Basi Di DATI II

LEZIONE 13. *NoSQL*

a.a 2017/2018

Prof.ssa G. Tortora

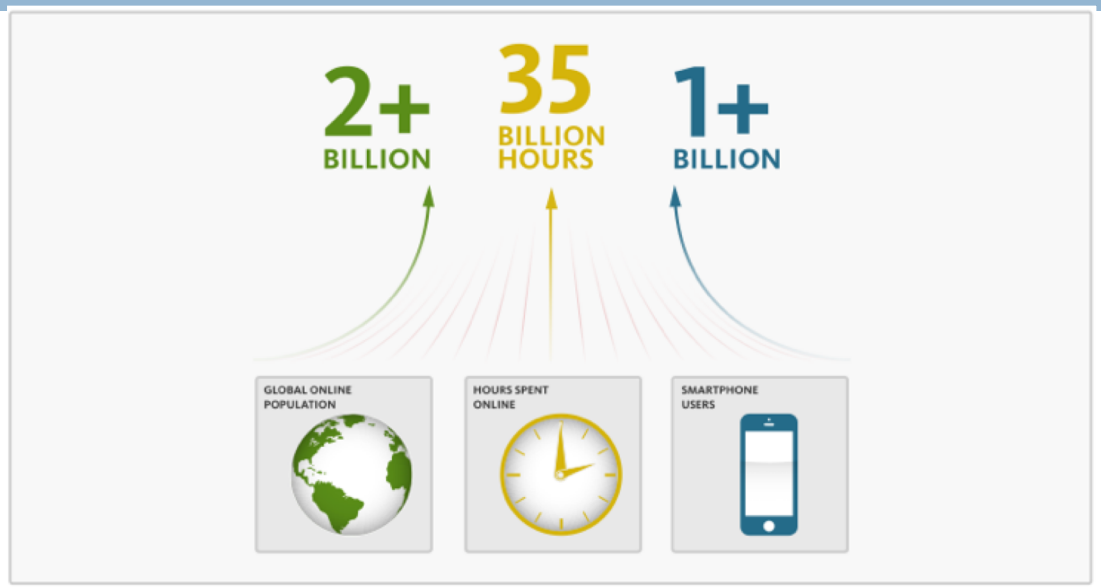
WHY RDBMS?

RDBMS - Motivazioni

- • Schema predefinito per lo storage di dati strutturati
- • Struttura BCNF già familiare
- • Strong consistency
- • Transazioni
- • Maturi e accuratamente testati (la maggior parte)
- • Facile adozione/integrazione
- • Basati sulle proprietà ACID
- • Data Retrieval: Standard Query Language (SQL) - versatile e potente
- • Scalabilità verticale: se volessimo rendere un DB SQL scalabile, l'unica
- alternativa sarebbe quella di potenziare l'hardware sul quale il DBMS è installato

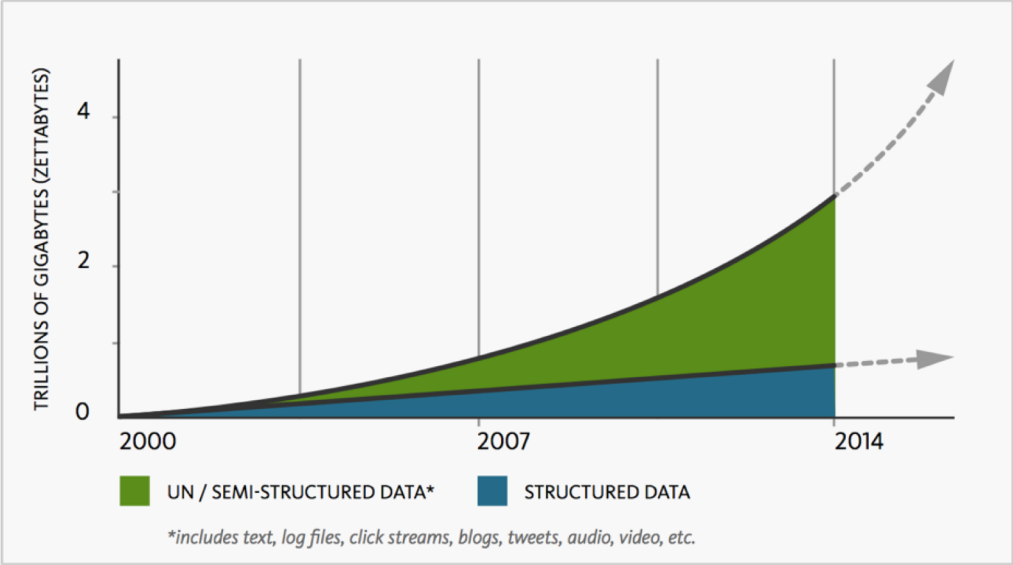
WHY NOSQL?

Introduzione - BIG USERS



Un gran numero di utenti combinato con una natura fortemente dinamica dei pattern di gestione dei dati sta guidando il bisogno di nuovi database scalabili. Con le tecnologie relazionali, molti sviluppatori trovano difficile raggiungere la scalabilità dinamica richiesta dalle applicazioni per garantire il livello di performance richiesto dagli utenti…
… molti si stanno rivolgendo ai NoSQL!

Introduzione - BIG DATA



L'ammontare dei dati è in rapida crescita e la natura di questi dati sta cambiando in conseguenza all'utilizzo che gli sviluppatori fanno di nuovi tipi di dati (per la maggior parte non strutturati o semi strutturati) nelle loro applicazioni

Introduzione – The Internet of Things

- Ci sono 14 milioni di “cose” connesse ad Internet. Si tratta di fabbriche, aziende agricole, ospedali e warehouses. Ci si connette da casa o dall’auto. Ci si connette dallo smarthphone e dal tablet. Si ricevono continuamente dati su ambiente, posizioni geografiche, spostamenti, temperature, meteo da oltre 50 bilioni di sensori
- La chiave è l’accesso globale real-time!
- I dati relativi alle telemetrie sono di piccole dimensioni, semi-strutturati e continui.
- Si tratta di una grossa sfida per i database relazionali i quali richiedono dati strutturati e con uno schema rigido
- I NoSQL raccolgono questa sfida! Sempre più aziende innovative si affidano a questi database richiedendo una tecnologia per sia in grado di scalare con i milioni di “cose” connesse ad Internet

Caratteristiche: Parole chiave NoSQL

- **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"
- **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

Caratteristiche

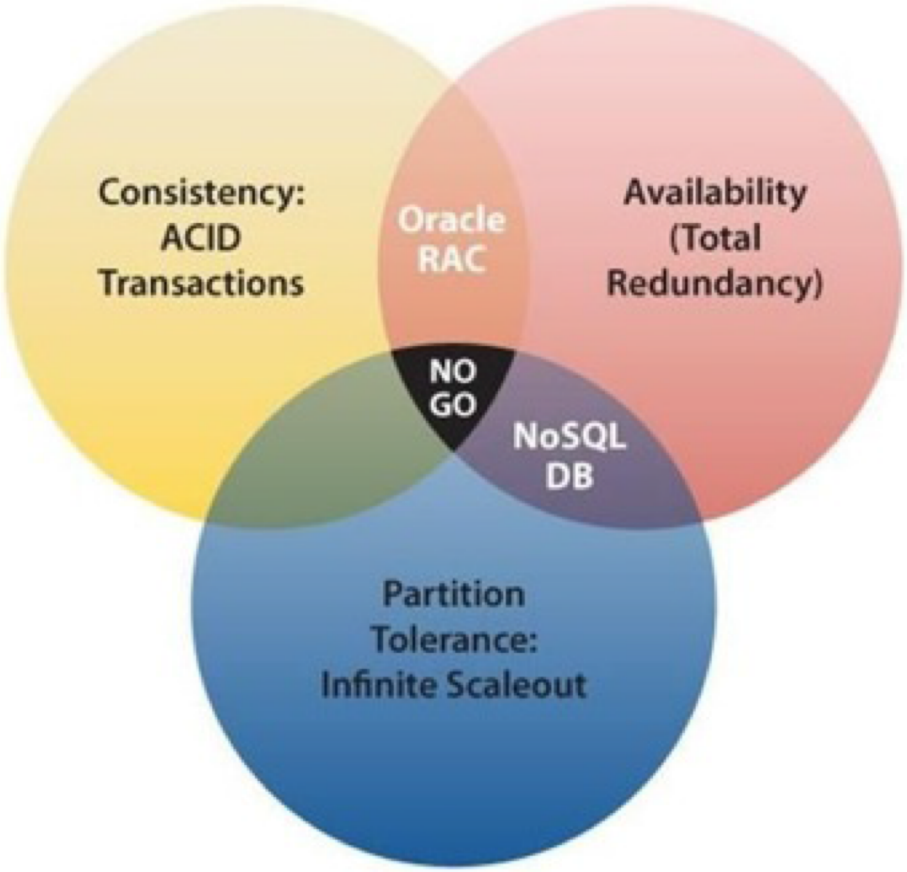
- Grossi volumi di dati
- Goodbye DBAs: un sistema RDBMS di alto livello può essere mantenuto solamente con l'assistenza di amministratori altamente esperti (e costosi). I Database NoSQL sono generalmente ideati per richiedere una minore manutenzione.
- Velocità di risposta alle query!
- BASE - le proprietà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

Transazioni BASE

- I NoSQL si basano su un modello più morbido e flessibile. Il modello BASE si sposa con la flessibilità offerta dai NoSQL e da approcci similari per il management dei dati non strutturati
- **Basically Available**: l'approccio dei database NoSQL si basa sul garantire la disponibilità dei dati anche in pre- senza di fallimenti multipli. Questo obiettivo è raggiunto attraverso un approccio fortemente distribuito
- **Soft state**: i database BASE abbandonano il requisito della consistenza dei modelli ACID quasi completamente. La consistenza dei dati è un problema dello sviluppatore e non deve essere gestita dal database.
- **Eventually Consistent**: l'unico requisito riguardante la consistenza è garantire che in un certo punto nel futuro i dati possano convergere ad uno stato consistente

Brewer's CAP Theorem

- Un sistema distribuito è in grado di supportare solamente 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



Challenges

- Troppo entusiasmo…?
- **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 CIO invece, la maturità di un RDBMS è rassicurate
- **Supporto**: tutte le aziende che sviluppato e vendono RDBMS forniscono un alto livello di supporto alle aziende. I sistemi NoSQL invece sono spesso progetti open source
- **Business Intelligence**: la business intelligence è un elemento chiave per le aziende IT; spesso i tools per la BI non supportano connettività con i database NoSQL
- **Amministrazione**: un obiettivo di design per i database 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 amministrazione RDBMS che non un esperto in NoSQL

Tipologie di Database 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**.

- I principali database NoSQL utilizzano paradigmi anche abbastanza differenti l'uno dall'altro, che ovviamente ne condizionano la progettazione, la manutenzione e l'interfacciamento.

- Analizziamo nel seguito le principali tipologie di database NoSQL attualmente disponibili, in modo da evidenziarne le caratteristiche più distintive nonchè i casi d'uso che ad esse meglio si addicono.

Classificare i DBMS NoSQL

- Le categorie che analizzeremo sono tre, ormai molto diffuse nella pratica comune:
- **Document data store**
- **key/value data store**
- **Graph-based data store**
- **Column-oriented data store**

Document data store (1)

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

Document data store (2)

- 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.
- Ad es.: 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 (3)

- 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 database NoSQL sono particolarmente adatti a rappresentare delle gerarchie.

Key-value data store (1)

- 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 (2)

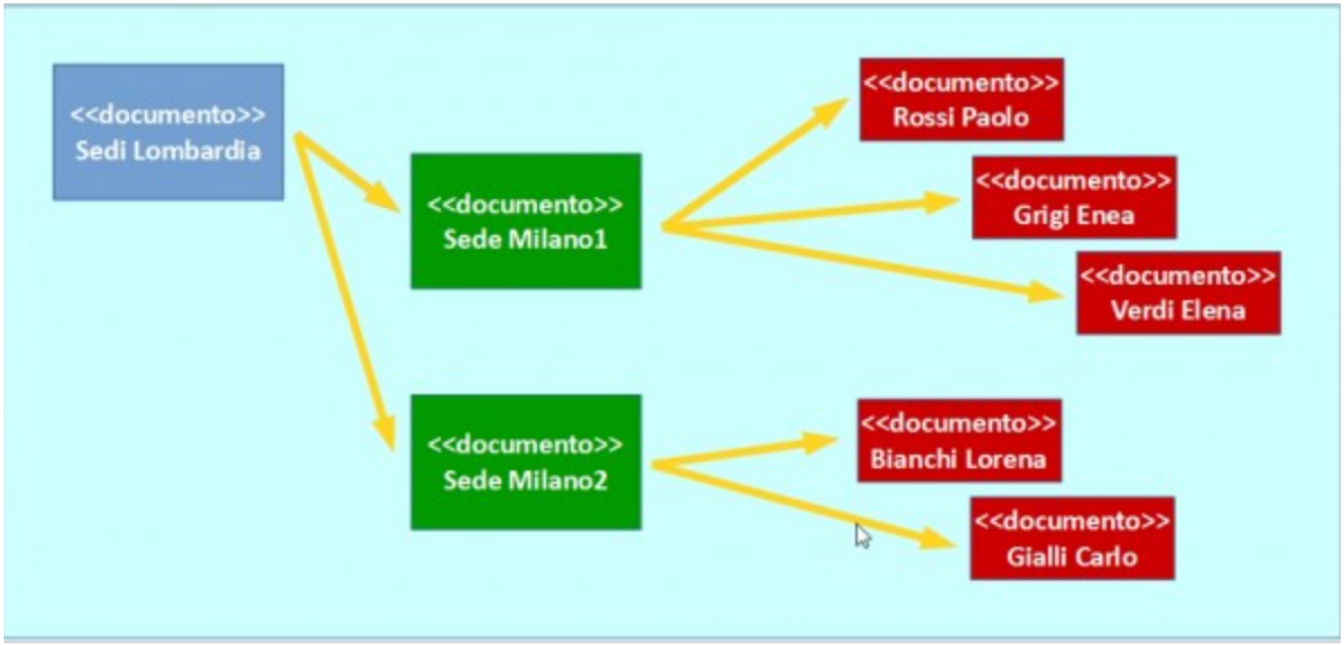
- Costruiti come *dizionari* o *mappe*, in cui vengono inseriti due valori alla volta,
- una chiave (identificatore che permette l'estrazione rapida del valore)
- il valore ad essa associato (l'informazione vera e propria)
- Il loro utilizzo principale è quello di offrire la possibilità di effettuare ricerche rapide di singoli blocchi di informazione.

- 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.

Key-value data store (3)

- Esempio: struttura di un'azienda (dotata di infrastrutture e personale) dislocata sul territorio nazionale.
- Ad ogni nodo corrisponde un elenco di sedi specifiche, ognuna caratterizzata da ulteriori dettagli e da una lista di documenti, contenenti i dati di ogni dipendente.
- Tutti i rettangoli costituiscono documenti



Key-value data store (4)

- La rapidità di interazione – oltre alla struttura a dizionario – rende tali database ideali per salvare, ad esempio, dati di sessione in ambiente client/server, dove la chiave rappresente l'ID di sessione, mentre i dati sono memorizzati come valori;

Graph-based data store (1)

- 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 (2)

- Le informazioni possono essere custodite sia nei nodi sia negli archi.
- La forza di questa tipologia 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)
- Esempio: i Social Network: la navigazione del grafo consente di proporre ad un utente nuove potenziali conoscenze recuperando gli "amici di amici".

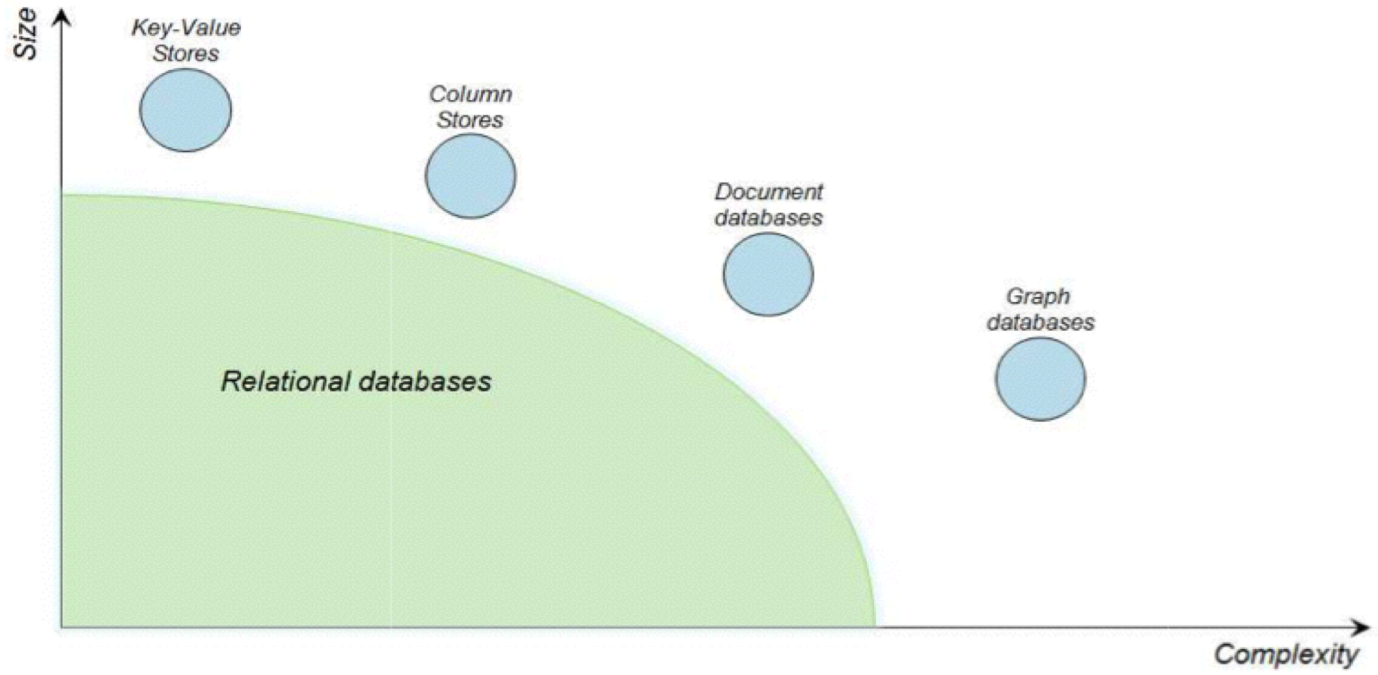
Column-oriented data store (1)

- 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

Database Column-oriented

- **Column-oriented**: sono database che immagazzinano dati in sezioni di colonne.
- La rapida 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.

Classificazione



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

Introduzione a MongoDB

- MongoDB è un **database NoSQL orientato ai documenti**, che nasce nel 2007 in California come servizio da utilizzare nell'ambito di un progetto più ampio, ma che presto è diventato un prodotto indipendente ed open-source.
- Memorizza i documenti in **JSON**, formato basato su JavaScript e più semplice di XML, ma comunque dotato di una buona espressività.
- Ma: cosa significa avere a che fare con un database NoSQL, e qual è la differenza tra questa categoria di basi di dati, ed i più tradizionali database SQL?

SQL vs NoSQL (1)

- **SQL** (*Structured Query Language):* linguaggio di programmazione usato per l'interrogazione e la gestione dei **database relazionali** (RDBMS): pertanto database relazionali = database SQL
- Basati sui concetti di:

- **Tabelle (con schema fisso) e campi**
- Relazioni e **vincoli di integrità referenziale**
- Transazione con le **proprietà ACID**

SQL vs NoSQL (2)

- **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 delle caratteristiche precedenti.
- Questo significa che 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.
- Vediamo quali sono i più importanti modelli logici alternativi al modello relazionale:

SQL vs NoSQL (3)

- Vediamo quali sono i più importanti modelli logici alternativi al modello relazionale:
- i **database orientati ai documenti** memorizzano i dati in documenti, ossia in oggetti complessi codificati in qualche modo e senza uno schema rigido. MongoDB è basato proprio su questo modello;
- i database **a grafo** usano strutture a grafo con relazioni libere (non prefissate come nel caso dei database relazioni) tra nodi del grafo. Un esempio che utilizza questo schema è **Neo4j**;
- i database chiave-valore utilizzano il modello dell'**array associativo** (**Memcached** è un esempio);
- altri modelli più complessi come **HBase** o **Cassandra** fanno corrispondere, alle chiavi, valori formati da famiglie di colonne anch'esse indicizzate.

SQL vs NoSQL (4)

- Per quanto riguarda le proprietà ACID, alcuni database NoSQL ne garantiscono solo alcune.
- Per esempio 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.
- Altri non garantiscono la **durabilità**: ad esempio, 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: Vantaggi e svantaggi

- Se implementiamo un'applicazione che gestirà molte relazioni tra oggetti di vario tipo, un modello logico di dati come quello a grafi può rendere l'applicazione molto più semplice da sviluppare.
- Un noto problema di quando si sviluppa con linguaggi orientati ad oggetti è, infatti, il cosiddetto **O/R impedance mismatch**, ossia il fatto che i due modelli, relazionale e ad oggetti, sono molto diversi tra loro. Ciò può generare problematiche che vanno dalla gestione del polimorfismo alla conversione dei tipi di dati.
- Tra gli svantaggi più significativi dei database NoSQL possiamo riportare la **carenza di tool**: per una tecnologia consolidata da molto tempo come SQL, esistono tantissimi strumenti di gestione e sviluppo, a differenza di quanto accade per la maggior parte dei database NoSQL.

MongoDB: caratteristiche di base (1)

- Come già detto, MongoDB è un **database orientato ai documenti**, ognuno dei quali è memorizzato nel formato JSON. Il documento è fondamentalmente un albero che può contenere molti dati, anche annidati. Un primo esempio è il seguente:

```
nome: "Dante", cognome: "Alighieri",
nato: 1265,  morto: 1321,
lingue: ["italiano", "latino" ],
opere: [
  { titolo: "Divina Commedia",
    iniziata: 1300,
    lingua: "italiano",
    tipo: "poesia",
    versi: "endecasillabi",
    libri: ["Inferno", "Purgatorio", "Paradiso"] }
]
}
```

MongoDB: caratteristiche di base (2)

- I documenti sono raggruppati in **collezioni** che possono essere anche eterogenee:
- *non c'è uno schema fisso per i documenti*.
- Tra le collezioni non ci sono relazioni o legami garantiti da MongoDB: in altri termini, *non esiste un concetto analogo al vincolo di integrità referenziale*.
- Modello logico a parte, le caratteristiche chiave di MongoDB sono:
- consente servizi di **alta disponibilità**, dal momento che la replicazione di un database (**Replica Set**) può avvenire in modo molto semplice;
- garantisce la **scalabilità automatica**, ossia la possibilità di distribuire (**Sharding**) le collezioni in cluster di nodi, in modo da supportare grandi quantità di dati senza influire pesantemente sulle performance.
- MongoDB si adatta a molti contesti, in generale quando si manipolano grandi quantità di **dati eterogenei e senza uno schema**. Non è invece opportuno quando si devono gestire molte relazioni tra oggetti, e si vuole garantire l'integrità referenziale tra essi (ad esempio in un ERP).

Vantaggi degli approcci NoSQL (1)

- **strutturazione dei dati**: va oltre le rigide strutture dell'approccio rtelazionale e dà maggiore **centralità alle informazioni e alla loro varietà**, supportando l'uso di dati non omogenei pur mantenendo le possibilità di interrogazione, analisi ed elaborazione efficiente;
- **scalabilità**: a differenza dei database relazionali, quelli di tipo NoSQL sono generalmente basati su strutture fisiche che si prestano meglio alla distribuzione dei dati su più nodi di una rete (**sharding**), permettendone pertanto un'espandibilità maggiore;

Vantaggi degli approcci NoSQL (2)

- **prestazioni**: la maggiore distribuibilità dei dati sulle reti permette migliori performance. Infatti la **replica** dei dati su più macchine le mette in condizione di rispondere più rapidamente alle richieste degli utenti. (Non è un caso che la crescita del movimento NoSQL sia coincisa con la diffusione dei Social Network, caratterizzati da grandi moli di dati ma anche da un enorme numero di accessi simultanei;
- **flessibilità nella progettazione**: la struttura flessibile usata nei database NoSQL non obbliga necessariamente ad una stereotipazione dei dati durante la progettazione, ma lascia liberi i programmatori di risolvere eventuali casi particolari direttamente in fase di sviluppo dell'applicazione.

OrientDB: cos'è e dove si colloca (1)

- **OrientDB**, prodotto e sviluppato da Orient Technologies: database **multi-model**, (modello "ibrido" con varie funzionalità NoSQL).
  - **Object Model**, permette di definire classi di dati, supporta ereditarietà e polimorfismo.
- Il **Document Model** permette di raccogliere i documenti in strutture molto flessibili (le collezioni) legate tra loro da link.
- Il **Graph Model** raccoglie documenti in nodi collegati tra loro da relazioni. La rete di interconnessioni così creata permetterà di estrapolare ulteriori informazioni utili.
- il **Key/Value Model**, consente di immagazzinare i dati in strutture indicizzate tramite chiavi.

OrientDB: cos'è e dove si colloca (2)

- A differenza di quanto avviene in altri gestori di database, OrientDB è **realmente Multi-Model**, nel senso che i vari approcci appena descritti non sono solo delle interfacce offerte dal sistema, ma quattro veri modelli di gestione dei dati di cui il motore del DBMS è stato dotato.
- La varietà di usi cui si presta OrientDB è proprio uno dei fattori che ne ha determinato la rapida diffusione. Tra i vari paradigmi NoSQL non ne esiste uno migliore degli altri, ma alcuni di essi possono essere particolarmente adatti ad alcuni tipi di progetti. A seconda delle circostanze quindi si possono sfruttare i vantaggi di un approccio piuttosto che degli altri.

Quando il NoSQL 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;
- è necessario interagire molto frequentemente con il database, volendo evitare conversioni onerose tra record e oggetti, nè tanto meno ricorrere all'utilizzo di O/RM o altre librerie specifiche;
- 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 relazioni consolidati da decenni, supportati da una gran varietà di strumenti. Inoltre, la struttura rigida che richiede non è necessariamente un problema, e talvolta può costituire un vantaggio (soprattutto se si devono modellare dati molto strutturati).
- In tali contesti, la duttilità del NoSQL non appare un requisito fondamentale. Inoltre non è da dimenticare che le 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 introdotti in PostgreSQL

Progettare database a documenti

- I **database a documenti**, per certi versi, ricordano il modello degli oggetti, che domina gli scenari dello sviluppo software.
- Questo modello consente di strutturare le informazioni in aggregazioni di dati, dette appunto *documenti*, dalla forte valenza descrittiva: a seconda delle problematiche da gestire avremo documenti che rappresentano utenti di un sistema, libri di una biblioteca, quotazione di titoli finanziari e quant'altro. I documenti sono disposti in strutture dati lineari dette *collection* o *collezioni*, e potranno essere collegati tra loro mediante riferimenti.
- Il progettista viene pertanto messo in condizione di modellare il sistema informativo creando gerarchie di informazioni congruenti con le esigenze specifiche.

Modellazione del sistema informativo a piccoli passi (1)

- Progettazione:
- 1) **individuazione delle possibili entità.**
- 2) nei limiti del prevedibile, **selezionare un insieme di proprietà** che dovrebbero apparire in questi documenti.
- Queste prime attività non sono obbligatorie, in quanto Mongo DB e molte altre realtà NoSQL non richiedono una definizione a priori della struttura interna dei documenti. Ciò risulta comunque utile a livello progettuale soprattutto quando si lavora in gruppo e si vuole cercare di avere un'idea di massima sulla varietà di informazioni che si dovranno gestire.
- Poichè, su MongoDB, i documenti sono rappresentati in BSON (un formato binario derivato e molto simile a JSON), utilizzeremo questo stesso formato nel seguito di questa lezione. Ogni documento, inoltre, sarà identificato da un valore univoco, detto *ObjectID*.
- BSON permette anche che le proprietà di un documento abbiano come valore un altro oggetto BSON o un array di oggetti BSON. Ciò è molto importante in quanto conferisce "profondità" al documento e pone la necessità, a questo punto, di affrontare un nuovo step nella progettazione:
- 3) **individuazione dei documenti da innestare in altri**. Molto spesso, infatti, capita di memorizzare *ObjectID* o oggetti *embedded* all'interno di un documento, e dover recuperare informazioni tramite questi.
- Un nodo embedded permette di scrivere un insieme di dati tra loro connessi all'interno di un altro documento:

Modellazione del sistema informativo a piccoli passi (2)


```
{
    "_id" : ObjectId("57bdd4a1eb426816499fb1be"),
    "Titolo" : "Promessi Sposi",
    "Autore" : "Alessandro Manzoni",
    "Pubblicazione" : {
        "Editore" : "PremiateEdizioni",
        "Anno_pubblicazione" : 2012,
        "Numero_pagine" : 732
    }
}
```

Modellazione del sistema informativo a piccoli passi (3)

- Nell'esempio precedente, abbiamo incluso in un unico documento che rappresenta un libro, anche tutte le informazioni sulla sua pubblicazione fisica, raccolte all'interno di un ulteriore oggetto innestato.
- Questa pratica risulta comoda per motivi di efficienza, in quanto con il recupero del documento-libro avremo implicitamente il recupero delle informazioni relative alla pubblicazione.
- Tuttavia, ciò potrebbe comportare la presenza di oggetti innestati simili in documenti diversi, con conseguente ridondanza dei dati – il male da cui i database relazionali hanno sempre tentato di fuggire. Ciò può risultare comunque accettabile in molti casi, perciò talvolta risulta una soluzione più che valida.

Considerazioni finali

- MongoDB è fortemente orientato all'interazione con le applicazioni, e non contempla alcune funzionalità come il controllo sui valori inseriti e la loro validazione, e la stessa navigazione attraverso gli *ObjectId* può essere organizzata in maniera efficiente attraverso il codice dell'applicazione che sfrutterà il database.
- Anche nella progettazione del database dovremo sempre considerare cosa vogliamo che sia insito nella struttura del sistema informativo, e cosa dovrà invece essere risolto a livello applicativo. Nella prossima lezione, metteremo in pratica i concetti visti sinora.

Progettare database KeyValue

- I database della famiglia Key Value sono ispirati alla struttura dati di dizionario.
- In pratica, questi database non possiedono strutture “classiche” come le tabelle, ma tutto ciò che vi viene archiviato dovrà essere fornito sotto forma di coppia chiave/valore: il valore è l’informazione vera e propria, la chiave è ciò che ne consente il recupero in fase di ricerca.
- Un database key/value può essere considerato come un’unica grande mappa (o dizionario). Illustreremo alcuni principi che saranno utili in fase di progettazione e ne esemplificheremo l’impiego mediante il DBMS Key/Value più diffuso: **Redis**.

Chiavi e valori (1)

- I database di questo tipo sono stati spesso criticati perchè troppo di nicchia, usati ad esempio per svolgere operazioni di cache in cui i dati sono salvati associandoli ad una chiave per il loro ripetuto impiego, senza doverli recuperare nuovamente alla fonte.
- Vengono spesso chiamati in causa per rappresentare cache o immagazzinamento di dati di sessione: tutto ciò è inoltre incoraggiato dalle alte prestazioni offerte da DBMS come Redis.
- Database di questo genere si prestano a moltissimi ulteriori scenari applicativi: ogni tabella di un database relazionale vive di una struttura Key/Value dove la “key” è costituita dalla chiave primaria, mentre il “value” è rappresentato dagli altri campi della stessa riga. Sotto questo aspetto, Redis e simili non sono poi così diversi dalle classiche tabelle relazionali.

Chiavi e valori (2)

- In un database key/value, **la chiave è l'elemento centrale della memorizzazione**: una scelta corretta delle chiavi sia come simbologia che come formato può essere la carta vincente nella progettazione di un database. Le chiavi in Redis sono *binary safe*, e pertanto potranno essere delle comuni sequenze alfanumeriche o il contenuto di un file non testuale, sebbene chiavi di dimensioni eccessive rischiano di penalizzare le prestazioni in fase di ricerca. È comunque consigliabile definire chiavi "leggibili", che abbiano un formato significativo. Buona norma prevedere di anteporre un prefisso seguito da un separatore (tipicamente per questo si usano i due punti ':').

Esempio (1)

- Modellare il sistema informativo di una biblioteca: inserimento dei dati di un utente usando come chiave il suo numero di tessera, e quelli di un libro mediante il numero di inventario: *users:12346* e *books:567890*. Entrambe le chiavi faranno parte della stessa struttura dati – che costituisce il database – ma il loro prefisso li classificherà su due domini diversi.
- Esempio: rappresentiamo un record della tabella Users, definita in un database relazionale, strutturato come segue:

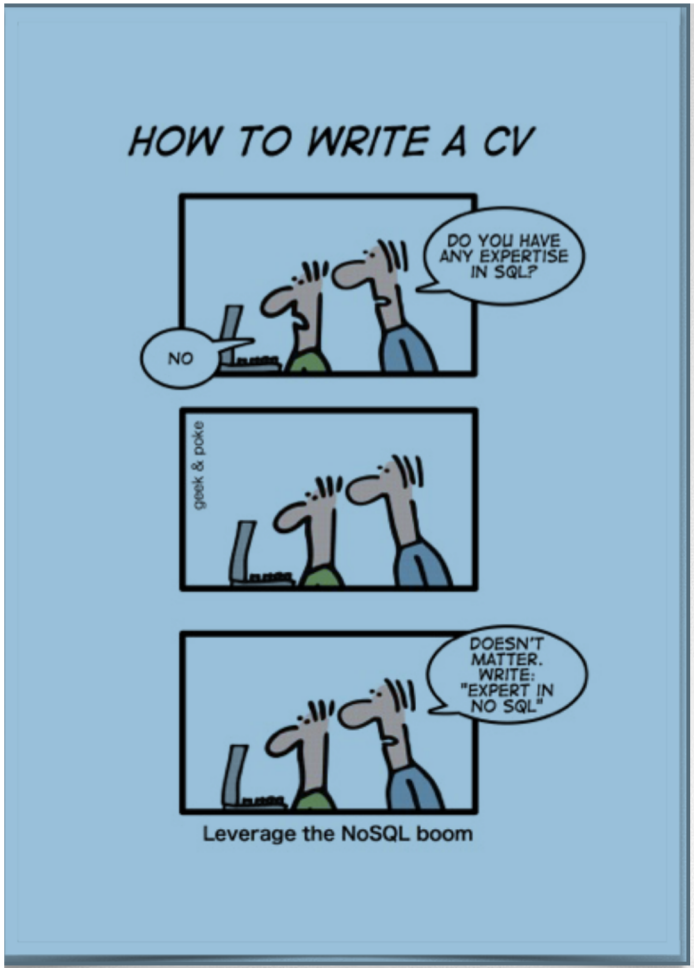
Esempio (2)

id	nome	cognome	indirizzo	città	CAP
123456	Giulio	Rossi	Via Monte Bianco 78	Torino	10100

Chiave	Valore

Users:123456	HashMap
	{
	“nome”:”Giulio”
	“cognome”:”Rossi”
	“indirizzo”:”Via Monte Bianco 78”
	“città”:”Torino”
	“cap”:”10100”
	}

The end


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom