in the data to generate the analysis target variable: the country of destination may in fact be "United States", "FR", "CA", …, "NDF" (No destination) and "Other". "NDF" is different from "Others", because "Others" means that there was actually a reservation, but in a country not included in the list, while "NDF" means that there was no reservation. The training dataset contains a sample of 213,451 users with accounts created from 1/1/2010 until 30/6/2014. In the dataset containing the web sessions, the 01/01/2014 behavior data, while the users dataset dates back to 2010.

Reading of Data Files
The analysis flow start from reading two important files: the data of users included in the analysis and the file relating to activity sessions recorded by users themselves from 1/1/2014 to 06/30/2014. The first analysis involves the verification of the correct data read and data quality for any subsequent data cleaning stages. In KNIME the Statistics node allows you to calculate the

distributions of qualitative variables and some summary measures of quantitative variables in this case the unique numerical are ages in train users and secs_elapsed in sessions). The Interactive Tables node enables you to display the data files to better understand the nature of any problems in the data. The analysis shows the necessity of careful data preparation, working both on data cleansing and on the transformation of the available variables.



Processing of Relevant Data

The first part of the analysis is linked to the need of selecting only active customers starting from 2014, and consider the sessions preceding their reservation. Before doing this, it has been necessary to obtain the date of the first activity business from the timestamp available, through appropriate string functions and subsequent transformation into a date field. Secondly, it has been calculated the number of seconds elapsed between the date of initial activity and the reservation date (with a daily level approximation); this figure will serve to isolate, at least by approximation, previous sessions with respect to the booking day. As can be seen from the data, about 1 out of 6 customers that book, book on the first day of activities on the website.

Selection of Active Visitors since 2014

The first part of the analysis, however, is linked to the necessity of selecting only active visitors since 2014, to select then the sessions before their reservation (considering the sessions dates available). From 213,451 visitors contained in the training set, 76,430 started surfing the website in 2014 and then they are selected for analysis. Using the number of seconds that have passed since their first action on the website (file sessions.csv) and the difference between the date of first visit previously built and the eventual date of reservation, we have selected the relevant sessions as predictors of booking. For visitors without booking all actions carried out in the six months available were considered.