database NoSQL

Oltre i database relazionali

DBMG

Gestione dei dati al giorno d'oggi

Milioni di database relazionali utilizzati da applicazioni che funzionano molto bene, ma ...

Nuovi trend

Organizzare dati non strutturati o semi-strutturati

Salvare dataset di grandi dimensioni offrendo scalabilità e prestazioni in modo economico

Reinventare i database relazionali

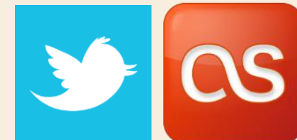
Nuove architetture

Sono emersi database non relazionali

redis

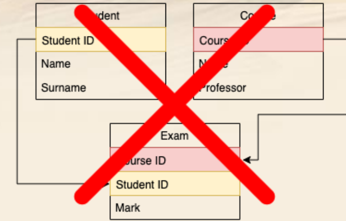
Nascita di “NoSQL”

- Nel **1998** Carlo Strozzi ha creato un database relazionare open-source, che richiedeva poche risorse, e che non utilizzava la classica interfaccia SQL
- Nel **2009** Johan Oskarsson’s (Last.fm) ha organizzato un evento per discutere i vantaggi dei database non-relazionali. È stato coniato un nuovo **hashtag** per promuovere l’evento su Twitter: **#NoSQL**



Principali caratteristiche di NoSQL

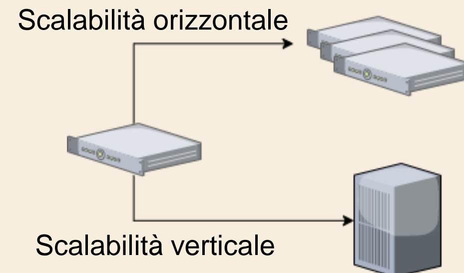
- **Nessuna operazione di join**



- **Senza Schema**
(nessuna tabella, schema implicito)

[illegible]	Name	[illegible]
S123456	[illegible]	Rossi
[illegible]	Paolo	[illegible]

- **Scalabilità orizzontale**



NoSQL databases

Database Distribuiti

DBMG

Replicazione dei Dati

Canada Central
Canada East
US Gov Iowa
Central US
Central US 2
Oregon
N. California
West US
AWS GovCloud
North Central US
South Central US
N. Virginia
East US
East US 2
US Gov Virginia
Ireland
UK West
West Europe
North Europe
Frankfurt
UK South
China North
Beijing
Seoul
Japan East
Tokyo
Japan West
China East
East Asia
West India
Central India
South India
Mumbai
Southeast Asia
Singapore
São Paulo
Brasil South
Australia East
Sydney
Australia Southeast
Microsoft Azure
amazon web services
Atomia

Stessi dati
ma in luoghi **diversi**
(stesso contenuto e
stesso schema)

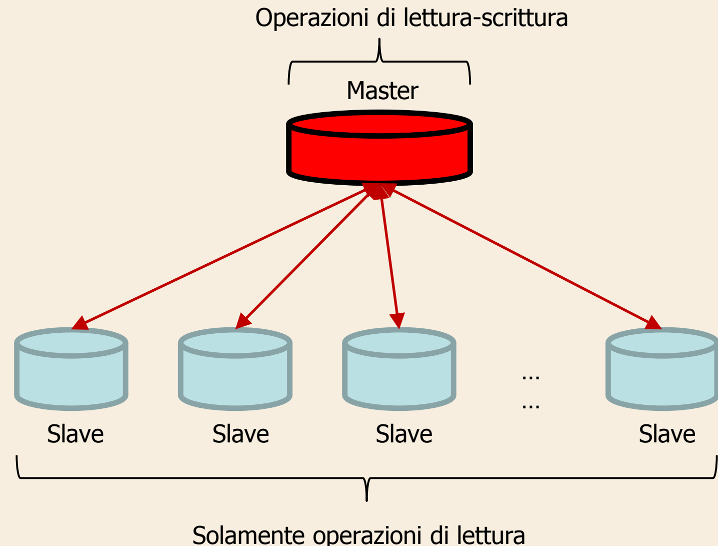
DBMG

Replicazione dei Dati

- **Stessi** dati
  - Porzioni dei dati oppure interi dataset (**chunks**)
- in **luoghi diversi**
  - Server nello stesso luogo e/o lontani fra loro, cluster, data center
- Obiettivi
  - Sopravvivenza ai guasti (availability) grazie alla ridondanza
  - Miglioramento delle performance
- Possibilità
  - Replicazione **Master-Slave**
  - Replicazione **Sincrone**
  - Replicazione **Asincrone**

Replicazione Master-Slave

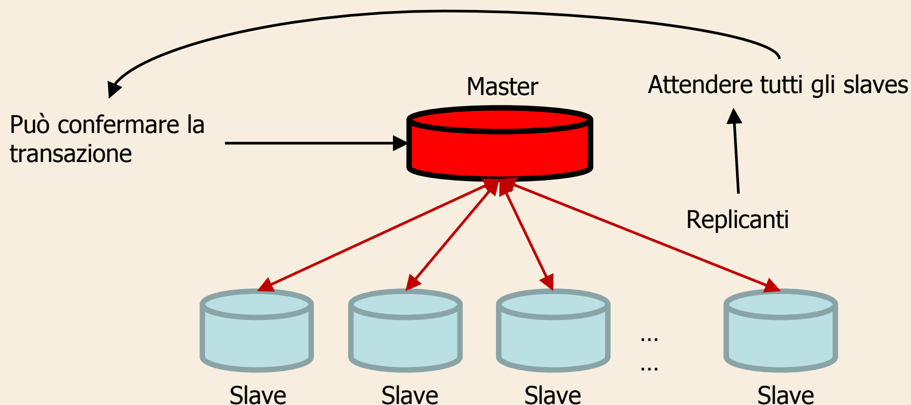
- Master-Slave
  - Un server **master** riceve tutte le richieste di scrittura, aggiornamento, inserimento
  - Uno (o più) server **slave** effettuano le operazioni di lettura (**non** possono scrivere)
  - **Scalabilità** solamente in lettura
  - Il master è il punto debole: un guasto al master impedisce il funzionamento



DBMG

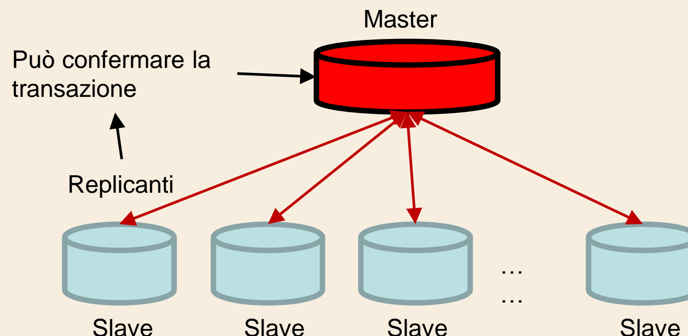
Replicazione Sincrona

- Prima di confermare una transazione, il master **aspetta** che tutti gli slave confermino la transazione tramite il *commit*.
- Abbattimento delle **Performance**, specialmente per repliche che coinvolgono il cloud (server remoti).
- Compromesso: aspettare un sottoinsieme di slave prima di confermare la transazione, per esempio attendendo la **maggioranza**.



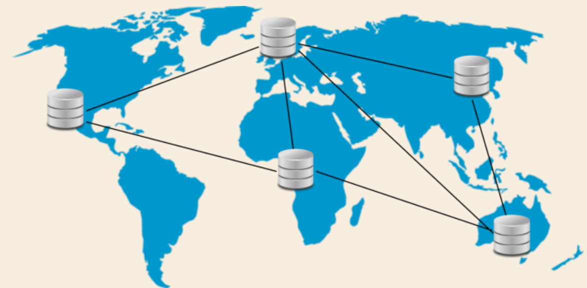
Replicazione Asincrona

- Il master conferma la transazione **localmente**, senza attendere gli Slave
- Ogni slave è indipendente, e richiede gli aggiornamenti direttamente dal master, il quale **potrebbe smettere di funzionare**
  - Se nessuno slave ha replicato la transazione, allora i dati del **master** interessati dalla transazione **sono persi**
  - Se alcuni slave hanno replicato la transazione, ed altri slave no, è necessaria una politica per concordare quali sono i dati attendibili
- Veloce ma inaffidabile



Database Distribuiti

Diversi server autonomi, che **cooperano** per gestire lo stesso **dataset**

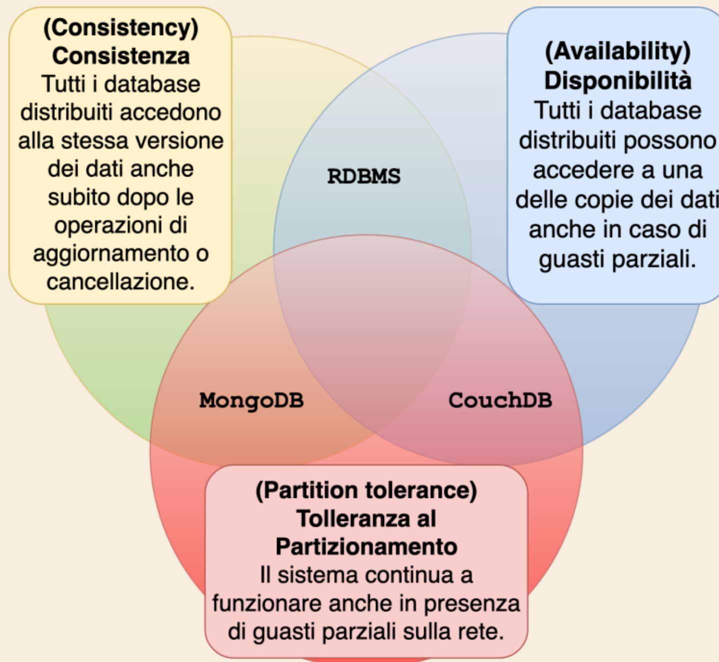


Funzionalità chiave dei database distribuiti

- Ci sono 3 problemi tipici nei database distribuiti:
  - **Consistency - Consistenza**
    - Tutti i database distribuiti forniscono gli stessi dati alle applicazioni.
  - **Availability - Disponibilità**
    - Uno (o più) guasti al database (incluso il master) non impediscono ai server rimasti di continuare a funzionare.
  - **Partition tolerance - Tolleranza al Partizionamento**
    - Il sistema continua a funzionare nonostante la perdite di messaggi e quando problemi di connettività causano un partizionamento della rete.

Teorema CAP

Il teroema CAP, anche conosciuto come teorema di Brewer, afferma che, per un Sistema distribuito è **impossibile** offrire **simultaneamente tutte e tre** le funzionalità precedentemente descritte.

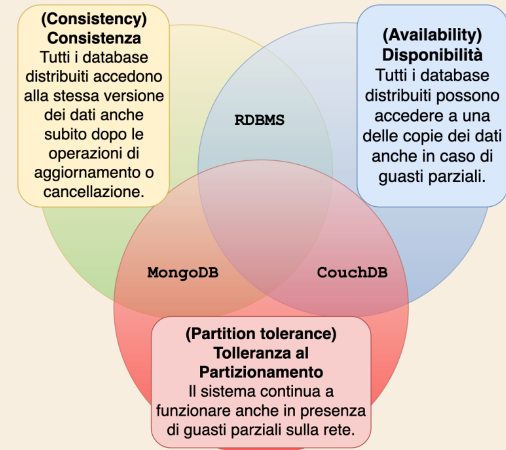


Teorema CAP

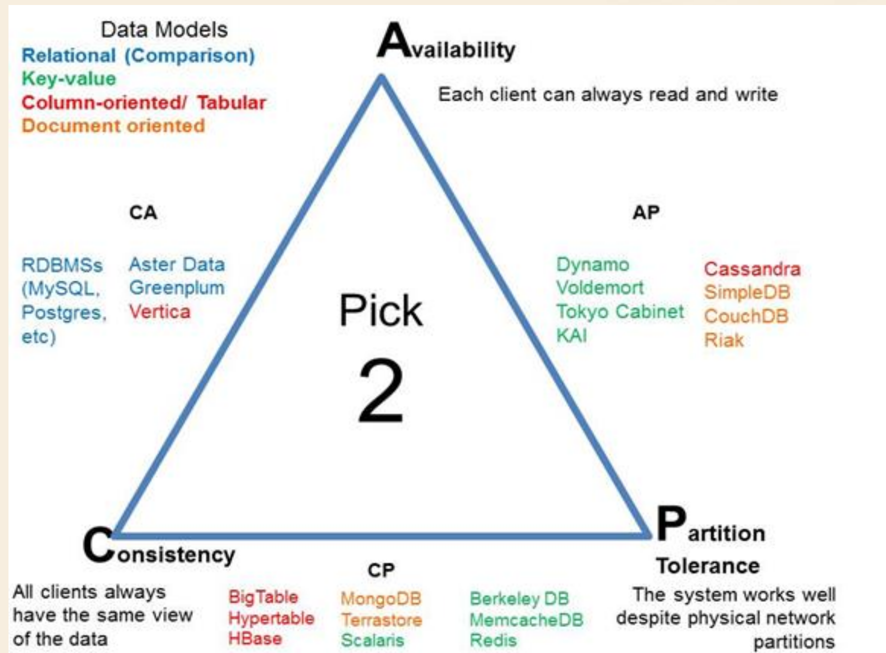
- Il teorema nasce come una **congettura** suggerita dall'Università della California nel 1999-2000
  - Armando Fox and Eric Brewer, "Harvest, Yield and Scalable Tolerant Systems", Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99), IEEE CS, 1999, pg. 174-178.
- Nel 2002 è stato formalmente dimostrato, creando il **teorema**
  - Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51-59
- Nel 2012, un nuovo studio da Eric Brewer, "CAP twelve years later: How the "rules" have changed"
  - IEEE Explore, Volume 45, Issue 2 (2012), pg. 23-29.

Teorema CAP

- Il modo più semplice per capire il CAP è pensare a **due nodi** sui lati opposti di una **partizione**.
- Se si consente ad almeno un nodo di aggiornare lo stato, i nodi diventeranno **incoerenti**, perdendo quindi la *C*.
- Se la scelta è di preservare la coerenza, un lato della partizione deve agire come se **non fosse disponibile**, perdendo così *A*.
- Solo quando non esiste alcuna **partizione** di rete, è possibile preservare sia la coerenza (C) che la disponibilità (A), perdendo così P.
- La convinzione generale è che per i sistemi altamente distribuiti, i progettisti **non possono rinunciare a P** e quindi devono scegliere se perdere la C o la A.
  - La scelta è spesso complicata



Teorema CAP



http://blog.flux7.com/blogs/nosql/cap-theorem-why-does-it-matter

CA senza P (consistente localmente)

- **Il partizionamento** (interruzione della comunicazione) causa un guasto.
- Possiamo ancora avere **Coerenza** e **Disponibilità** dei dati condivisi **all'interno di ciascuna partizione**, ignorando le altre partizioni.
  - Coerenza / disponibilità locale anziché globale
- Coerenza locale per una parte dei sistema: disponibilità del 100% dei dati per questa parte di sistema. Non avere partizioni non esclude il partizionamento dove le diverse partizioni hanno la propria CA "locale".
- Partizionare significa avere **più sistemi indipendenti** con CA al 100% ognuna e che non hanno bisogno di interagire.

DBMG

CP senza A (blocco delle transazioni)

- Un Sistema è autorizzato a *non* rispondere alle richieste (eliminando la "A").
- Si tollera il **partizionamento/guasti**, perché se avviene un partizionamento, semplicemente vengono bloccate tutte le risposte, assumendo quindi che il sistema non possa continuare a funzionare correttamente senza i dati dall'altro lato della partizione.
- Una volta che il partizionamento è risolto e la **consistenza** può essere nuovamente verificata, è possibile riabilitare la disponibilità ("A") uscendo da questa modalità di blocco.
- In questo tipo di configurazioni è richiesta la consistenza **globale**, per questo, il comportamento prevede che **l'accesso ai server replicanti** non sincronizzati **sia bloccato**.
- Per tollerare la P in ogni momento, dobbiamo sacrificare la A in qualsiasi momento per avere globalmente la consistenza.

AP senza C (sforzo minimo)

- Se non interessa la **coerenza globale** (simultanea), ogni parte del sistema può rendere disponibile ciò che sa.
- Ogni parte potrebbe essere in grado di rispondere, anche se il sistema nel suo insieme è stato suddiviso in regioni che non possono comunicare (**partizioni**).
- In questa configurazione penalizziamo la coerenza che non può essere garantita come globale in **nessun momento**.

Una conseguenza di CAP

“Ogni nodo in un sistema dovrebbe essere in grado di prendere decisioni esclusivamente in base allo **stato locale**. Se è necessario fare operazioni con un carico elevato sul server, e c’è la possibilità di **errori**, se è richiesto un accordo tra i server replicanti, il sistema non può funzionare. Se si è preoccupati per la **scalabilità**, qualsiasi algoritmo che costringe ad avere un accordo tra i server replicanti diventerà il collo di bottiglia. Questo è un fatto.”

Werner Vogels, Amazon CTO and Vice President

Oltre CAP

- La scelta di "2 su 3" è ingannevole per diversi motivi.
- Innanzitutto, poiché le **partizioni** sono rare, ci sono poche ragioni per scegliere C o A quando il sistema non è partizionato.
- In secondo luogo, la **scelta tra C e A** può avvenire più volte all'interno dello stesso sistema con granularità diverse (anche fini); non solo i sottosistemi possono fare scelte diverse, ma la scelta può cambiare in base all'operazione, o anche ai dati specifici, o all'utente interessato.
- Infine, tutte e tre le proprietà sono più **una scelta continua che binaria**. La disponibilità è ovviamente continua dallo 0 al 100 percento, ma ci sono anche molti livelli di coerenza e persino le partizioni presentano sfumature, incluso il disaccordo all'interno del sistema sull'esistenza di una partizione.

ACID versus BASE

- ACID e BASE rappresentano due filosofie progettuali opposte nello spettro di coerenza-disponibilità
- Le proprietà ACID si concentrano sulla **coerenza** e rappresentano l'approccio tradizionale dei database
- **BASE**: **B**asically **A**vailable, **S**oft state, **E**ventually consistent, consente di lavorare in presenza di **partizioni** e quindi promuove la **disponibilità**

ACID

Le quattro proprietà ACID sono:

- **Atomicity (A) – Atomicità**: Tutti i sistemi beneficiano delle operazioni atomiche, la transazione del database devono avere esito positivo o negativo, non è consentito il successo parziale.

- **Consistency (C) – Consistenza:** Durante la transazione del database, il database passa da uno stato valido a un altro valido. In ACID, la C indica che una transazione preserva i vincoli di integrità del database, per esempio le chiavi univoche. Al contrario, la C in CAP si riferisce solo alla **coerenza della copia singola**.

- **Isolation (I) – Isolamento:** L'isolamento è al centro del teorema CAP: se il sistema richiede l'isolamento ACID, può operare al massimo da una parte durante una partizione, perché la transazione di un cliente deve essere isolata dalla transazione di un altro cliente.

- **Durability (D) – Persistenza (o Durabilità):** I risultati dell'applicazione di una transazione sono permanenti, devono persistere dopo il completamento della transazione, anche in presenza di errori.

DBMG

BASE

- **Basically Available – Fondamentalmente disponibile**: il sistema fornisce la disponibilità, come enunciato nel teorema CAP

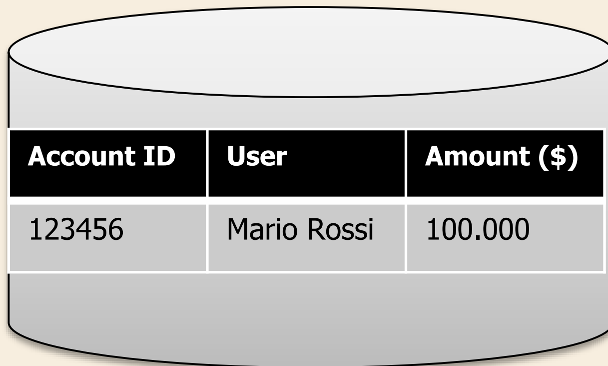
- **Soft state – Stato debole:** indica che lo stato del sistema può cambiare nel tempo, anche senza input, a causa dell'eventuale modello di coerenza.

- **Eventual consistency – Eventualmente consistente:** indica che il sistema diventerà coerente nel tempo, dato che il sistema non riceve input durante tale periodo.

Problema della risoluzione di conflitti

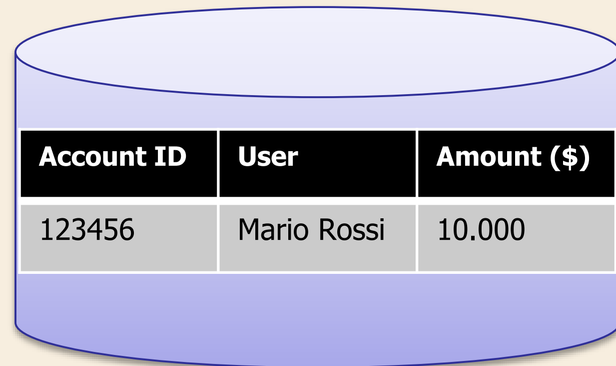
- Conflitto: quando repliche diverse, allo stesso tempo *t* hanno dati non allineati.

Replica 1



Account ID	User	Amount ($)
123456	Mario Rossi	100.000

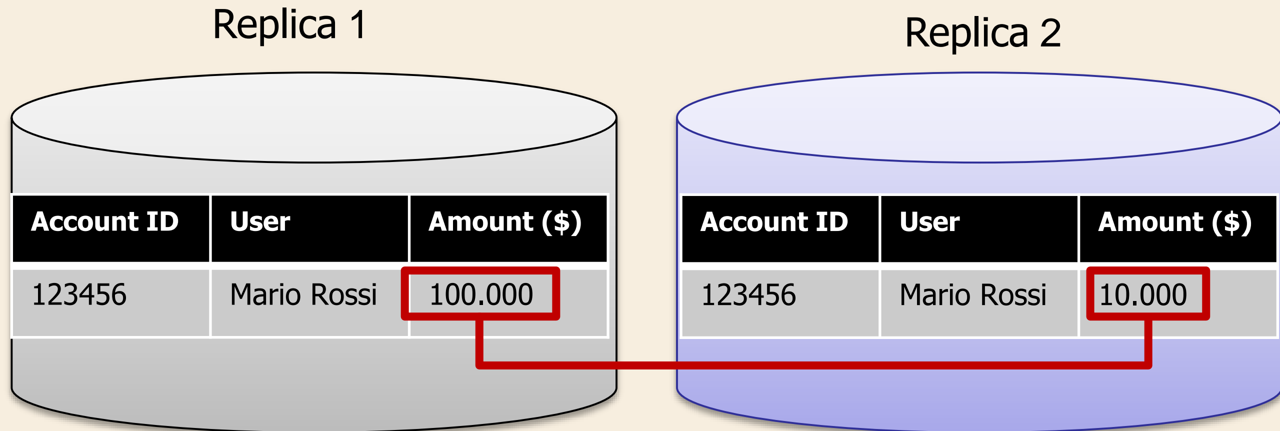
Replica 2



Account ID	User	Amount ($)
123456	Mario Rossi	10.000

Problema della risoluzione di conflitti

- Conflitto: quando repliche diverse, allo stesso tempo *t* hanno dati non allineati.

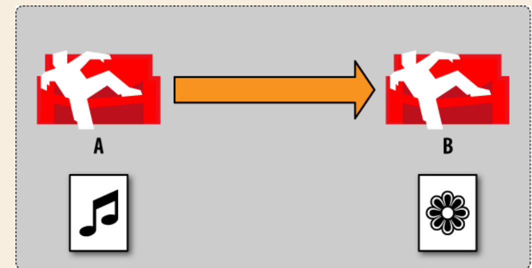


Problema della risoluzione di conflitti

- **Apache CouchDB** è un database *NoSQL* open-source basato sulla nozione di *documenti*
- Una delle caratteristiche distintive di CouchDB è la replica multi-master, che consente di scalare tra macchine per creare sistemi ad alte prestazioni.
- Come risolve i conflitti?

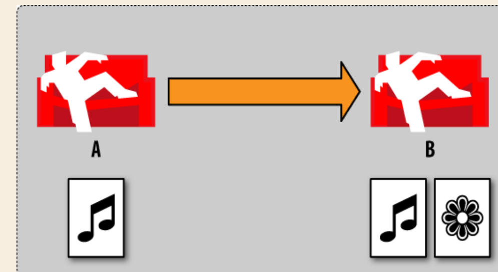
Problema della risoluzione di conflitti

- Dati due clienti, **A** e **B**
- **A** prenota una camera di albergo, l'ultima disponibile
- **B** fa lo stesso, ma in un nodo diverso del sistema, che **non è coerente**



Problema della risoluzione di conflitti

- Il documento relativo alla camera di hotel è ora affetto da un errore dovuto a due **aggiornamenti in conflitto**
- Le applicazioni dovrebbero risolvere il conflitto seguendo regole personalizzate (è una decisione aziendale)
- Il database può:
  - **Individuare** il conflitto
  - Fornire una **soluzione** locale, tipo: l'ultima versione è stata salvata ed è la versione "vincitrice"



Conflitti

- CouchDB garantisce che lo **stesso conflitto**, finirà sempre con la **stessa revisione "vincente"**, e la stessa revisione "perdente".

- Questo è possible perchè CouchD esegue un **algoritmo deterministico** per individuare la revisione "vincente".
  - La revisione con la cronologia più lunga diventa la revisione vincente.
  - Se due revisioni hanno la stessa lunghezza, i valori di _rev vengono confrontati come codici ASCII, il valore maggiore vince.

DBMG

database NoSQL

Oltre i database relazionali

DBMG

Confronto

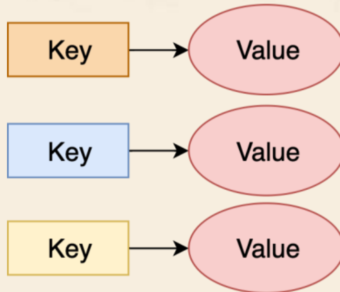
Database relazionale	Database non-relazionale
Basato su tabella, ogni record è una riga strutturata.	Soluzioni di archiviazione specializzate, ad esempio coppie di valori-chiave basate su documenti, database di grafi, archiviazione colonnare.
Schema predefinito per ogni tabella, modifiche consentite ma generalmente bloccanti (costose in ambienti distribuiti e live).	Senza schema: lo schema può cambiare dinamicamente per ogni documento, adatto per dati semi-strutturati o non strutturati.
Scalabile verticalmente, cioè tipicamente ridimensionato aumentando la potenza dell'hardware.	I database NoSQL sono scalabili orizzontalmente: vengono ridimensionati aumentando i server di database nel pool di risorse per ridurre il carico.
Utilizzo di SQL (Structured Query Language) per definire e manipolare i dati, un linguaggio molto versatile.	Linguaggi di query personalizzati, incentrati sul concetto di documenti, grafici e altre strutture di dati specializzate.

Confronto

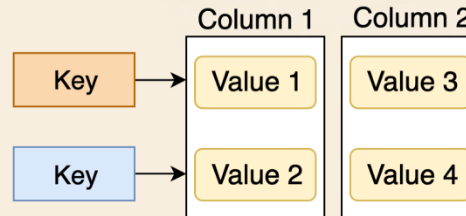
Database relazionale	Database non-relazionale
Adatto a query complesse, basato su join di dati.	Nessuna interfaccia standard per eseguire query complesse, nessun join.
Adatto per l'archiviazione di dati "piatta" e strutturata.	Adatto a dati complessi (ad esempio gerarchici), simili a JSON e XML.
Per esempio: MySql, Oracle, Sqlite, Postgres e Microsoft SQL Server.	**Per esempio:** MongoDB, BigTable, Redis, Cassandra, Hbase e CouchDB.

Tipi di database NoSQL

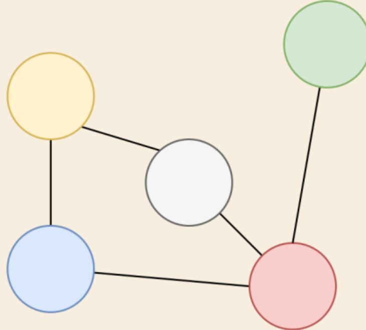
Chiave Valore



Orientati per Colonna



Database sui grafi

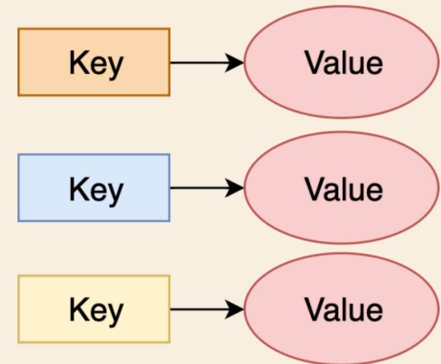


Basati sui documenti

```
{
first_name: "Mario",
last_name : "Rossi",
SSN: "AAAA00000000",
city: "Torino",
job: "Engineer",
cars: [
  {
    model: "Model S",
    year: "2018"
  },
  {
    model: "Model X",
    year: "2016"
  }
],
}
```

Database Chiave-valore

- Archivi di dati NoSQL **più semplici**
- Abbina le chiavi ai valori
- Nessuna struttura
- Ottime **performance**
- Scalabile facilmente
- Molto veloce
- Esempi: **DynamoDB**, Redis, Riak, Memcached



Database Chiave-valore – DynamoDB

Modello di dati

Coppie (chiave, valore):

- Chiave = id univoco
- Valore = un oggetto piccolo (< 1 Mbyte)

Transazione più semplice

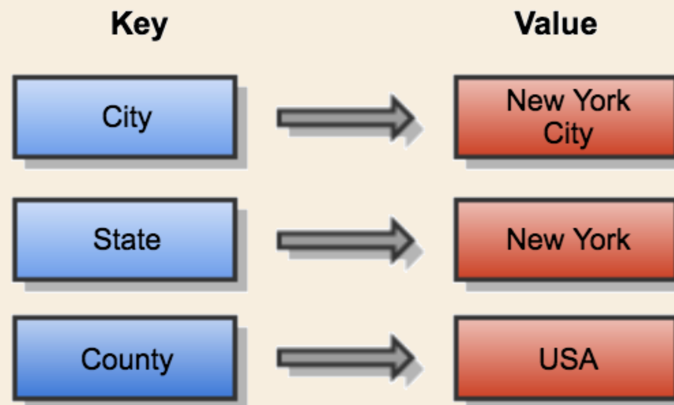
- Put(chiave, valore)
- Get(chiave)

- Replica ed eventualmente coerenza
- Record distribuiti tra i nodi in base alla chiave
- Presuppone che l'ambiente sia protetto (cloud)

Database Chiave-valore: caso d’uso

Caso d’uso reale:

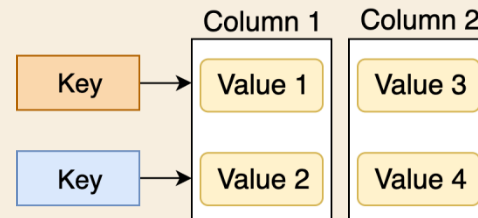
- Schema semplice
- Letture/scritture rapide con aggiornamenti poco frequenti
- Alte prestazioni e scalabilità



DBMG

Database orientati per Colonna

- Salvare i dati in un formato **colonnare**
  - Name = "*Mario*":row1,row3; "*Paolo*":row2,row4; …
  - Cognome = "*Rossi*":row1,row5; "*Bianchi*":row2,row6,row7…
- Una colonna è un **attributo** (anche complesso)
- Coppie chiave-valore archiviate e restituite sfruttando un sistema parallelo (simile agli **indici**)
- **Le righe** possono essere costruite da valori in colonna
- Le colonne salvate possono produrre in ouput dati semplici (**tabelle**)
- Esempi: **Cassandra**, Hbase, Hypertable



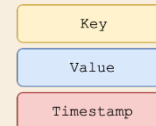
Database orientati per Colonna – Cassandra

Principali Caratteristiche

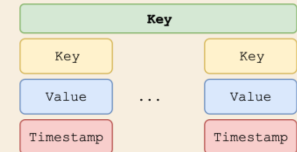
- ✓ **Disponibilità continua:** Nessun singolo punto di guasto.
- ✓ **Multi-data center:** I dati sono replicati in diversi data center.
- ✓ **Scalabilità orizzontale:** è possibile aggiungere più hardware per fornire più memoria e prestazioni. Scalabilità lineare.

Modello dei dati

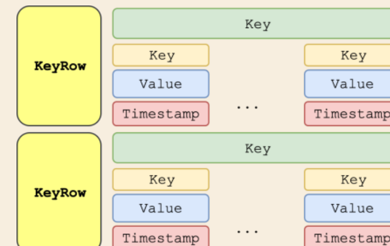
Colonna: tupla (chiave, valore, timestamp)



SuperColonna: un nome (chiave) e una mappa ordinata delle colonne



Famiglia di colonne: Analoghe ad una tabella in un database relazionale. È un contenitore per raggruppare insiemi di colonne.



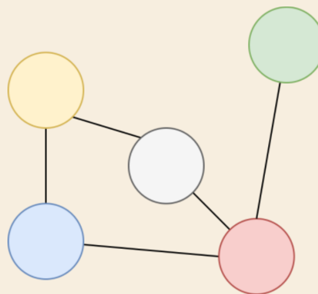
Database orientati per Colonna: caso d’uso

Caso d’uso reale:

- Applicazioni distribuite geograficamente su più data center
- Applicazioni che possono tollerare alcune incoerenze a breve termine nelle repliche
- Applicazioni con campi dinamici
- Applicazioni che necessitano potenzialmente di volumi di dati veramente grandi, come centinaia di terabyte

Database sui grafi

- Basati sulla teoria dei grafi
- Creati con **Vertici** ed **Archi** orientati e non orientati tra coppie di vertici
- Utilizzati per salvare informazioni relativi alle **reti**
- Adatto per diverse applicazioni del mondo reale
- Esempi: Neo4J, Infinite Graph, OrientDB



DBMG

Database sui grafi – Neo4J

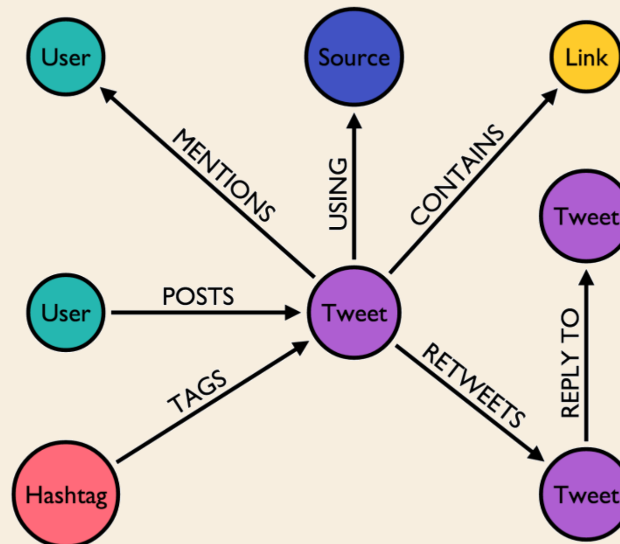
Modello dei dati: Nodi e archi.
Proprietà: coppie (chiave, valore)

- Transazioni ACID complete.
- Altamente scalabile, fino a diversi miliardi di nodi/relazioni/proprietà.
- Espressivo, con un linguaggio di query grafico e leggibile (**Cypher**).
- Plugin esistenti per altri linguaggi di programmazione.

Database sui grafi: caso d’uso

Caso d’uso reale:

- I database basati sulla teoria dei grafi sono adatti per casi d'uso in cui il dominio può essere modellato naturalmente come un un grafo. (Per esempio le relazioni tra le persone nei social network).



Database basati sui Documenti

- Memorizza e recupera documenti
- I documenti sono coppie auto descrittive (**attributo = valore**)
  - Come `city: "Torino"`
- Le chiavi sono mappate nei documenti
- I documenti hanno una natura **eterogenea**
- Tra le soluzioni più utilizzate
- Esempi: **MongoDB**, CouchDB, RavenDB

```
{
first_name: "Mario",
last_name : "Rossi",
SSN: "AAAA00000000",
city: "Torino",
job: "Engineer",
cars: [
  {
    model: "Model S",
    year: "2018"
  },
  {
    model: "Model X",
    year: "2016"
  }
],
}
```

DBMG

Concetti utili: gli oggetti JSON

- JSON è un linguaggio indipendente per salvare e scambiare dati

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }  ],
}
```

DBMG

Concetti utili: gli oggetti JSON

- JSON è un linguaggio indipendente per salvare e scambiare dati

Chiave — Valore

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }  ],
}
```

Concetti utili: gli oggetti JSON

- JSON è un linguaggio indipendente per salvare e scambiare dati

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }  ],
}
```

Chiave: "firstName" — Valore: "John"

Chiave: "address" — Valore composto: { "streetAddress": ..., "postalCode": "10021-3100" }

Concetti utili: gli oggetti JSON

- JSON è un linguaggio indipendente per salvare e scambiare dati

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }  ],
}
```

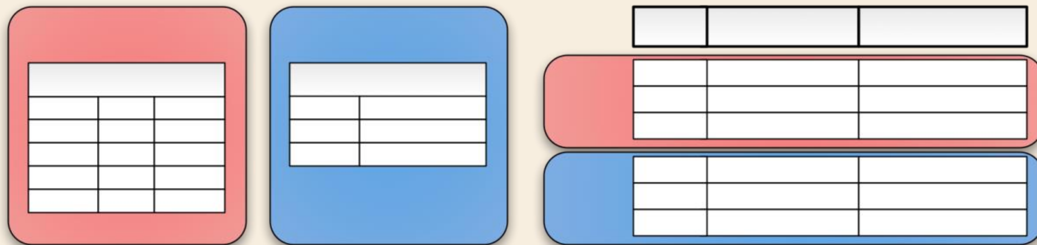
Chiave: "firstName" — Valore: "John"

Chiave: "address" — Valore composto

Chiave: "phoneNumbers" — Vettore di valori

Concetti utili: Sharding

- Uno **shard** è una partizione dei dati del database.
- Ogni shard è salvato sul server in un **istanza separata del database**, per bilanciare il carico di lavoro.



Verticale

Orizzontale

Database basati sui Documenti - MongoDB

Sistema NoSQL basato sui documenti

- Master-Slave / Server replicanti & Autosharding.
- Sistemi automatici per bilanciare il carico di lavoro grazie ai dati divisi in shard.
- Utilizza documenti codificati in formato JSON.
- Diversi tipi di indici come Indici B-tree, geospaziali.
- Memorizza documenti di qualsiasi dimensione senza complicare le applicazioni che lo utilizzano.

Database basati sui Documenti: caso d’uso

Caso d’uso reale:

- Applicazioni che richiedono la possibilità di archiviare attributi di tipo diverso e grandi quantità di dati.
- Applicazioni che richiedo un intensivo utilizzo dei dati con bassa latenza.

Aziende leader che utilizzano MongoDB