Data Management

Indice

1 Modelli di dati. **2**
1.1 SQL. 2
1.2 NoSQL. 3
1.2.1 Document Based: MongoDB[1, 2]. 4
1.2.2 GraphDB: Neo4J[4]. 6

1 Modelli di dati.

Il *data modelling* è uno strumento per descrivere parte del mondo reale, permettendo di immagazzinare dati in un database ed effettuare interrogazioni sfruttando un linguaggio creato ad hoc. Il linguaggio di interrogazione deve essere comprensibile sia alla macchina che all'uomo, espressivo, semplice, flessibile e, se possibile, deve seguire degli standard. Il modello di dati più famoso è quello relazionale (rappresentato da SQL).

I modelli di dati seguono il Teorema di CAP, introdotto da Eric Brewer nel 2000:

- Consistency: tutti i nodi vedono gli stessi dati allo stesso istante;
- Availability: ogni nodo deve rispondere alle richieste rivoltegli;
- Partition Tolerance: il sistema funziona anche con dati frammentanti su una rete di calcolatori.

È impossibile per un modello soddisfare contemporaneamente tutte e tre le caratteristiche[3]. Il modello relazionale segue le leggi CA, mentre i modelli NoSQL sono CP o AP (dipende dal modello). Si tende a preferire la disponibilità alla coerenza per ottenere dati non aggiornati piuttosto che messaggi di errore.

1.1 SQL.

Nato negli anni '70, il modello relazionale è ben sviluppato e conosciuto e gode di un'ottima reputazione: è stato lo standard *de facto* per anni ed è ancora considerato il migliore per garantire l'integrità dei dati (permessa dal suo schema rigido). Il modello relazionale segue quattro proprietà rappresentate dalla sigla ACID:

- Atomicity: l'operazione è *atomica*, ovvero o avviene per intero (`COMMIT`) o restituisce un errore e il database ritorna alla forma iniziare (`ROLLBACK`);
- Consistency: i nuovi dati inseriti rispecchiano lo schema prestabilito (o l'operazione di inserimento fallisce);
- Isolation: le singole operazioni non influiscono sulle altre; per fare questo il database costruisce una coda di esecuzione dei processi, così lo stato del database non muta durante l'esecuzione di una singola richiesta;
- Durability: la persistenza del file è garantita (virtualmente a tempo indefinito) anche in caso di crash del sistema; per ottenere questo risultato sono utilizzati backup e log files.

Il modello relazionale tuttavia è fortemente influenzato dalla tecnologia degli anni in cui è nato: permette infatti di archiviare la maggior quantità di informazione occupando il minor spazio possibile su disco (allora gli hard disk erano costosi e ingombranti). Nel tempo dunque si sono notati difetti nel modello:

- un attributo può avere solo un tipo di valore;
- SQL non è compatibile con i moderni linguaggi di programmazione ad oggetti;

- la struttura del modello è molto rigida;
- non permette loop nei dati;
- la modifica di tabelle esistenti è difficile e dispendiosa.

Negli anni dunque si sono cercati modelli alternativi. Nei RDBMS la performance (cioè la velocità di esecuzione) dipende da vari fattori:

- numero delle righe;
- tipo di operazione;
- algoritmo scelto;
- struttura dei dati.

Inoltre il modello relazionale rende difficile scalare l'hardware: è stato pensato per poter girare su un'unica macchina fisica[1].

1.2 NoSQL.

Per risolvere i problemi dei database relazionali, nasce un movimento informatico chiamato NoSQL (*Not Only SQL*) che non rifiuta il modello ma propone approcci alternativi: i database NoSQL non hanno un modello prestabilito rendendo facile archiviare dati non strutturati e permettendo di aggiungere attributi senza modificare l'intero modello. Mentre il modello relazionale si basa sull'assunzione del *mondo chiuso*[2] (se il valore di verità non è noto, si considera la proposizione falsa), i modelli NoSQL adottano il modello del *mondo aperto*[3] secondo il quale non si può stabilire il valore di verità di cosa non è noto.

È possibile *scalare* (*scaling*) un DBMS potenziando l'hardware su cui è ospitato. Si distingue lo *Scaling Up*, il potenziare la singola macchina, dallo *Scaling out*, aggiungere macchine in una rete di calcolatori; quest ultimo caso è molto difficile da effettuare su un database di tipo relazionale. Il costo è un altro dei problemi: l'acquisto di macchine di una certa potenza è molto alto e non garantisce un aumento lineare delle prestazioni, che anzi scendono asintoticamente.

Opponendosi alla rigidità del modello relazionale, le proprietà dei database NoSQL non potevano che essere riassunte dall'acronimo BASE:

- Basic Availability: la consistenza può anche non essere garantita interamente, mostrando solamente una parte dei dati realmente disponibili (è il tipico caso di crash di un nodo che quindi non può trasmettere i dati al resto della rete);
- Soft State: i dati possono avere schemi diversi (a differenza del modello relazionale);
- Eventual Consistency: i sistemi NoSQL richiedono che, ad un certo punto, i dati convergano a uno stato consistente senza specificare quando; prima del raggiungimento della consistenza si possono avere valori non veritieri.

[1] Il modello *NewSQL* proposto nel 2011 tenta di risolvere a questo inconveniente

[2] `https://it.wikipedia.org/wiki/Ipotesi_del_mondo_chiuso`

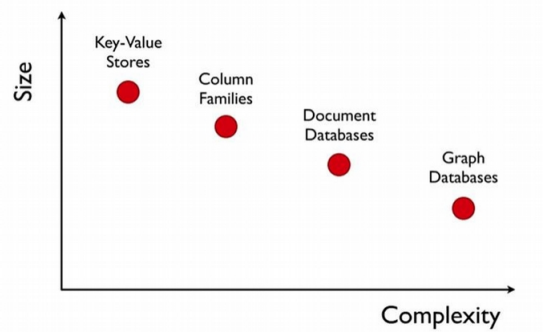
[3] `https://it.wikipedia.org/wiki/Ipotesi_del_mondo_aperto`

ACID	BASE
Forte Coerenza	Coerenza debole
Poca disponibilità	Forte disponibilità
Difficilmente scalabile	Facilmente scalabile
Rigido	Flessibile

I modelli NoSQL si dividono in quattro macrocategorie[3, 4]:

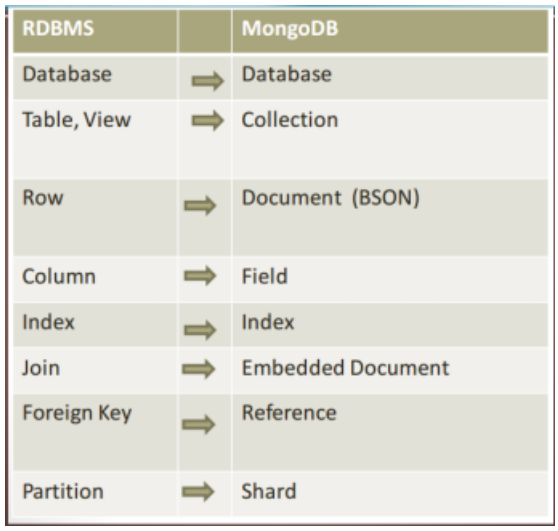
1. Document based (CouchDB, MongoDB): di solito salvati con file JSON, contenente un insieme ordinato di coppie <chiave - valore>; è facile ricercare dati in questo formato;
2. Graph based (Neo4J, FlockDB): sfrutta il concetto matematico di grafo per archiviare i dati come nodi ed esaltare i legami tra di essi costruendo archi; è difficile da scalare per quanto riguarda l'immagazzinamento (è difficile tagliare un grafo) ma è molto rapido nelle query.
3. Key Value (Dynamo, Voldemort, Rhino DHT): sono delle tabelle di coppie <chiave - valore> con chiavi che si riferiscono (o puntano) a un certo dato, è molto simile ai Document based.
4. Column family (Big Table, Cassandra): sono tabelle sparse e annidate contenenti coppie <chiave - valore>, sono in grado di salvare grandi quantità di dati.

Costando di meno lo spazio di archiviazione, questi modelli non tentano di minimizzare lo spazio occupato su disco ma puntano a massimizzare le prestazioni nelle operazioni di R/W. La ridondanza dei dati facilita questo compito pur aumentando notevolmente le dimensioni dei dati stessi: si sacrifica spazio di archiviazione a favore della velocità di lettura. La semplicità del modello permette di archiviare un maggior numero di dati:



1.2.1 Document Based: MongoDB[1, 2].

MongoDB è, come già affermato, un sistema di gestione basato sui documenti (*Document Based Management System*), in cui i dati sono archiviati in formato BSON (Binary JSON) e letti tramite indici. Mongo DB non prevedere operazioni di join. Cambiano i nomi delle entità rispetto al modello relazionale:



RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Lavorando su una rete di calcolatori, MongoDB può impiegare un tempo considerevole a importare documenti da un database presistente; nel caso in cui si si voglia velocizzare il processo, si potrebbe decidere di apportare alcune modifiche:

- disabilitare il riconoscimento dei dati, che è un segnale trasmesso tra processi di comunicazione, computer o dispositivi, per indicare il riconoscimento o la ricezione del messaggio, come parte di un protocollo di comunicazione;
- disabilitare la scrittura su un file di tipo log che ha funzione di backup.

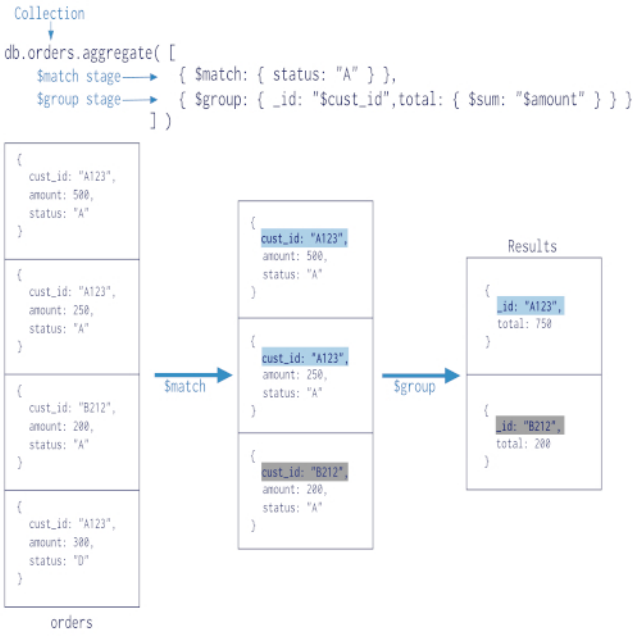
Bisogna stare molto attenti nel fare ciò, perché ogni ogni perdita non sarà registrata e di conseguenza sarà definitiva.

Per dataset di grandi dimensioni risulta molto utile l'utilizzo di indici: simili agli indici dei libri (o delle tabelle SQL), rappresentano un modo più veloce per recuperare informazioni. L'uso di indici rallenta l'inserimento di nuovi dati (perché appunto devono essere indicizzati) ma velocizza notevolmente le ricerche. Il campo `_id` è sempre indicizzato. Senza un indice, MongoDB analizza a tappeto tutti i documenti (esattamente come SQL). Metaforicamente dovrebbe leggere ogni volta tutto il libro per recuperare l'informazione. L'indicizzazione evita proprio questo problema (che aumenta con le dimensioni della *collection*) organizzando il contenuto in una lista ordinata. Dato che l'indicizzazione rallenta le modifiche, è buona norma utilizzare solamente un paio di indici per ogni collection; il limite di MongoDB è di 64 indici. Il comando è il seguente:

```
db.<collection>.ensureIndex({
  <field1> : <sorting>,
  <field2> : <sorting> ...
});
```

dove `<sorting>` assume il valore $+1$ per ordinamenti in ordine crescente e -1 per ordine decrescente. I campi `field` devono essere presenti in tutti i documenti della collection.

È possibile effettuare una sorta di `join` in MongoDB grazie all'algoritmo *pipeline*[3], che però impiega più tempo del corrispettivo SQL. Pipeline permette di eseguire operazioni di `$match` e `$group` su una collection così da eseguire una operazione su insiemi di risultati.



Per ragioni commerciali, MongoDB offre anche un'interfaccia SQL tramite il connettore BI (Business Intelligence): genera lo schema relazionale e lo usa per accedere ai dati.

1.2.2 GraphDB: Neo4J[4].

Un grafo è una collezione di nodi e archi, i quali rappresentano le relazioni tra i nodi stessi. È facile modellare numerose realtà, come: social media, raccomandazioni, luoghi geografici, reti logistiche, grafici per le transazioni finanziarie (per il rilevamento delle frodi), master data management, bioinformatica, sistemi di autorizzazione e controllo degli accessi. Il modello a grafo con etichette (*labeled-property graph model*) ha le seguenti caratteristiche:

- contiene *nodi* e *relazioni*;
- i nodi contengono *proprietà* (coppie <chiave-valore>);
- i nodi possono essere etichettati con una o più *etichette*;
- le relazioni sono nominali e direzionali, con un nodo d'inizio e uno di fine;
- le relazioni, come i nodi, possono avere delle proprietà (e queste possono avere anche valori).

Le proprietà delle relazioni possono essere assegnate con un criterio *fine-grained* o *generic*. Considerando il caso della relazione `ADDRESS`, si può fare distinzione tra:

- fine-grained: `HOME_ADDRESS`, `WORK_ADDRESS` o `DELIVERY_ADDRESS` (sono tutte etichette diverse);

- generic: `ADDRESS:home`, `ADDRESS:work` o `ADDRESS:delivery` (l'etichetta è sempre `ADDRESS`, con un valore che ne indica il tipo).

Generalmente è preferito il secondo metodo per la minore complessità nello scrivere query (come ad esempio elencare tutti gli indirizzi di una data persona).

Un database a grafo può usare un motore di archiviazione nativamente a grafo o usare altri sistemi di archiviazione. Il primo ottimizza la gestione dei grafi, mentre il secondo archivia i dati in formato tabellare o tramite documenti per poi interrogare il database come se fosse un grafo. Il metodo tabellare per esempio archivia le relazioni su una tabella relazionale che può essere interrogata tramite *join bomb* (join con se stessa), tuttavia questo sistema degenera in fretta all'aumentare della distanza tra due nodi.

Il database a grafo risolve quindi il problema dei database relazionali (e di molti altri database NoSQL) a gestire le relazioni interne ai dati. Infatti anche altri modelli NoSQL, indipendentemente dal modello adottato, soffrono perdite di prestazioni quando sono effettuate aggregazioni (soprattutto non indicizzati) dal momento che i collegamenti non sono nativi e manca il concetto di prossimità, presente invece nel grafo. Si può tentare di risolvere il problema con dati annidati tra di loro ma la struttura del database risulterebbe eccessivamente complessa e non permetterebbe altre query. Il DBA in base ai suoi bisogni (integrazione con altre applicazioni) può benissimo decidere di usare un database a grafo con una gestione dei dati non nativa senza che questo impatti sulla qualità del prodotto finale. In un archivio nativo a grafo gli attributi, i nodi e i nodi referenziati sono memorizzati insieme per ottimizzare l'engine di processamento a grafo.

Per eseguire una query nel modello a grafo, il tempo di risposta non tipende strettamente dal numero totale di nodi (che rimane più o meno costante) perché la query viene processata nella porzione locale del grafo connessa al nodo base, mentre nei modelli relazionali e altri modelli NoSQL le prestazioni calano al aumentare dei dati (spesso in una maniera lineare). Inoltre è possibile aggiungere altri nodi e relazioni senza disturbare il modello già esistente anche nel caso in cui i dati non abbiano la stessa struttura. Il processing engine usa *"index-free adjacency"* cioé i nodi connessi sono collegati fisicamente tra di loro, ciò velocizza il loro recupero da una query, ma questa velocità ha un prezzo: l'efficacia delle query che non sfruttano le proprietà del grafo viene peggiorata, ad esempio nel caso delle operazioni di R/W.

Neo4j implementa un linguaggio Cypher, di tipo dichiartivo, che permette query al grafo usando una sitassi simile a SQL o SPARQL, ma comunque ottimizata per i grafi. Cypher è facile da leggere e capire ed espressivo, pur rimanendo compatto. Il linguaggio astrae il concetto di grafo permettendo all'autore di una query di indicare solamente cosa desidera tralasciando come.
Neo4J utilizza in parallelo il linguaggio Gremlin, parte del framework Apache TinkerPop, che invece permette di esplicitare le modalità con cui deve essere svolta una richiesta al database.

Letture di approfondimento

[1] Kristina Chodorow. *MongoDB: The Definitive Guide*. Second Edition. United States of America: O'Reilly, 2013. ISBN: 978-1-449-34468-9.

[2] Kristina Chodorow. *Scaling MongoDB*. United States of America: O'Reilly, 2011. ISBN: 978-1-449-30321-1.

[3] Marina Pernini. *NoSql DB. Analisi prestazionale e confronto col modello relazionale*. EAI, 2018. ISBN: 978-620-2-08288-4.

[4] Ian Robinson. *Graph Databases*. Second Edition. O'Reilly, 2015. ISBN: 978-1-491-93200-1.