DBMS NoSQL (Not Only SQL)

FITSTIC 2022

whoami

Joseph Giovanelli

- Email: j.giovanelli@unibo.it
- Ph.D. candidate in Computer Science and Engineering @ UniBO

Research topics

- Big Data Analytics
- Automated Machine Learning

https://big.csr.unibo.it/

Cosa significa NoSQL

Il termine viene usato per la prima volta nel '98 da Carlo Strozzi

- Indicava un RDBMS open-source che usava un linguaggio diverso da SQL per le interrogazioni

Nel 2009 viene usato da un meetup di San Francisco

- Ospitavano discussioni di progetti open-source ispirati ai nuovi database di Google e Amazon
- Gruppi partecipanti: Voldemort, Cassandra, Dynamite, HBase, Hypertable, CouchDB, MongoDB

Oggi, il termine NoSQL indica dei DBMS (DataBase Management System) in cui il meccanismo di persistenza è diverso dal modello relazionale (usato dagli RDBMS)

- NoSQL = Not Only SQL
- Secondo Strozzi, il termine NoREL sarebbe stato più consono

I punti forti degli RDBMS

Meccanismo delle transazioni

- Garanzia nella gestione della consistenza e degli accessi concorrenti

Integrazione

- Applicazioni diverse possono condividere e riutilizzare le stesse informazioni

Standard

- Il modello relazionale ed il linguaggio SQL sono standard affermati
- Un unico background teorico condiviso da diverse tecnologie

Solidità

- In uso da oltre 40 anni

I punti deboli degli RDBMS

Conflitto di impedenza

- La memorizzazione del dato si basa sul modello relazionale, ma la manipolazione del dato si basa tipicamente sul modello a oggetti
- Tante soluzioni proposte, nessuno standard
  - E.g.: Object Oriented DBMS (OODBMS), Object-Relational DBMS (ORDBMS), Object-Relational Mapping (ORM) frameworks

Difficile scalabilità orizzontale

- I Big Data sono una realtà; un unico server non può gestire tutto
- Distribuire un RDBMS non è una soluzione facile

Consistenza vs efficienza

- Garantire la consistenza dei dati è un must – anche a costo delle performance
- Le applicazioni odierne richiedono letture e scritture con grande frequenza e a bassa latenza

Rigidità dello schema

- Una modifica "a regime" può essere molto costosa

Caratteristiche comuni ai NoSQL

Non solo righe e tabelle

- Varietà di modelli per la gestione e la manipolazione dei dati

Libertà dai join

- Possibilità (in alcuni casi, necessità) di estrarre i dati senza eseguire dei join

Libertà dagli schemi

- Possibilità di caricare e interrogare i dati senza definire degli schemi prefissati (*schemaless* o *soft-schema*)

Architettura distribuita e *shared-nothing*

- Semplice scalabilità del sistema in ambiente distribuito senza degrado di performance
- Ogni workstation dispone della propria RAM e dei propri dischi, che non sono condivisi

Cosa non è NoSQL

Addio ad SQL

- Esistono sistemi NoSQL che utilizzano SQL (o un linguaggio SQL-like)

Open-source

- Molti sistemi NoSQL sono open-source, ma ne esistono anche molti commerciali

Cloud Computing

- Il Cloud favorisce le capacità di scalare dei sistemi NoSQL, ma è possibile usarli anche in-house

Ottimizzazione delle risorse hardware

- A parità di risorse, un DBMS relazionale centralizzato offre prestazioni migliori

I precursori del movimento NoSQL

LiveJournal, 2003

- Obiettivo: ridurre il numero di interrogazioni al DB da un insieme di web server
- Soluzione: Memcached, studiato per mantenere in RAM query e risultati

Google, 2005

- Obiettivo: gestire l'enorme quantità di dati raccolti (indicizzazione del web, Maps, Gmail, ecc.)
- Soluzione: BigTable, ottimizzato per garantire scalabilità e performance su Petabyte di dati

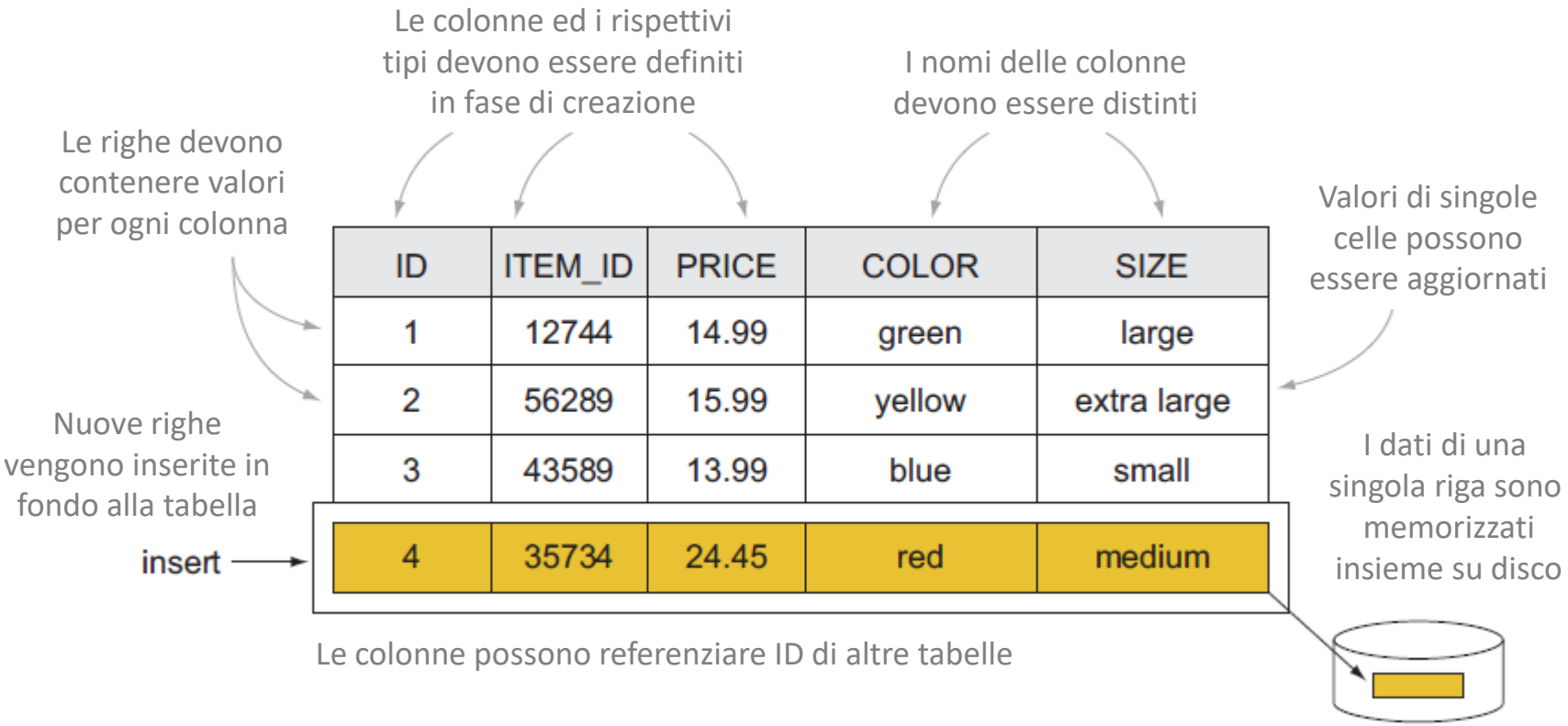
Amazon, 2007

- Obiettivo: assicurare l'affidabilità e la continuità del servizio di vendita online, 24/7
- Soluzione: DynamoDB, caratterizzato da una forte semplicità d'uso e di gestione dei dati

Modelli dati

Modello relazionale

Il modello relazionale si basa su righe e tabelle



ID	ITEM_ID	PRICE	COLOR	SIZE
1	12744	14.99	green	large
2	56289	15.99	yellow	extra large
3	43589	13.99	blue	small
4	35734	24.45	red	medium

NoSQL: tanti modelli

Una delle principali difficoltà è capire quale modello adottare

Modello	Descrizione	Casi d'uso
Key-value	Associa un qualunque valore ad una stringa di testo	Dizionari, tabelle di lookup, cache, memorizzazione file e immagini
Document	Memorizza informazioni gerarchiche con una struttura ad albero	Documenti, qualunque dato idoneo ad una struttura gerarchica
Column family	Memorizza matrice sparse usando sia la riga che la colonna come chiave	Crawling, sistemi con elevata variabilità, matrici sparse
Graph	Memorizza nodi e archi	Query su reti sociali, inferenza, pattern matching

Key-value: modello

Un DB contiene una o più collezioni (corrispettive delle tabelle)

Ogni collezione contiene un elenco di coppie chiave-valore

- Chiave: una stringa di testo univoca
  - Esempi: id, hash, path, query, chiamate REST
- Valore: un BLOB (binary large object)
  - Esempi: testo, documenti, pagine web, file multimediali

Livello di atomicità: la coppia chiave-valore

Key	Value
image-12345.jpg	Binary image file
http://www.example.com/my-web-page.html	HTML of a web page
N:/folder/subfolder/myfile.pdf	PDF document
9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

Ricorda un semplice dizionario

- La collezione è indicizzata per chiave
- Il valore può contenere informazioni diverse: una o più definizioni, sinonimi e contrari, immagini, ecc.

Key-value: interrogazione

Tre semplici modalità di interrogazione:

- put($key as xs:string, $value as item())
  - Aggiunge una coppa chiave-valore alla collezione
  - Se la chiave esiste già, il valore viene rimpiazzato
- get($key as xs:string) as item()
  - Restituisce il valore corrispondente alla chiave indicata (se esiste)
- delete($key as xs:string)
  - Elimina la coppia con la chiave indicata

Key	Value
user:1234:name	Enrico
user:1234:age	30
post:9876:written-by	user:1234
post:9876:title	NoSQL Databases
comment:5050:reply-to	post:9876

Il valore è come una *black box*: non lo si può interrogare!

- Non è possibile fare query con filtri
- Non è possibile indicizzare i valori
- Per questo motivo, le chiavi sono spesso "parlanti"

Document: modello

Un DB contiene una o più collezioni (corrispettive delle tabelle)

Ogni collezione contiene un elenco di documenti (tipicamente JSON)

- Un documento ha una struttura ad albero auto-descrittiva

Ogni documento contiene un insieme di campi

- L'unico campo obbligatorio è l'ID,
  il cui valore identifica il documento nella collezione

Ogni campo è strutturato come una coppia chiave-valore

- Chiave: stringa di testo univoca all'interno del documento
- Valore: può essere semplice (stringa, numero, booleano)
  o complesso (oggetto, array, BLOB)
  - Un valore può contenere campi a sua volta

Livello di atomicità: il documento

```
{
    "_id": 1234,
    "name": "Enrico",
    "age": 30,
    "address": {
        "city": "Ravenna",
        "postalCode": 48124
    },
    "contacts": [ {
        "type": "office",
        "contact": "0547-338835"
    }, {
        "type": "skype",
        "contact": "egallinucci"
    } ]
}
```

Document: interrogazione

A differenza dei key-value, il contenuto del documento è visibile al DBMS

Di conseguenza, l'espressività dei linguaggi di interrogazione è molto più elevata

- Possibile creare indici sui campi
- Possibile filtrare sui campi
- Possibile ottenere più documenti con un'unica query
- Possibile effettuare proiezioni sui documenti
- Possibile eseguire degli update sui campi di un documento

Implementazioni diverse offrono funzionalità diverse

- Meccanismi per gestire viste materializzate o dinamiche
- Interrogazione dei dati secondo il paradigma MapReduce
- Connettori con diversi strumenti Big Data (e.g., Spark, Hive)
- Esecuzione di interrogazioni *full-text*

Column-family: modello

Un DB contiene una o più famiglie di colonne (o tabelle)

Ogni famiglia di colonne contiene un elenco di righe nella forma di coppie chiave-valore

- Chiave: stringa di testo univoca all'interno della famiglia di colonne
- Valore: un insieme di colonne

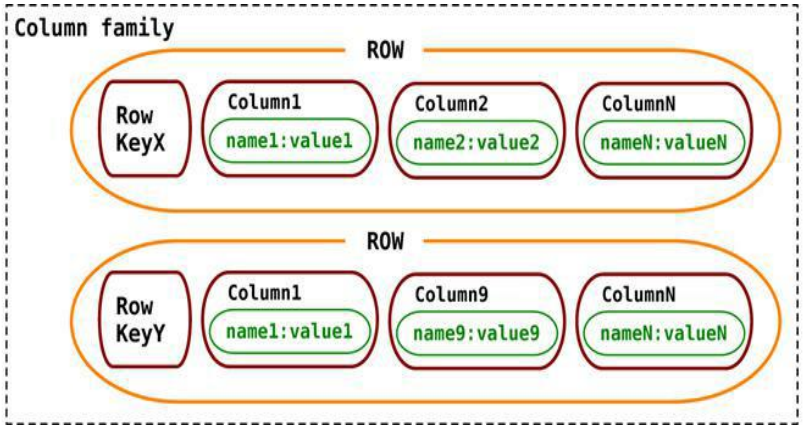
Ogni colonna è, a sua volta, una coppia chiave-valore

- Chiave: stringa di testo univoca all'interno della riga
- Valore: semplice o complesso (*supercolonna*)

Livello di atomicità: la riga

Rispetto al modello relazionale:

- Le righe specificano solo le colonne per cui esiste un valore
  - Soluzione particolarmente adeguata per matrici sparse
- Il valore di una colonna può essere "versionato" sulla base di un timestamp



Famiglia di colonne	Chiave di riga	Chiave di colonna	Timestamp	Valore

Column-family: interrogazione

Column-family = wide-column

L'espressività dei linguaggi è una via di mezzo tra i key-value ed i document

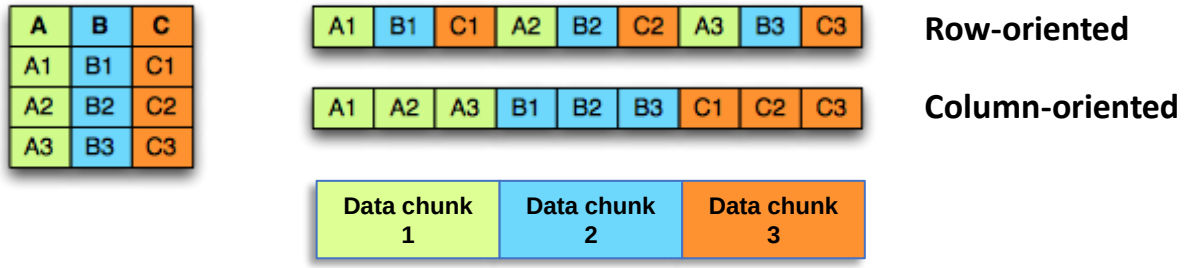
- Sconsigliato (ove possibile) creare indici sulle colonne
- Possibile filtrare sulle colonne (con alcune limitazioni)
- Possibile ottenere più righe con un'unica query
- Possibile effettuare proiezioni sulle righe
- Impossibile eseguire degli update sulle colonne di un documento
  - Possibile solo rimpiazzarla interamente, come nei key-value

Data la similarità col modello relazionale, è spesso disponibile un linguaggio SQL-like

Column-family: ≠ colonnare

I database colonnari memorizzano i dati per colonne anziché per righe

- Vantaggi: minore I/O, miglior compressione dei dati, ecc.
- Utilizzati in contesti OLAP
- Non rientrano nella categoria dei NoSQL!



Graph: modello

Un DB contiene uno o più grafi

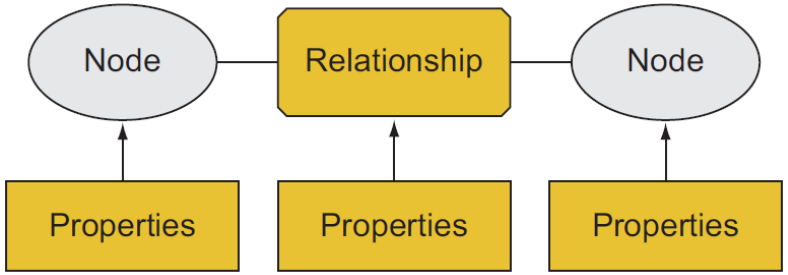
Ogni grafo è composto da nodi e relazioni

- Nodi: rappresentano solitamente entità reali
  - E.g.: persone, organizzazioni, pagine web, workstation, cellule, libri, ecc.
- Relazioni: rappresentano connessioni direzionate tra i nodi
  - E.g.: amicizie, relazioni di lavoro, collegamenti ipertestuali, collegamenti ethernet, diritti d'autore, ecc.
- Nodi e relazioni sono descritti da proprietà

Livello di atomicità: la transazione

Specializzazioni più conosciute:

- Modello reticolare
  - Principalmente relazioni di tipo parent-child o owner-member
- Triplestore
  - Relazioni soggetto-predicato-oggetto (e.g., RDF)



Graph: interrogazione

I database a grafo modellano situazioni completamente diverse

Di conseguenza, anche linguaggi e modalità di interrogazione cambiano

- Supporto alle transazioni
- Supporto a indici, selezioni e proiezioni
- Linguaggio di interrogazione basato su pattern

Query	Pattern
Trova amici di amici	(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)
Trova il percorso più breve tra A e B	shortestPath((userA)-[:KNOWS*..5]-(userB))
Cosa ha comprato chi ha comprato i miei stessi prodotti?	(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED]->(otherProduct)

Graph vs Aggregate-oriented

I modelli a grafo sono intrinsecamente diversi rispetto agli altri

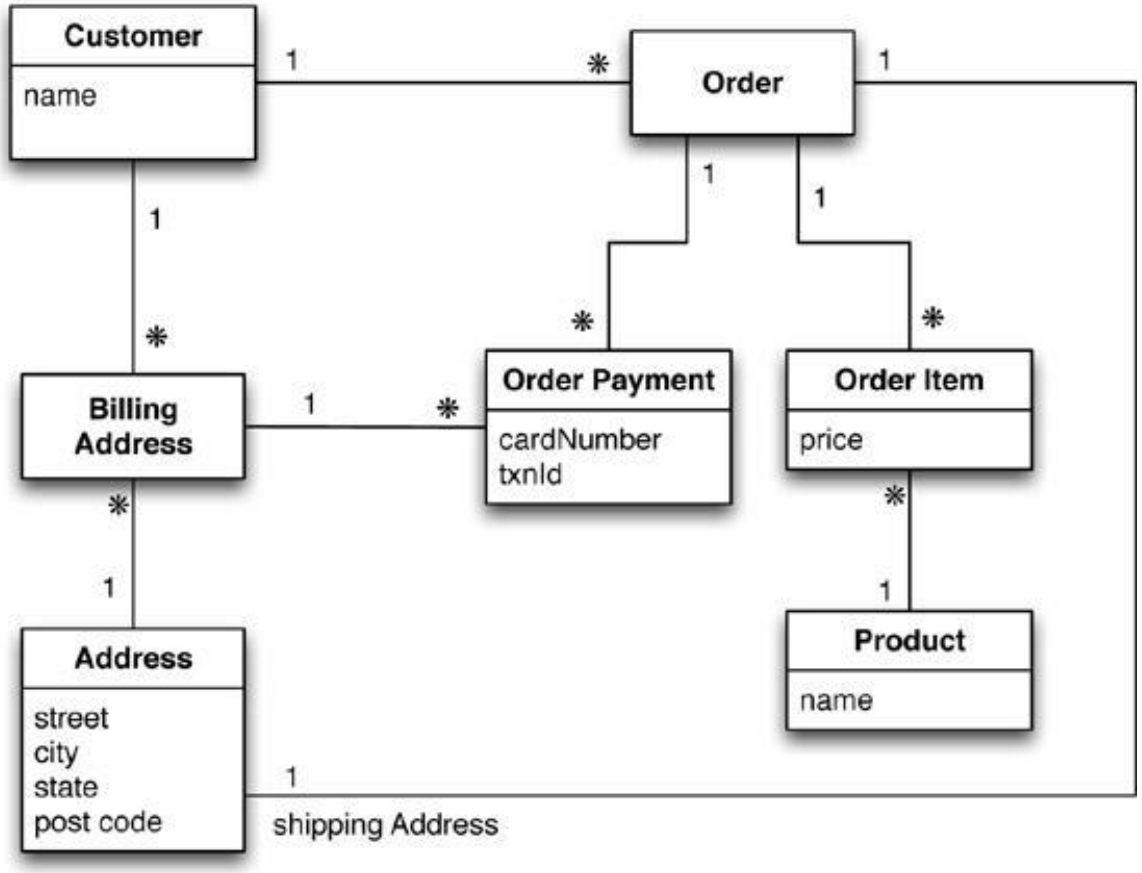
- Focus sulle relazioni tra entità piuttosto che sulle entità stesse
- Scalabilità ridotta
- Modellazione data-driven

Gli altri modelli (key-value, document e column-family) sono definiti anche aggregate-oriented

- L'aggregato è il blocco atomico del modello
- Più incapsulamento, meno join
- Elevata scalabilità
- Modellazione query-driven

Modellazione dati: un esempio

Tipico caso d'uso: clienti, ordini, prodotti



Modellazione dati: relazionale

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

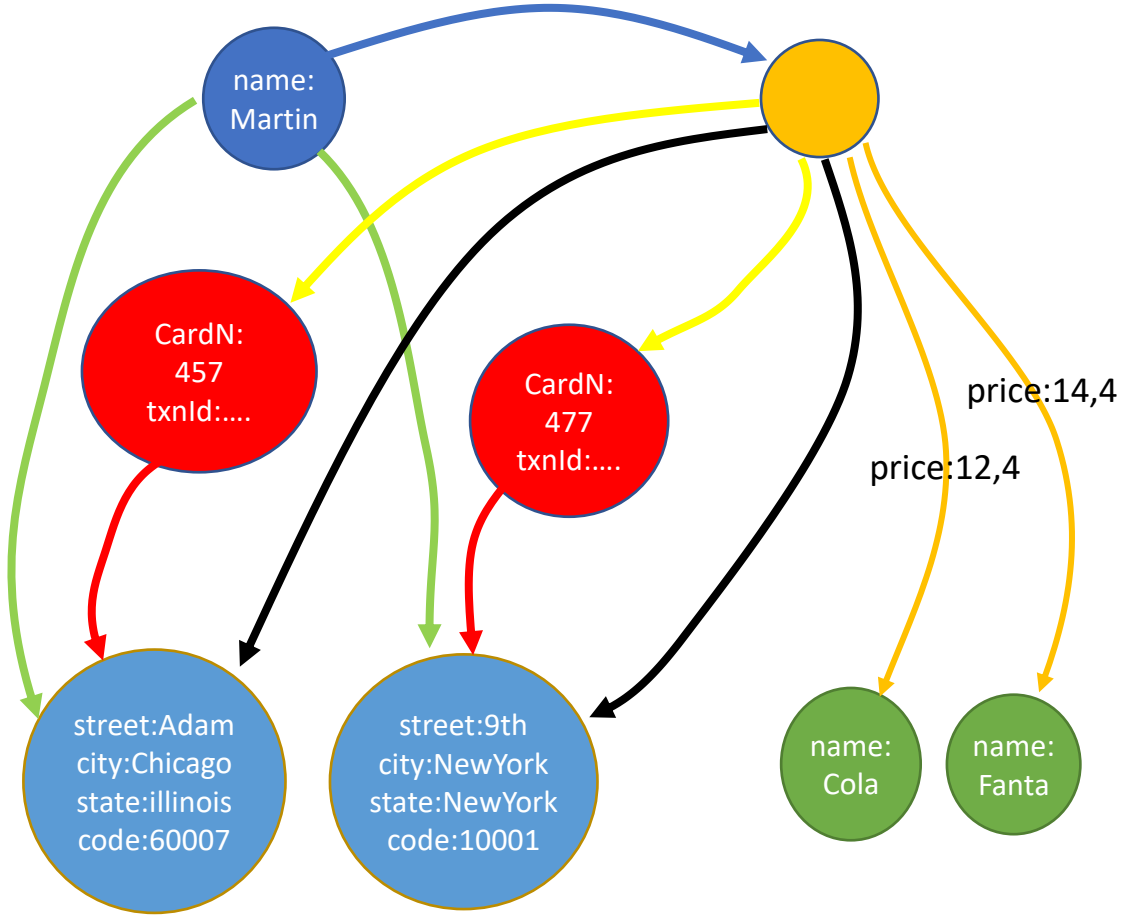
Id	City
77	Chicago

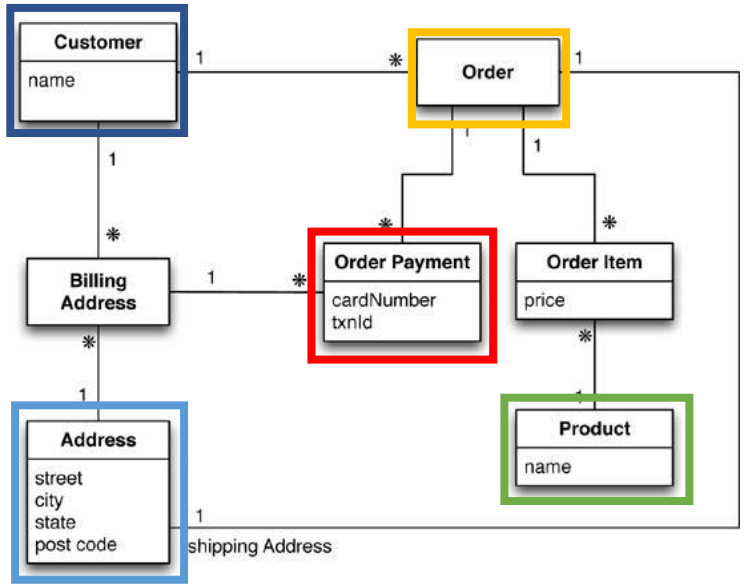
OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Modellazione dati: grafo

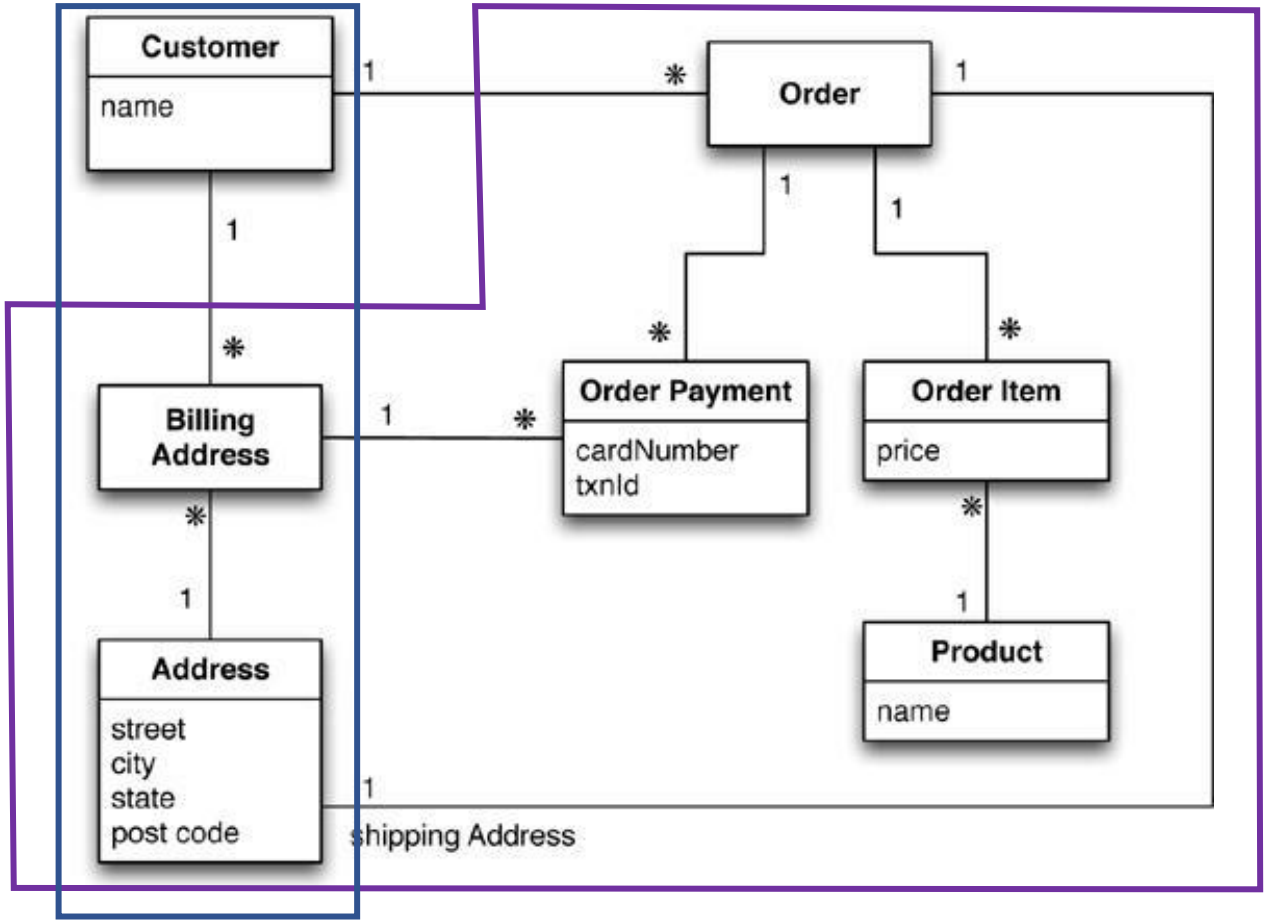
- Gli ID sono gestiti implicitamente





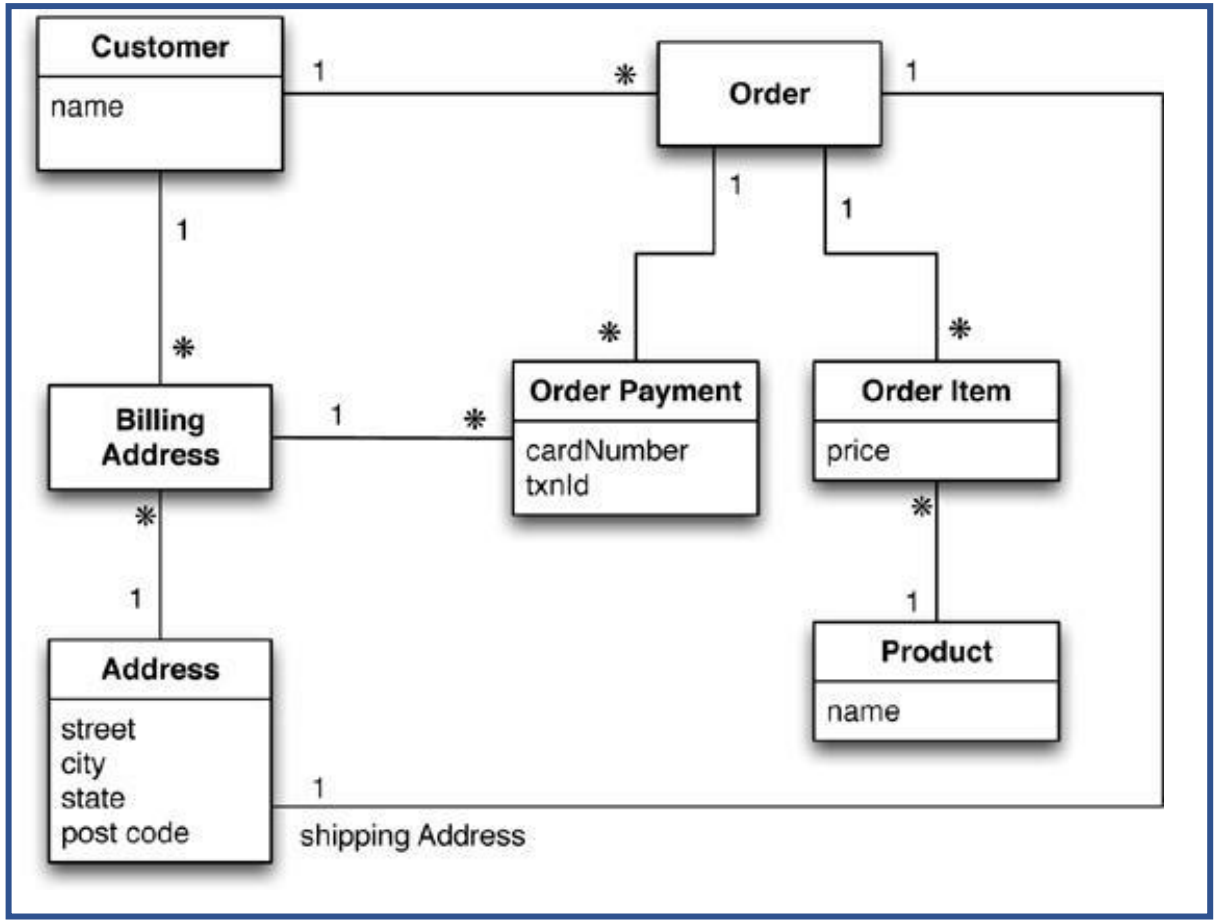
Modellazione dati: orientata agli aggregati

Una possibile modellazione di aggregati



Modellazione dati: orientata agli aggregati

Una modellazione alternativa



Modellazione dati: chiave-valore

Customer collection

key	value
cust-1:name	Martin
cust-1:adrs	[{"street":"Adam", "city":"Chicago", "state":"Illinois", "code":60007}, {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}]

cust-1:ord-99	{
	"orderpayments": [
	{"card":477, "billadrs":
	{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007} },
	{"card":457, "billadrs":
	{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
	],
	"products": [
	{"id":1, "name":"Cola", "price":12.4},
	{"id":2, "name":"Fanta", "price":14.4}
	],
	"shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", code":10001}
	}

Product collection

key	value
p-1:name	Cola
p-2:name	Fanta

Modellazione dati: documentale (1)

Customer collection

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ],
  "orders": [ {
    "orderpayments":[
      {"card":477, "billadrs": {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
      {"card":457, "billadrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
    ],
    "products":[
      {"id":1, "name":"Cola", "price":12.4},
      {"id":2, "name":"Fanta", "price":14.4}
    ],
    "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  }]
}
```

Product collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

Modellazione dati: documentale (2)

```
{
  "_id": 1,
  "name": "Martin",
  "adrs": [
    {"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007},
    {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
  ]
}
```

Customer collection

```
{
  "_id":1,
  "name":"Cola",
  "price":12.4
}
```

```
{
  "_id":1,
  "name":"Fanta",
  "price":14.4
}
```

Product collection

```
{
  "_id": 1,
  "customer":1,
  "orderpayments":[
    {"card":477, "billadrs":{"street":"Adam", "city":"Chicago", "state":"illinois", "code":60007}},
    {"card":457, "billadrs":{"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}}
  ],
  "products": [
    {"id":1, "name":"Cola", "price":12.4},
    {"id":2, "name":"Fanta", "price":14.4}
  ],
  "shipAdrs": {"street":"9th", "city":"NewYork", "state":"NewYork", "code":10001}
}
```

Order collection

Modellazione dati: wide-column

Order table > Order details column family

Ord	CustName	Pepsi	Cola	Fanta	…
1	Martin		12.4	14.4	
2	…	…			…

Order table > Order payments column family

Ord	OrderPayments
1	Card | Steet | City | State | Code 477 | 9th | NewYork | NewYork | 10001 457 | Adam | Chicago | Illinois | 60007
2	…

Modellazione di aggregati: progettazione

Il termine deriva dal Domain-Driven Design

- Un aggregato è un insieme di oggetti strettamente correlati tra loro e che vengono trattati in blocco
- Un aggregato è un'unità per la manipolazione dei dati e la gestione della consistenza

Vantaggi degli aggregati:

- Facilitano il lavoro degli sviluppatori software, che spesso manipolano i dati attraverso strutture aggregate
- Facili da gestire in un sistema distribuito

Non esiste una strategia universale per la definizione dei confini degli aggregati

- Dipende unicamente da come si intende manipolare i dati

Gli aggregati sono di grande aiuto per la gestione su cluster

- Pro: i dati che devono essere manipolati insieme (e.g., gli ordini ed i relativi dettagli) vengono modellati nello stesso aggregato – e quindi risiedono nello stesso nodo
- Contro: gli aggregati facilitano solo alcuni tipi di interazione

I DBMS relazionali sono agnostici da questo punto di vista

Gestione della distribuzione dei dati

Uno sguardo dietro le quinte

Distribuzione dei dati

Uno dei punti di forza dei sistemi NoSQL è la capacità di *scalare*

- L'aggregato è un'unità che si presta bene alla distribuzione all'interno di un cluster

In generale, gestire un cluster introduce una certa complessità

- L'utilizzo di un database NoSQL in un server singolo non è da escludere

In un sistema distribuito ci sono due aspetti ortogonali da valutare:

- **Sharding**: spacchettare i dati su nodi diversi
- **Replication**: copiare gli stessi dati su nodi diversi
  - Modalità master-slave
  - Modalità peer-to-peer

Obiettivi di distribuzione e scalabilità

Robustezza

- Replicazione: in caso di perdita di nodi, la replicazione impedisce la perdita di dati

Efficienza

- Distribuzione: partizioni differenti possono essere lette/scritte in parallelo da nodi diversi
- Replicazione: gli stessi dati possono essere letti in parallelo da utenti diversi attraverso nodi diversi

Single-server

La soluzione più semplice: utilizzare il database in un'unica macchina

- Molti database NoSQL progettati per lavorare in forma distribuita…
- … ma non significa che non lavorino bene in locale
- I database a grafo lavorano meglio in locale

Quando ha senso l'approccio single-server?

- Mole di dati non è enorme
- Si vogliono sfruttare le caratteristiche dei database NoSQL (e.g., schemaless, modello dati)

Un sistema distribuito è inevitabilmente più complesso; non è detto che ne valga sempre la pena

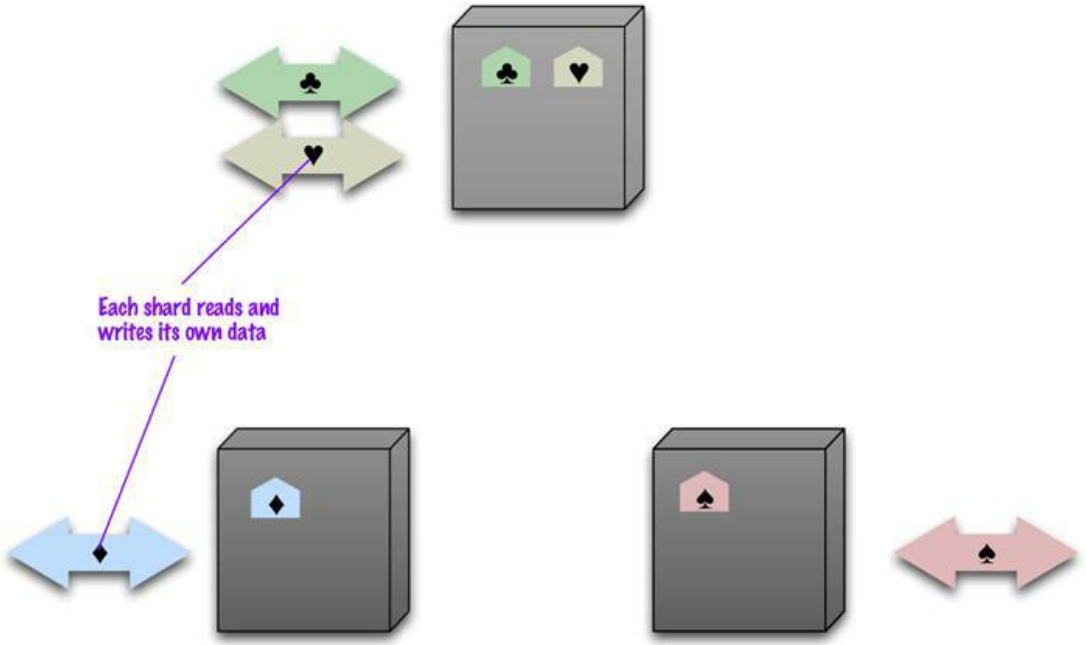
Sharding

Sharding: spezzare i dati in parti (*shard*) che vengono memorizzate su macchine diverse

- I.e., scalare orizzontalmente

Una buona *strategia di sharding* è **fondamentale** per ottimizzare le performance

- Tipicamente basata sul valore di uno o più campi



Strategie di sharding

Regole principali per una strategia di sharding:

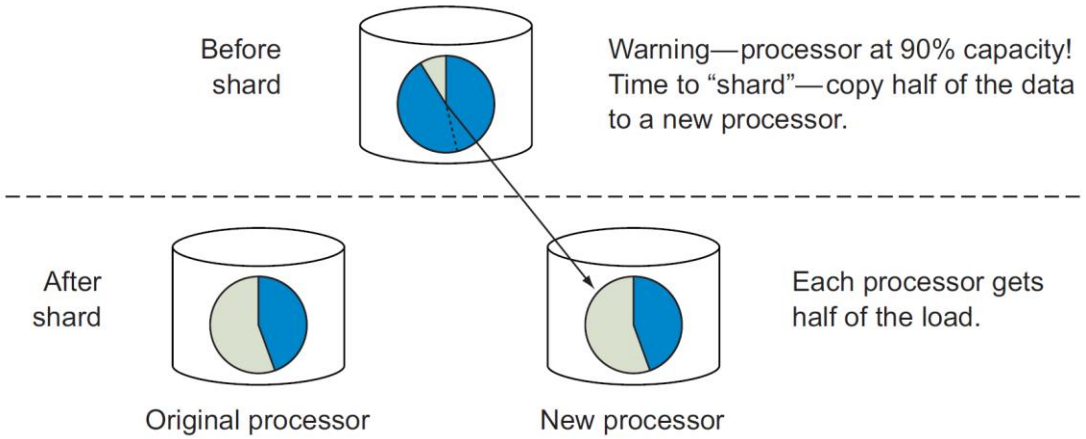
1. Memorizzare i dati vicini al luogo da cui devono essere acceduti
   - E.g., memorizzare gli ordini per l'Italia nel data center europeo
2. Mantenere una distribuzione bilanciata
   - Ogni nodo dovrebbe ricevere (più o meno) la stessa percentuale di dati
3. Mettere insieme dati che potrebbero essere letti in sequenza
   - Stesso cliente, stesso nodo

Auto-sharding

Molti database NoSQL offrono politiche di auto-sharding

- Il database gestisce la distribuzione dei dati in base al carico di lavoro abituale

Attenzione: ridefinire (o definire tardivamente) la strategia di sharding può risultare molto oneroso



Replication

Replication: i dati vengono copiati su diversi nodi

Come gestire la distribuzione delle repliche nei nodi?

- Principio comunemente adottato:
  - Una replica nel nodo che riceve il dato
  - Una replica in un nodo diverso all'interno dello stesso rack
  - Una replica in un nodo all'interno di un rack diverso

Ogni aggiornamento ad un dato richiede la propagazione delle modifiche a tutte le sue repliche

Due tecniche per gestire gli aggiornamenti:

- Master-slave
- Peer to peer

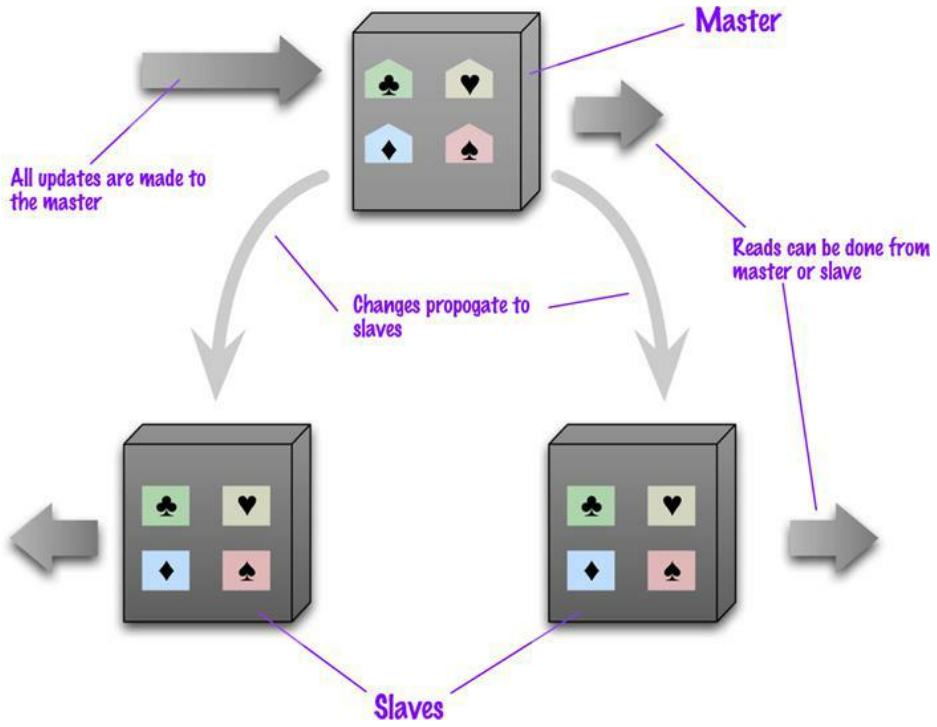
Master-Slave Replication

Master

- E’ il "manager" dei dati
- Gestisce tutti i processi di aggiornamento
- Può essere deciso manualmente o sorteggiato

Slaves

- Consentono la lettura dei dati
- Sincronizzati con il master
- Possono diventare master in caso di failure di quest’ultimo



Master-Slave Replication: pro e contro

Pro

- Gestisce facilmente molte richieste di lettura
  - Basta aggiungere più nodi e assicurarsi che le letture vengono gestite degli slave
- Gli slave possono gestire le letture anche senza master
- Utile quando il carico di lavoro è principalmente in lettura

Contro

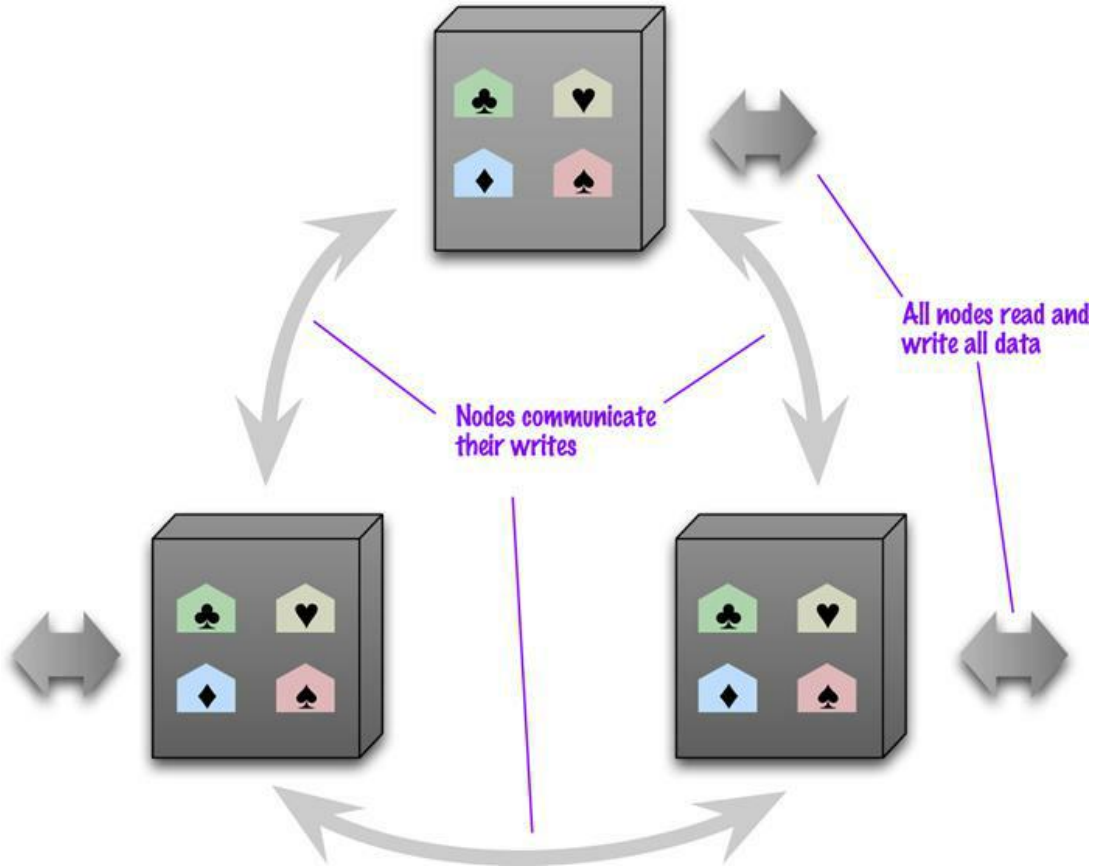
- Il master è un collo di bottiglia
  - **Solo il master può gestire gli aggiornamenti dei dati**
  - In caso di failure, bisogna attendere il ripristino del master o una nuova elezione
- La lenta propagazione dei cambiamenti può generare inconsistenza
  - Due utenti possono leggere valori diversi nello stesso momento
  - Le inconsistenze in lettura possono essere problemantiche, ma hanno una durata limitata
- Non ideale in presenza di un forte carico di lavoro in scrittura

Peer-to-Peer Replication

Tutti i nodi hanno la stessa importanza

Tutti i nodi gestiscono gli aggiornamenti

La perdita di un nodo non compromette
né letture né scritture



Peer-to-Peer Replication: pro e contro

Pro

- Il fallimento (failure) di un nodo non interrompe le richieste di letture e di scritture
- Aggiungendo nodi di aumentano facilmente le performance

Contro

- Inconsistenza!
- La lenta propagazione dei cambiamenti può generare inconsistenza
  - Due utenti possono leggere valori diversi nello stesso momento
  - Le inconsistenze in lettura possono essere problematiche, ma hanno una durata limitata
  - E' lo stesso problema che occorre in replicazione master-slave
- Due persone possono aggiornare diverse copie dello stesso dato (memorizzate in nodi diversi) nello stesso momento
  - Le inconsistenze in scrittura sono per sempre

Consistenza con replicazione: gestire i conflitti

Conflitti in lettura

- Tolleranza: la finestra di inconsistenza è solitamente limitata
- Read-your-writes: la consistenza in lettura è garantita rispetto alle proprie modifiche
  - Si associa un utente ad un unico nodo (rischio: carico di lavoro sbilanciato)
  - Si usano campi di versionamento per assicurarsi che nessuno abbia modificato il valore dall'ultima lettura

Conflitti in scrittura (modello P2P)

- Last write wins: l'ultimo update sovrascrive i precedenti
- Prevenzione: nuove scritture solo sulla versione più recente
- Rilevamento: si mantiene la cronologia delle modifiche e si segnala il problema all'utente, lasciandogli facoltà di decidere quale versione confermare

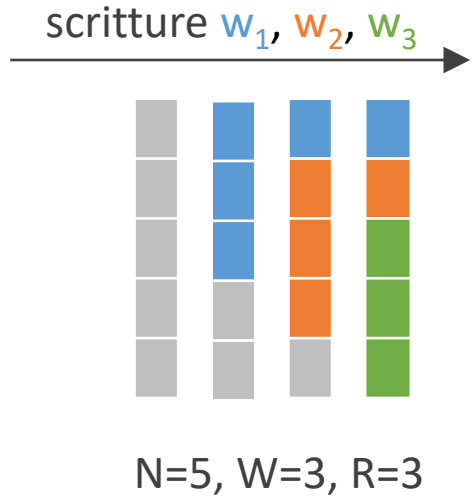
Consistenza con replicazione: il quorum

Il meccanismo di quorum permette di consistenza sia in lettura che in scrittura in presenza di replicazione dei dati

- Si basa sulla necessità di contattare una determinata maggioranza di nodi responsabili di un certo dato
- Il quorum corrisponde al numero minimo di repliche che devono essere contattate per poter avere la possibilità di eseguire un'operazione su un dato replicato

Ogni dato ha N repliche

- Quorum in scrittura: W > N/2,
  - La scrittura è consentita solo se W repliche possono essere aggiornate
  - Garantisce che due operazioni non avvengano contemporaneamente
- Quorum in lettura: R > N-W
  - L'operazione è consentita solo se R repliche possono essere lette
  - Garantisce che almeno una copia aggiornata venga letta



Gestione della consistenza

Uno sguardo dietro le quinte

RDBMS vs NoSQL: filosofie diverse

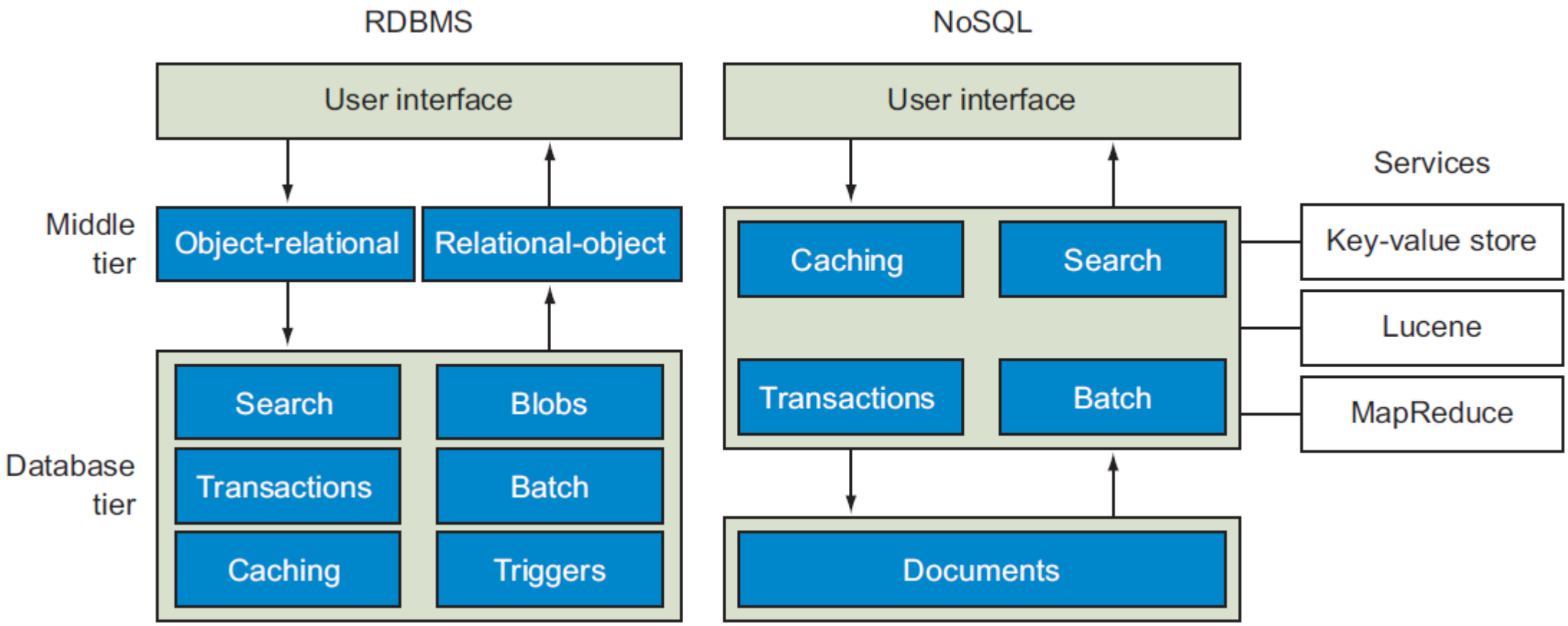
I DBMS relazionali vengono da decenni di utilizzo assodato

- Forte focus sulla consistenza dei dati
- Ottimizzazione delle performance maturata con anni di ricerca
- Sistemi altamente complessi (trigger, caching, sicurezza, ecc.)

I sistemi NoSQL nascono per far fronte alle limitazioni degli RDBMS

- Forte focus sulla distribuzione e sulla disponibilità dei dati
- Sistemi molto semplici (per ora)
- Rapidità e maneggevolezza piuttosto che consistenza a tutti i costi

RDBMS vs NoSQL: filosofie diverse



RDBMS vs NoSQL: filosofie diverse

Pro RDBMS

- Problematiche di consistenza e concorrenza gestita dal DBMS
- Politiche e meccanismi di sicurezza affidabili e ad alto livello di dettaglio
- Standard SQL: stesso codice, tecnologie diverse
- Schema e vincoli garantiscono la qualità del dato
- ER ed SQL conosciuti da tutti
- Integrazione immediata con strumenti di reportistica e OLAP

Pro NoSQL

- Non è necessario definire lo schema per poter cominciare a lavorare (fase di test)
- Gestione efficiente della distribuzione dei dati su più nodi
- Scalabilità lineare delle performance rispetto al numero di nodi
- Funzionalità di text-search integrate (non sempre) ed efficienti
- Libertà dagli ORM
- Malleabilità nella gestione di dati ad alta varietà

Consistenza: un esempio

Immaginiamo di fare un giroconto di 1000€ dal conto A al conto B; il passaggio richiede due operazioni

- Prelievo di 1000€ dal conto A
- Deposito di 1000€ sul conto B

In nessun caso deve poter capitare che:

- A causa di un errore, i 1000€ non vengano depositati sul conto B e vengano persi
- A causa di un errore, i 1000€ vengono depositati due volte sul conto B (magari!)
- Interrogando il database, venga visualizzato uno stato intermedio
  - E.g., se in A c'erano solo 1000€ e in B 0€, non deve poter essere visualizzato uno stato intermedio in cui A+B = 0€

Negli RDBMS esiste un meccanismo infallibile: le transazioni

Consistenza negli RDBMS: ACID

Le transazioni garantiscono quattro proprietà fondamentali, denominate ACID

Atomicity

- La transazione è indivisibile: o viene completata interamente con successo, o fallisce
- Non è possibile che venga completata solo a metà

Consistency

- La transazione lascia il DB in uno stato consistente
- Nessuno dei vincoli di integrità del DB può mai essere violato

Isolation

- Una transazione esegue indipendentemente dalle altre
- Se più transazioni eseguono in concorrenza, l'effetto netto equivale a quello di un'esecuzione seriale

Durability

- Il DBMS protegge il DB a fronte di guasti

Consistenza negli RDBMS: ACID

L'implementazione delle transazioni ACID richiede la gestione di un meccanismo di lock e log

- Blocco delle risorse e tracciamento delle modifiche
- In caso di problemi, ripristino dello stato iniziale
- Alla fine, sblocco delle risorse

Garanzia di consistenza a discapito di velocità e disponibilità del dato

- Ricadute sui tempi di attesa degli utenti
- Difficili da gestire se il database è distribuito

In alcune situazioni, la garanzia di consistenza è meno importante

- E.g.: acquisto di prodotti
- La gestione del carrello richiede velocità e disponibilità
- L'emissione dell'ordine richiede consistenza

Consistenza nei NoSQL

Diversi tentativi sono stati fatti per descrivere le proprietà dei sistemi NoSQL in contrapposizione alle proprietà ACID delle transazioni

- Proprietà BASE
- Teorema CAP

Non vanno intese come proprietà garantite dai sistemi NoSQL…

… piuttosto come proprietà che *tentano* di descriverne il comportamento

Consistenza nei NoSQL: BASE

Basic **A**vailability

- Il sistema deve essere sempre disponibile

Soft-state

- E' accettabile che il sistema presenti delle inconsistenze, purché temporanee

Eventual consistency

- Prima o poi, il sistema verrà lasciato in uno stato consistente
- Il concetto più importante

ACID

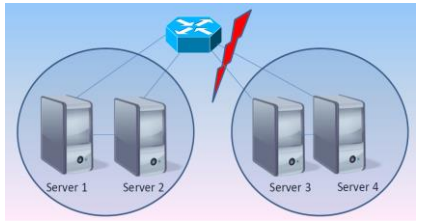
- Approccio pessimista (prevenire meglio che curare)

BASE

- Approccio ottimista (prima o poi si sistema tutto)
- Focus su throughput piuttosto che su consistenza

Consistenza nei NoSQL: teorema CAP

"Teorema": date le seguenti tre proprietà, solo due possono essere garantite



Consistency: il sistema è sempre consistente

Availability: il sistema è sempre disponibile

Partition tolerance: il sistema può subire partizionamenti di rete

Tre situazioni

- CA: il sistema non può subire partizionamenti (single server)
- AP: in caso di partizionamenti, il sistema sacrifica la consistenza (overbooking)
- CP: in caso di partizionamenti, il sistema sacrifica la disponibilità (prenotazioni bloccate)

Teorema controverso e di difficile interpretazione

- Asimmetria delle proprietà: i sistemi che sacrificano la consistenza lo fanno sempre (per favorire la velocità), non solo in caso di partizionamento
- A seconda dei requisiti applicativi, il rapporto tra queste proprietà viene gestito in maniera più o meno rigida

Consistenza nei NoSQL: soluzioni

Consideriamo due utenti che vogliono prenotare la stessa stanza d'albergo, nonostante avvenga un partizionamento di rete

CP: nessun utente può prenotare stanze (si sacrifica A)

- Situazione non ideale

AP: entrambi i nodi accettano le richieste di prenotazione (si sacrifica C)

- Possibile overbooking: conflitto in scrittura da gestire

caP: solo un utente può prenotare

- L'altro vedrà la stanza disponibile ma non prenotabile

La gestione di queste situazioni dipende strettamente dal contesto

- Trading finanziario? Blog? E-commerce?

L'importante è capire:

- Qual è la tolleranza sulla lettura di dati obsoleti
- Quanto può essere ampia la finestra di inconsistenza

Consistency in NoSQL: summary

Sorgente	Causa	Effetto	Soluzione
Replicazione (MS, P2P)	Le propagazione delle modifiche sulle repliche è lenta	Conflitto in lettura	- Tollerare - Read-your-writes - Quorum
Replicazione (P2P)	Due scritture vengono eseguite su repliche diverse dello stesso dato	Conflitto in scrittura	- Last write wins - Prevenire - Rilevare - Quorum
Partizionamento di rete	Impossibilità di comunicare con tutte le repliche di un certo dato	- Conflitto in lettura - Possibile conflitto in scrittura	- Rilassare il CAP - Prevenire conflitti in scrittura - Gestire conflitti in scrittura come sopra
Assenza di proprietà ACID	- Un aggiornamento di più record fallisce a metà query - Due aggiornamenti di più record si sovrappongono	Inconsistenza irrecuperabile	- Ogni sistema prevede dei meccanismi più o meno limitati di proprietà ACID
Denormalizzazione dei dati	Lo stesso dato è replicato su record diversi con valori diversi	Impossibilità di individuare i valori corretti	- Evitare la denormalizzazione se è necessaria una forte consistenza - Pulire i dati prima di analizzarli

One size does not fit all

Ad ogni applicazione il giusto modello dati

DB Key-Value popolari

Riak: http://basho.com/riak/

Redis (Data Structure server): http://redis.io/

- Più di un semplice key-value perché supporta la memorizzazione di strutture più complesse (liste, set, …) e l'esecuzione di operazioni sui valori (range, diff, …)

Memcached DB: http://memcached.org/

Berkeley DB: http://www.oracle.com/us/products/database/berkeley-db/

HamsterDB: http://hamsterdb.com/

- Specialmente per usi embedded

Amazon DynamoDB: https://aws.amazon.com/dynamodb/

- Un servizio non open-source

Project Voldemort: http://www.project-voldemort.com/

- Un'implementazione open-source di Amazon DynamoDB

DB Key-Value: quando usarli

Caso d'uso molto semplici

- Dati indipendenti (nessuna necessità di modellare relazioni)
- Le query sono solitamente delle semplici lookup
- Sono necessarie performance molto elevate

Esempi

- **Memorizzare informazioni di sessione**
  - Ogni sessione web è unica e ha un valore univoco di `sessionId`. Tutte le informazioni su una sessione possono essere memorizzare con una richiesta `PUT` e recuperate con una richiesta `GET`.
- **Profili utente, preferenze**
  - Ogni utente ha il suo identificatore (`userId`, username, o altro) e ha le sue preferenze in termini di lingua, colori, timezone, elenco dei prodotti accessibili, ecc. – tutte informazioni che possono essere messe in un unico aggregato.
- **Shopping Cart, servizi di chat**
  - Ogni sito di e-commerce associa un carrello ad ogni utente, che può essere memorizzato in un aggregato la cui chiave è l'ID dell'utente.

DB Key-Value: casi d'uso reali

Crawling di pagine web

- La chiave è l'URL, il valore è il contenuto intero della pagina (HTML, CSS, JS, immagini, ..)

Key	Value
http://www.example.com/index.html	<html>…
http://www.example.com/about.html	<html>…
http://www.example.com/products.html	<html>…
http://www.example.com/logo.png	Binary…

Twitter timeline

- Per ogni utente viene materializzata la lista dei tweet da mostrare nella timeline

Amazon S3 (Simple Storage Service)

- Un servizio per memorizzare file nel cloud
- Utile per gestire backup personali, condivisione di file con gruppi di utenti, pubblicazione di siti e applicazioni
- Si paga al consumo
  - Storage: approx. $0.03 per GB per month
  - Uploading files: approx. $0.005 per 1000 items
  - Downloading files: approx. $0.004 per 10,000 files* PLUS $0.09 per GB (first GB free)

DB Key-Value: quando non usarli

Dati con relazioni

- Se c’è bisogno di gestire le relazioni fra dati in dataset diversi (o fra dati con set di chiavi diversi)
- Alcuni sistemi forniscono meccanismi di link-walking

Operazioni multi-record

- Perché non si possono fare operazioni su più record

Interrogazioni sui dati

- Se si vogliono effettuare interrogazioni sulle informazioni memorizzate nel valore dell’aggregato anziché sulla chiave
- Alcuni prodotti offrono limitate funzionalità di interrogazione sui valori

DB documentali popolari

MongoDB: http://www.mongodb.org

CouchDB: http://couchdb.apache.org

Terrastore: https://code.google.com/p/terrastore

OrientDB: http://www.orientechnologies.com

RavenDB: http://ravendb.net

Lotus Notes: http://www-03.ibm.com/software/products

DB documentali: quando usarli

Elevata espressività

- Memorizzare i dati secondo un modello innestato
- Formulare query complesse che coinvolgono molti campi

Esempi

- **Log di eventi / web services**
  - Repository centrali per la memorizzazioni di log di eventi di diverse applicazioni; sharding sulla base del nome dell'applicazione sorgente o del tipo di evento.
- **CMS, piattaforme di blogging**
  - L'assenza di uno schema predefinito rende i database documentali adatti all'utilizzo per content management systems (CMS) o per applicazioni di gestione di siti web, per la gestione di commenti, registrazioni e profili utente
- **Web Analytics o Real-Time Analytics**
  - La possibilità di aggiornare solo alcune parti di un documento permette di salvare e aggiornare facilmente delle metriche di analisi
  - L'indicizzazione di contenuti testuali abilita anche applicazioni di real-time sentiment analysis e social media monitoring.
- **Applicazioni di e-commerce**
  - Le applicazioni di e-commerce hanno spesso bisogno di flessibilità sullo schema per memorizzare prodotti e ordini e per poter permettere l'evoluzione del proprio modello dati senza incorrere in costi di refactory o di migrazione dei dati

DB documentali: casi d'uso reali

Servizi di advertising

- MongoDB nasce come sistema di gestione di banner pubblicitari
  - Il servizio deve essere disponibile 24/7 e molto efficiente
  - Necessarie regole complesse per trovare il banner giusto in base agli interessi della persona
  - Necessità di gestire tipologie diverse di ad e di avere una reportistica dettagliata

Internet of Things

- Gestione real-time dei dati generati da sensori
- Bosch utilizza MongoDB per catturare dati da automobili (sistema di frenata, servosterzo, tergicristalli, ecc.) e da strumenti di manutenzione di velivoli
  - Implementate regole di business che avvisano il pilota in caso di pressione dei freni calata sotto un livello critico, o avvisano l'operaio se uno strumento è utilizzato in maniera impropria
- Technogym utilizza MongoDB per catturare dati dagli attrezzi connessi

DB documentali: quando non usarli

Transazioni complesse e multi-operazione

- A meno di qualche eccezione (e.g., RavenDB), i database documentali non sono adatti per operazioni atomiche cross-documento.

Interrogazioni su strutture aggregate variabili

- Se la struttura degli aggregati cambia continuamente, anche le query devono essere costantemente modificate. Una soluzione potrebbe essere quella di disaggregare i dati (i.e., normalizzarli), ma verrebbe meno il concetto di aggregato.

DB column-family popolari

Cassandra: http://cassandra.apache.org

HBase: https://hbase.apache.org

Hypertable: http://hypertable.org

SimpleDB: https://aws.amazon.com/simpledb

Google BigTable: https://cloud.google.com/bigtable

DB column-family: quando usarli

Compromesso tra espressività e semplicità, sia a livello di modellazione che di interrogazione dei dati

Esempi

- **Log di eventi; CMS, piattaforme di blogging**
  - Similmente ai database documentali, applicazioni diverse possono usare colonne diverse
- **Matrici sparse**
  - Mentre un RDBMS gestisce tutti i *null*, un database colonnare permette di allocare solamente le colonne per cui è necessario memorizzare un dato.
- **Applicazioni GIS**
  - "Pezzi" di mappa possono essere memorizzati come coppie di longitudine (chiave di riga) e latitudine (colonna)

DB column-family: casi d'uso reali

Applicazioni Google

- BigTable è il database di riferimento per molti servizi di Google, tra cui Search, Analytics, Maps e Gmail

Profili utente e preferenze

- Spotify usa Cassandra per gestire tutti i metadati relativi ad utenti, artisti, canzoni, playlist, ecc.

Manhattan

- Dopo aver provato Cassandra, Twitter ha sviluppato un suo DBMS NoSQL proprietario per gestire buona parte dei propri servizi

DB column-family: quando non usarli

Stesse considerazioni dei database documentali

- Transazioni complesse e multi-operazione
- Interrogazioni su strutture aggregate variabili

Necessità di elevata espressività nell’interrogazione dei dati

- I join sono fortemente sconsigliati
- Supporto limitato per filtri e group-by

DB a grafo popolari

Neo4J: http://neo4j.com

InfiniteGraph: http://www.objectivity.com/products/infinitegraph/

OrientDB: http://orientdb.com/orientdb/

FlockDB: http://github.com/twitter-archive/flockdb

DB a grafo: quando usarli

Dati interconnessi

- I social network sono un'applicazione in un cui i database a grafo sono molto efficienti (non solo per memorizzare amicizie, ma, per esempio, per memorizzare relazioni di lavoro); ogni dominio ricco di relazioni è un buon contesto

Servizi di instradamento e location-based

- Applicazioni che affrontano il problema del commesso viaggiatore (TSP, Travelling Salesman Problem)
- Applicazioni location-based per suggerire, ad esempio, i migliori ristoranti nelle vicinanze. In questi casi, le relazioni modellano il concetto di distanza tra i nodi

Applicazioni di recommendation, fraud-detection

- Sistemi che suggeriscono «i prodotti acquistati dai tuoi amici», o «i prodotti acquistati da chi ha acquistato questo prodotto»
- Quando le relazioni sono talmente tante da evidenziare chiari pattern comportamentali, la ricerca di outlier può servire per la rilevazione di frodi

DB a grafo: casi d'uso reali

Analisi delle connessioni

- Trovare di amici/relazioni comuni (friend-of-a-friend) in un social network
- Individuare concentrazioni di telefonate che identificano una rete criminale
- Analizzare flussi di denaro tra conti corrente che riscontrino pattern tipici di riciclaggio di denaro o furti di carte di credito
- Principali utilizzatori: studi legali, forze di polizia, agenzie di intelligence
  - https://neo4j.com/use-cases/fraud-detection/
- Utile anche in attività di analisi del testo (Natural Language Processing)

Inferenza

- Creazione di regole che permettano di estrarre nuova conoscenza in presenza di determinati pattern (e.g., transitività di relazioni, meccanismi di fiducia)

DB a grafo: quando non usarli

Applicazioni data-intensive

- Attraversare il grafo è semplice, ma analizzare tutto il grafo può diventare particolarmente oneroso.
- Esistono framework per l'elaborazione distribuita di grafi (e.g., Apache Giraph), ma non si appoggiano su database a grafo

Persistenza poliglotta

Database diversi sono progettati per risolvere problemi diversi

Utilizzare un singolo DBMS per gestire tutti i requisiti…

- Memorizzare dati operazionali
- Gestire informazioni temporanee di sessione
- Attraversare grafi di clienti
- Effettuare operazioni OLAP
- …

… porta solitamente a soluzioni non efficienti.

Ogni attività ha i suoi requisiti in termini di availability, consistenza, backup, ecc.

La scelta del database

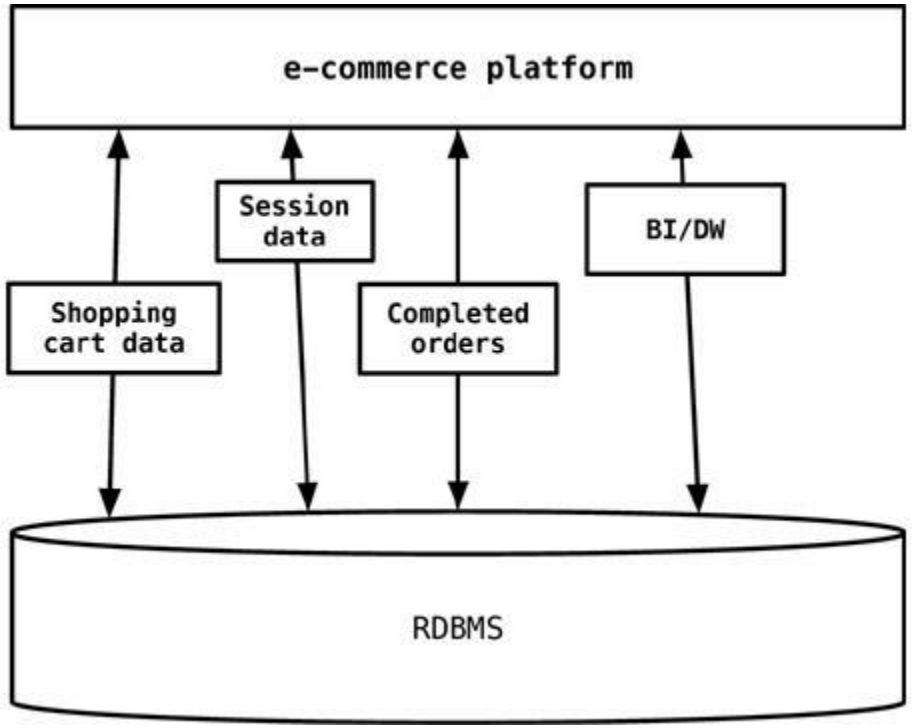
Perché scegliere un database NoSQL invece di un RDBMS?

- Scalabilità!
- Necessità di funzionalità esclusive dei sistemi NoSQL (ad esempio, grafi, strutture innestate, proprietà schemaless)
- Migliorare le performance in casi in cui la consistenza non è un vincolo
- Evitare il problema del conflitto di impedenza
  - Anche se molti altri problemi vengono caricate sulle spalle degli sviluppatori

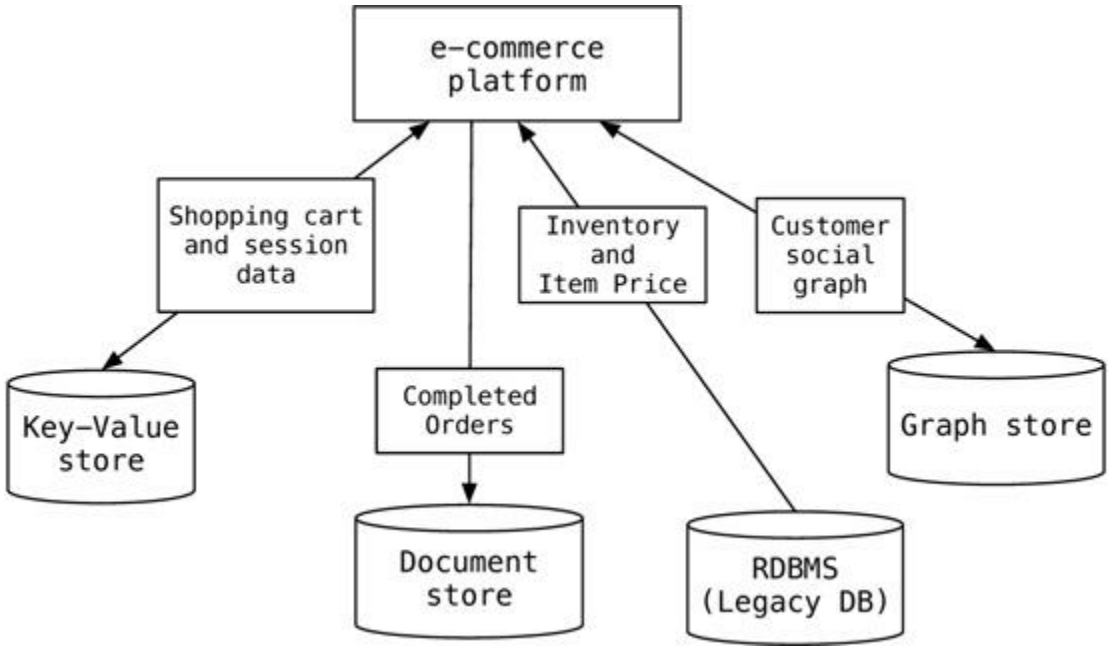
L'ecosistema NoSQL non è maturo come quello relazionale

- Ma è in forte evoluzione
- E' fondamentale verificare le aspettative e i miglioramenti previsti prima di cambiare/estendere il database di riferimento

Approccio tradizionale



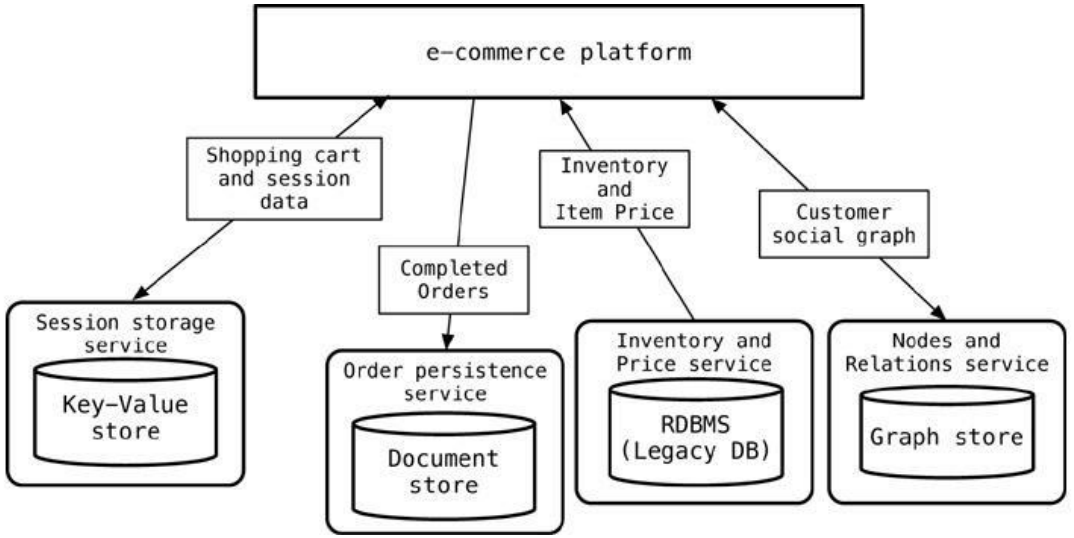
Gestione poliglotta dei dati



Gestione service-oriented

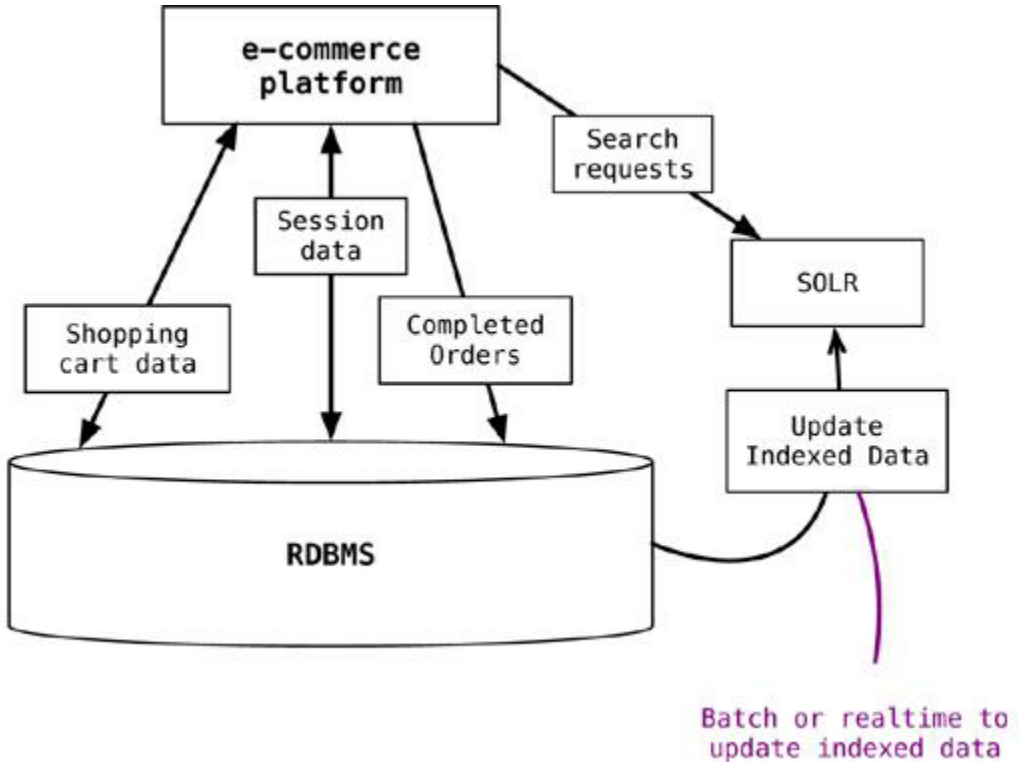
I singoli database possono essere "impacchettati" all'interno di servizi, che mettono i dati a disposizione di più applicazioni attraverso un meccanismo di API

- Diversi prodotti NoSQL (e.g., Riak, Neo4J) forniscono già API REST



Estensione delle tecnologie esistenti

Se non si può cambiare tecnologia, si possono comunque utilizzare le nuove tecnologie a supporto di quelle esistenti



Persistenza poliglotta in azienda

I DBA tendono a diventare sempre più poliglotti

- E' importante: capire come funzionano le tecnologie NoSQL, come monitorare questi sistemi e come sfruttarli al meglio delle loro potenzialità.

Questione sicurezza: verificare la possibilità di creare utenti e assegnare loro determinati privilegi.

- La maggior parte dei DBMS NoSQL **non** offre una robusto sistema di sicurezza.
- La responsabilità viene «scaricata» sulle applicazioni.

Attenzione alle problematiche legate a licenze, supporto, upgrade, driver, auditing, strumenti di supporto, ecc.

- Molti prodotti sono open-source e hanno una comunità attiva di collaboratori; alcune aziende offrono supporto a livello commerciale.
- Alcuni strumenti di supporto stanno venendo progressivamente rilasciati (e.g., MongoDB Monitoring Service, Datastax Ops Center, Rekon browser)
- Gli strumenti di ETL stanno aggiungendo il supporto ai DBMS NoSQL