

# Machine Learning

## 2 – Data Engineering

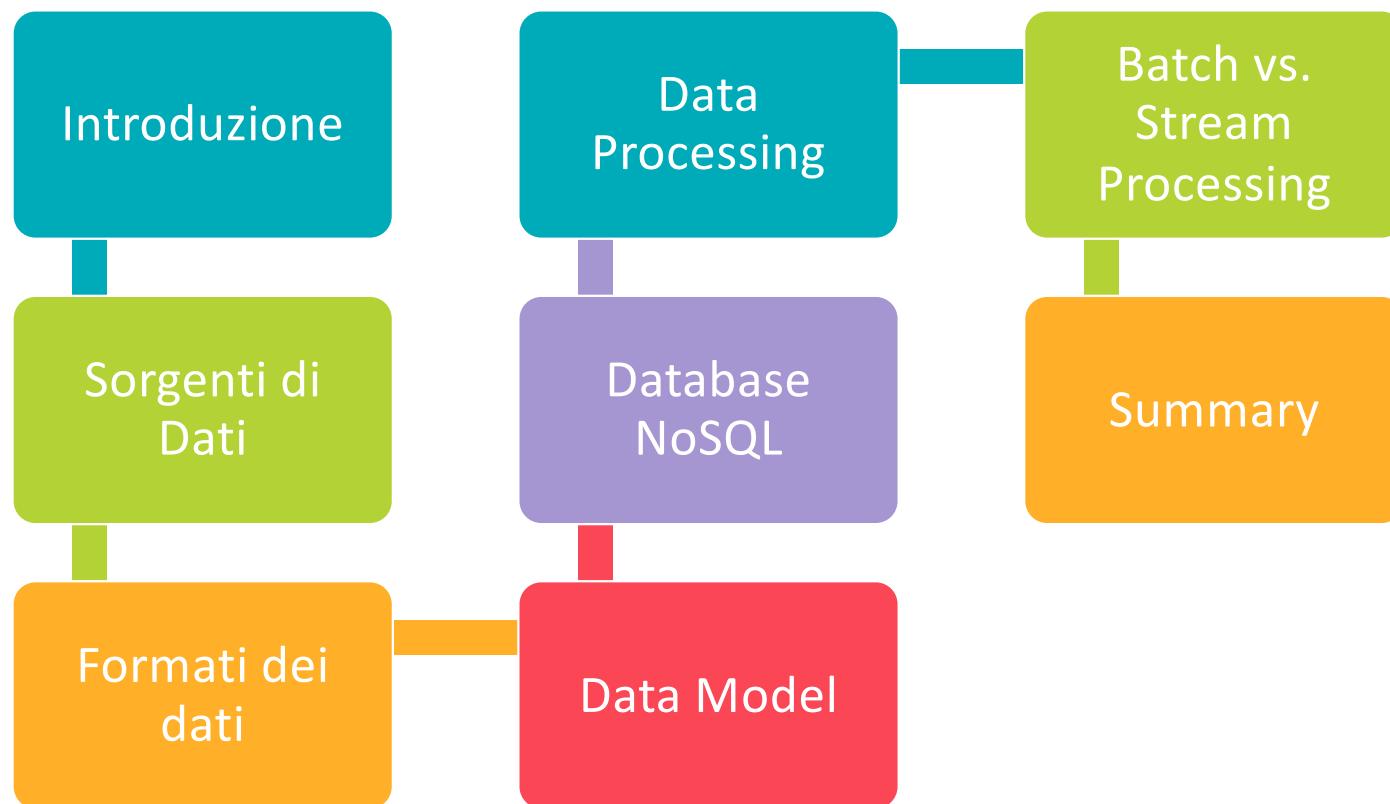


Prof. Giuseppe Polese

Prof.ssa Loredana Caruccio



# Outline



# Introduzione

# Introduzione



## Big Data: ... qualcuno ha sbagliato

### Oggi utilizziamo

- PetaByte ( $10^{15}$ )
- ExaByte ( $10^{18}$ )
- ZettaByte ( $10^{21}$ )
- YottaByte ( $10^{24}$ )

*NOBODY WILL  
EVER NEED MORE  
THAN 640K RAM.*

---

Bill Gates, 1981

---



# Big Data: tra ieri e Oggi

**Michael Cox** e **David Ellsworth** hanno introdotto il termine "Big Data", per esprimere il fatto che i dati delle simulazioni del flusso d'aria intorno agli aeromobili non potessero essere elaborati e/o visualizzati

1997

Oggi

I **big data** raccolgono informazioni da social network, video, comunicazioni su Internet, dispositivi mobili, settore sanitario (cartelle cliniche, MRI, ecc.), scienza (astronomia, meteorologia, ecc), trasporti (traffico terrestre, aereo, marittimo), finanza, sensori e dispositivi intelligenti provenienti dall'Internet of Things (IoT)

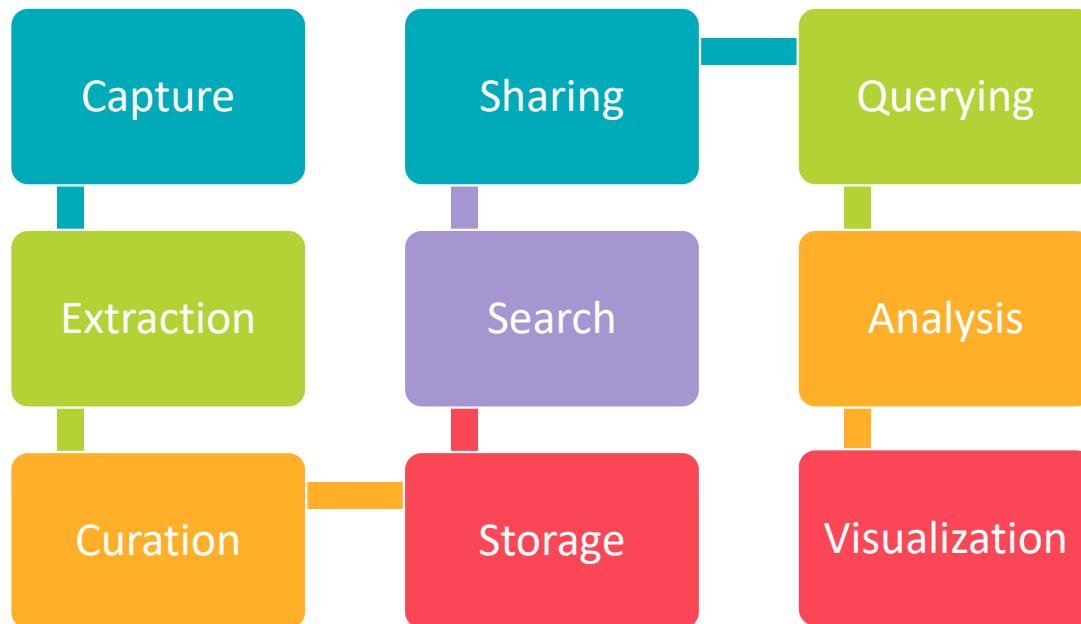
## Big Data: Definizione

La definizione del NIST (National Institute of Standards and Technology dice:

*I Big Data consistono in ampi insiemi di dati, principalmente caratterizzati per **volume, varietà, velocità** e/o **variabilità**, che richiedono un'architettura scalabile per l'efficiente archiviazione, manipolazione e analisi*

- **Volume:** processare un'enorme quantità di dati (raccolta, archiviazione, recupero, elaborazione, aggiornamento)
- **Varietà:** raccogliere dati da molte sorgenti che possono essere rappresentati in forme omogenee (strutturate) o eterogenee (non strutturate)
- **Velocità:** raccogliere dati ad alta velocità ed in tempo reale
- **Variabilità:** cambiare l'interpretazione dei dati in base al contesto in cui sono collezionati ed analizzati

## Big Data: Un'altra definizione



*I big data sono una raccolta di insiemi di dati così grandi e complessi da diventare difficili da elaborare con gli strumenti di gestione dei database disponibili o con le applicazioni tradizionali di elaborazione dei dati*

- Sapere come raccogliere, elaborare, archiviare, recuperare e gestire una quantità sempre crescente di dati è essenziale per coloro che desiderano costruire sistemi di ML

## Big Data nel business

### **Amazon.com**

- Milioni di operazioni di back-end ogni giorno
- Catalogo, ricerche, click, liste dei desideri, carrelli, venditori terzi, ...



### **Walmart**

- > 1 milione di transazioni di clienti all'ora
- 2,5 petabyte (2560 terabyte)



### **Facebook**

- 4PT di dati aggiunti ogni giorno (2018)
- 1 miliardo di foto in un giorno (Halloween)



### **FICO** - Rilevamento delle frodi sulle carte di credito

- Protegge 2,1 miliardi di conti attivi



# Big Data nella PA

## USA: **Big Data Research and Development Initiative**

- Ha esplorato il modo in cui i big data affrontano i problemi importanti del governo
- 84 diversi programmi sui big data distribuiti in sei dipartimenti

## Dati.gov

- > 104.000 di dataset

Il governo possiede sei dei dieci supercomputer più potenti al mondo.

## NASA Center for Climate Simulation

- 32 petabyte di osservazioni e simulazioni

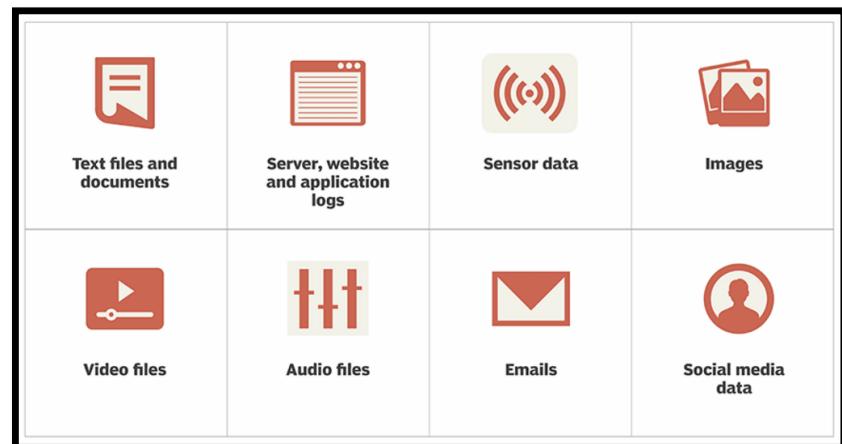
Topics	
A-Z	1-9
Manufacturing (70)	
Ecosystems (75)	
Climate (108)	
Law (120)	
World Wide Human Ge... (145)	
Education (147)	
BusinessUSA (218)	
Agriculture (223)	
Research (227)	
Finance (235)	
Safety (327)	
Consumer (329)	
Ocean (376)	

# Sorgenti di Dati

2

## Dati generate dagli utenti

- Prima di iniziare a lavorare con il Machine Learning (ML), è essenziale capire le diverse sorgenti di dati, le loro caratteristiche e il loro scopo
- Una fonte è costituita dai dati immessi dagli utenti, ovvero i dati inseriti esplicitamente dagli utenti.
- L'input dell'utente può essere
  - Testo
  - Immagini
  - Video
  - File



## Dati generati dagli utenti (2)

- Gli utenti possono facilmente inserire dati errati
  - testi troppo lunghi o troppo corto
  - valori numerici al posto del testo
  - file con formati errati
- I dati di input utente richiedono processi di verifica e validazione aggiuntivi per garantire la **qualità dei dati**.
- È necessaria, tuttavia, anche garantire un'elaborazione rapida dei dati inseriti dagli utenti data la loro poca pazienza nell'attendere i risultati.



## Dati generati dai sistemi

- I log possono registrare lo stato e gli eventi significativi del sistema
  - l'utilizzo della memoria,
  - il numero di istanze,
  - i servizi chiamati,
  - i pacchetti utilizzati, ecc.
- È molto meno probabile che questo tipo di dati siano malformattati come i dati inseriti dagli utenti
- È necessaria, tuttavia, anche garantire un'elaborazione rapida dei dati inseriti dagli utenti data la loro poca pazienza nell'attendere i risultati.
- Anche se si tratta di dati generati dal sistema, sono comunque considerati parte dei dati dell'utente e potrebbero essere soggetti alle norme sulla privacy.

## Database Aziendali

- Esistono poi i database aziendali generati dalle applicazioni aziendali. Gli stessi gestiscono vari asset come
  - l'inventario,
  - le relazioni con i clienti,
  - gli utenti, ecc.
- Questo tipo di dati può essere utilizzato dai modelli di ML direttamente o da vari componenti di un sistema di ML.

### Esempio:

*Se qualcuno digita "frozen", sta cercando alimenti surgelati o Frozen della Disney?*

*Amazon deve verificare la disponibilità di questi prodotti nei suoi database interni prima di classificarli e mostrarli agli utenti.*

## Dati delle terze parti

- I dati delle **prime parti** sono i dati sui clienti che le aziende collezionano direttamente
- I dati delle **seconde parti** sono i dati altre aziende collezionano sui proprio clienti e che li rendono disponibili
- I dati delle **terze parti** sono i dati sui clienti che collezionano aziende preposte per la raccolta di dati di persone che non sono propri clienti

*È possibile acquistare dati di tutti i tipi, come le attività sui social media, la cronologia degli acquisti, le abitudini di navigazione sul web, l'orientamento politico, ecc.*

- Questi dati possono essere particolarmente utili per sistemi come i sistemi di raccomandazione per generare risultati pertinenti agli interessi degli utenti.
- Le nuove regolamentazione sulla data privacy ha ridotto in modo significativo la quantità di dati di terze parti.

# Formati dei Dati

3

## La scelta del formato dei dati

- Dato che i dati provengono da più fonti con diversi modelli di accesso, l'archiviazione dei dati non è sempre semplice
- È importante è importante pensare a come verranno utilizzati i dati in futuro, in modo che il formato utilizzato abbia senso
- Alcune domande da porsi sono:
  - Come si memorizzano i dati multimodali, ad esempio un campione che può contenere sia immagini che testi?
  - Dove posso archiviare i miei dati in modo economico e in modo che siano veloci da consultare?
  - Come posso archiviare modelli complessi in modo che possano essere caricati ed eseguiti correttamente su hardware diverso?

## La serializzazione dei dati

- La **serializzazione dei dati** rappresenta il processo che effettua la conversione delle strutture dati o dello stato degli oggetti allo scopo che i dati possano essere memorizzati o trasmessi ed eventualmente ricostruiti dopo.
- Quando si considera un formato di dati con cui lavorare, si possono prendere in considerazione diverse caratteristiche, come:
  - la leggibilità
  - i modelli di accesso
  - il tipo di rappresentazione (binaria, testuale, ecc.), che influenza la dimensione dei file

## Tipi di formato

- In Tabella sono riportati i principali tipi di formato<sup>1</sup> che si utilizzano

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	<i>Text</i>	Yes	<i>Everywhere</i>
<i>Parquet</i>	<i>Binary</i>	No	<i>Hadoop, Amazon Redshift</i>
Avro	Binary primary	No	Hadoop
Protobuf	Binary primary	No	Google, TensorFlow (TFRecord)
Pickle	Binary	No	Python, PyTorch serialization

1. [https://en.wikipedia.org/wiki/Comparison\\_of\\_data-serialization\\_formats](https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats)

# JSON

```
{  
  "firstName": "Boatie",  
  "lastName": "McBoatFace",  
  "isVibing": true,  
  "age": 12,  
  "address": {  
    "streetAddress": "12 Ocean  
    "city": "Port Royal",  
    "postalCode": "10021-3100"  
  }  
}
```

## Esempio

- **JSON** (JavaScript Object Notation) è tra i formati di rappresentazione maggiormente utilizzati oggi.
- Anche se deriva da JavaScript, è **indipendente dai linguaggi**: la maggior parte dei linguaggi di programmazione moderni possono generare e analizzare dati rappresentati in JSON.
- Il suo paradigma **chiave-valore** è semplice ma potente, in grado di gestire dati con diversi livelli di struttura.

## JSON: Issues

```
{  
  "firstName": "Boatie",  
  "lastName": "McBoatFace",  
  "isVibing": true,  
  "age": 12,  
  "address": {  
    "streetAddress": "12 Ocean  
    "city": "Port Royal",  
    "postalCode": "10021-3100"  
  }  
}
```

Esempio

I principali problemi del formato JSON sono:

1. Una volta che i dati nei file JSON sono stati formattati in uno schema è abbastanza **difficile tornare indietro** per modificare lo schema.
2. I file JSON sono file di testo, quindi **occupano molto spazio**.

# CSV e Parquet

- I formati CSV e Parquet rappresentano due paradigmi distinti
  - CSV è di tipo **row-major**, ovvero gli elementi consecutivi di una riga sono memorizzati uno accanto all'altro
  - Parquet è di tipo **column-major**, ovvero gli elementi consecutivi di una colonna sono memorizzati uno accanto all'altro.

Column-major:			
Row-major:			
<ul style="list-style-type: none"><li>• Data is stored and retrieved column by column</li><li>• Good for accessing features</li></ul>			
Example 1	...	...	...
Example 2	...	...	...
Example 3	...	...	...

## Formati Row-Major

- Poiché i computer moderni elaborano i dati sequenziali in modo più efficiente rispetto ai dati non sequenziali, se una tabella è row-major, **l'accesso alle sue righe sarà più veloce** dell'accesso alle sue colonne.

*Immaginiamo di avere un dataset di 1.000 sample e che ogni sample abbia 10 feature. Se consideriamo ogni sample come una riga e ogni features come una colonna, come spesso accade in ML, allora i formati di tipo row-major come CSV sono migliori per accedere ai sample*

### Esempio:

Accedere a tutti gli sample raccolti oggi

## Formati Column-Major

- I formati Column-major consentono una lettura flessibile basata sulle colonne, soprattutto se i dati sono grandi con migliaia, se non milioni, di feature.

*Immaginiamo di avere un dataset di transazioni di ride-sharing con 1.000 feature. Se consideriamo ogni features come una colonna allora i formati di tipo column-major come Parquet sono migliori per filtrare le feature*

### **Esempio:**

Accedere ai dati delle sole quattro feature ora, posizione, distanza e prezzo

## Row-Major Vs. Column-Major

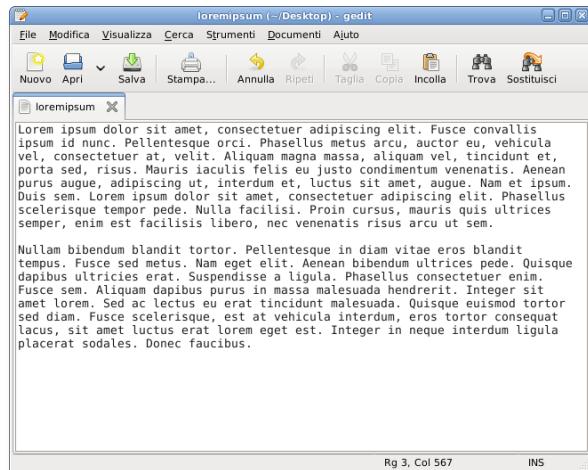
- I formati row-major consentono una scrittura più rapida dei dati. Considerate la situazione in cui dovete aggiungere sempre nuovi sample individuali ai dati.
  - Per ogni singolo esempio, sarebbe molto più veloce scriverlo in un file in cui i dati sono già in un formato row-major.
- In generale
  - I formati a row-major sono migliori quando si devono effettuare molte scritture
  - I formati column-major sono migliori quando si devono leggere molte colonne.

## Formato Testo Vs. Formato Binario

- I **file di testo** sono file di testo che sono testo in chiaro
  - File che sono leggibili dall'uomo
- I **file binari** sono il termine generico che si riferisce a tutti i file non testuali
  - File che contengono solo 0 e 1 e sono destinati a essere letti o utilizzati da programmi che li sanno interpretare
  - Un programma deve sapere esattamente come sono disposti i dati all'interno del file binario

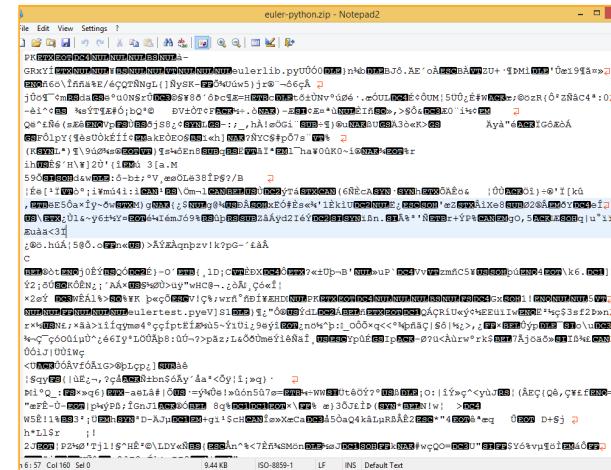
# Formato Testo Vs. Formato Binario

## File di Testo



Se si aprono i file di testo nell'editor di testo (ad esempio, VS Code, Notepad), si è in grado di leggere i testi in essi contenuti.

## File binario



Se si apre un file binario nell'editor di testo, si vedranno blocchi di numeri, probabilmente in valori esadecimales.

## Formato Testo Vs. Formato Binario: Esempi

- I file binari sono più leggeri in termini di memoria richiesta per memorizzarli.
- Supponiamo di voler memorizzare il numero 1.000.000:
  - Se lo si memorizza in un file di testo, richiederà 7 caratteri, e se ogni carattere è 1 byte, richiederà  $7 \times 1 = \mathbf{7 \text{ bytes}}$
  - Se lo si memorizza in un file binario come **int32**, occorreranno solo **4 byte**

```
Path("data/interviews.csv").stat().st_size  
14200063 ←  
  
df.to_parquet("data/interviews.parquet")  
Path("data/interviews.parquet").stat().st_size  
6211862 ←
```

**interviews.csv**, un file CSV (formato testo) di 17.654 righe e 10 colonne.

**interviews.parquet**, il precedente dataset convertito in formato binario.

# Data Processing

4

## Il concetto di transazione

- Una **transazione** si riferisce a qualsiasi tipo di azione:
  - *Twittare, ordinare un passaggio attraverso un servizio di ride-sharing, caricare un nuovo modello, guardare un video su YouTube, ecc.*
- Anche se queste diverse transazioni coinvolgono diversi tipi di dati, il modo in cui vengono elaborate è simile in tutte le applicazioni.
- Questo tipo di elaborazione è noto come transazioni online (**OLTP – Online Transaction Processing**)

## Database Transazionali

- Un DBMS multiutente deve consentire **accesso a più utenti** contemporaneamente.
  - Le transazioni devono essere elaborate velocemente (**bassa latenza**) in modo da non far aspettare gli utenti.
  - **Condivisione di dati** e trattamento di transazioni multiutente.
  - Deve includere software per il **controllo della concorrenza** che garantiscano l'aggiornamento corretto  
*Esempio: il problema della prenotazione di posti per una compagnia aerea (applicazione di transaction processing).*

## Proprietà delle Transazioni

- Le transazioni dovrebbero possedere alcune proprietà (dette **ACID properties**, dalle loro iniziali):

**Atomicità:** una transazione è un'unità atomica di elaborazione da eseguire o completamente o per niente (*responsabilità del recovery subsystem*).

**Consistency preserving:** una transazione deve far passare il database da uno stato consistente ad un altro (*responsabilità dei programmatori*).

**Isolation:** Una transazione non deve rendere visibili i suoi aggiornamenti ad altre transazioni finché non è committed (*responsabilità del sistema per il controllo della concorrenza*).

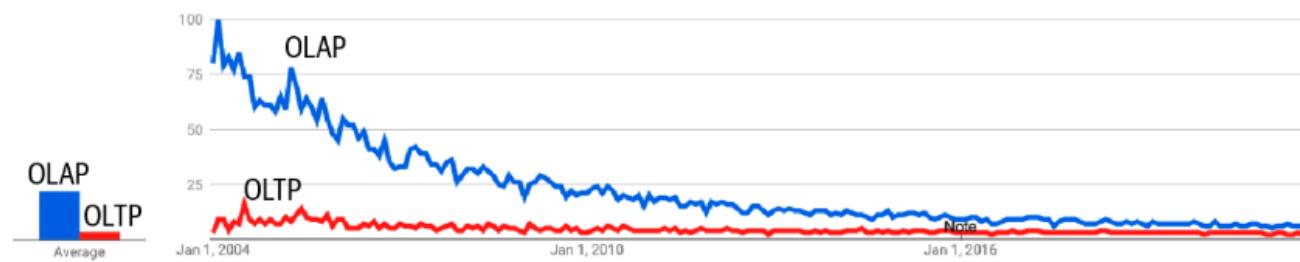
**Durability:** Se una transazione cambia il database, e il cambiamento è committed, queste modifiche non devono essere perse a causa di fallimenti successivi (*responsabilità del sistema di gestione dell'affidabilità*)

## Operazioni analitiche

- I database transazionali potrebbero non essere efficienti per domande come "*Qual è il prezzo medio di tutte le corse a settembre a San Francisco?*"
  - Questo tipo di domanda analitica richiede aggregare i dati in colonne su più righe di dati.
  - I database analitici sono progettati per questo scopo.
  - Sono efficienti con le query che consentono di esaminare i dati da dati da diversi punti di vista.
  - Questo tipo di elaborazione si chiama **online analytical processing (OLAP)**.

## OLTP e OLAP

- Tuttavia, i termini OLTP e OLAP sono diventati obsoleti per tre motivi:
  - La separazione tra database transazionali e analitici era dovuta ai limiti della tecnologia. Tuttavia, questa separazione è stata eliminata.
  - Nei paradigmi OLTP o OLAP tradizionali, l'archiviazione e l'elaborazione sono strettamente accoppiate. È stato introdotto disaccoppiamento.
  - "Online" è diventato un termine che può significare molte cose diverse.



# Data Model

5

## Modello dei Dati



Un modello deve **rappresentare una certa realtà**

- Esempio:  
Una mappa rappresenta una porzione di territorio. È quindi un modello del territorio, che rappresenta alcune caratteristiche, nascondendone altre

## Modello dei Dati



Un modello deve **fornire un insieme di strutture** simboliche per rappresentare una realtà, nascondendo alcuni dettagli

- Esempio:
  - Una mappa ha una serie di simbolismi grafici per rappresentare gli aspetti salienti di un territorio

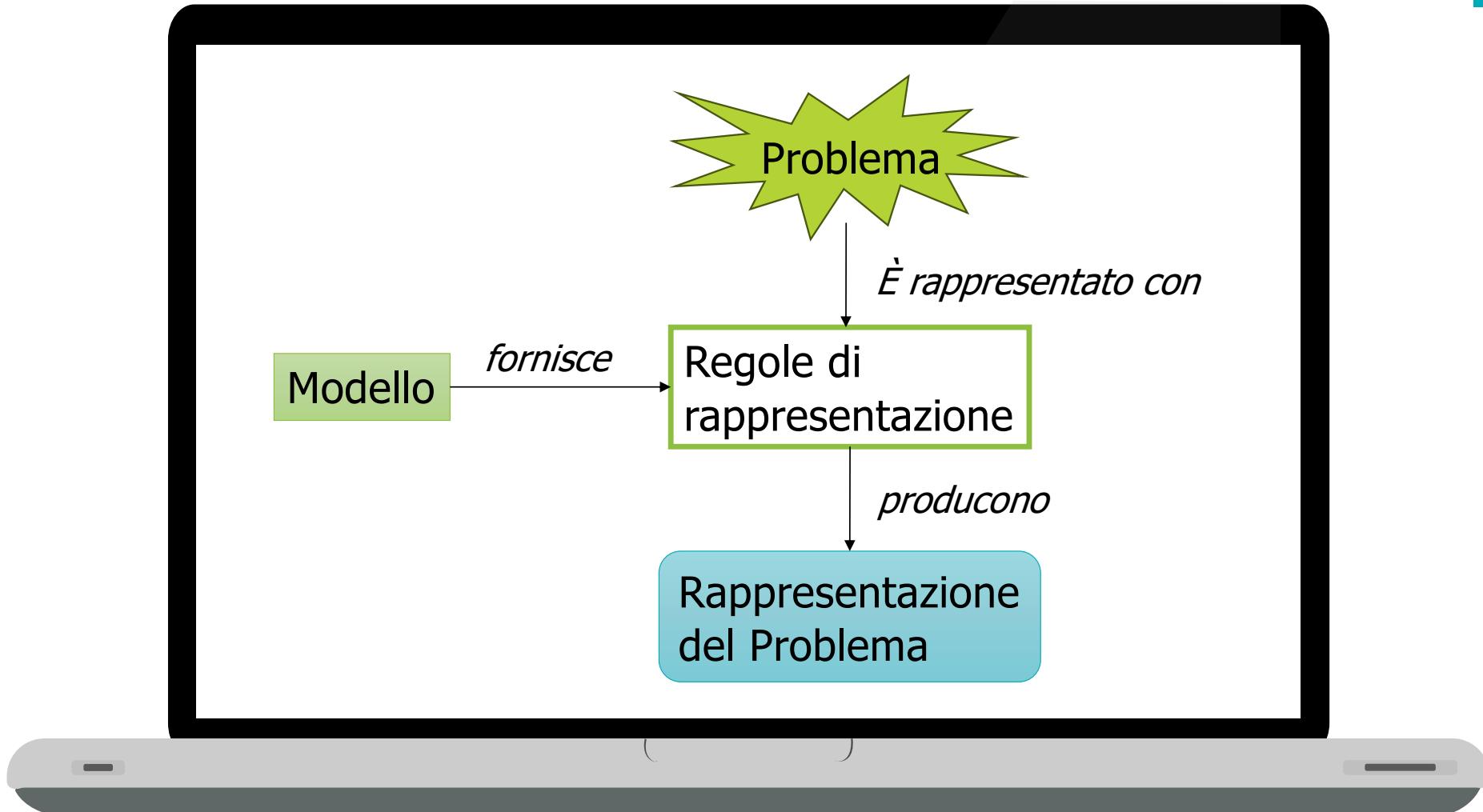
## Modello dei Dati



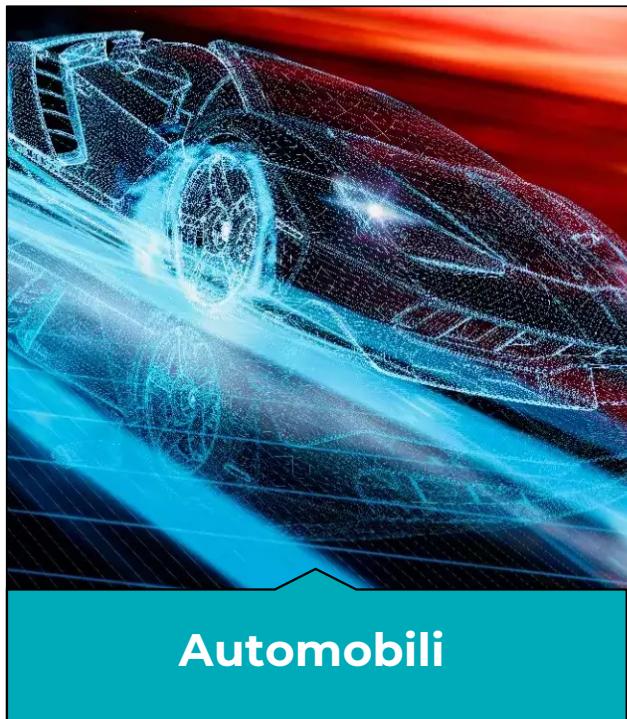
Un modello deve fornire un insieme di costrutti utilizzati per descrivere i dati e le strutture atte a memorizzarli, nascondendo alcuni dettagli di memorizzazione

- Esempio:  
il modello relazionale fornisce il costrutto relazione (tabella) che permette di definire insiemi di record omogenei

ESAME	STUDENTE	CORSO	VOTO
	056/000484	Diritto Privato	24
	056/000484	Procedura Civile	28
	011/120579	Procedura Penale	30
	...	...	...



## Modello dei Dati



- I modelli dei dati permettono di rappresentare un problema utilizzando le sue regole di rappresentazione.
- Consideriamo le automobili nel mondo reale
  - un'auto può essere descritta utilizzando la marca, il modello, l'anno, il colore e il prezzo
  - un'auto può essere descritta utilizzando il suo proprietario, la sua targa e la sua storia di indirizzi registrati

## Il Modello Relazionale

- Proposto da E. F. Codd nel 1970 per favorire l'indipendenza dei dati
- Disponibile in DBMS reali nel 1981 Si basa sul concetto matematico di relazione (con una variante)
- Due accezioni:
  - **Relazione matematica:** sottoinsieme del prodotto cartesiano di insiemi
  - Relazione secondo il **modello relazionale** dei dati

## Relazione Matematica

- $D_1, \dots, D_n$  ( $n$  insiemi anche non distinti)
- **Prodotto cartesiano**  $D_1 \times \dots \times D_n$ :
  - L'insieme di tutte le  $n$ -uple  $(d_1, \dots, d_n)$  tali che  $d_1 \in D_1, \dots, d_n \in D_n$
- **Relazione matematica** su  $D_1, \dots, D_n$ :
  - Un sottoinsieme di  $D_1 \times \dots \times D_n$ .
- $D_1, \dots, D_n$  sono i **domini** della relazione

## Relazione Matematica

- $D_1 = \{a, b\}$
- $D_2 = \{x, y, z\}$
- Prodotto cartesiano  $D_1 \times D_2$

a	x
a	y
a	z
b	x
b	y
b	z

- Una relazione  $r \subseteq D_1 \times D_2$

a	x
a	z
b	y

## Relazione Matematica: Proprietà

- Essendo la relazione matematica un insieme, esso gode delle seguenti **proprietà**:
  - Non c'è ordinamento fra le n-uple  $(d_1, \dots, d_n)$
  - Le n-uple sono distinte
  - Ciascuna n-upla è ordinata: l'i-esimo valore proviene dall' i-esimo dominio

## Tabelle e Relazioni

- Le relazioni hanno naturale rappresentazione per mezzo di tabelle
- Una tabella rappresenta una relazione se
  - I valori di ogni colonna sono fra loro omogenei (dello stesso tipo)
  - Le righe sono diverse fra loro (altrimenti non è un insieme)
  - Le intestazioni delle colonne sono diverse tra loro
- In una tabella che rappresenta una relazione
  - L'ordinamento tra le righe è irrilevante
  - L'ordinamento tra le colonne è irrilevante

## Struttura non posizionale

- A ciascun dominio si associa un nome (attributo), che ne descrive il "ruolo"

Casa	Fuori	RetiCasa	RetiFuori
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	0	2
Roma	Milan	0	1

## Normalizzazione di schemi

- Spesso è auspicabile che le relazioni siano normalizzate.
  - La normalizzazione dei dati può seguire forme normali come la prima forma normale (1NF), la seconda forma normale (2NF), la terza forma normale (3NF), ecc.
- Si consideri la relazione **Libro** mostrata nella Tabella

- Ci sono molti duplicati in questi dati.
- Ad esempio, le righe 1 e 2 sono quasi identiche, tranne che per il formato e il prezzo

Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	\$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	\$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	UK	\$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$15

# Normalizzazione di schemi

- Se le informazioni sull'editore cambiano, ad esempio il nome da "Banana Press" a "Pineapple Press", o se cambia il paese, dovremo aggiornare le righe 1, 2, 3 e 4.

Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	\$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	\$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	UK	\$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$15

# Normalizzazione di schemi

- Se le informazioni sull'editore cambiano, ad esempio il nome da "Banana Press" a "Pineapple Press", o se cambia il paese, dovremo aggiornare le righe 1, 2, 3 e 4.
- Se si separano le informazioni sull'editore in una tabella a sé stante, quando le informazioni di un editore cambiano, dobbiamo aggiornare solo la relazione dell'editore

Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	\$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	\$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	UK	\$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	\$15

Title	Author	Format	Publisher ID	Price
Harry Potter	J.K. Rowling	Paperback	1	\$20
Harry Potter	J.K. Rowling	E-book	1	\$10
Sherlock Holmes	Conan Doyle	Paperback	2	\$30
The Hobbit	J.R.R. Tolkien	Paperback	1	\$30
Sherlock Holmes	Conan Doyle	Paperback	2	\$15

Publisher ID	Publisher	Country
1	Banana Press	UK
2	Guava Press	US

## SQL: Un linguaggio dichiarativo

- Il linguaggio di interrogazione più diffuso per i database relazionali è oggi SQL
- La cosa più importante da notare riguardo a SQL è che si tratta di un linguaggio **dichiarativo**, al contrario di Python, che è un linguaggio **imperativo**.
  - Nel paradigma **imperativo**, si specificano i passi necessari per un'azione e il computer li esegue per restituire i risultati
  - Nel paradigma **dichiarativo** si specificano i risultati che si vogliono ottenere e il computer scopre i passi necessari per ottenere gli output richiesti.

## SQL: Caratteristiche

- Con un database SQL, si specifica lo schema dei dati che si desidera (le tabelle da cui si desidera ottenere i dati), le condizioni che i risultati devono soddisfare, le trasformazioni di base come join, sort, group, aggregate e così via, ma non come recuperare i dati.
- Altre caratteristiche:
  - Potrebbe essere utilizzato per risolvere qualsiasi problema (Turing-complete) aggiungendo qualche altra caratteristica
  - Alcune query potrebbero essere molto lunghe e difficili da interpretare/modificare

# Database NoSQL

6

## Perché non usare sempre il modello relazionale?

- Il modello di dati relazionale è stato in grado di generalizzarsi a molti casi d'uso, dall'e-commerce alla finanza ai social network
- Per alcuni casi d'uso, questo modello può essere restrittivo
  - Richiede che i dati seguano uno schema rigido
  - La gestione degli schemi è onerosa
  - È difficile scrivere ed eseguire query SQL per applicazioni specializzate

## Database NoSQL

**NoSQL** non è uno specifico linguaggio, ma è il termine che raggruppa un insieme di tecnologie per la persistenza dei dati che funzionano in modo sostanzialmente diverso dai database relazionali, quindi non rispettano una o alcune caratteristiche dei RDBMS.



**N.o**nlySQL

<http://blog.sqlauthority.com>

## Database NoSQL

- I database NoSQL possono avere le caratteristiche più disparate:
  - alcuni non utilizzano il modello relazionale,
  - altri usano tabelle e campi ma senza schemi fissi,
  - alcuni non permettono vincoli di integrità referenziale,
  - altri ancora non garantiscono transazioni ACID.
  - oppure ci sono varianti che combinano le precedenti.
  - interpretare/modificare

## SQL vs NoSQL

- Per quanto riguarda le proprietà ACID, alcuni database NoSQL ne garantiscono solo alcune:
  - Non sempre è garantita **la consistenza** (si parla in questo caso di *eventual consistency*), ossia ci può essere una certa latenza prima che una modifica al database sia visibile.
  - Non sempre è garantita **la durabilità**: in alcuni database distribuiti il malfunzionamento di un nodo dopo una transazione potrebbe impedire la corretta sincronizzazione di tutta la rete.

*Queste proprietà vengono rilassate allo scopo di fornire performance migliori e alta disponibilità*

## NoSQL: Parole Chiave

- **Non relazionali:** l'approccio rigido dei DB relazionali non permette di memorizzare dati fortemente dinamici
  - I DB NoSQL sono “schemaless” e consentono di memorizzare “on the fly” attributi, anche senza averli definiti a priori.
- **Distribuiti:** la flessibilità nella clusterizzazione e nella replicazione dei dati permette di distribuire su più nodi lo storage, in modo da realizzare potenti sistemi “fault tolerant”.

## NoSQL: Parole Chiave

- **Scalabili orizzontalmente:** in contrapposizione alla scalabilità verticale, abbiamo architetture enormemente scalabili, che consentono di memorizzare e gestire una grande quantità di informazioni.
- **Open-Source:** filosofia alla base del movimento NoSQL.

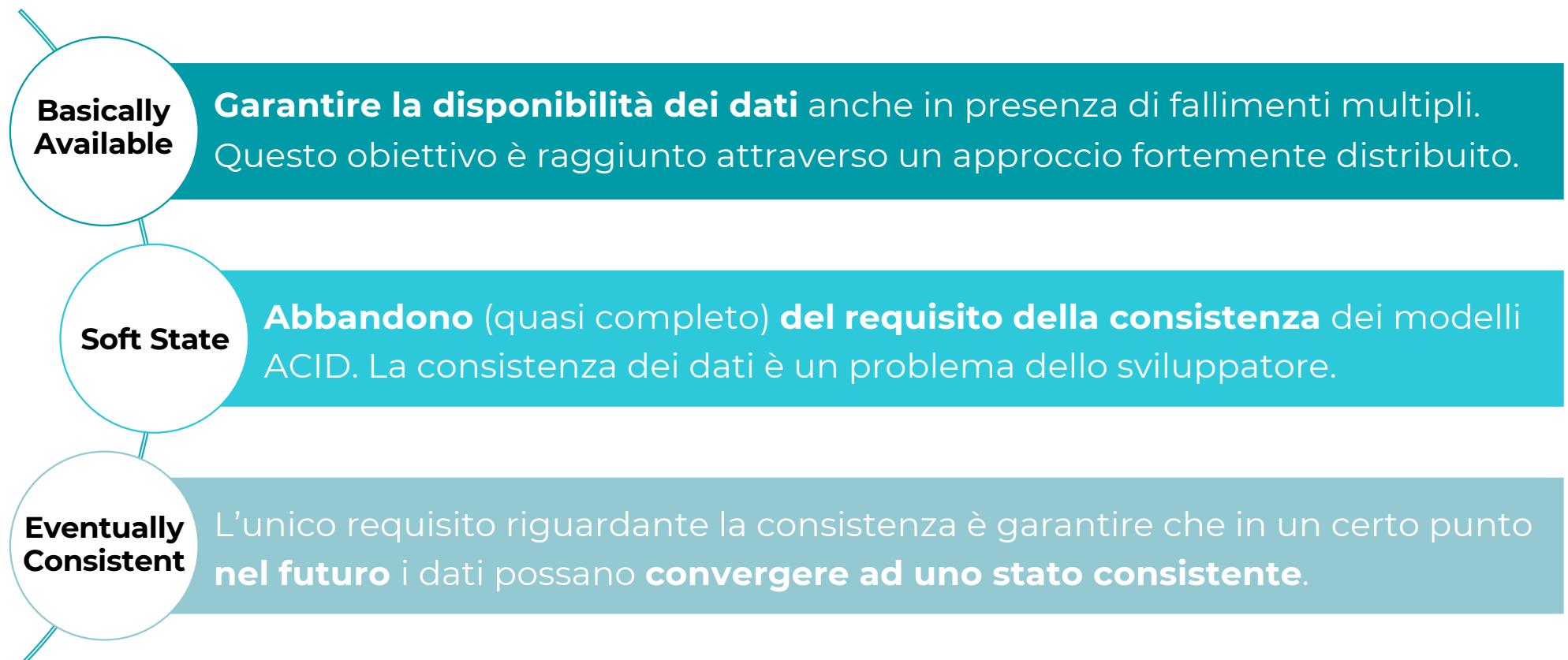


## NoSQL: Caratteristiche

- Grossi **volumi di dati**.
- Goodbye DBA: un sistema RDBMS di alto livello può essere mantenuto solamente con l'assistenza di amministratori altamente esperti (e costosi).
- I Database NoSQL sono generalmente ideati per richiedere una minore manutenzione.
- **Velocità** di risposta alle query!
- Transazioni **BASE** - le proprietà ACID non sono richieste.
- **CAP Theorem**.

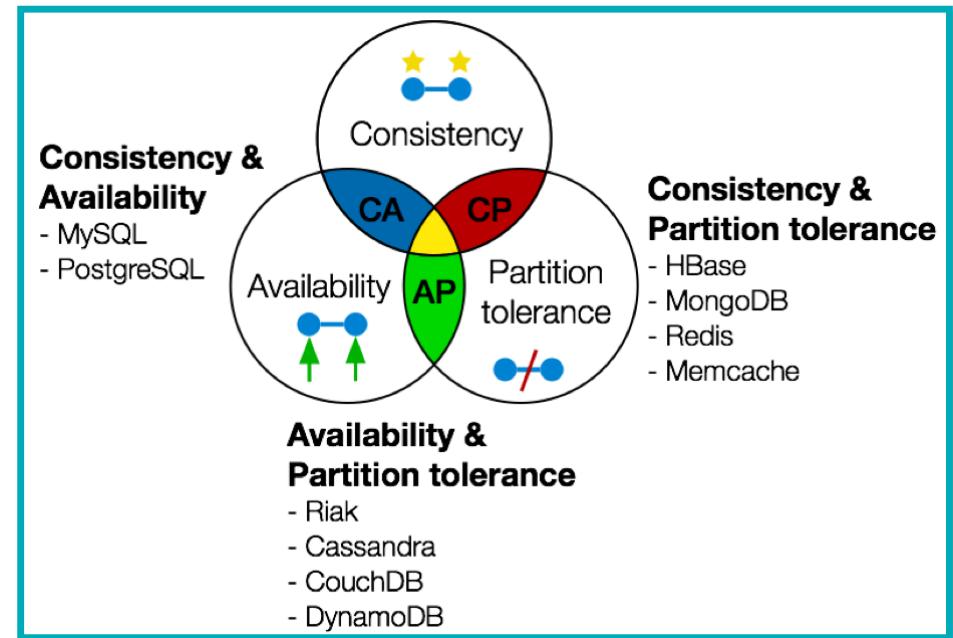
## Transazioni BASE

- I NoSQL si basano su un modello più morbido e flessibile.



# CAP Theorem

- Un sistema distribuito è in grado di supportare solo due tra le seguenti caratteristiche:
  - **Consistency**: tutti i nodi vedono lo stesso dato allo stesso tempo.
  - **Availability**: ogni operazione deve sempre ricevere una risposta.
  - **Partition Tolerance**: capacità di un sistema di essere tollerante ad una aggiunta, una rimozione di un nodo nel sistema distribuito o alla non disponibilità di un componente singolo.



## NoSQL: Challenges

- **Maturità:** i sistemi RDBMS sono in esercizio da tanto tempo.
  - Per i sostenitori del movimento NoSQL questo è un segno della loro obsolescenza; per molti, invece, la maturità di un RDBMS è rassicurante.
- **Supporto:** tutte le aziende che hanno sviluppato e vendono RDBMS forniscono un alto livello di supporto alle aziende.
  - I sistemi NoSQL invece sono spesso progetti open source.
- **Amministrazione:** un obiettivo di design per i DB NoSQL è quello di non necessitare di gran supporto amministrativo ma la realtà è che al giorno d'oggi sono necessarie grosse competenze per la loro installazione e manutenzione.
- **Expertise:** è più semplice trovare un esperto per l'amministrazione degli RDBMS che non un esperto in NoSQL.

## NoSQL: Quando conviene

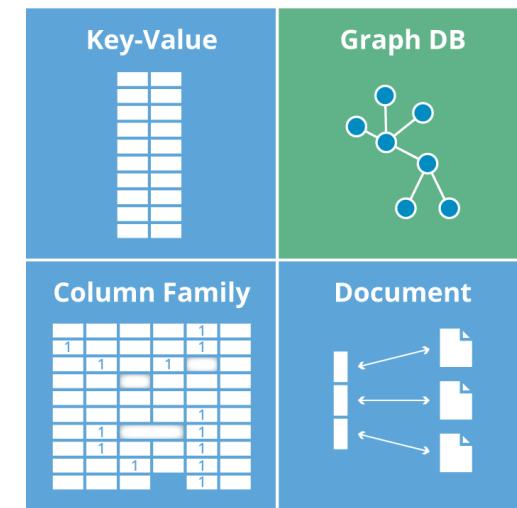
- La **struttura dei dati non è definibile** a priori.
- I dati disposti nei vari oggetti sono molto collegati tra loro e il JOIN rischia di non essere lo strumento ottimale.
  - Sarebbe da prediligere la **navigazione** tra oggetti sfruttando i riferimenti tra i vari nodi di informazione.
- Necessitiamo di **prestazioni più elevate**, potendo considerare troppo stringenti le proprietà ACID per il nostro campo di applicazione.

# Quando non rinunciare al modello relazionale

- In moltissimi casi, il modello relazionale rimane l'opzione migliore.
- Esistono DBMS relazionali consolidati da decenni, supportati da una gran varietà di strumenti.
  - La struttura rigida che richiede non è necessariamente un problema, e talvolta può costituire un vantaggio (soprattutto se si devono modellare dati molto strutturati).
- Nuove tipologie di approccio ai dati offerte dal NoSQL si stanno facendo largo in prodotti relazionali blasonati, offrendo comode vie di fuga e scenari di integrazione nuovi, senza rinunciare alla persistenza in senso tradizionale.
  - Si pensi, ad esempio, ai nuovi tipi di dato (key-value) introdotti in PostgreSQL.

## Tipologie di DB NoSQL

- La parola NoSQL, non indica una ben precisa tipologia di database, bensì è generalmente usata per raggruppare tutti quei DBMS che non possono essere definiti relazionali
- Le categorie più note sono
  - **Document data store**
  - **Key-value data store**
  - **Graph-based data store**
  - **Column-oriented data store**



## Document Data-Store



- Utilizzano dati non strutturati.
- Schema-less.
- Supporto a diversi tipi di documento.
- Un documento è identificato da una chiave primaria.
- Scalabilità orizzontale.

## Document Data-Store

- La rappresentazione dei dati è affidata a strutture simili ad oggetti, dette **documenti**, ognuno dei quali possiede un certo numero di proprietà che rappresentano le informazioni.
  - Per i documenti non è obbligatorio avere una struttura rigida fissa.

*Se i documenti devono rappresentare delle persone, probabilmente avremo in tutti proprietà quali (nome, cognome, data di nascita), ma potremmo decidere di dotare un documento di un numero di telefono, mentre un altro di una email, in base alle informazioni che possediamo*

## Document Data-Store

- Il documento può essere visto come l'equivalente del record delle tabelle.
- I documenti possono essere messi in relazione tra loro con dei riferimenti, pertanto i DB NoSQL sono **particolarmente adatti a rappresentare delle gerarchie.**

## Key-value data store



- Utilizza un associative array (chiave-valore) come modello fondamentale per lo storage.
- Storage, update e ricerca basato sulle chiavi.
- Tipi di dati primitivi familiari ai programmatori.
- Semplice.
- Veloce recupero dei dati.
- Grandi moli di dati.

## Key-value Data-Store

- Costruiti come **dizionari** o **mappe**, in cui vengono inseriti due valori alla volta:
  - Una chiave (identificatore che permette l'estrazione rapida del valore);
  - Il valore ad essa associato (l'informazione vera e propria).
- Offrono la possibilità di effettuare ricerche rapide di singoli blocchi di informazione.

## Key-value Data-Store

- Esempio: **un sistema di messaggistica**:
  - la chiave rappresenta un “account” cui verranno indirizzate comunicazioni,
  - il valore corrispondente è la lista di messaggi ricevuti.
- Tali database puntano tutto sulla ricerca della velocità:
  - sono spesso ottimizzati per conservare i dati in memoria dinamica e salvarli periodicamente su disco in modo da garantire, sia un certo livello di efficienza della risposta, sia la persistenza dei dati.

## Graph-based data store



- Utilizza nodi (entità), proprietà (attributi) e archi (relazioni).
- Modello logico semplice e intuitivo.
- Ogni elemento contiene un puntatore all'elemento adiacente.
- Attraversamento del grafo per trovare i dati.
- Efficiente per la rappresentazione di reti sociali o dati sparsi.
- Relazioni tra i dati centrali.

## Graph-based data store

- Le informazioni sono custodite sia nei nodi e sia negli archi.
- La sua forza è tutto l'insieme del valore informativo che si può estrapolare ricostruendo percorsi attraverso il grafo.
  - Con un database a grafo si può trovare un percorso tra due città lontane – quindi non collegate direttamente – e, sommando la distanza di ogni tratto, si potrebbe calcolare la distanza totale dalla partenza alla destinazione, più ogni altra grandezza derivata (come tempi, costi, eccetera).

*i Social Network - la navigazione del grafo consente di proporre ad un utente nuove potenziali conoscenze recuperando gli "amici di amici"*

# Column-oriented data store



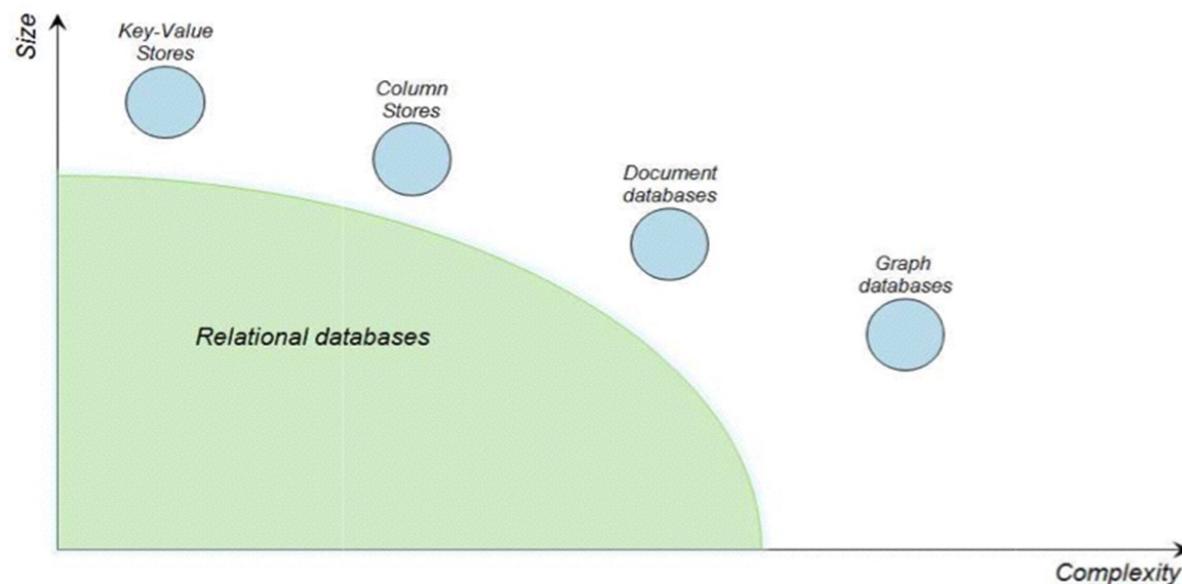
- I dati sono nelle colonne anziché nelle righe.
- Un gruppo di colonne è chiamato famiglia e vi è un'analogia con le tabelle di un database relazionale.
- Le colonne possono essere facilmente distribuite.
- Scalabile.
- Performante.
- Fault-tolerant.

## Column-oriented data store

- **Column-oriented**: sono database che immagazzinano dati in sezioni di colonne.
- La **rapidità di aggregazione** che permettono è stata apprezzata dai grandi produttori di motori di ricerca e Social Network.
- Alcune delle principali soluzioni di questo tipo (Cassandra, Big Table, SimpleDB) sono state realizzate rispettivamente da colossi come Facebook, Google ed Amazon.

## Database NoSQL: Classificazione

- **Operational complexity:** all'aumentare della complessità nella struttura dati diminuisce la capacità di memorizzazione dei dati stessi (parametro size).



# Dataflow

7

# Flussi di dati

*Come passiamo i dati tra processi diversi che non condividono la memoria?*

- Quando i dati vengono passati da un processo all'altro, essi fluiscano da un processo all'altro, il che ci dà un flusso di dati.
- Esistono tre modalità principali di flusso di dati:
  - Passaggio di dati attraverso i database.
  - Passaggio di dati attraverso servizi che utilizzano richieste come quelle fornite da API REST e RPC (ad esempio, richieste POST/GET).
  - Passaggio di dati attraverso un trasferimento in tempo reale.

## Passaggio di dati attraverso i database

- Rappresenta la modalità di passaggio di dati più semplice

*Per passare i dati dal processo A al processo B, il processo A può scrivere i dati in un database e il processo B può semplicemente leggerli da quel database*

- Non funziona sempre per due motivi:
  - Richiede che entrambi i processi siano in grado di accedere allo stesso database.
  - La lettura/scrittura dai database potrebbe essere troppo lenta.

## Passaggio di dati attraverso servizi

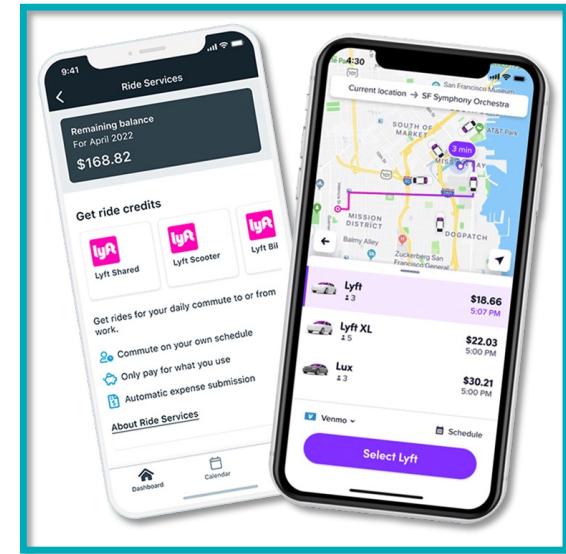
- Un modo per passare i dati tra due processi è quello di inviarli direttamente attraverso una rete che li collega

*Per passare i dati dal processo B al processo A, il processo A invia prima una richiesta al processo B che specifica i dati di cui A ha bisogno, e B restituisce i dati richiesti attraverso la stessa rete.*

- Si dice che questa modalità è guidata dalle richieste
  - È strettamente legata all'architettura orientata ai servizi.

# Passaggio di dati attraverso servizi: Esempio

- immaginate di lavorare sul problema dell'ottimizzazione dei prezzi un'applicazione di ride-sharing come Lyft. Consideriamo 3 servizi tra i centinaia di servizi che offre Lyft:
  - **Gestione degli autisti:** stabilisce quanti autisti saranno disponibili nel prossimo minuto in una determinata area.
  - **Gestione delle corse:** prevede quante corse saranno richieste nel minuto successivo in una determinata area.
  - **Ottimizzazione dei prezzi:** Prevede il prezzo ottimale per ogni corsa.



## Passaggio di dati attraverso servizi: Esempio

*Poiché il prezzo dipende dall'offerta (i conducenti disponibili) e dalla domanda (le corse richieste), il servizio di ottimizzazione dei prezzi ha bisogno di dati provenienti sia dalla gestione dei conducenti sia dalla gestione delle corse.*

- Ogni volta che un utente richiede una corsa, il servizio di ottimizzazione del prezzo richiede il numero previsto di corse e il numero previsto di autisti per prevedere il prezzo ottimale per questa corsa.

## Passaggio di dati attraverso trasferimento in tempo reale

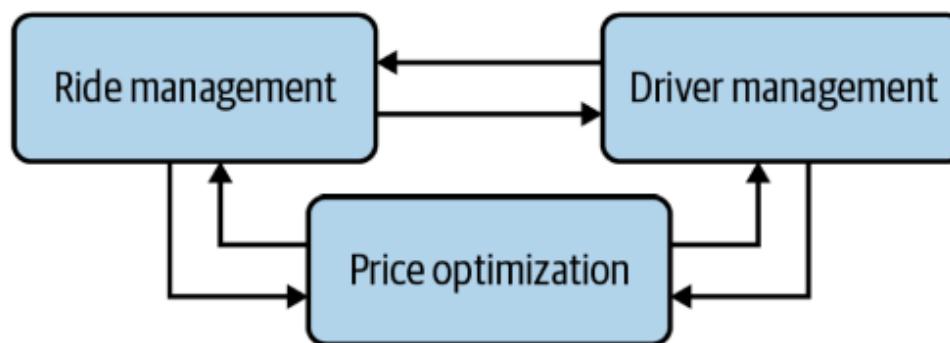
- Riconsideriamo l'esempio precedente con tre semplici servizi: la gestione del conducente, la gestione della corsa e l'ottimizzazione del prezzo

*Immaginiamo che anche il servizio di gestione degli autisti abbia bisogno di conoscere il numero di corse dal servizio di gestione delle corse per sapere quanti autisti mobilitare*

- Immaginiamo che voglia conoscere anche i prezzi previsti per usarli come incentivi per gli autisti:
  - Se ti metti in viaggio ora puoi ottenere un sovrapprezzo di 2 volte.

## Passaggio di dati attraverso trasferimento in tempo reale

- Se i dati passano attraverso i servizi come discusso nella sezione precedente, ognuno di questi servizi deve inviare richieste agli altri due servizi, come mostrato in Figura.
- Con solo tre servizi, il passaggio dei dati diventa già complicato.
- Il passaggio di dati tra i servizi può esplodere e diventare un collo di bottiglia, rallentando l'intero sistema.

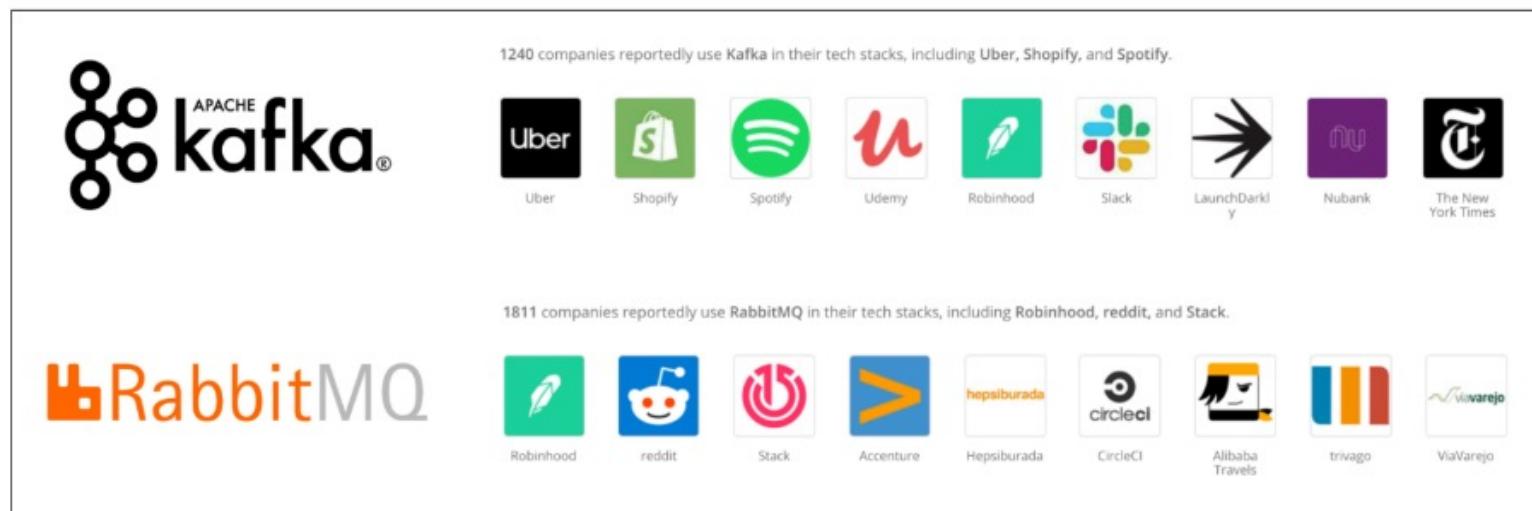


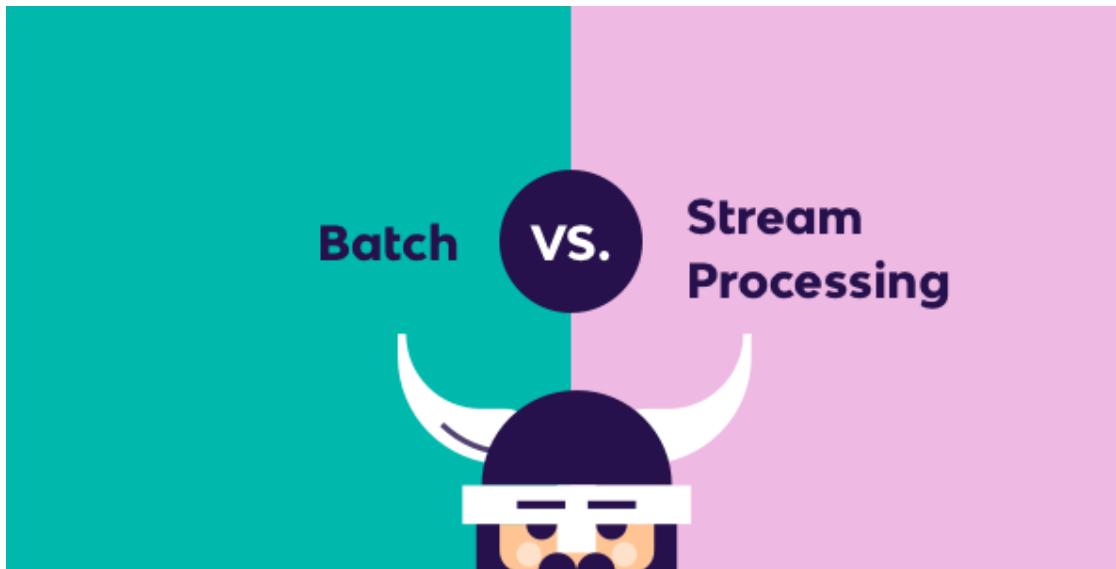
## Passaggio di dati attraverso trasferimento in tempo reale

- Un dato trasmesso a un trasporto in tempo reale è chiamato evento
  - Questa architettura è quindi chiamata anche event-driven.
- I due tipi più comuni di trasmissione in tempo reale sono:
  - **Publish-Subscribe**: qualsiasi servizio può *pubblicare* su diversi argomenti e qualsiasi servizio può *consumare* un argomento in modo da poter leggere tutti gli eventi in quell'argomento.
    - Le soluzioni Pubsub hanno spesso una politica di conservazione dei dati per un certo periodo di tempo.

## Passaggio di dati attraverso trasferimento in tempo reale

- I due tipi più comuni di trasmissione in tempo reale sono:
  - Message Queue:** un evento ha spesso dei *consumatori* previsti (un evento con consumatori previsti è chiamato messaggio) e la coda di messaggi è responsabile di far arrivare il messaggio ai consumatori giusti.





- Una volta che i dati arrivano nei motori di archiviazione dati come database, data lake o data warehouse, diventano **dati storici**.
- Ciò si contrappone ai **dati in streaming**
  - **dati che continuano ad arrivare**

## Dati Storici e Elaborazione Batch

- I dati storici vengono spesso elaborati in batch job, che vengono avviati periodicamente
  - Ad esempio, una volta al giorno, si potrebbe voler avviare un batch job per calcolare il costo medio delle corse dell'ultimo giorno.
- L'**elaborazione batch** è stata oggetto di ricerca per molti decenni e le aziende hanno creato sistemi distribuiti come MapReduce e Spark per elaborare in modo efficiente batch di dati.

## Elaborazione in Streaming

- L'**elaborazione in streaming** si riferisce all'esecuzione di calcoli sui dati in streaming.
  - Può anche essere avviata periodicamente, ma i periodi sono di solito molto più brevi.
  - Può essere avviato ogni volta che se ne presenta la necessità.
    - Ad esempio, ogni volta che un utente richiede un passaggio, si elabora il flusso di dati per vedere quali autisti sono attualmente disponibili

## Batch Processing Vs. Stream Processing

- Poiché l'elaborazione in batch avviene molto meno frequentemente rispetto all'elaborazione in flusso, in ML, questa viene solitamente utilizzata per calcolare caratteristiche che cambiano meno spesso.
  - Ad esempio le valutazioni degli autisti: se *un autista ha fatto centinaia di corse, è meno probabile che la sua valutazione cambi in modo significativo da un giorno all'altro*
- L'elaborazione a flusso viene utilizzata per calcolare le caratteristiche che cambiano rapidamente:
  - Ad esempio, *quanti autisti sono disponibili in questo momento?*

# Batch Vs. Online Learning

- **Batch Learning:** si utilizza quando abbiamo pieno accesso al dataset per il training.
- **Online Learning:** si utilizza quando abbiamo la necessità di far evolvere il modello ogni volta che si processano nuovi dati.

