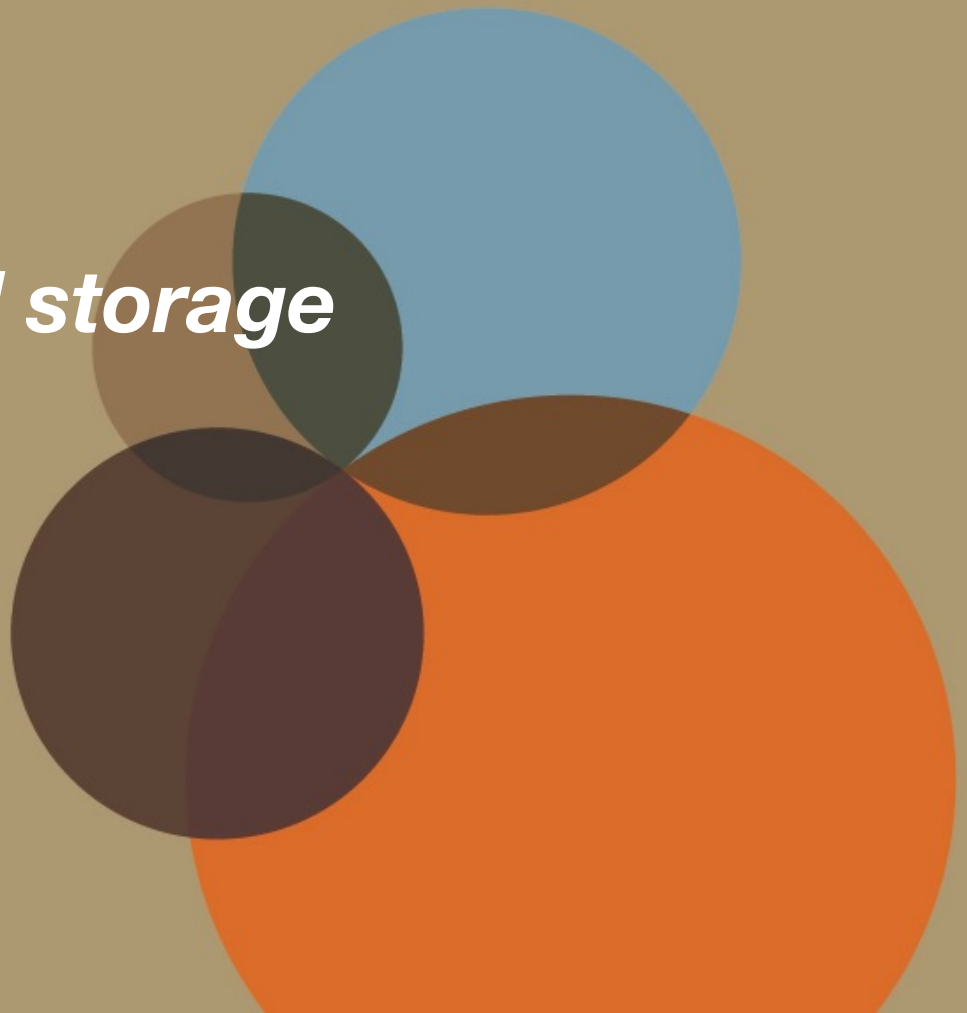
UNIVERSITÀ
DEGLI STUDI
DI UDINE

Big Data Management, Analysis, and Presentation

Data provenance and storage



Andrea Brunello
andrea.brunello@uniud.it

Today's agenda

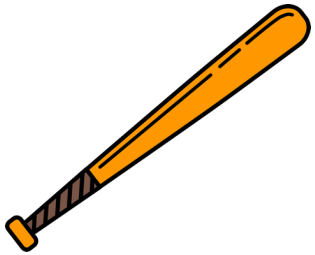
We are going to spend today's four hours talking about:

- The need for databases
- Different kinds of data
- Relational databases and SQL
- NoSQL solutions
- The twilight zone: NewSQL
- Data warehousing & data lakes
- The ETL process
- Types of analytics

- Regardless of the domain considered, data are the lifeblood of corporate information systems
- **Data** is a collection of raw, unorganised and unanalysed material relating to a phenomenon (e.g., a sensor, connected to a machinery, outputting decimal numbers)
- Through a processing and interpretation phase, the data become meaningful **information** for the recipient (e.g., the numbers are interpreted as a temperature in degrees °C)
- Combining information, intuition and experience we obtain **knowledge**, which can be compared with knowledge already acquired and which allows us to interpret and guide our actions (e.g., the engine is overheating, thus a maintenance should be scheduled within a week)

The importance of context

- Interpretation happens also in Natural Language:
  - The word “BAT” alone carries little to no information
  - Compare “BAT” paired with the words “CAVERN” and “FLY”; and “BAT” paired with “FIELD”, “GAME”, and “SWING”
- We interpret also through the context!
- Collecting data in organized structures, and establishing links between them, makes the data informative
- **Information systems** are those (fundamental) systems that support the storage and management of information of an organization for all its purposes



Database Management System (DBMS)

A DBMS contains information on a particular domain

- Collection of **interrelated data** (database)
- Set of **programs** to manage the data

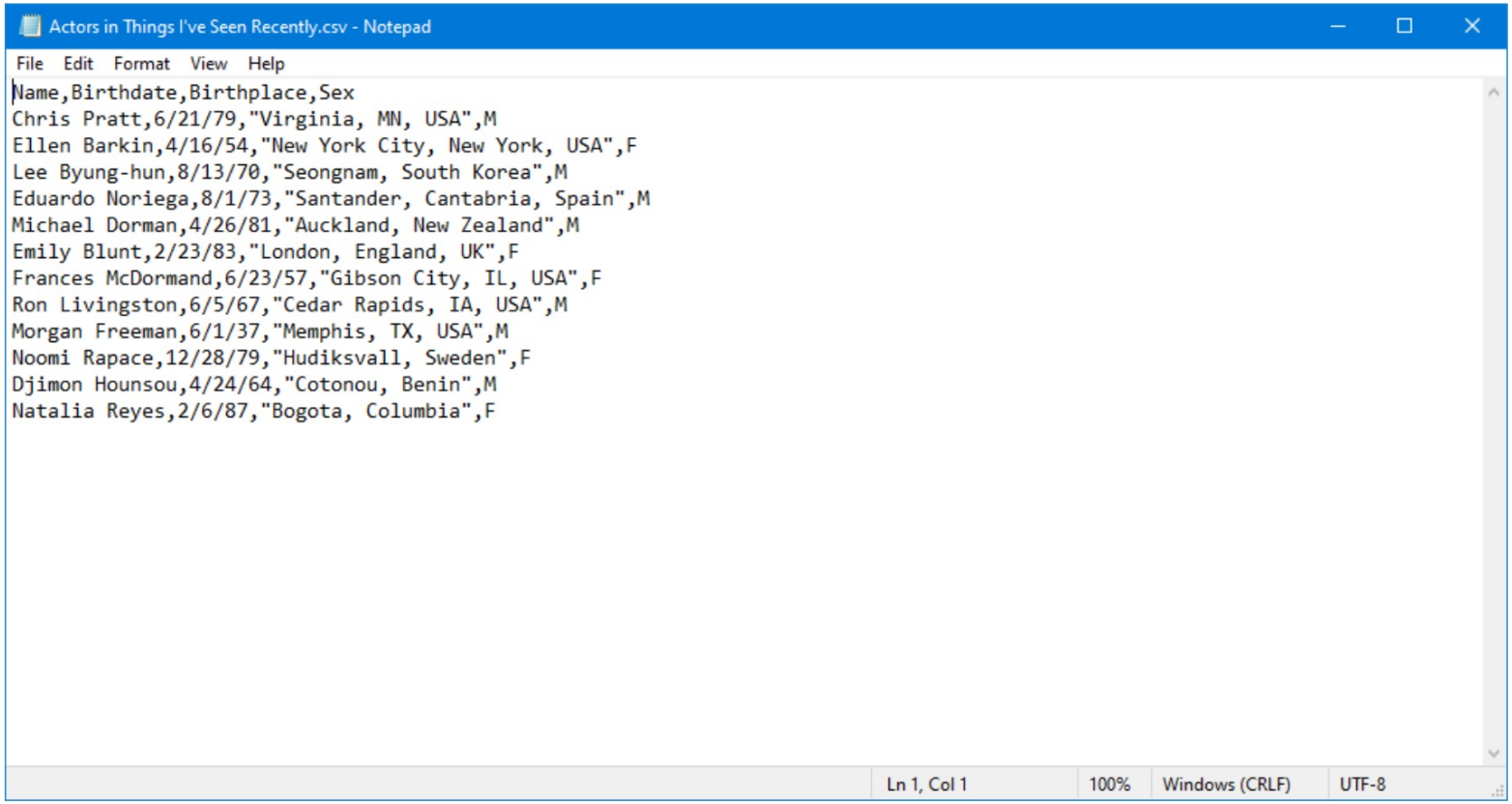
For example, in the insurance domain:

- **Data:** customers, insurance policies, claims, insured goods, …
- **Programs:** add a new customer, open a claim, perform a risk assessment, …

… Databases are really pervasive!

In the early days, data-centric applications were built by means of:

- **Files**, e.g., in “csv” format
- **Programs** specifically designed to access such files



Files to store the data and relationships among the data

Vehicles.txt

```
Plate,Year,Manufacturer,Model
AX252KF,2015,Ford,Mustang
BK356PQ,2020,Dodge,Charger
FH665HG,2019,Chevrolet,Corvette
CD124LP,2021,Chevrolet,Camaro
```

Customers.txt

```
Name,Surname,Birth_date,Gender
John,Doe,1974-07-12,M
Mary,Jane,1990-08-15,F
Darell,Smart,1983-03-07,M
Jessica,Miller,1995-11-03,F
Belinda,Barton,1967-12-05,F
Ned,Fitzgerald,1977-10-26,M
```

Belongs_to.txt

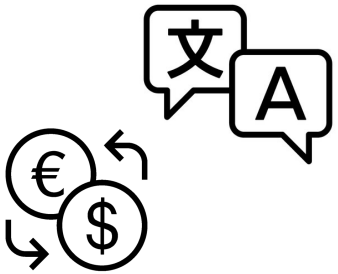
```
Plate,Customer_name,Customer_surname
AX252KF,John,Doe
BK356PQ,Mary,Jane
FH665HG,Jessica,Miller
CD124LP,Ned,Fitzgerald
```

Drawbacks of relying on files and programs

Data **redundancy and heterogeneity**

- Multiple file formats and programming languages
- Duplication of information in different files

  → Possible data inconsistency!



Difficulty in **accessing data**

- Need to write a new program to carry out each new task
- Low interactivity

Drawbacks of relying on files and programs

Constraints management

- Domain constraints (e.g., car value >= 0) become «buried» in program code rather than being stated explicitly
- Hard to add new constraints and manage existing ones

Atomicity of updates

- Failures may leave the system in an inconsistent state
- E.g., a transfer of funds from one account to another should be either carried out in its entirety or not happen at all

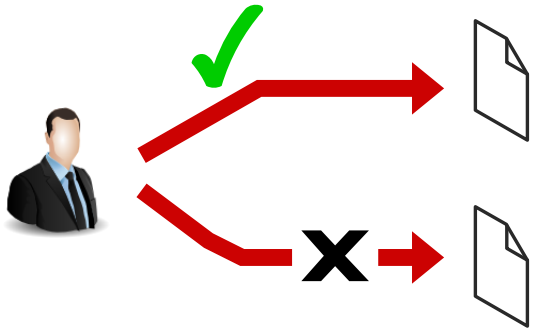
Drawbacks of relying on files and programs

Concurrent access by multiple users

- What happens if 2 users try to access and modify the same data?
- E.g., two users booking the last airline ticket

Security problems

- Hard to provide a user access to some, but not all, data



→ (Relational) DBMSs provide **solutions** to all these problems!

Data considered by information systems used to be very simple

Things started to change in the **2000s**, following the rise of the Internet and its applications, such as social networks

We can broadly distinguish between the following kinds of data:

- **Structured** data
- **Semi-structured** data
- **Unstructured** data

It can be seen as tabular data, represented by **rows** and **columns**

- Each row belonging to a same table has a fixed format, think about an Excel file
- For instance, personal data regarding customers
- Easy to store and process, but they convey a limited amount of information

	A	B	C	D	E	F
1	Name	Surname	Birth date	Gender	Address	Phone number
2	John	Doe	12/07/74	M	660 North Gonzales Ave. Canyon City, CA 91387	202-555-0123
3	Mary	Jane	15/08/90	F	7772 Shore St. Zion, IL 60099	202-555-0176
4	Darell	Smart	07/03/83	M	2 Pine Ave. Royal Oak, MI 48067	202-555-0197
5	Jessica	Miller	03/11/95	F	7015 Wilson St. South Portland, ME 04106	202-555-0197
6	Belinda	Barton	05/12/67	F	22 Wakehurst Street Jackson, NJ 08527	202-555-0197
7	Ned	Fitzgerald	26/10/77	M	12 Sage Rd. Hope Mills, NC 28348	202-555-0197

Semi-structured data do not have a fixed format, but still some structuring

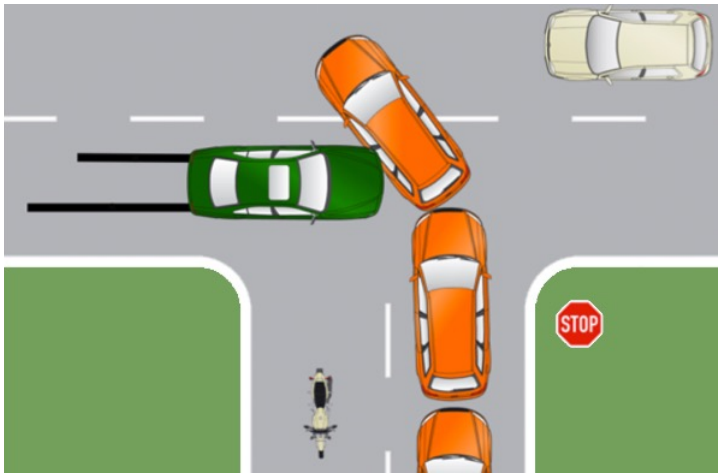
- They contain **tags** or other markers to separate semantic elements and enforce hierarchies of fields within the data
- For example, this can be the case with closed form answers given to telephone surveys

```
<Phone campaign>
    <Survey ID = 1>
        <Name> John Easter </Name>
        <Question ID = 1> yes </Question>
        <Question ID = 2> yes </Question>
        <Question ID = 2.1> 5 </Question>
        <Question ID = 2.2> 7 </Question>
        <Question ID = 3> no </Question>
    </Survey>
    <Survey ID = 2>
        <Name> Mary Christmas </Name>
        <Question ID = 1> yes </Question>
        <Question ID = 2> no </Question>
        <Question ID = 3> yes </Question>
        <Question ID = 3.1> yes </Question>
        <Note> The person seems to be rather interested in the offered product </Note>
    </Survey>
</Phone campaign>
```

Unstructured data is **not organized** in any predefined manner

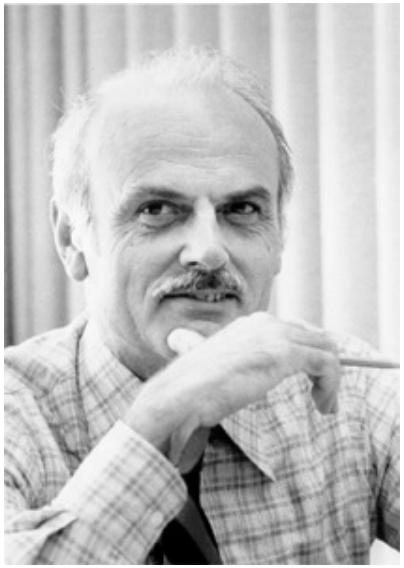
- Free text
- Audio and visual materials
- Difficult to store, index, and analyse

"Vehicle A crossed the road failing to give way to vehicle B, which subsequently hit it on its left side."



Relational databases represent information (data model) by means of **rows** (tuples) that are grouped into **tables** (relations)

Since their proposal, in the **1970s**, by Ted Codd, they have been playing a prominent role in the database realm, because of their **simplicity** and elegant formalization



In addition, relational databases come with **SQL** (Structured Query Language), a user-friendly language that allows for an intuitive interaction with the database and its content

Relational DBs: Tables to represent data and relationships among data

	ssn [PK] text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text
1	386-64-8608	Lorrie	Gillaspy	Female	1968-04-04	68490	sales
2	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market
3	682-79-1556	Johnny	Binder	Male	1969-02-28	43672	[null]
4	622-17-9461	Mark	Robinson	Male	1960-08-17	84741	[null]
5	205-35-1353	John	Smith	Male	1984-02-20	70825	sales
6	678-24-1113	Veronique	Pauleit	Female	1987-01-28	58604	prod
7	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service

Table “Employee”

Table “Department”

	code [PK] text	name text	budget numeric	address text
1	prod	Product development	513086	68152 Schmedeman Junction
2	market	Marketing	715091	763 Shoshone Avenue
3	sales	Sales	748657	25370 Tennyson Drive
4	c_service	Customer service	780377	61 Dorton Park
5	res	Research and development	981007	57 Cucumber Street

Intra-relational constraints:

- Attribute domains
- Not-NULL
- Unique
- Primary key

Inter-relational constraints:

- Foreign key

Other than these, RDBMSs offer many other solutions to guarantee **data consistency**, e.g., CHECK constraints, triggers, …

In addition, these systems are quite sophisticated for what concerns the management of **atomicity** and **concurrency** of operations

Atomicity. All operations in a transaction succeed, or every operation is rolled back (like nothing happened).

Consistency. On transaction completion, the database is moved from a consistent state to a different, but always consistent state (wrt the defined constraints).

Isolation. Transactions do not interfere with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially

Durability. The results of a transaction are permanent, even in the presence of failures of the system. Results can be changed only by a subsequent successful transaction

Relational databases: Major RDBMS vendors

PostgreSQL: https://www.postgresql.org/

Microsoft SQL Server: https://www.microsoft.com/en-us/sql-server/

MySQL: https://www.mysql.com/

ORACLE: https://www.oracle.com/database/

SQL is the (largely) most popular language used to interact with relational databases, due to its intuitive syntax

It encompasses several functionalities:

- Data Definition Language, **DDL**
  - Definition and update of table schemas
  - Management of integrity constraints
- Data Manipulation Language, **DML**
  - Insert, update, delete of tuples
- Management of data access criteria and concurrency aspects

SQL was born in the **1970s**, with the development of language *Sequel* by IBM

It then became an ANSI/ISO standard in 1986; latest version **SQL:2023**

Although standardized, different DBMS vendors implement **slightly different versions** of the language, especially concerning its advanced features

Still, the core functionalities of the language are largely equivalent across DBMSs

A brief history of SQL

1970 — THEORY FOUNDATION

E.F.Codd from IBM published *A Relational Model of Data for Large Shared Data Banks*.

MID 1970s — SEQUEL (STRUCTURED ENGLISH QUERY LANGUAGE)

SEQUEL was designed to manipulate and retrieve data stored in IBM's System R.

1979 — SQL (STRUCTURED QUERY LANGUAGE)

Relational Software introduced one of the first commercial implementations of SQL, Oracle V2 (Version2) for VAX computers.

1986 — ANSI X3.135-1986 AND ISO 9075:1987

This was the first formal SQL standard.

1989 — SQL-89

A revision included minor enhancements and introduced integrity constraints.

1992 — SQL2 ANSI X3.135-1992 AND ISO 9075:1992

A major revision that introduced JOINs, subqueries, set operations, and enhance-ments in DDL and DML.

1999 — SQL3 ISO/IEC 9075:1999

This revision introduced object-oriented features, triggers, recursive queries, and the ability to create and use stored

2003 — ISO/IEC 9075:2003

Enhancements to SQL's capabilities with XML, and the SQL/PSM (Persistent Stored Modules).

2006 — ISO/IEC 9075:2006

Enhanced integration of the XML data type and specifications related to XML querying and handling.

2008 — ISO/IEC 9075:2008

Enhancements like in TRUNCATE statement, FETCH clause, merge statements and temporal data support.

2011 — ISO/IEC 9075:2011

Substantial improvements in temporal data handling, support for over-clause and window functions.

2016 — ISO/IEC 9075:2016

Added features like JSON support and introduced Polymorphic Table Functions.

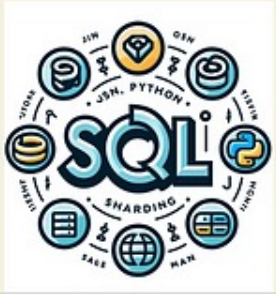
2019 — ISO/IEC 9075:2019

Further enhancements in JSON, SQL routines and types using Python, and advanced sharding capabilities.

© Fanghua Yu, 2024

2023 — ISO/IEC 9075:2023

New features for property graph queries (**PGQ**) over data stored in tables, and more supports of JSON data types.



The **fundamental block** of an SQL query is composed of three clauses

SELECT	← list of attributes, aggr. functions, expressions
FROM	← data sources, tables
WHERE	← filtering conditions for the data extraction

Other clauses: ORDER BY, HAVING, GROUP BY

Structured Query Language (SQL): Example 1

Extract the name, surname, and monthly salary all employees of gender female and with a total salary greater than 60000

SELECT first_name, last_name, salary/12

FROM employee

WHERE gender = 'Female' **AND** salary > 60000;

	ssn [PK] text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text
1	386-64-8608	Lorrie	Gillaspy	Female	1968-04-04	68490	sales
2	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market
3	682-79-1556	Johnny	Binder	Male	1969-02-28	43672	[null]
4	622-17-9461	Mark	Robinson	Male	1960-08-17	84741	[null]
5	205-35-1353	John	Smith	Male	1984-02-20	70825	sales
6	678-24-1113	Veronique	Pauleit	Female	1987-01-28	58604	prod
7	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service

	first_name text	last_name text	round numeric
1	Lorrie	Gillaspy	5707.50
2	Lynde	Kitchenside	6356.08

Structured Query Language (SQL): Example 2

Extract the average salary of employees working in the sales department

SELECT AVG(salary)

FROM employee

WHERE department = ‘sales’;

	ssn [PK] text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text
1	386-64-8608	Lorrie	Gillaspy	Female	1968-04-04	68490	sales
2	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market
3	682-79-1556	Johnny	Binder	Male	1969-02-28	43672	[null]
4	622-17-9461	Mark	Robinson	Male	1960-08-17	84741	[null]
5	205-35-1353	John	Smith	Male	1984-02-20	70825	sales
6	678-24-1113	Veronique	Pauleit	Female	1987-01-28	58604	prod
7	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service

	avg numeric
1	69657.500000000000

Structured Query Language (SQL): Example 3

Extract the name and surname of employees working at a department with a budget greater than 720000

```
SELECT employee.first_name, employee.last_name
FROM employee
        JOIN department ON employee.department = department.code
WHERE department.budget > 720000;
```

Structured Query Language (SQL): Example 3

Each row of employee is combined with every other row of department; then, only the resulting rows that satisfy the JOIN condition are kept

	ssn text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text	code text	name text	budget numeric	address text
1	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	prod	Product development	513086	68152 Schmedeman Junction
2	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	market	Marketing	715091	763 Shoshone Avenue
3	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	sales	Sales	748657	25370 Tennyson Drive
4	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	c_service	Customer service	780377	61 Dorton Park
5	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	res	Research and development	981007	57 Cucumber Street
6	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	prod	Product development	513086	68152 Schmedeman Junction
7	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	market	Marketing	715091	763 Shoshone Avenue
8	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	sales	Sales	748657	25370 Tennyson Drive
9	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	c_service	Customer service	780377	61 Dorton Park
10	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	res	Research and development	981007	57 Cucumber Street
11	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market	prod	Product development	513086	68152 Schmedeman Junction

Structured Query Language (SQL): Example 3

At this point, the WHERE condition is applied over the remaining rows

	ssn text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text	code text	name text	budget numeric	address text
1	205-35-1353	John	Smith	Male	1984-02-20	70825	sales	sales	Sales	748657	25370 Tennyson Drive
2	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service	c_service	Customer service	780377	61 Dorton Park
3	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market	market	Marketing	715091	763 Shoshone Avenue
4	386-64-8608	Lorrie	Gillaspy	Female	1968-04-04	68490	sales	sales	Sales	748657	25370 Tennyson Drive
5	678-24-1113	Veronique	Pauleit	Female	1987-01-28	58604	prod	prod	Product development	513086	68152 Schmedeman Junction

We then obtain the final result, keeping just the attributes specified in the SELECT clause

	first_name text	last_name text
1	Lorrie	Gillaspy
2	John	Smith
3	Frank	Doyle

Structured Query Language (SQL): Example 4

For each department, extract the minimum and maximum salary of its employees

SELECT department, **MIN**(salary), **MAX**(salary)

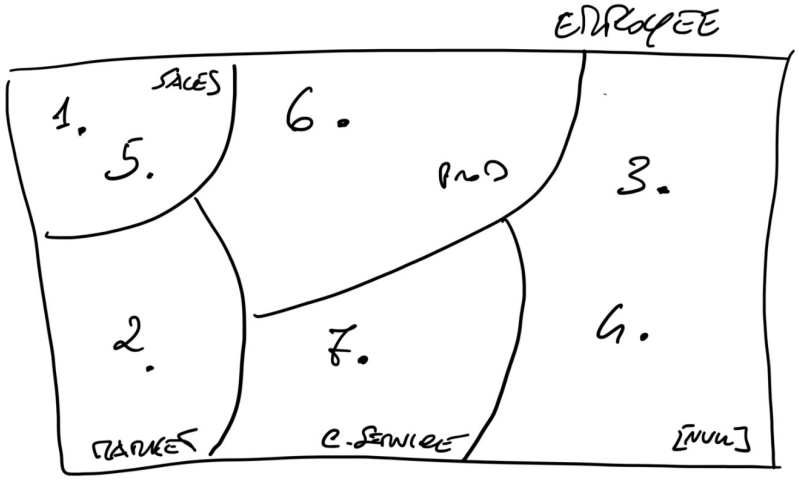
FROM employee

GROUP BY department;

	ssn [PK] text	first_name text	last_name text	gender text	birth_date date	salary numeric	department text
1	386-64-8608	Lorrie	Gillaspy	Female	1968-04-04	68490	sales
2	258-77-4074	Lynde	Kitchenside	Female	1987-03-16	76273	market
3	682-79-1556	Johnny	Binder	Male	1969-02-28	43672	[null]
4	622-17-9461	Mark	Robinson	Male	1960-08-17	84741	[null]
5	205-35-1353	John	Smith	Male	1984-02-20	70825	sales
6	678-24-1113	Veronique	Pauleit	Female	1987-01-28	58604	prod
7	210-35-1000	Frank	Doyle	Male	1994-02-20	70825	c_service

Structured Query Language (SQL): Example 4

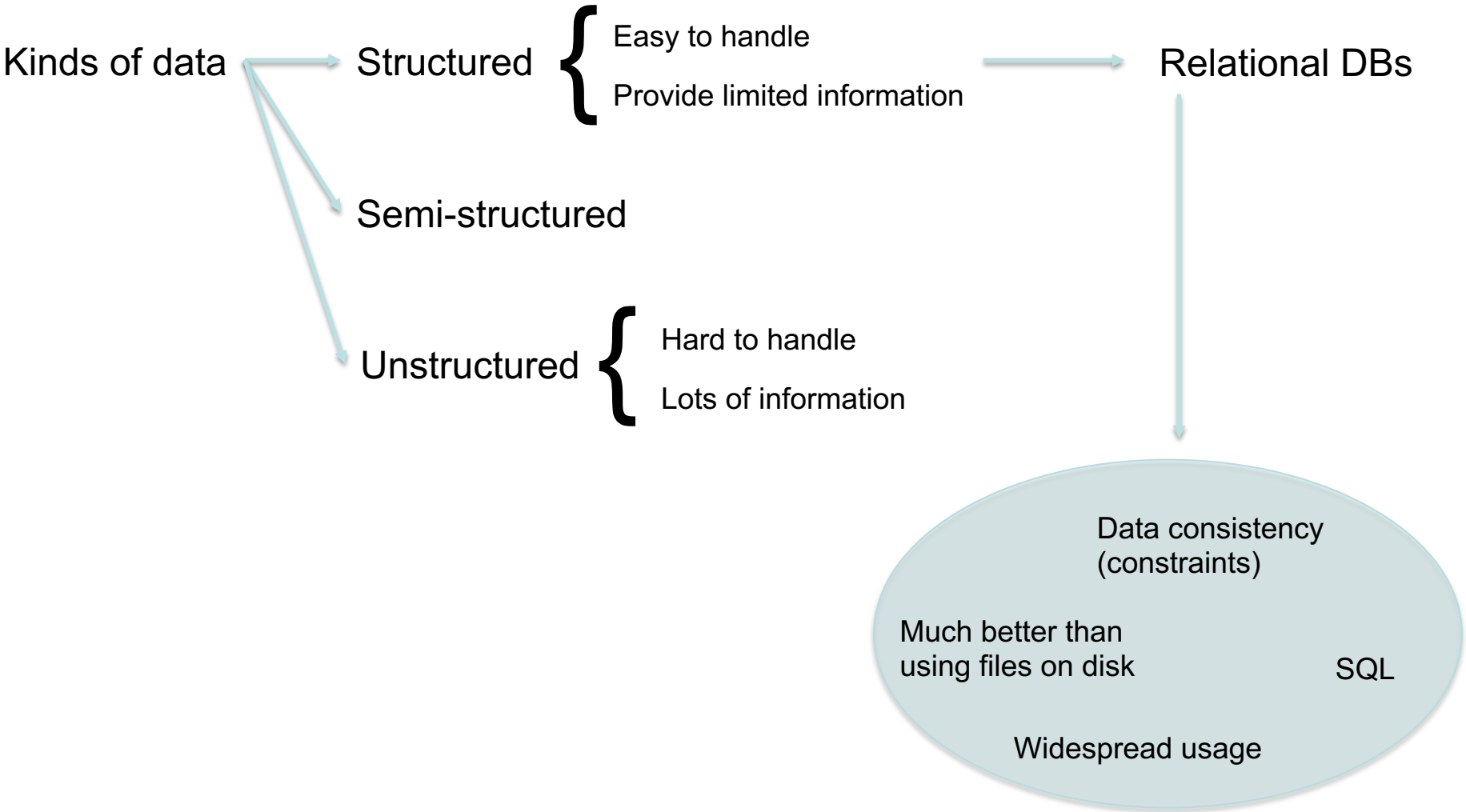
The table rows are first partitioned according to the value of *department*



For each partition, its name, minimum and maximum *salary* is returned

	department character varying (20)	min numeric	max numeric
1	[null]	43672	84741
2	prod	58604	58604
3	sales	68490	70825
4	c_service	70825	70825
5	market	76273	76273

Recap


Kinds of data
Structured
Easy to handle
Provide limited information
Relational DBs
Semi-structured
Unstructured
Hard to handle
Lots of information
Data consistency
(constraints)
Much better than
using files on disk
SQL
Widespread usage

The term NoSQL (Not only SQL) refers to data stores that are **not relational databases**, rather than explicitly describing what they are

A possible definition (http://nosql-database.org/): *"Next Generation DBs mostly addressing some of the points: being non-relational, distributed, open source and horizontally scalable"*

NoSQL storage technologies have **very heterogeneous** operational, functional, and architectural characteristics

NoSQL proposals have been developed starting from **2009** trying to address new challenges that emerged in that decade

Over the years, data got bigger in volume, increased in their heterogeneity, and started to be produced faster and faster

This put classical RDBMS solutions under strain

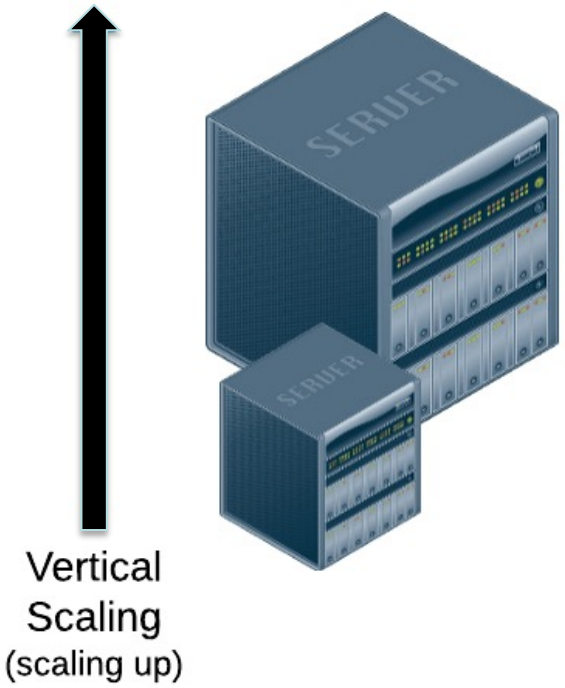
NoSQL systems are specifically designed to accommodate data characterized by a high:

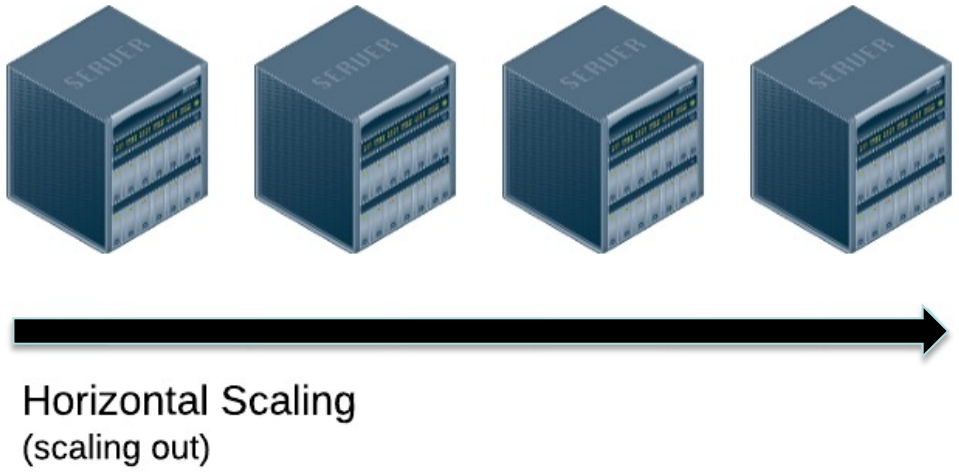
- **Volume**: e.g., insurance data of customers worldwide
- **Variety**: e.g., accident descriptions
- **Velocity**: e.g., continuous monitoring of car trackers

} Big Data

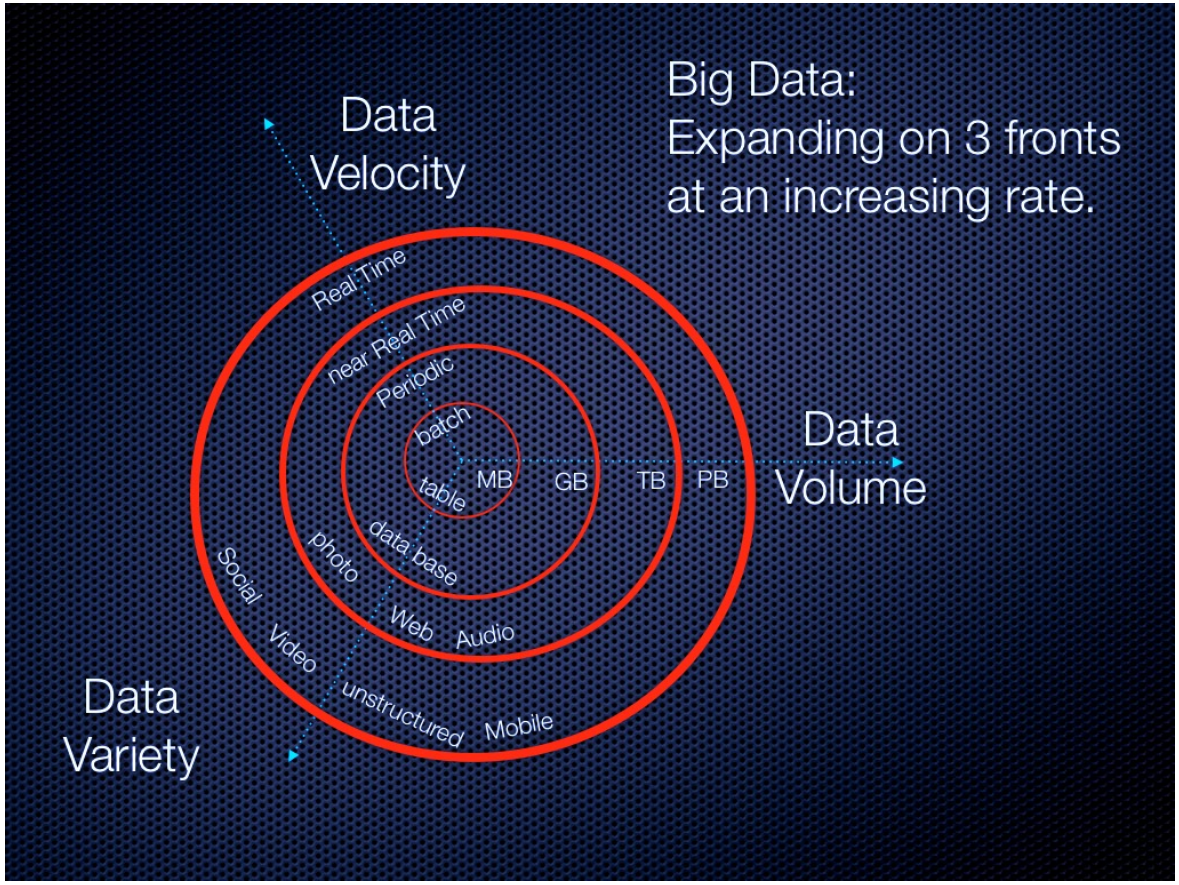
Key characteristics of NoSQL solutions: **flexibility** and (horizontal) **scalability**

Vertical VS horizontal scaling





Gartner analyst Doug Laney introduced the 3Vs concept in a 2001 MetaGroup research publication: "*3D data management: Controlling data volume, variety and velocity*"



Data Never Sleeps 11.0

DOMO

Domo has been keeping tabs on the world's data usage—in a minute—for over a decade now. What the numbers consistently show is that how we use data is always evolving—and that data isn't slowing down. We're also seeing some big changes. The rise of Artificial Intelligence (AI) is reshaping the way we communicate, work, and create. Digital payments continue to replace traditional transactions. Taylor Swift streams in countless headphones. And a rash of cybercrime grows alongside these digital experiences.

In Domo's 11th edition of Data Never Sleeps, we take the pulse of our digital age, where every click, swipe, and stream fuels an ever-expanding digital universe. These are not just numbers; they are the heartbeat of a world where data reigns supreme.

Volume, Variety, Velocity – Big Data

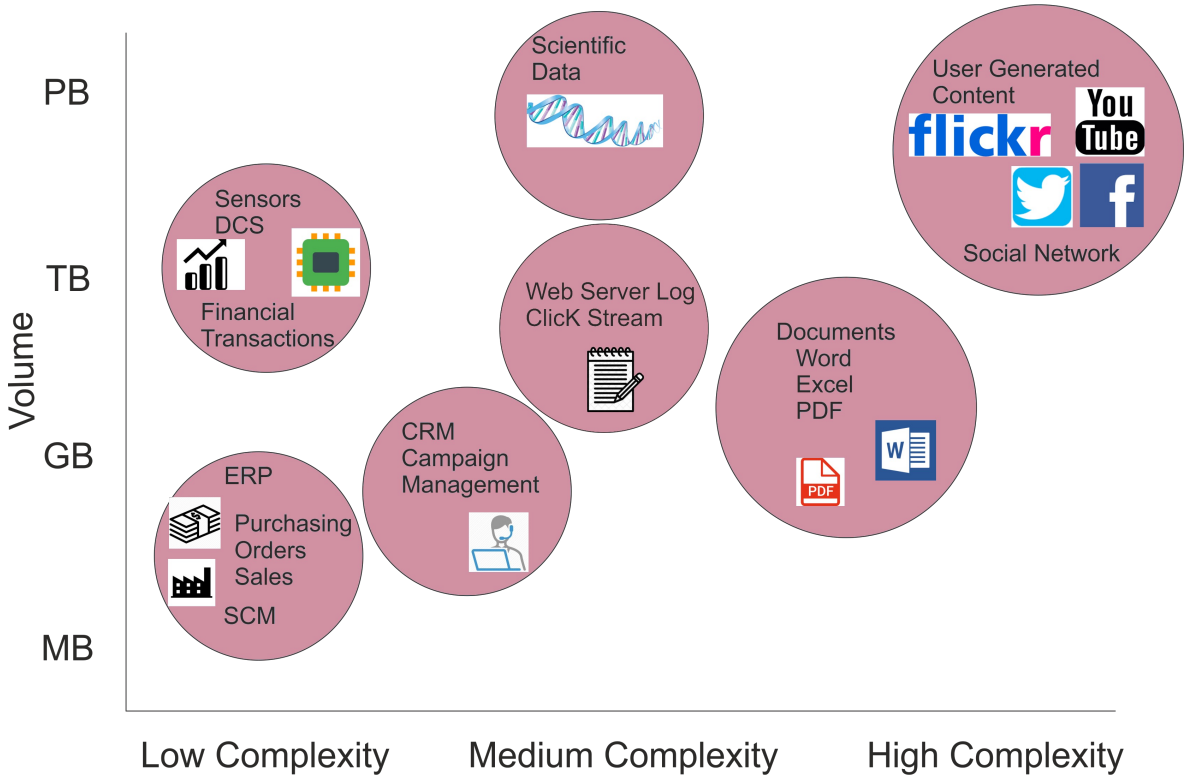
Data acquired via sensors or scientific instruments may come at a high speed

They may have to be stored or analyzed as soon as they arrive, since they are transient due to their size and velocity (for example: logs, data streams)

For companies that rely upon fast-generated data it is also important to exploit/analyze such data as fast as possible

"*Just in its 1st phase, the SKA telescope will produce some 160 TB of raw data per second that the supercomputers will need to handle*"

Volume, Variety, Velocity – Big Data



Drawbacks of NoSQL solutions

They have very simple data models and do not enforce the same number of constraints as traditional RDBMSs → **potential data quality issues**

Flexibility in accommodating heterogeneous data can become a problem if not properly managed

The design process is not as straightforward and consolidated as that of relational solutions

Lack of SQL means that the interaction is more difficult!

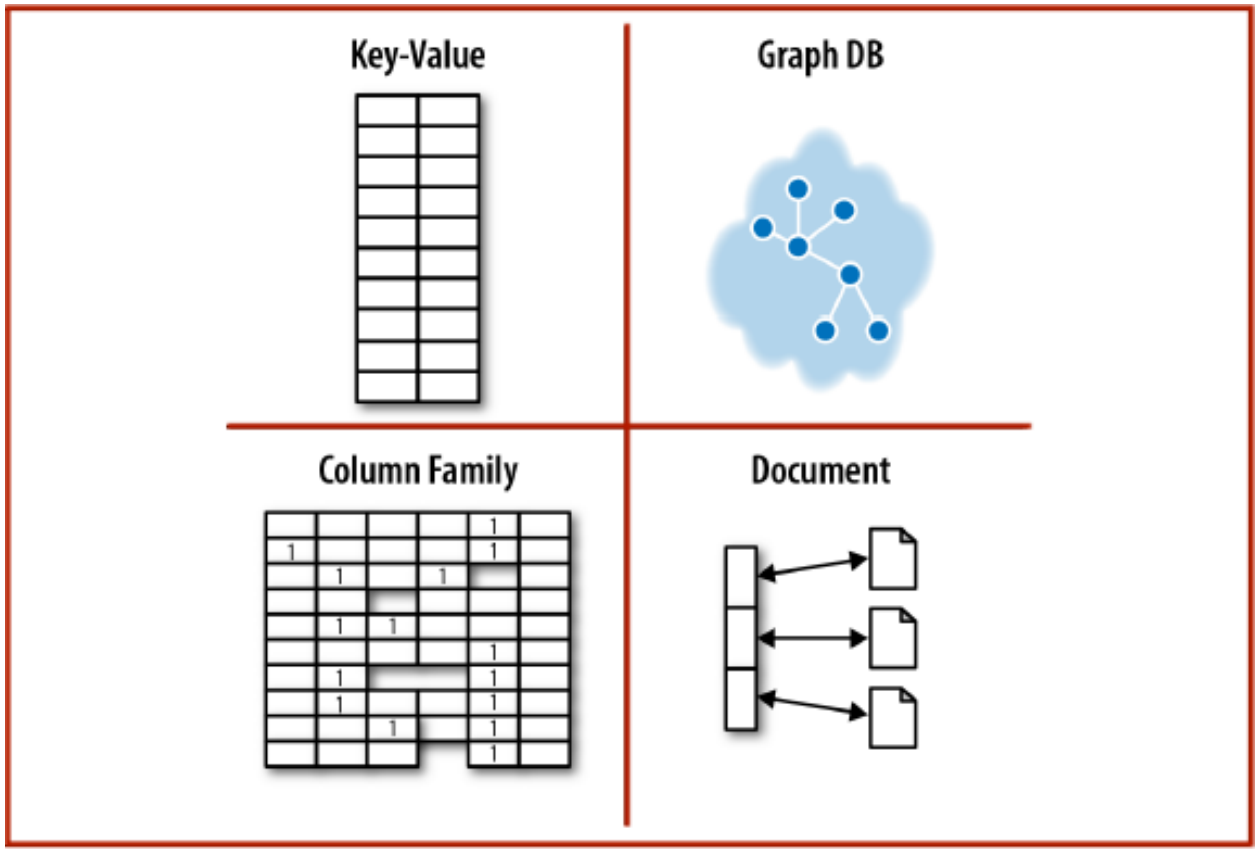
Often, **lack of one or more ACID properties**

In order to better support the characteristics of the novel kinds of data, NoSQL systems rely on the BASE properties instead:

- **Basically available.** The store appears to be accessible most of the time (for instance, tolerance to node failures)
- **Soft state.** Stores do not need to be write-consistent, nor do different replicas have to be mutually consistent all the time
- **Eventual consistency.** The system eventually exhibits consistency at some later point

Of course, this kind of relaxed consistency would be a problem in a banking database. In other cases, it is perfectly acceptable, e.g., for social networks

The NoSQL Quadrant



Key-value stores are based on the concept of **associative array**, i.e., a data structure that contains pairs of <*key, value*>; the key is used to access the values

For example, we could have the arrays: <SSN, phone>, <SSN, dept_name>, <dept_name, budget>

The fundamental point is that accessing a value knowing the key is extremely **fast**

Operations allowed over an associative array are:

- *Add:* add an element to the array
- *Remove:* remove an element from the array, knowing its key
- *Modify:* change the value associated to a key
- *Find:* search for a value knowing the key

Array “emp_name”:

[<XA13, John Doe>, <XZ42, Mary Jane>, <XG56, Jhonny Donny>, …]

Array “emp_department”:

[<XA13, Physics>, <XZ42, Physics>, <XG56, Computer Science>, …]

Array “emp_salary”:

[<XA13, 45000>, <XZ42, 50000>, <XG56, 52000>, …]

Array “dept_budget”:

[<Physics, 950000>, <Computer Science, 780000>, …]

Key-Value stores: Main characteristics

Key-value stores are **extremely simple** when it comes to their data model

Typical relational operations, such as complex filters and JOINS are not possible

Important RDBMS functionalities such as foreign keys and several other kinds of constraints are not supported

Nevertheless, these stores can be extremely sophisticated regarding their implementation of **horizontal scalability**

Typical use case: storing simple information that has to be accessed fast

- Storage of profiles, preferences, and configurations
- Storage of multimedia objects
- Storage of fast sensors data

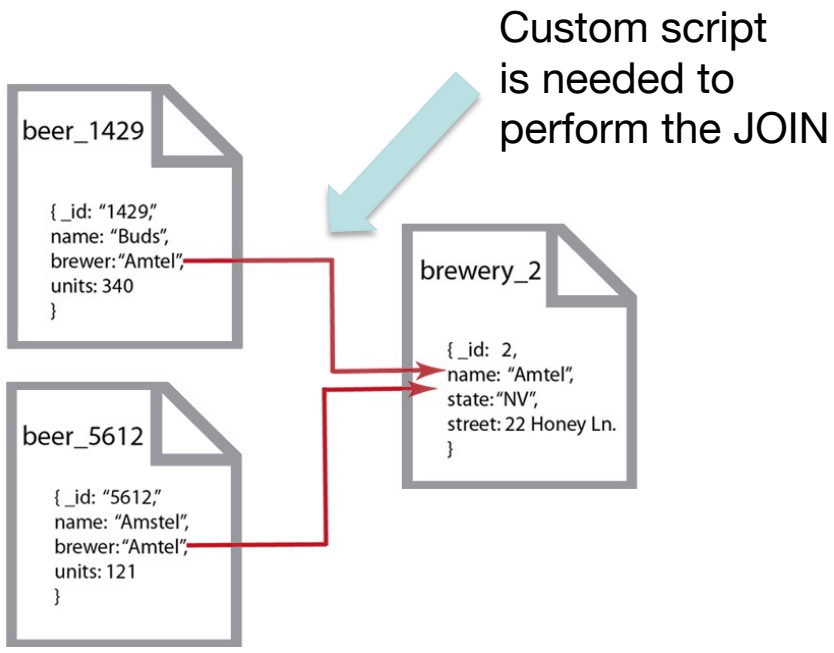
Available systems

- Redis: https://redis.io/
- Berkeley DB: oracle.com/database/technologies/related/berkeleydb.html
- Aerospike: https://aerospike.com/

This kind of databases store, retrieve and manage **document-oriented information**, also referred to as semi-structured data

Documents do not have a fixed structure, nonetheless they make use of **tags** or other markers to organize their content (e.g., XML and JSON file formats)

Document stores represent a step up with respect to key-value stores, since they allow a structure to be defined over keys and values, which can be operated upon by the users



CRUD:

- *Creation:* of a new document
- *Retrieval:* based on key, content, or metadata (tags)
- *Update:* the content or metadata of the document
- *Deletion:* of a document

Each document in the database is uniquely identified by a **key**, which can be used to retrieve the document from the database

Also here, there is no explicit support support for foreign keys or JOIN operations. Relationships are represented by **document nesting** (for example, all the books written by a given author)

Document-oriented databases
Example

USERS

ID	first_name	last_name	cell	city	latitude	longitude
1	Leslie	Yepp	8125552344	Pawnee	39.170344	-86.536632

HOBBIES

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

JOB HISTORY

ID	user_id	job_title	year_started
20	1	Deputy Directory	2004
21	1	City Councilor	2012
22	1	Director, National Parks Service, Midwest Branch	2014

USERS

```
{
   "_id": 1,
   "first_name": "Leslie",
   "last_name": "Yepp",
   "cell": "8125552344",
   "city": "Pawnee",
   "location": [ -86.536632,
                 39.170344 ],
   "hobbies": [
     "scrapbooking",
     "eating waffles",
     "working"
   ],
   "jobHistory": [
     {
       "title": "Deputy Director",
       "yearStarted": 2004
     },
     {
       "title": "City Councillor",
       "yearStarted": 2012
     },
     {
       "title": "Director, National Parks
                 Service, Midwest Branch",
       "yearStarted": 2014
     }
   ]
}
```

Typical use case: similar to key-value stores, but more complex data can be handled, e.g., management of product data, inventory

Available systems

- MongoDB: https://www.mongodb.com/
- RavenDB: https://ravendb.net/
- Apache CouchDB: https://couchdb.apache.org/

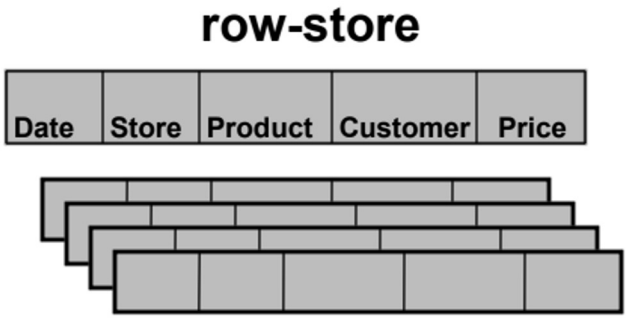
The roots of column family stores can be traced back in the 1970s

However, due to market needs and non-favourable technology trends it was not until the 2000s that they really took off

A column-oriented DBMS stores each database table **column separately**, in different disk locations, or even different machines

Values belonging to the same column are **packed together**, as opposed to traditional DBMSs that store entire rows one after the other

Rows VS columns



2021-04-20,Rome,A01,Bill,50
2021-04-23,Rome,A03,John,150
2021-04-22,Milan,B05,Mary,25
2021-04-19,Milan,C09,Jane,80
2021-04-25,Venice,F12,Jim,75

2021-04-20,2021-04-23,2021-04-22,2021-04-19,2021-04-25
Rome,Rome,Milan,Milan,Venice
A01,A03,B05,C09,F12
Bill,John,Mary,Jane,Jim
50,150,25,80,75

Column family stores: Advantages and disadvantages

- Column stores are beneficial when the typical application is **reading a subset of columns**, or **performing aggregate functions** over them (AVG, MIN, MAX, …)
- They offer efficient storing capabilities due to an **easier compression of data**, which is performed column by column
- Newly inserted tuples have to be **broken down** to their component attributes, and each attribute must be written separately
- Reading overhead for queries that access many attributes at the same time, due to the **tuple reconstruction** process

Column family stores: Available systems

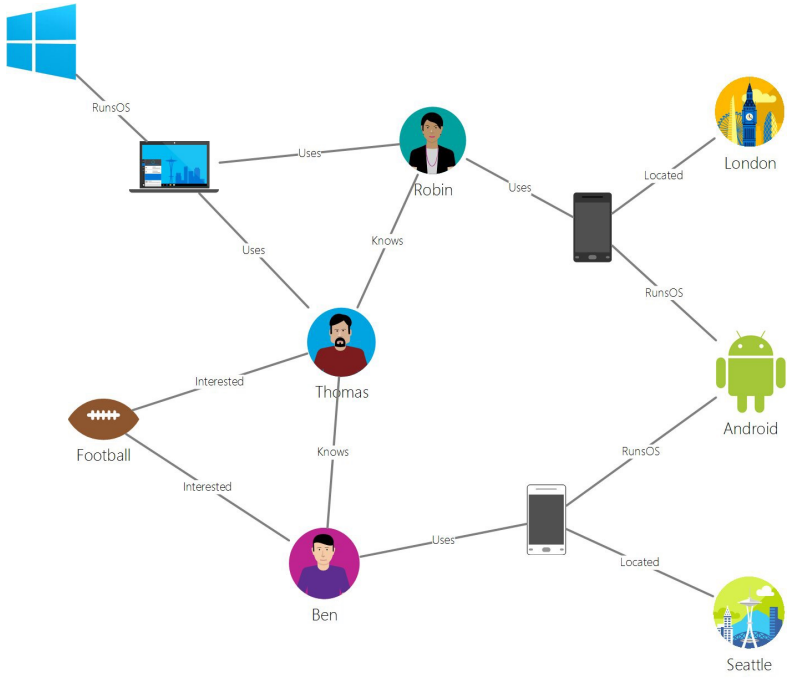
Apache Cassandra: https://cassandra.apache.org/

Vertica: https://www.vertica.com/

HBase: https://hbase.apache.org/

A graph database uses **graph structures** with nodes, edges, and properties to represent and store data

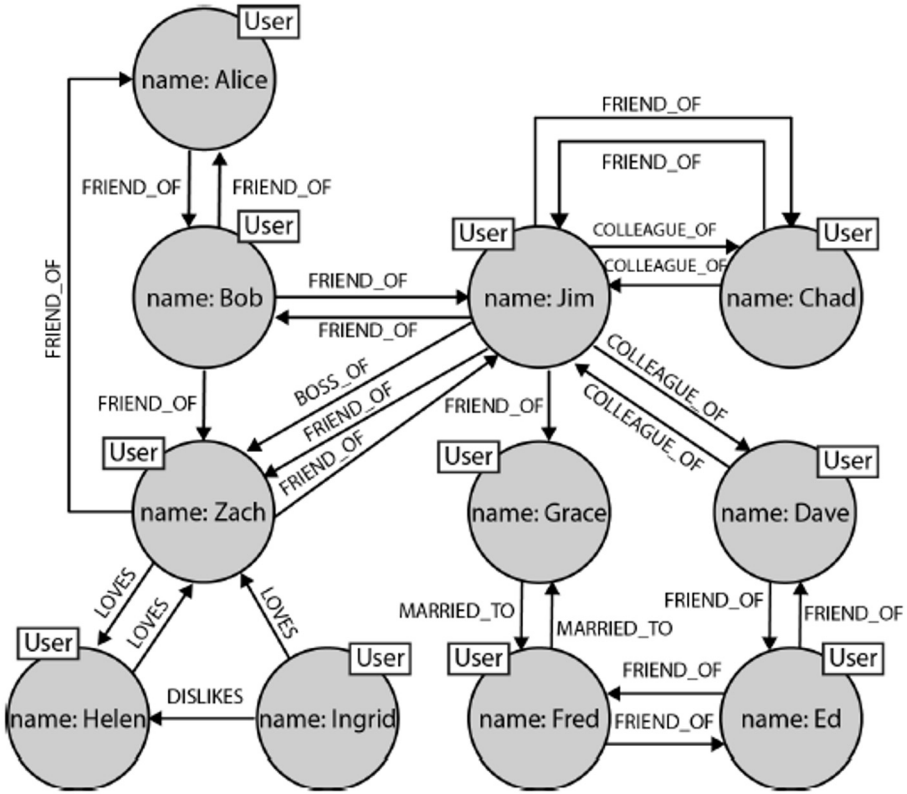
Graph databases can naturally represent certain kinds of semi-structured, **highly interconnected data**, such as those present in social networks, or in geospatial and biotech applications



Graph databases: Example

Connected nodes have **direct pointers** to each other, thus, navigating the graph is extremely fast and simple

The same graph could be modelled using other kinds of NoSQL approaches, as well as RDBMSs, nevertheless, the resulting databases would be **very difficult** to query, update, and populate



Graph VS tables

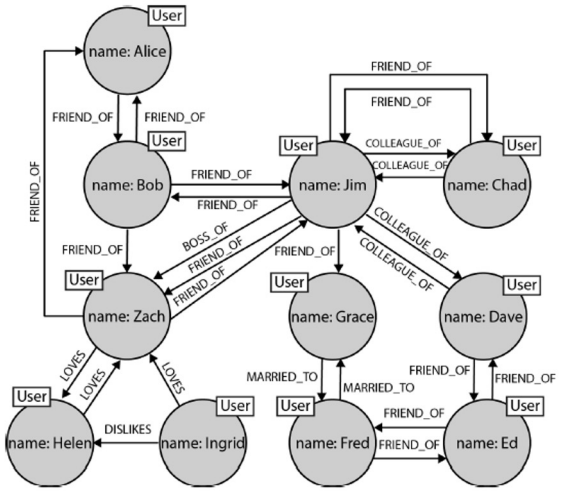


Table “node”

Node kind	Node Name
User	Alice
User	Bob
User	Zach
User	Helen
…	…

Table “edge”

Edge kind	Node from	Node to
Friend of	Bob	Zach
Loves	Zach	Helen
Loves	Helen	Zach
…	…	…

Graph databases can easily answer to queries such as:

- What is the shortest path connecting node X with node Y?
- What are the friends of Bob?
- What are the friends of friends of Bob?

Available systems:

- **Neo4j:** https://neo4j.com/
- **AllegroGraph:** https://allegrograph.com/

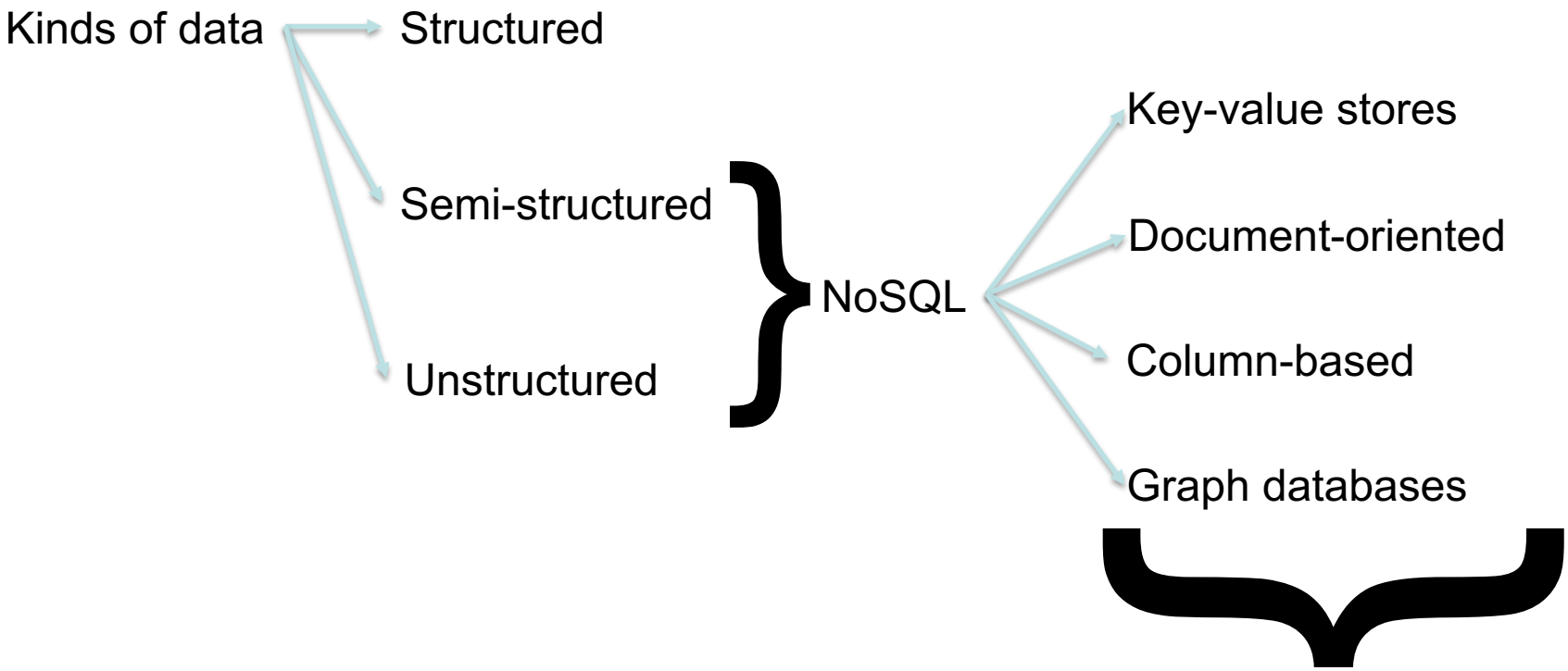
Interactive examples: Neo4J Sandbox

In recent years there has been a progressive (re)approach between relational solutions and NoSQL, with the birth of the so-called **NewSQL**

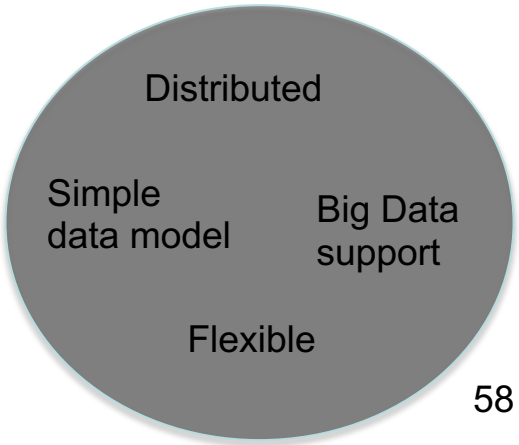
NewSQL refers to DBMSs that try to **combine the performance capabilities of NoSQL with the data guarantees offered by relational databases**

For example, there are NoSQL systems capable of effectively supporting transactions (Neo4j is an example), and others that offer languages inspired by SQL, although generally less powerful than the latter (e.g., Cassandra's CQL)

On the other hand, relational databases such as Postgres allow, today, to store semi-structured data inside tables, thanks to the JSON and JSONB formats, while others offer support to store time series data, generated quickly and in large quantities (e.g., TimescaleDB)


Kinds of data
Structured
Semi-structured
Unstructured
NoSQL
Key-value stores
Document-oriented
Column-based
Graph databases


Relational
NoSQL
NewSQL


Distributed
Simple data model
Big Data support
Flexible

Irrespective from the considered business domain, medium to large sized companies typically have to deal with **several kinds of data**

Such data can be stored into different kinds of systems, or even simply saved as files on disk

This makes **data analyses quite hard**

Motivating example: The contact centre domain

Multi-channel contact centres are an important component of business world

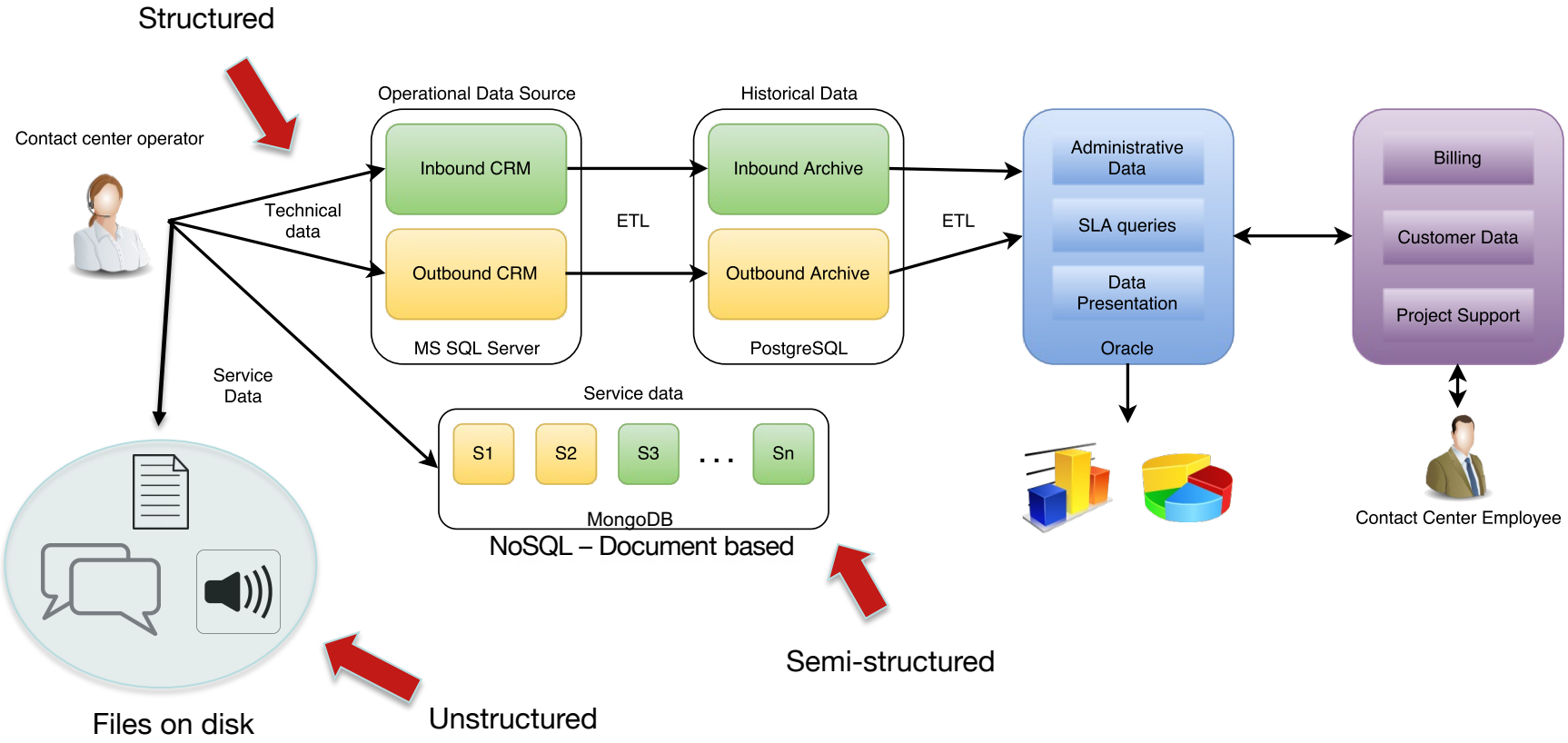
They serve as a primary customer-facing channel for firms in many different industries, and employ millions of agents across the globe

During their operation, they generate vast amounts of **heterogeneous data**: automatic call logs, survey answers, hand-written notes, chat/email messages, voice recordings, …

Operations carried out by contact centres can be broadly classified into:

- **Inbound**: handling incoming traffic
- **Outbound**: performing outgoing calls, sending emails, etc
- **Backoffice**: e.g., data preparation and data analysis tasks

The contact centre domain: A possible starting situation



What are the issues here?

- **Heterogeneous systems** require ad-hoc solutions for reading and writing data
- Different databases adopt **different conventions/formats** for storing the data
- Possibly (and probably) replicated and **inconsistent information**
- Difficult to perform queries and analyses involving **more than one repository**
- Some of the data are **not even considered** for analytics purposes
- Classical, operational systems are **not designed to perform complex analyses**, but to support applications and daily operations (OLTP, On Line Transaction Processing)

How can we solve them?

There is the necessity of having a **clear and uniform view** over all the company data, currently scattered among several systems

This can be obtained by means of an **enterprise-wide repository**, in which information coming from different sources are brought together

Such a repository should be explicitly designed to **support analytics tasks** (OLAP, On Line Analytical Processing)

→ **Data warehousing** provides a solution to all these problems!

According to William Inmon, a data warehouse is a *subject-oriented, integrated, consistent, non-volatile, and time-variant collection of data in support of management's decisions*

Thus, data warehousing is a technique for collecting and managing data originating from multiple sources, so to provide meaningful business insights

The data warehouse makes it easier to perform analysis tasks:

- Single, integrated, source of truth
- It typically poses at the heart of a **decision support system**, i.e., an information system that supports business or organizational decision-making activities

Subject-oriented

The data warehouse focuses on **enterprise-specific concepts**, as defined in the high-level corporate data model

Subject areas may include:

- Customer
- Agent
- Policy
- Claim
- Accident

On the contrary, classical databases may hang on **enterprise-specific applications** (e.g., to provide support to user interfaces and production software)

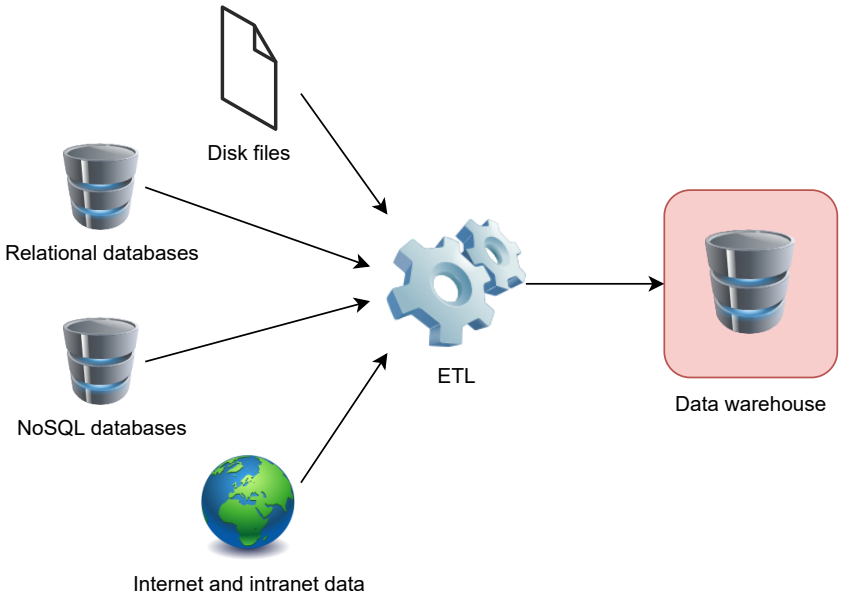
Integrated and consistent

Data is fed from multiple, **disparate sources** into the warehouse

As the data is fed, it is converted, reformatted, restructured, summarized, to conform to the **data warehouse schema**

Data is entered into the data warehouse in such a way that the many inconsistencies at the operational level are **resolved**

Data migration is carried out by means of the **ETL** (Extract, Transform, Load) process



Integrated and consistent: ETL definition

Extract: data is gathered from multiple, heterogeneous sources

Transform:

- *Data cleansing*: removal of errors and inconsistencies
- *Data integration*: reconciliation of data on the same item coming from different sources
- *Data aggregation*: transformation/aggregation of data to match the data warehouse schema (typically, relational)

Load: initial bulk load, and subsequent continuous feed

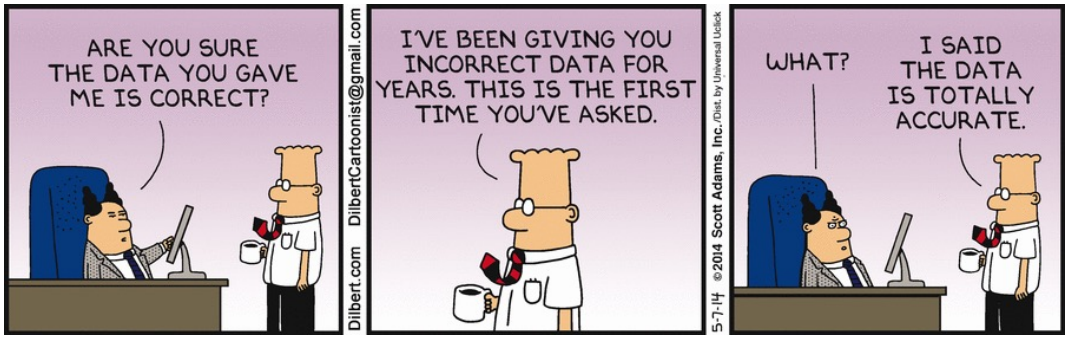
Integrated and consistent: ETL caveats

Before processing all the dirty data, it is important to determine the cleansing and integration cost for every dirty data element (part of the **data quality process**)

Everyone would like to have clean data only, but it is also a **matter of cost and time** to perform ETL

Nevertheless, always remember:

- Garbage in = garbage out
- Your analyses are as good as your data



After the data is inserted in the warehouse, it is **neither changed nor removed**

The only exceptions happen when false data is inserted or the capacity of the data warehouse is exceeded and **archiving** becomes necessary

This means that DWs can be essentially viewed as **read-only databases**

When subsequent changes occur, a new **snapshot** record is written. In doing so, a historical record of data is kept in the data warehouse

Time variancy implies that the warehouse stores data representative as it existed at **many points in time in the past**

A time horizon is the length of time data is represented in an environment; a **5-to-10-year** time horizon is normal for a data warehouse

While operational databases contain current-value data (no history of changes), data warehouses contain **sophisticated series of snapshots**, each taken at a specific moment in time

In OLTP applications, which are typical of classical databases, the user reads and writes small pieces of detailed current-value data, e.g., to perform a bank transfer from one account to another

On the contrary, data warehouses support OLAP operations, in which the user is interested in performing **read-only operations aggregating historical data** over large datasets

E.g., calculate the average amount of money that customers under the age of 20 withdrew in Italy in the years 2018-2022

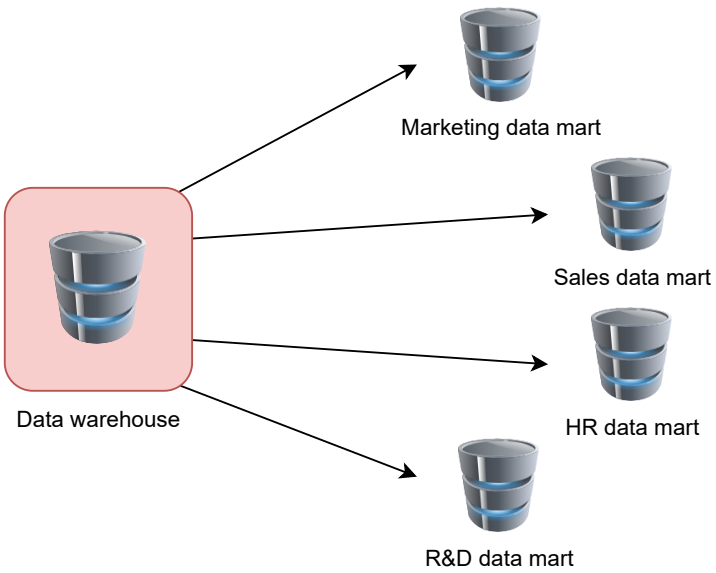
Such operations are typically complex and time consuming

→ **Multidimensional model**

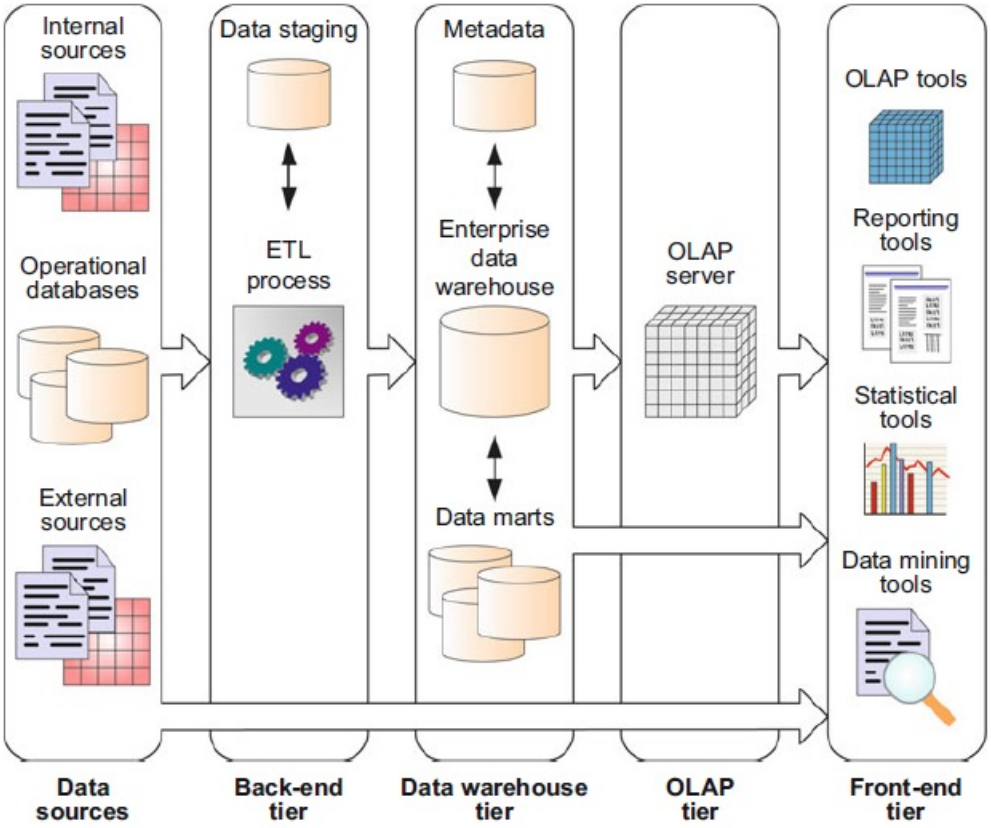
A data mart is focused on a **single functional area** of an organization and it contains a subset of the data stored in a data warehouse

A data mart is a condensed version of a data warehouse and is designed for use by a specific department, unit or set of users in an organization

Data marts are **smaller in size and are more flexible** compared to a data warehouse, and can be fed starting from the data contained in the latter



Overall data warehouse architecture



A data lake is a repository that can store large amounts of structured, semi-structured, and unstructured **data in its native format**, allowing to access them before the ETL phase

Data is only transformed upon usage (**schema on read** vs. schema on write)

Storing information in a data lake is relatively inexpensive w.r.t. a data warehouse

A data lake is not a substitute for a data warehouse; the latter guarantees quick answers to interactive queries thanks to its schema on write approach

If not properly managed, the data lake can easily grow into a **data swamp**!

The distinctive features of OLAP applications suggest the adoption of a **multidimensional representation of data**, since running analytical queries against traditionally stored information would result in complex query specification and long response times

The key idea is that of **pre-aggregating** some of the data

The multidimensional model relies on the concepts of **fact**, **measure**, and **dimension**

A fact is the part of your data that indicates that a **specific event or transaction** has happened, like the sale of a product or receiving a shipment

A fact is composed of multiple **measures**, that describe it

For example, a fact may be that of receiving an order for some shoes, detailed by the two *measures:*

- “price”
- “quantity”

Dimensions provide a way to **categorize/label/index facts**, e.g., considering spatial or temporal aspects. Thus, they allow to filter and group facts

The previous order may be detailed by the following two *measure* and three *dimension* attributes:

- Total amount US$ 750
- Quantity purchased is 10
- Received yesterday at 2 pm
- Served by our store in New York
- Placed by customer #XAZ19

In the multidimensional model, data is represented in an *n*-dimensional space, usually called a **data cube or hypercube**

A data cube is defined by dimensions (cube edges) and facts (cube cells):

- Dimensions are perspectives used to analyse the data
- Facts have related numeric values, the measures

Data cubes can be sparse, meaning that there may not be a cell value for each combination of dimensions

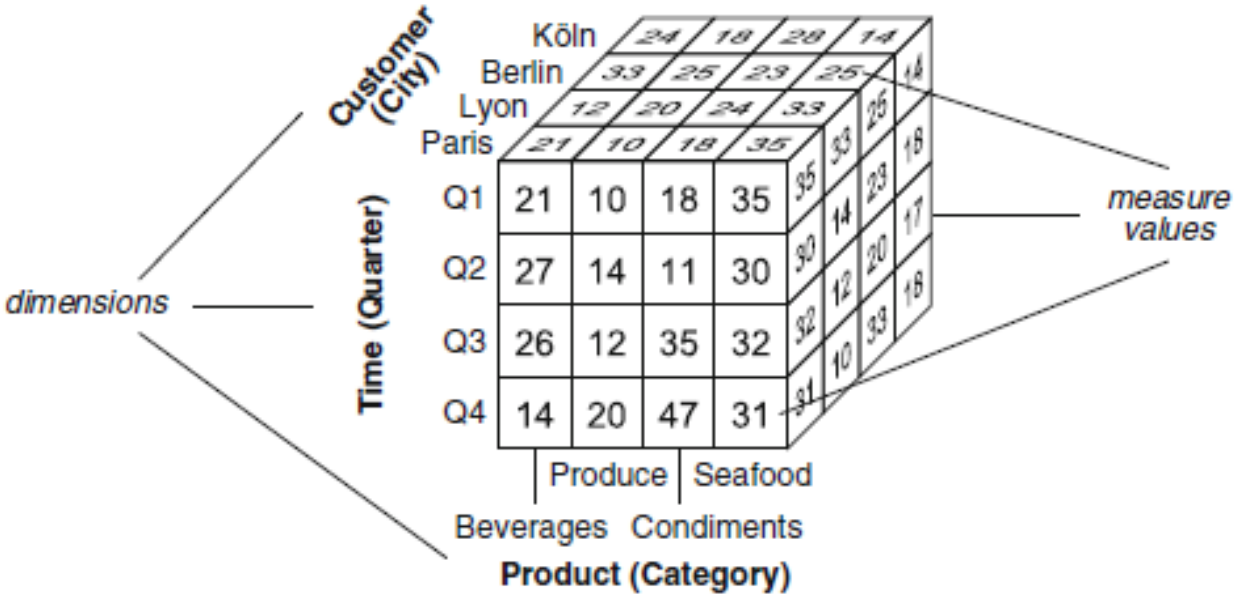
Bi-dimensional pivot table, that considers:

- Measure “Amount”
- Dimensions “Place” and “Product”
- Facts are the amounts of products sold in each country

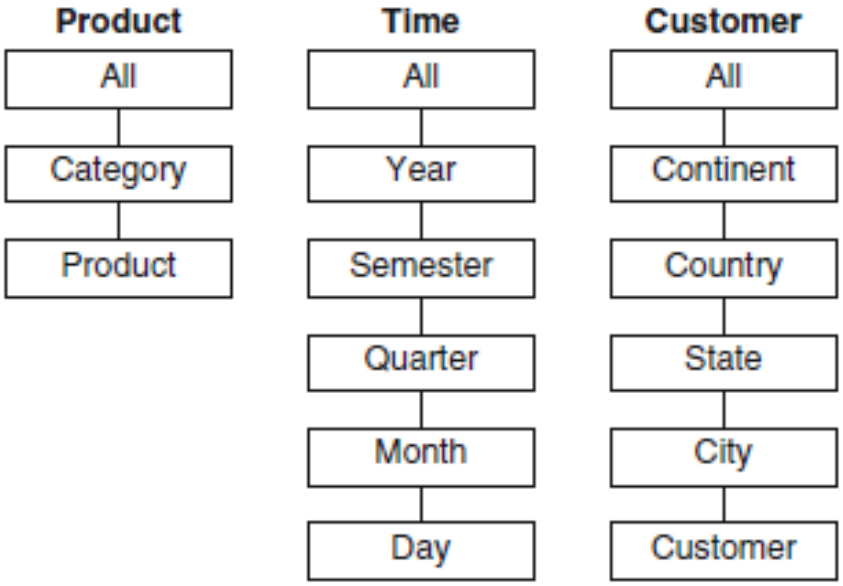
	A	B	C	D	E	F	G	H	I	J
1	Category	(All)								
2										
3	**Sum of Amount**	**Column**								
4	**Row Labels**	**Apple**	**Banana**	**Beans**	**Broccoli**	**Carrots**	**Mango**	**Orange**	**Grand Total**	
5	Australia	20634	52721	14433	17953	8106	9186	8680	131713	
6	Canada	24867	33775		12407		3767	19929	94745	
7	France	80193	36094	680	5341	9104	7388	2256	141056	
8	Germany	9082	39686	29905	37197	21636	8775	8887	155168	
9	New Zealand	10332	40050		4390			12010	66782	
10	United Kingdom	17534	42908	5100	38436	41815	5600	21744	173137	
11	United States	28615	95061	7163	26715	56284	22363	30932	267133	
12	**Grand Total**	**191257**	**340295**	**57281**	**142439**	**136945**	**57079**	**104438**	**1029734**	
13										

3-dimensional OLAP cube example

Amount of products sold in each quarter in different cities



To extract strategic knowledge from a cube, it is necessary to view its data at **several levels of detail**. Dimension hierarchies provide such a capability



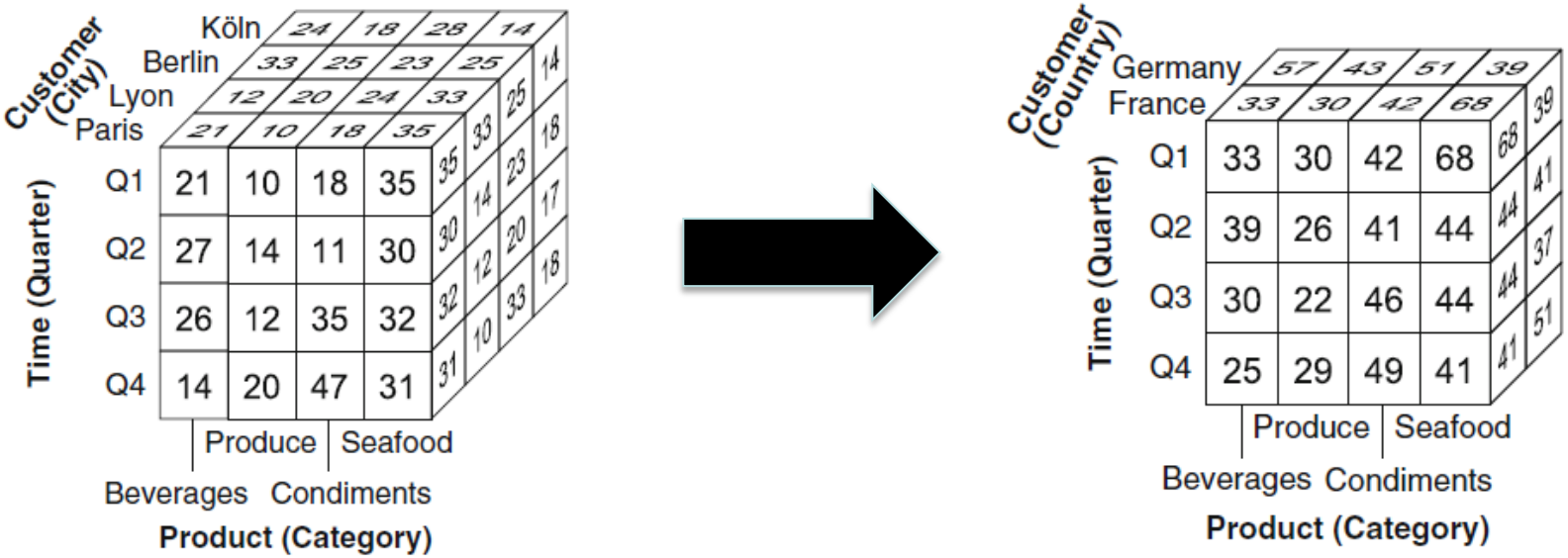
The four types of analytical operations performed on OLAP cubes are:

- Roll up
- Drill down
- Slice and dice
- Pivot (rotate)

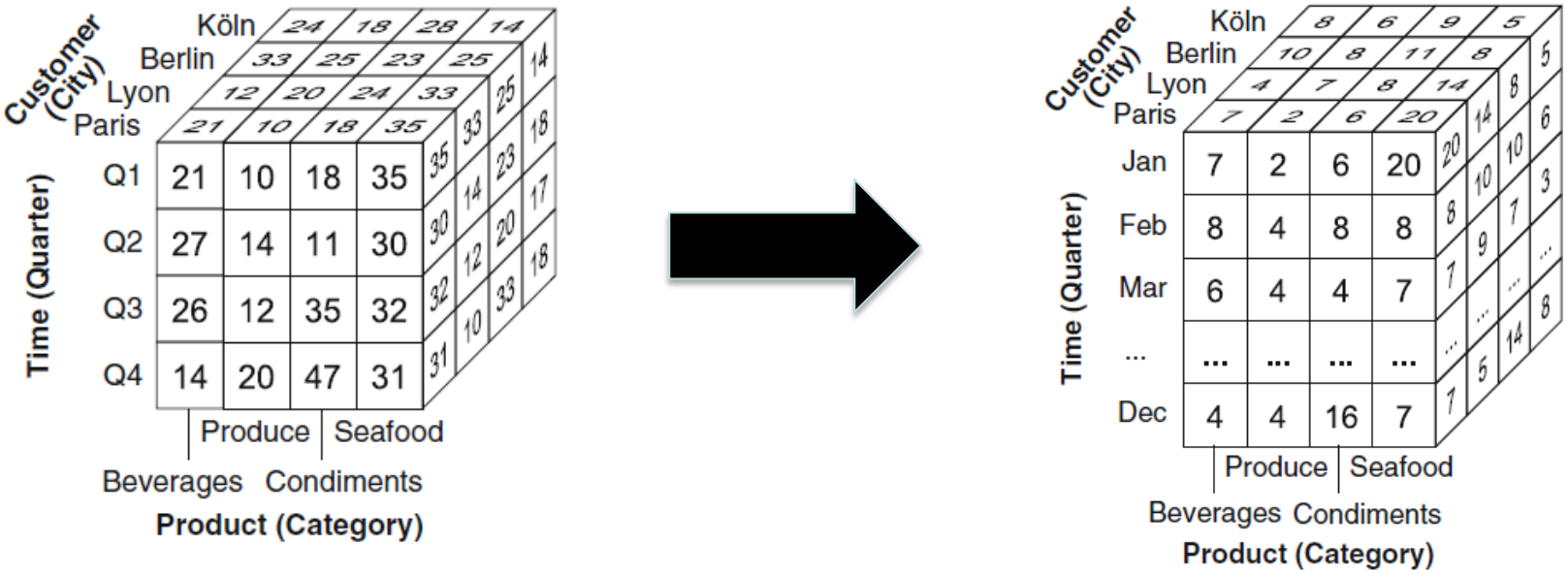
In practice, OLAP cubes are implemented by **pre-aggregating** the data at the maximum level of detail allowed by the dimension hierarchies

"If you don't reveal some insights soon, I'm going to be forced to slice, dice, and drill!"

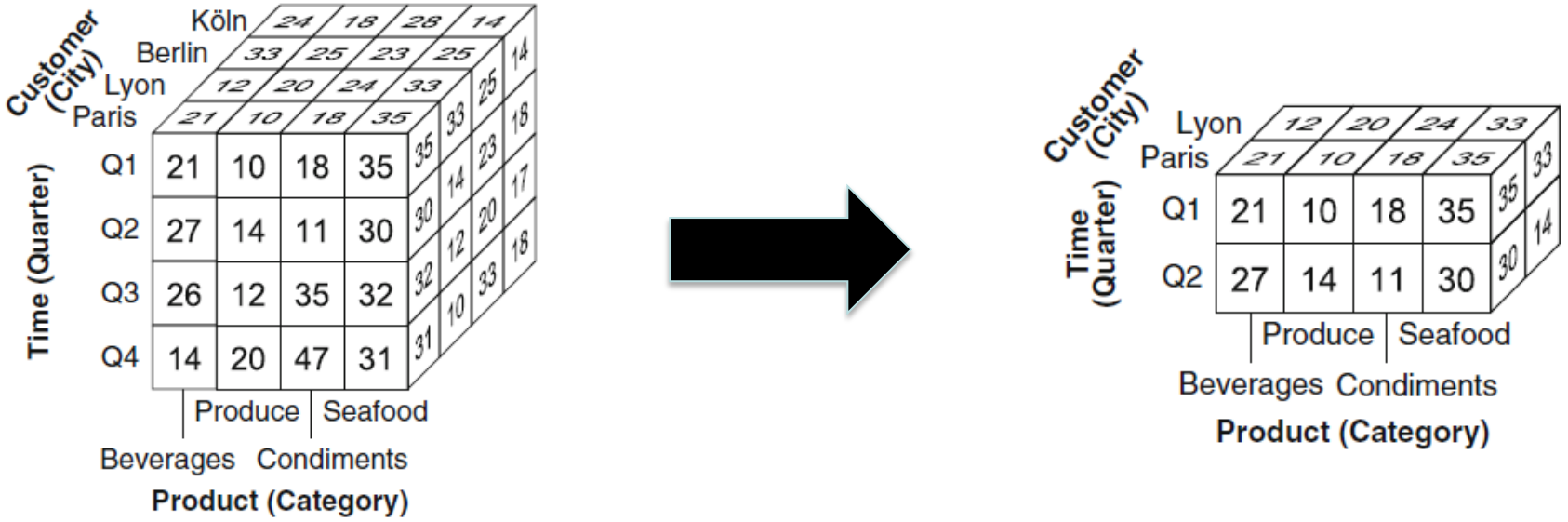
It involves summarizing the data along a chosen dimension (e.g., sum), navigating from a finer level of detail (down) to a coarser (up) along the associated hierarchy



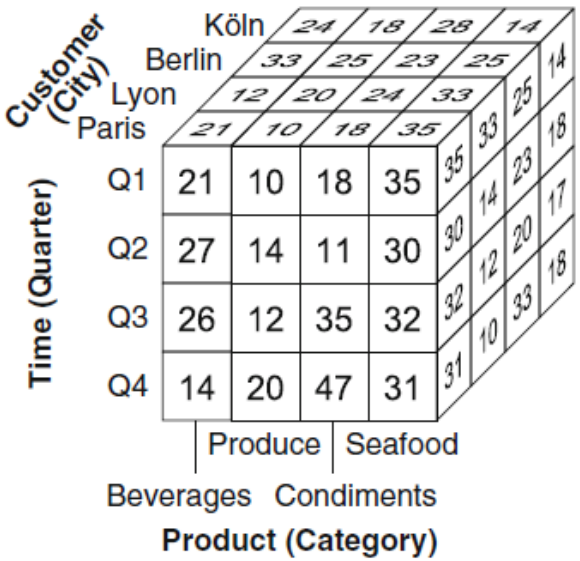
It allows the user to move from summarized (up) to more detailed (down) data along a dimension hierarchy; useful to investigate interesting patterns further



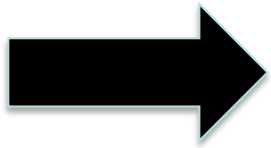
It is the act of picking a subset of a cube by fixing one or more values for one or more of its dimensions

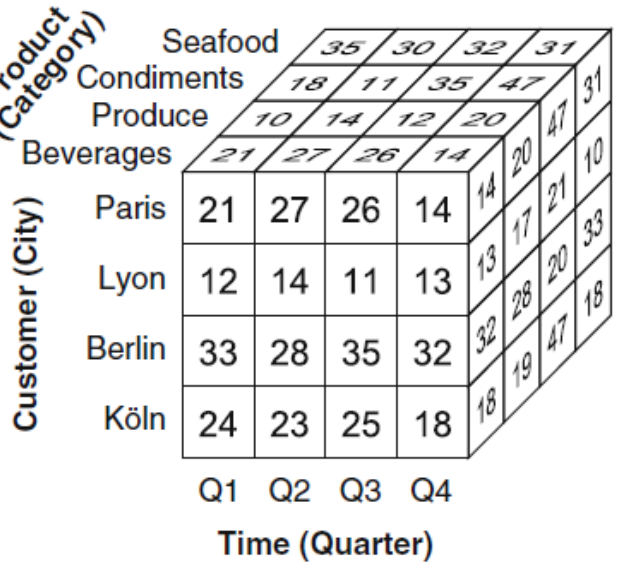


This operation allows an analyst to rotate the cube in space to see its various faces. In other words, the pivot operation allows to look at the data from a different perspective



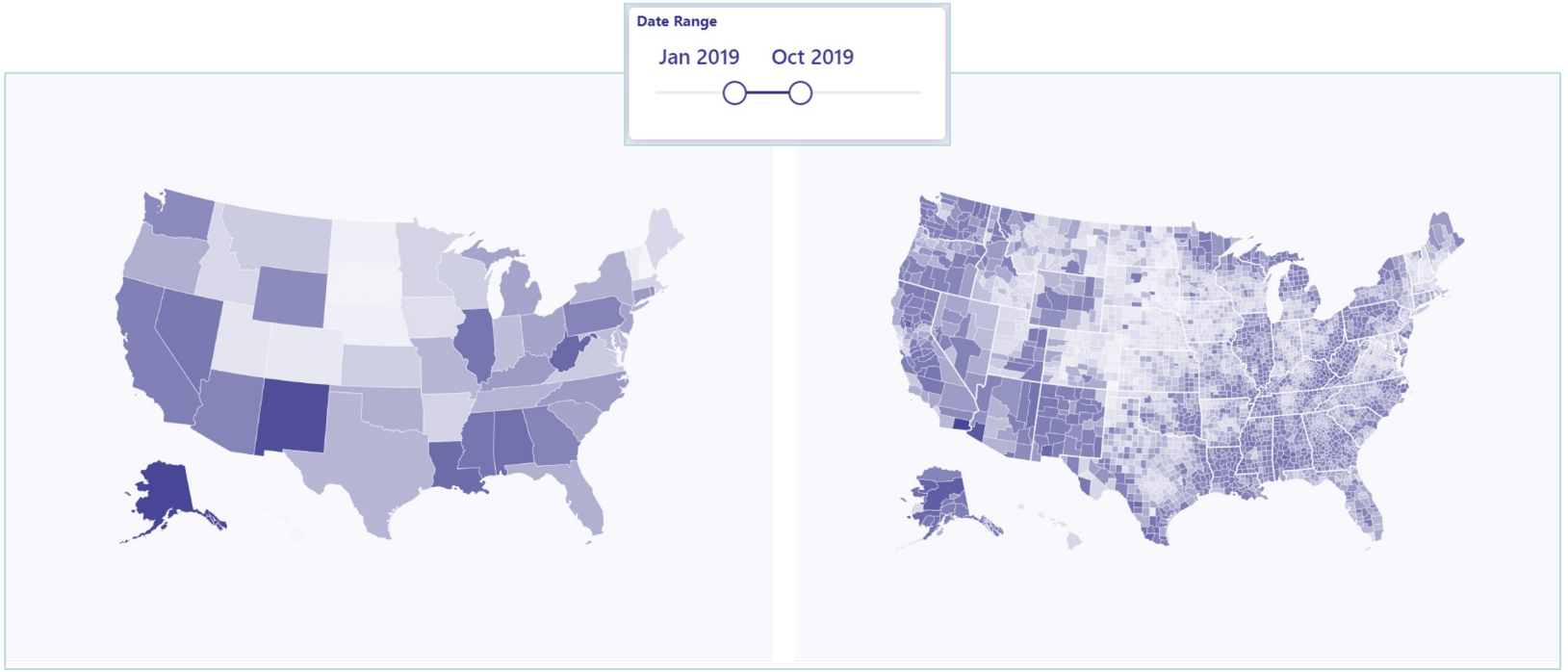
For each City, look at the data in terms of Time and Product





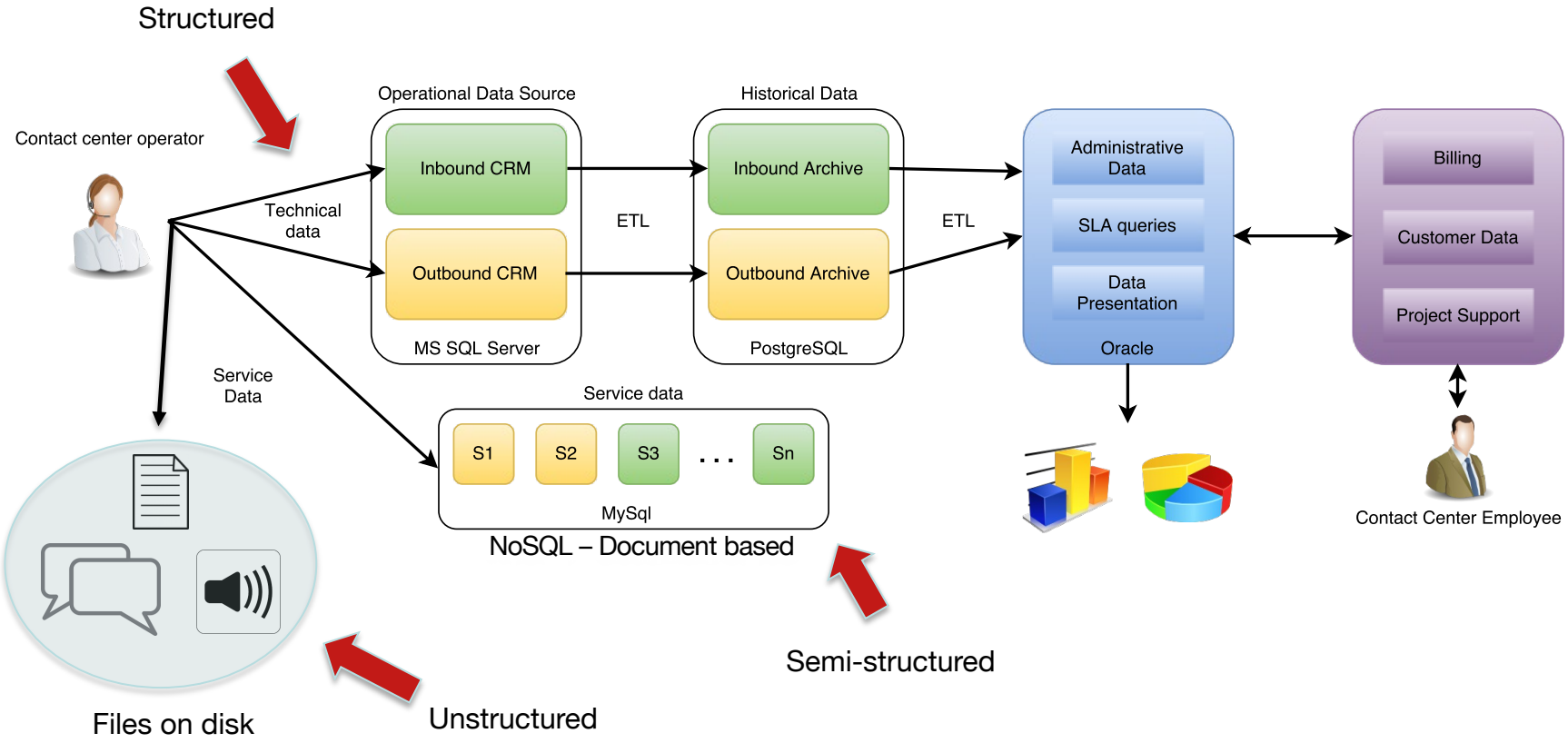
For each product, look at the data in terms of City and Time

Note that hypercubes, like pivot tables, are just an **intuitive representation** of how the data are pre-aggregated and arranged within the system. The information can then be conveyed to the user in several manners, e.g., relying on different (interactive) graphs

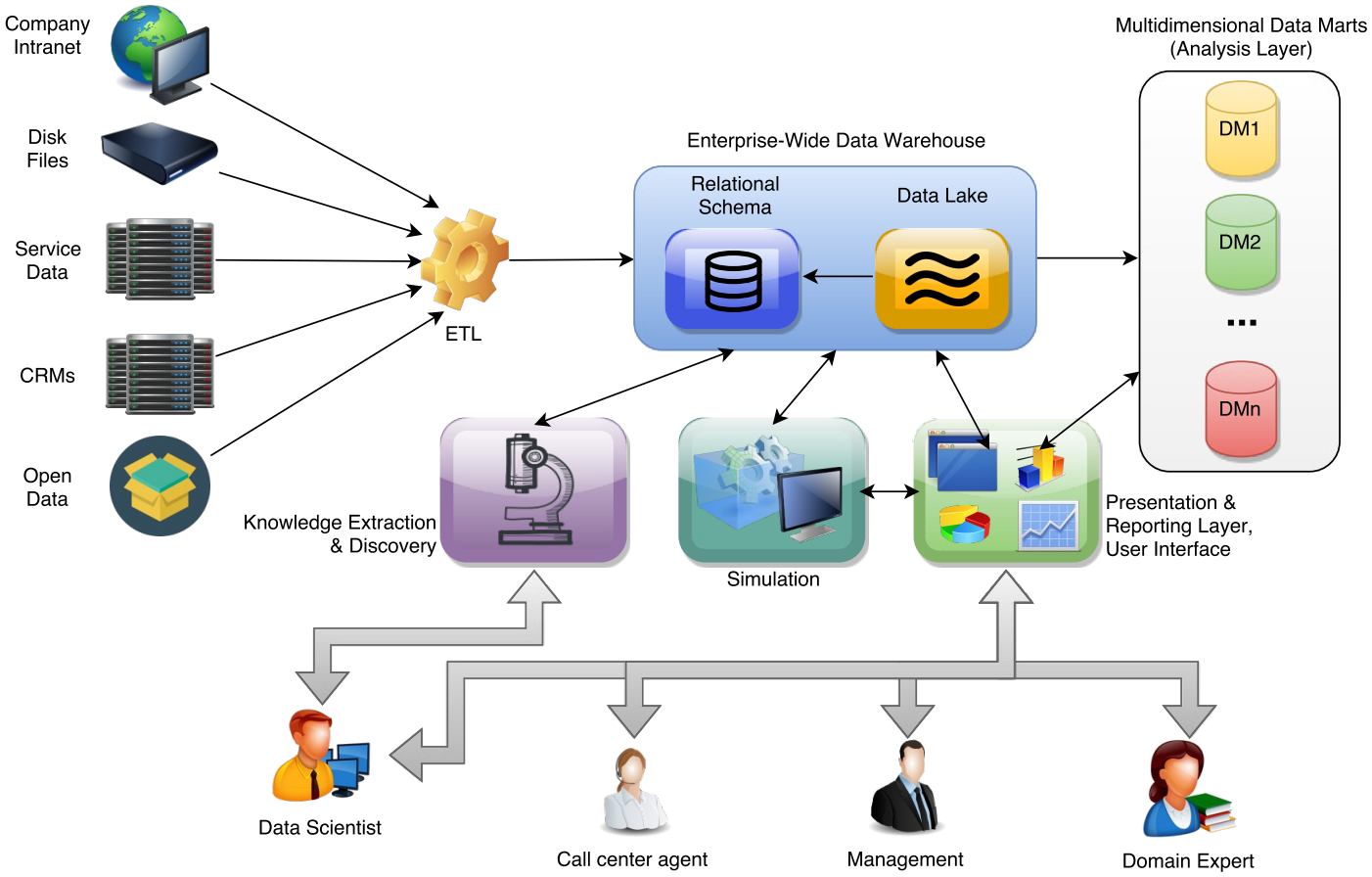


Total number of sales in the, drill-down operation over the spatial dimension

The contact centre domain: A possible starting situation



The contact centre domain: Final decision support system



Collecting data is important but, without analyses, there is no value from them

Creating value from data is also referred to as **data monetization**

Not only analyses… sometimes data monetization pertains to just selling data (e.g., by social networks)

There are three main analysis types to extract value from data:

- **Descriptive** analytics
- **Predictive** analytics
- **Prescriptive** analytics

It is used in almost every company (Business Intelligence)

It is focused on **historical or current data**, and makes use of OLAP and statistical techniques to analyse/summarise data

E.g., drill down, roll up, slicing and dicing, and pivoting operations performed on data cubes

Typical questions:

- How many customers were lost to the competition in the last year?
- What customers are likely to be committing a fraud regarding their currently opened claims (= which customers are currently deviating from the usual behaviour)?

Predictive analytics aims at developing a **vision of the future** making use of past data (Business Analytics)

It heavily relies on statistical analyses and machine learning, in addition to proper data modelling, pre-processing and querying

Typical questions:

- What will the churn rate of customers be in the next three years?
- What customers are most likely to commit a fraud regarding their future claims?

Prescriptive analytics is a relatively recent concept which has its roots in predictive tasks, but it goes even further

It can **suggest** to the decision makers the actions to take in order to reach a desired goal

Typical example:

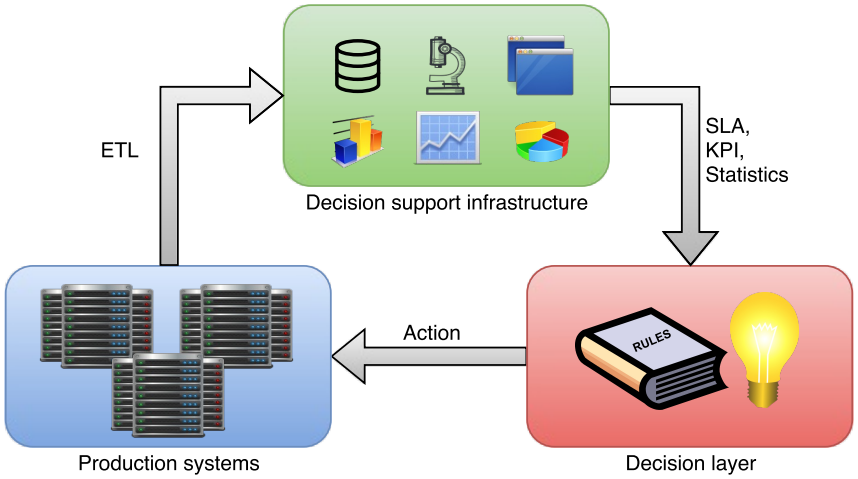
- What are the factors that mostly influence the probability of churn?
- How will the future churn rate be affected if I work on them?

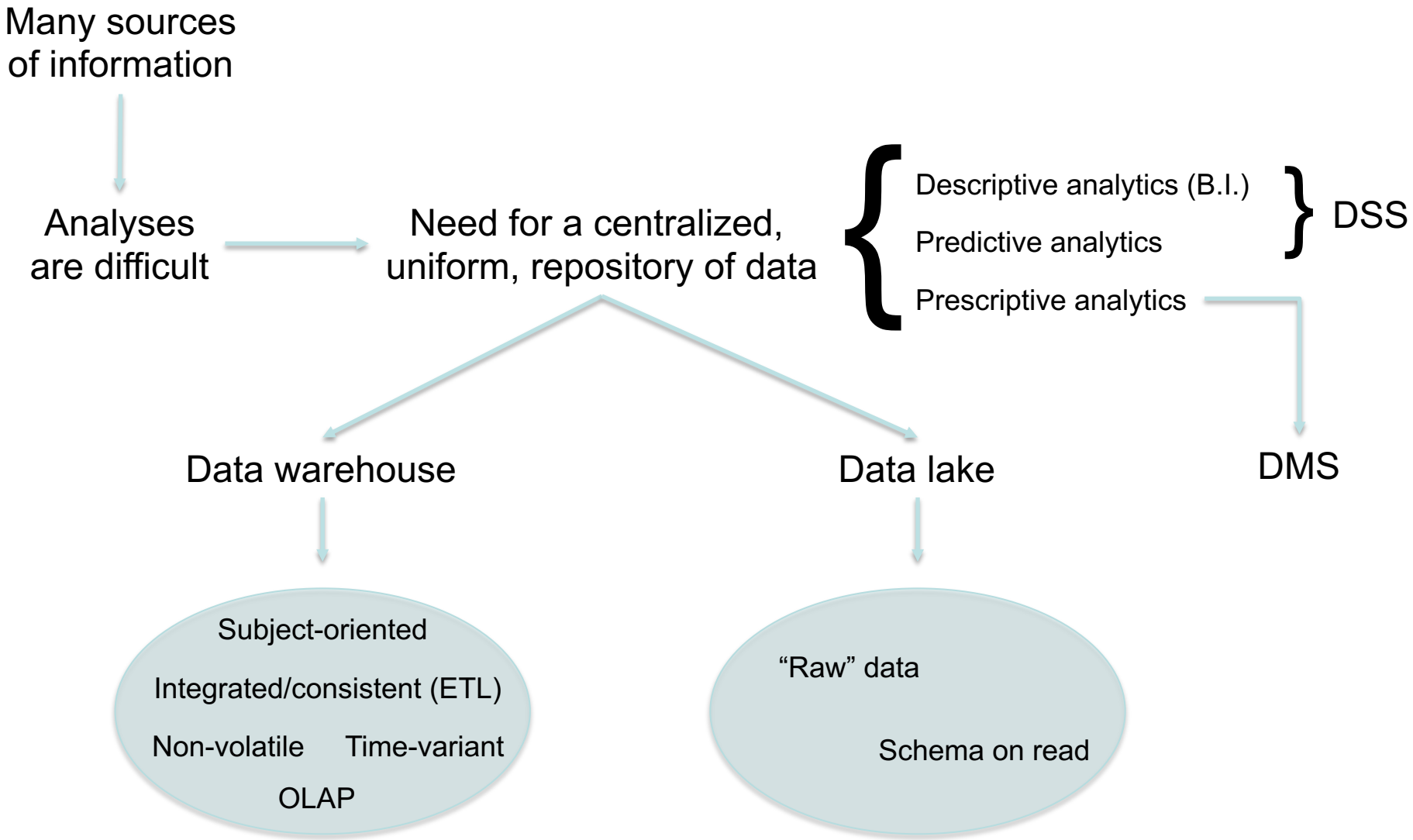
A DMS is an **“action-oriented”** evolution of a DSS

DSSs aim at recommending an action by offering managers information upon which to come up with an idea and ultimately to make a choice

DMSs make one step more and take actions without human intervention based on known information and a set of coded business rules

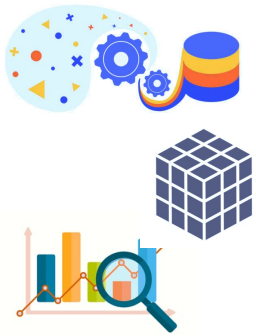
Of course, this mainly involves routinary decisions




Many sources of information
Analyses are difficult
Need for a centralized, uniform, repository of data
Descriptive analytics (B.I.)
Predictive analytics
DSS
Prescriptive analytics
DMS
Data warehouse
Data lake
Subject-oriented
Integrated/consistent (ETL)
Non-volatile
Time-variant
OLAP
“Raw” data
Schema on read

Main take aways

- Data management is a very complex topic, as there are **several kinds of data** (structured, semi-structured, unstructured)
- In addition, Big Data exhibit **Volume**, **Variety**, and **Velocity**
- **DBMS**s allow to store, manage, and retrieve data in a principled manner, addressing much of the inherent complexity
- There are **relational** and **NoSQL** DBMSs, each with pros and cons
- Due to this high heterogeneity, an enterprise information system can become quite complex, and **difficult to use** for data analytics tasks
- **Data warehousing** provides a solution to handle this heterogeneity
- Once the data are correctly modelled, **data monetization** can take place (descriptive, predictive and prescriptive analytics)



- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database system concepts*.
- Gudivada, V. N., Rao, D., & Raghavan, V. V. (2014, June). NoSQL systems for big data management. In *2014 IEEE World congress on services* (pp. 190-197).
- Harrison, G. (2015). *Next Generation Databases: NoSQL and Big Data*. Apress.
- Pavlo, A., & Aslett, M. (2016). What's really new with NewSQL?. ACM Sigmod Record, 45(2), 45-55.
- Inmon, W. H. (2005). *Building the data warehouse*. John Wiley & Sons.