

NoSql Database: Aggregate DB Data (traduzione da Lembo)

Francesco Pugliese, PhD

neural1977@gmail.com

Database NoSql: DB Aggregati

- ✓ Modelli di dati **NoSQL**
- ✓ **Database** Chiave-Valore, Documenti, Colonna
- ✓ Modelli di Distribuzione
- ✓ Consistenza
- ✓ Map-Reduce

NoSQL: aldilà dei Database a Grafo

- ✓ I Database a Grafo hanno la seguente capacità:
- ✓ forniscono una modellazione **schemaless** dei dati
- ✓ un **trattamento** nativo delle relazioni tra pezzi di informazione
- ✓ Le caratteristiche menzionate li rendono particolarmente adatti alla **gestione di complesse relazioni nei dati**, in particolare in quei contesti in cui le dinamiche del dominio rendono le soluzioni basate su modelli relazionali classici non efficacemente ed efficientemente applicabili (per esempio le connessioni utente in un social network, i sistemi di raccomandazione, le applicazioni geospaziali, ecc.)
- ✓ Abbiamo anche investigato un interessante uso dei database a grafo specificati attraverso l'**RDF W3C** che è uno standard per i dati (e per la conoscenza) condivisi alla scala del web.

NoSQL

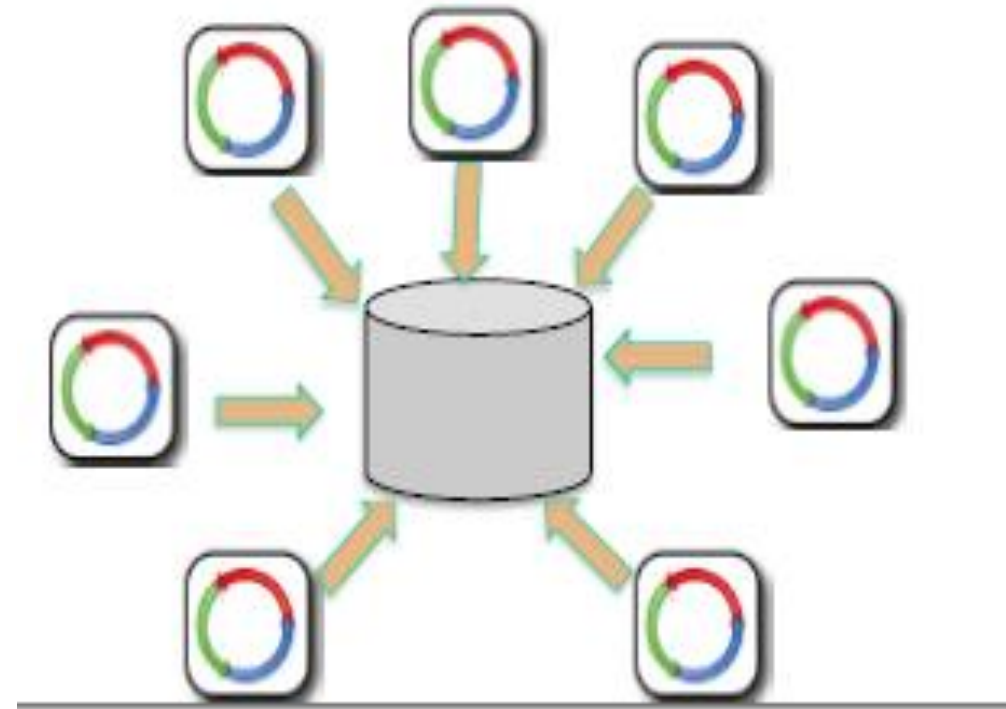
- ✓ I Database a Grafo sono una particolare famiglia di database che possiamo classificare come appartenenti al "**Movimento NoSQL**"
- ✓ Tuttavia, o database a grafo generalmente presentano solo alcune delle caratteristiche che sono tipiche delle soluzioni NoSQL, e che possiamo riassumere come segue (anche se non c'è una definizione generalmente accettata di NoSQL in letteratura):
 - ✓ schemaless
 - ✓ non usano SQL
 - ✓ sono generalmente **open-source** (anche se i NoSQL sono anche applicati ai sistemi cloud)
 - ✓ generalmente lavorano in cluster (anche se i database a grafo non ricadono in questo)
 - ✓ generalmente non gestiscono la consistenza attraverso le **transazioni ACID** (che i DB a grafo invece supportano)

Guardiamo al passato: predominio relazionale

- ✓ I **Database Relazionali** sono stati per decenni la scelta di default dei sistemi di data storage, specialmente per applicazioni enterprise
- ✓ La principale ragione di questo predominio è:
- ✓ Capacità di mantenere i dati in memoria di massa in un modo **strutturato** (esempio file system w.r.t., ossia un tipo di network file system sharato)
- ✓ **Semplicità** nel modello dei dati
- ✓ Gestione della **concorrenza** nell'accesso ai dati
- ✓ Basato su linguaggi **standardizzati**
- ✓ Usato come mezzo per **integrare applicazioni** (integrazione di database)

SQL come meccanismo di integrazione

- ✓ Il **fattore primario** che ha reso i db relazionali maggiormente di successo rispetto agli altri modelli di dati (come ad esempio gli Object Oriented DB) è probabilmente il ruolo giocato da **SQL come meccanismo di integrazione tra applicazioni**
- ✓ In questo scenario, applicazioni multiple immagazzinano i loro dati in un database integrato comune. Questo **migliora la comunicazione** perchè tutte le applicazioni operano su un insieme consistente di dati persistenti



Verso nuovi modelli dei dati

- ✓ Quali sono le ragioni per cui i modelli di dati **NoSQL** hanno iniziato a divenire così popolari?
- ✓ Alcune di queste motivazioni coincidono con le motivazioni che hanno originato lo sviluppo dell'ecosistema Big Data, e di cui abbiamo già discusso
- ✓ Dovrebbero in ogni caso essere enfatizzati i seguenti aspetti:
- ✓ Persone che hanno iniziato a modellare le applicazioni di db
- ✓ Avere a che fare con aggregati, ossia con una porzione rilevante concettualmente di informazione (oggetto, data record), è molto più facile per i nuovi database gestire **operazioni su cluster**, dal momento che il db aggregato rende una unità naturale per la replicazione e lo sharding (distribuzione)
- ✓ Inoltre, il problema della **Impedance Mismatch Problem**, ossia la differenza tra il modello relazionale e le strutture dati in memoria.

Database di Integrazione vs Applicazione

- ✓ **Esistono svantaggi** all'uso della condivisione dell'integrazione di database:
- ✓ Una struttura che sia progettata per integrare molte applicazioni finisce per essere molto complessa
- ✓ I cambiamenti ai dati di differenti applicazioni hanno bisogno di essere coordinati
- ✓ Differenti applicazioni hanno differenti necessità di performance, dunque chiamano differenti strutture ad indice
- ✓ Accesso control policies complesse
- ✓ Un approccio differente è quello di trattare il DB come un **database di applicaizone (application database)**

Application Database

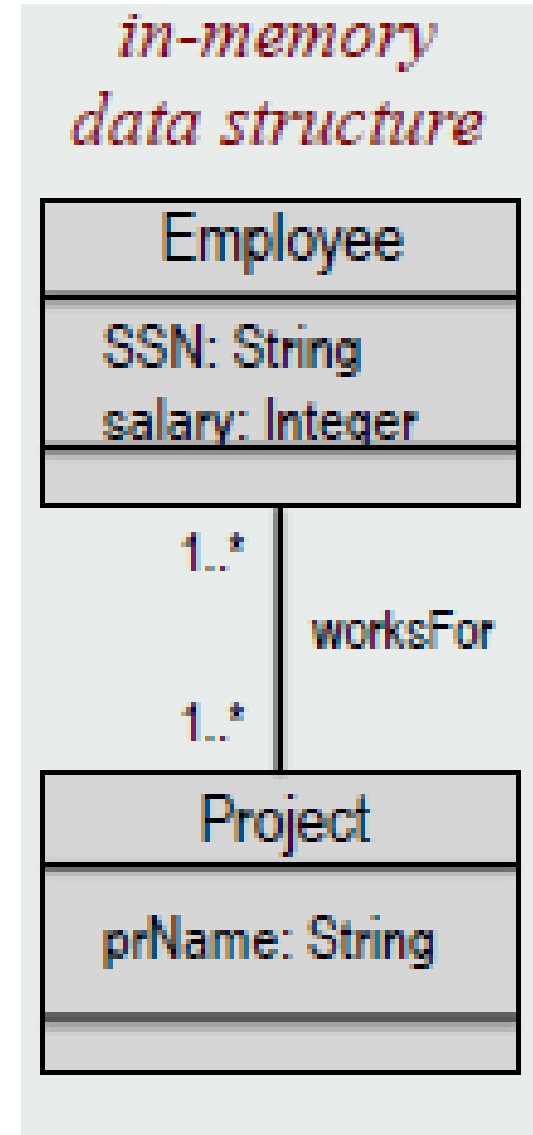
- ✓ Un **Application Database** viene solo direttamente acceduto da una singola applicazione, che lo rende **più facile da mantenere ed evolvere**
- ✓ **L'interoperabilità riguarda il fatto che si può ora passare alle interfacce dell'applicazione:**
 - ✓ Durante il 2000 abbiamo assistito ad un ben distinto spostamento verso i web service, dove le applicazioni **potrebbero comunicare sul HTTP (lavorare su Architetture Service-Orientate)**
- ✓ Se comunichiamo con SQL, i dati devono essere strutturati in relazioni. Tuttavia con un servizio, siamo capaci di usare strutture dati più ricche, possibilmente con dei record innestati e delle liste.
- ✓ Questi dati sono di solito rappresentati come documenti in **XML** o più recentemente **JSON (Javascript Object Notation)**, un formato di interscambio leggero

Impedance Mismatch

- ✓ L' **Impedance Mismatch** è una delle maggiori cause di frustrazione per gli sviluppatori di applicazioni, e nel 1990 molte persone credevano che i database relazionali sarebbero stati sostituiti da database che replicavano le strutture dati nella memoria anche sul disco.
- ✓ Quel decennio fu segnato dalla crescita dei linguaggi di programmazione orientati agli oggetti, e quindi dalla nascita dei **db orientati agli oggetti**.
- ✓ Tuttavia, dopo che OOP ebbe successo nella programmazione i DB orientati agli oggetti non ebbero per nulla successo: i db relazionali rimasero la principale tecnologia per il data storage, essendo altamente consolidati, ben noti, ottimizzati e soprattutto basati su linguaggi standard come SQL.
- ✓ Quindi, l'Impedance rimase un problema, framework di Object-relational mapping come Hibernate o iBatis sono stati proposti per rendere le cose più facili, ma non sono adatti a quegli scenari (frequenti) in cui molte applicazioni fanno affidamento allo stesso database integrato. Anche le performance delle query soffrono in questi framework.

Impedance Mismatch - Esempio

- ✓ Dati correnti immagazzinati in un DB:
- ✓ D2: Codice degli impiegati con salario e SSN
- ✓ D1: Impiegati e Progetti per cui gli impiegati lavorano
- ✓ Concettualmente:
- ✓ Un impiegato è identificato dal suo SSN.
- ✓ Un progetto è identificato dal suo nome.
- ✓ Quindi:
- ✓ Un impiegato dovrebbe essere creato dal suo SSN
- ✓ un progetto dovrebbe essere creato dal suo PrName



Attacco dei Cluster

- ✓ **Il 2000 ha assistito** a parecchie proprietà del web che sono drammaticamente aumentate in grandezza nel tempo!
- ✓ I siti web iniziarono a tracciare l'attività e la struttura in un modo molto dettagliato. Grandi data set sono apparsi: link, social network, attività nei log, dati di mapping. Con la crescita del volume dei dati è aumentato anche il numero degli utenti.
- ✓ Superare i problemi dell'incremento dei dati e del traffico ha richiesto più risorse computazionali.
- ✓ Scalare significa macchine più grandi, più processori, più storage e memoria. Ma macchine più grandi sono più costose ed anno dei limiti concreti nell'incremento delle loro dimensioni, hanno un tetto massimo teorico.

Attacco dei Cluster

- ✓ **L'alternativa a scalare le dimensioni delle macchine è quella di usare un elevato numero di machine all'interno di un cluster.** Un cluster di macchine piccole può usare dell'hardware di basso costo e finire per essere più economico alle varie scale.
- ✓ Inoltre un cluster può anche essere **più resiliente** dal momento che se le macchine singole si danneggiano, l'intero cluster può continuare a funzionare fornendo alta affidabilità oltre che alte performance
- ✓ Mentre le grandi compagnie si sono mosse verso i cluster, questo ha portato ad un nuovo problema: **i db relazionali non sono progettati per essere eseguiti sui cluster!**

Attacco dei Cluster

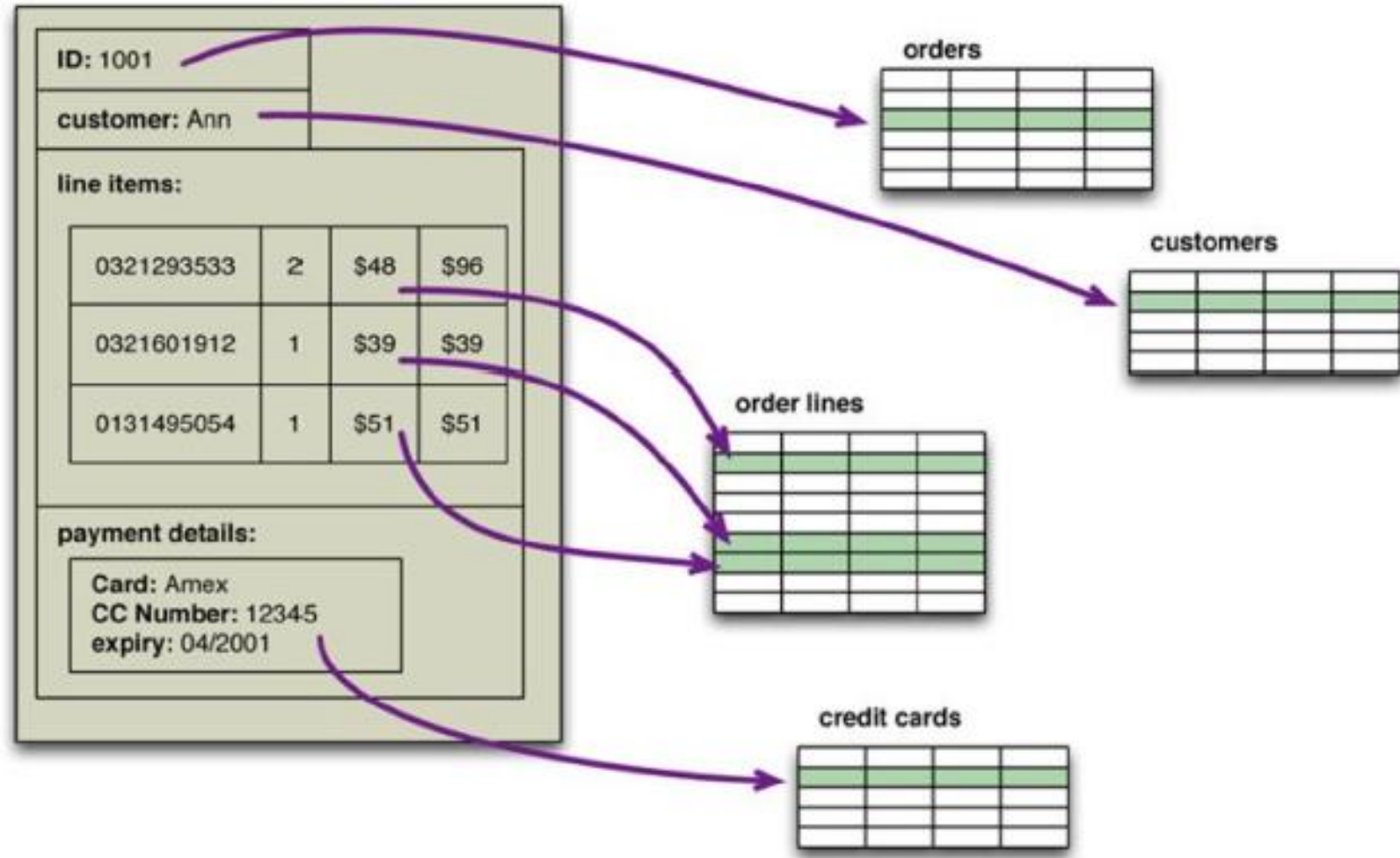
- ✓ **I DB relazionali possono anche essere avviati su server separati** per differenti insiemi di dati, questo è lo **sharding** dei database (ad esempio i dati vengono fisicamente segmentati su vari nodi di storage dei dati)
- ✓ Mentre si separa il caricamento, tutt lo sharding deve essere controllato a livello applicativo che deve tenere traccia di quale db server deve rispondere per comunicare ciascun blocco di dati
- ✓ Inoltre, perdiamo query, integrità referenziale, transazioni o controllo di consistenza con l'uso dello sharding
- ✓ **La decisione della granularità dello sharding** è una questione molto ma molto difficile!

Modelli dei Dati Aggregati

- ✓ **Il modello relazione divide l'informazione** che noi vogliamo storing in tuple (righe): questa è una struttura molto semplice per i dati.
- ✓ **L'orientazione Aggregata** prende una differente direzione nell'approccio. Essa riorganizza ciò che serve per operare sui dati in **unità** che hanno una struttura più complessa.
- ✓ Può essere maneggevole pensare in termini di un record complesso che permette liste e altre strutture di record che vengano innestate all'interno di esso.
- ✓ Tuttavia, non c'è un termine comune per questo tipo di record complesso; secondo il [SaFo13] usiamo il termine **aggregato**.
- ✓ **Aggregato** è un termine che viene dal Domain-Driven-Design (DDD). In DDD, un aggregato è una collezione di oggetti collegati che desideriamo trattare come una unica unità. In particolare, è un'unità che serve alla **data manipulation** e al **management della consistenza**.

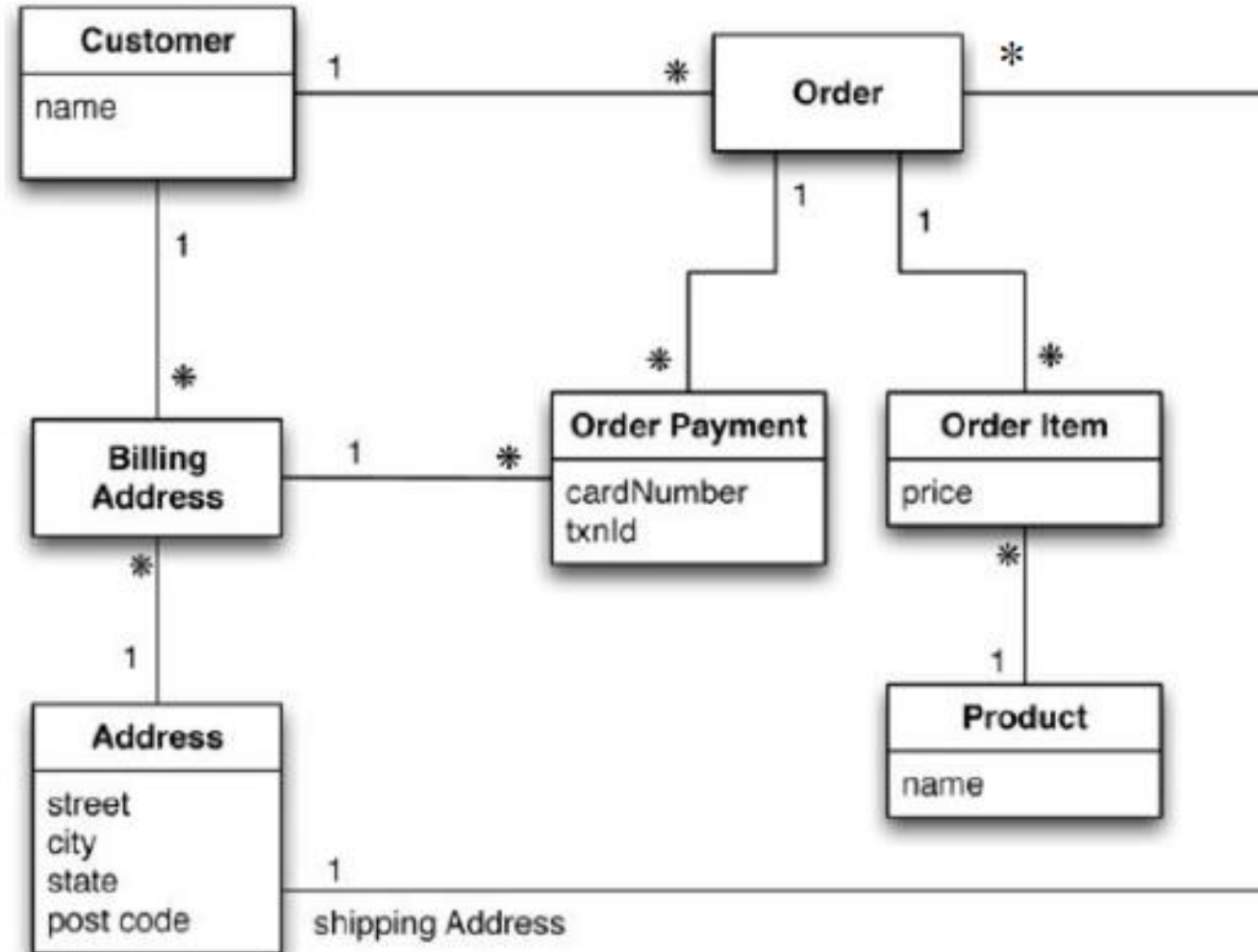
Modelli di Dati Aggregati

- ✓ **Un ordine** il quale ricerca un singolo aggregato:



Esempi di Relazioni e Aggregati

✓ Prospettiva dei DB relazionali: senza aggregati



Esempi di Relazioni e Aggregati

- ✓ **Nota:** per semplicità di presentazione, solo gli attributi interessanti per l'istanza dal lato della relazione **Address** vengono presentati qui. Di default ciascuna tabella relazione usa un Id (che identifica un oggetto).

Customer	
Id	Name
1	Martin

Order		
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

Conseguenze di un'Orientazione Aggregata

- ✓ Il fatto che un ordine sia costituito da item dell'ordine, un indirizzo di spedizione, e un pagamento possono essere espressi in un modello relazionale in termini di relazioni usando una chiave straniera ma **non esiste nulla per distinguere le relazioni che rappresentano aggregazioni da quelle che non le rappresentano**. Come risultato, il database non può usare una conoscenza di una struttura aggregata per aiutarla a immagazzinare e a distribuire i dati.
- ✓ Tuttavia l'**Aggregazione** non è una proprietà dei dati di tipo logico: si riferisce più che altro al modo in cui i dati sono utilizzati dalle applicazioni, un problem che è spesso fuori dai confini del data modelling
- ✓ Inoltre, **una struttura aggregata di dati potrebbe aiutare con alcune interazioni dei dati ma potrebbe essere un ostacolo per altri** (nel nostro esempio, per ottenere la storia delle vendite di un prodotto, è necessario scavare dentro ogni aggregato del database)

Conseguenze di un'Orientazione Aggregata

- ✓ La ragione intrigante per usare **l'orientazione aggregata** è che essa aiuta enormemente **quando si vuole avviare un db su un cluster!**
- ✓ **L'orientazione aggregata** si adatta bene allo scaling del sistema in quanto il db aggregati possono essere usati naturalmente per la distribuzione
- ✓ Dall'altro lato, gli aggregati sottili come lo slicing aggregate per accessi a più grana fine potrebbero essere davvero difficili da implementare.

Conseguenze dell'Orientazione Aggregata

- ✓ Gli aggregati hanno un importante conseguenza per le **transazioni**.
- ✓ I db relazionali permettono di manipolare qualsiasi combinazione di righe da qualsiasi tabella in una singola transazione ACID
- ✓ Si dice spesso che i db **NoSQL non supportano le transazioni ACID** e quindi si trascura la consistenza. Questo tuttavia non è proprio vero per i **DB a Grafo** che sono, come per i relazionali, alieni alle aggregazioni.
- ✓ In generale, è vero che i DB aggregati non hanno transazioni ACID che si estendono agli aggregati multipli. Invece, supportano una manipolazione atomica di un singolo aggregato alla volta: questo significa che se abbiamo bisogno di manipolare aggregati multipli in una modalità atomica, dobbiamo gestirlo da soli a livello di codice.
- ✓ In pratica, troviamo che la maggior parte del tempo dobbiamo tenere i nostri bisogni di atomicità all'interno di un singolo aggregato **che è parte della considerazione di decidere come dividere i nostri dati in aggregati.**

Modelli dei Dati Aggregati

✓ Si possono considerare tre differenti **Modelli Dei Dati Aggregati**:

1. Chiave - Valore

2. Documento

3. Colonna - Famiglia

Bibliografia
