



Couchbase

Perché scegliere i database NoSQL?

Perché le aziende di successo scelgono
applicazioni basate su database NoSQL



Cos'è NoSQL?

NoSQL è un moderno sistema di gestione dei dati tramite cui è possibile archiviare le informazioni in documenti JSON anziché in colonne e righe, come avviene nei database relazionali. La struttura dei dati che una soluzione NoSQL fornisce è pensata per snellire la fase di sviluppo e renderla più solida.

Di conseguenza i database NoSQL sono strutturalmente concepiti per essere flessibili e scalabili e rispondere con rapidità alle esigenze di gestione dei dati delle moderne aziende che operano in rete.

Questo documento offre una panoramica sulle problematiche che possono essere superate grazie ai sistemi NoSQL e indica quando e perché scegliere una soluzione NoSQL rispetto a un database relazionale.

Il focus sull'esperienza utente spinge le aziende all'adozione di soluzioni NoSQL

La rivoluzione NoSQL nasce dalla crescente domanda di interazioni utente complesse e dalla necessità di acquisire un vantaggio competitivo, specialmente nell'ambito delle applicazioni Web in tempo reale.

L'esperienza utente costituisce il più importante elemento di differenziazione dalla concorrenza: le aziende si stanno allineando alle nuove aspettative con un'offerta di servizi on-demand e in tempo reale, resilienti e reattivi.

Nelle aziende di oggi l'interazione è digitale come mai prima d'ora: non solo quella con i clienti, ma anche con dipendenti, partner, fornitori e persino con i prodotti.

Fulcro della rivoluzione NoSQL sono applicazioni di rete, applicazioni Edge e basate sul Web: l'Internet delle cose (IoT), i social media, l'analisi dei Big Data, il cloud esterno o interno di un'azienda, il mobile/edge computing... E l'elenco è lungo.

Per conciliare questo pool di nuovi sistemi è necessaria maggiore flessibilità, prestazioni migliori e una scalabilità tale da poter consolidare più tipi di sistemi in un'unica piattaforma.

Database relazionali e database NoSQL a confronto: le differenze

I database relazionali nascono nell'epoca dei computer mainframe e delle applicazioni aziendali di back-office: ben prima di Internet, del Cloud e dei Big Data, dei dispositivi mobili e dell'odierna economia digitale distribuita abilitata al 5G. La prima implementazione commerciale, di fatto, viene pubblicata da Oracle nel 1979. Questi database erano concepiti per essere eseguiti su un unico server, meglio se di grandi dimensioni. L'unico modo per aumentare la capacità dei database relazionali era aggiornare i server a livello di processori, memoria e capacità di archiviazione per ottenere una scalabilità commisurata a quanto stabilito nella Legge di Moore.

I database NoSQL nascono come risultato della crescita esponenziale di Internet e della comparsa delle applicazioni Web. Il 2006 è l'anno di pubblicazione della ricerca Google BigTable, il 2007 quello dell'articolo di ricerca Amazon Dynamo. Gli efficienti database chiave-valore distribuiti sono stati essenziali in questa fase rivoluzionaria e hanno portato la tecnologia a uno sviluppo ulteriore.



Vengono così sviluppati nuovi database per soddisfare i mutati requisiti aziendali, che aziende come Couchbase hanno ulteriormente sviluppato per rispondere alle esigenze con uno sguardo al futuro: la necessità di uno **sviluppo agile** e di **operare su qualsiasi scala**.

L'agilità significava fornire schemi flessibili, API utili, solidi sistemi di query basate su SQL, analisi di testo e molto altro. Con la scalabilità, invece, si poteva accrescere il volume di dati senza sacrificare prestazioni e stabilità.

Supporto per SQL, un aiuto per gli sviluppatori NoSQL

I sistemi relazionali tradizionali gestiscono i dati tabulari restituendoli sotto forma di righe e colonne. Anche i database NoSQL offrono questa funzione. Tuttavia, le moderne applicazioni NoSQL non obbligano gli sviluppatori a usare uno schema statico da riadattare ogni volta che si introduce una modifica. Al contrario, i database NoSQL offrono agli sviluppatori tutta la flessibilità di cui hanno bisogno per eccellere nel proprio lavoro.

I sistemi NoSQL conservano i dati JSON in forma gerarchica, ma li restituiscono all'applicazione sotto forma di strutture di dati JSON complete o parziali, corrispondenze di ricerca full-text, righe di query SQL tabulari, oggetti basati su chiavi o perfino sistemi di analisi dei Big Data.

Tale convergenza di tecnologie snellisce l'architettura delle informazioni delle aziende e aiuta gli sviluppatori a produrre applicazioni con maggiore efficienza e senza dover acquisire familiarità con decine di piattaforme diverse.

Oltre i limiti dei database SQL

Per operare su larga scala, il nuovo approccio dei sistemi NoSQL richiede capacità di computing basate su cluster con interconnessioni efficienti dei nodi per mantenere la sincronizzazione dei dati e garantire un flusso ad alta velocità.

Ad esempio, oggi si chiede ai responsabili tecnici di creare un'infrastruttura aziendale di gestione dei dati con le seguenti caratteristiche:

- Capacità di supportare un elevato numero di utenti simultanei (decine di migliaia, ma anche milioni)
- Capacità di offrire esperienze altamente reattive a una base di utenti distribuita in tutto il mondo
- Disponibilità costante, senza tempi di inattività
- Capacità di gestire dati semi-strutturati e non strutturati
- Rapidità di adattamento a requisiti mutevoli e con aggiunte frequenti di nuove funzionalità

I database relazionali basati su SQL non soddisfano questi nuovi requisiti, contrariamente ai database NoSQL.



Consideriamo alcuni esempi di aziende della classifica Forbes Global 2000 che hanno scelto di implementare soluzioni NoSQL per le applicazioni mission-critical e che figurano nelle recenti notizie:

- **Tesco**, uno dei migliori rivenditori di generi alimentari d'Europa, implementa NoSQL per l'e-commerce, il catalogo dei prodotti e altre applicazioni
- **Ryanair**, tra le compagnie aeree con maggiore traffico al mondo, usa NoSQL come base per la sua app per dispositivi mobili, che vanta oltre 3 milioni di utenti
- **Marriott** implementa NoSQL nel suo sistema di prenotazione, che registra ogni anno 38 miliardi di dollari
- **Viber** elabora oltre 15 miliardi di eventi al giorno per la sua base di oltre 1 miliardo di clienti
- **GE** implementa NoSQL nella sua piattaforma Predix per supportare la gestione dell'Internet industriale

Tendenze di oggi e sfide di domani

Oggi più che mai, gli obiettivi riguardo all'esperienza utente dipendono da un'integrazione tecnologica strettamente allineata, che deve adattarsi perfettamente alle tendenze mutevoli dell'economia digitale per non rischiare di essere già superata al momento dell'implementazione.

Tra gli obiettivi tecnologici di alto livello ci sono:

- Piattaforme consolidate con integrazione reciproca efficiente
- Architetture di semplice gestione
- Plumbing dei dati efficace per applicazioni Web in tempo reale e bassa latenza
- Creare un service-layer che spinga i dati il più possibile vicino agli utilizzatori

Cinque vantaggi dei database NoSQL nell'economia digitale di oggi

Le tendenze generali dell'economia digitale hanno introdotto nuove sfide che portano gli obiettivi appena citati a un livello persino superiore. Alcuni clienti ambiziosi con intuizioni interessanti su come usare i dati hanno posto CIO e responsabili tecnici davanti a nuova serie di requisiti tecnologici.

Di seguito sono presentate cinque tendenze che giocano sui vantaggi dei database NoSQL per superare le sfide che si pongono nell'operare in un'economia digitale.



Tendenze dell'economia digitale	Requisiti
1. Costante incremento dei clienti al mondo online	• Disporre della scalabilità per supportare migliaia o milioni di utenti • Soddisfare i requisiti dell'esperienza utente con prestazioni sempre elevate • Mantenere la disponibilità 24 ore su 24, 7 giorni su 7
2. Internet connette ogni cosa	• Supportare molte applicazioni diverse con differenti strutture di dati • Garantire software "sempre attivi" senza alcuna interruzione • Supportare flussi di dati continui dal Web in tempo reale
3. I volumi di Big Data sono in aumento	• Conservare dati semi-strutturati e non strutturati generati dai clienti • Conservare diversi tipi di dati da diverse fonti nella stessa infrastruttura o anche nello stesso cluster • Conservare i dati generati da migliaia o milioni di utenti e dispositivi
4. Le applicazioni passano al cloud	• Scalabilità su richiesta per supportare più clienti e conservare più dati • Impiegare applicazioni completamente gestite su scala globale per supportare i clienti in tutto il mondo • Ridurre al minimo i costi dell'infrastruttura, per un time to market più rapido
5. Il mondo è passato al mobile	• Creare app "offline-first", ossia che non richiedono una connessione di rete • Sincronizzare i dati di dispositivi mobili/periferici con database remoti nel cloud • Supportare più piattaforme per dispositivi mobili con un unico back-end



I requisiti sopracitati sono stringenti e pongono, anche ai migliori sistemi, la sfida di aumentare ulteriormente le prestazioni con meno risorse. I requisiti stringenti di oggi possono essere raggruppati approssimativamente in due categorie, che influiscono su due diversi livelli di utenti finali:

- Fornire piattaforme agili per consentire agli sviluppatori di applicazioni di eccellere
- Supportare architetture con sistema scalabile che offrano prestazioni migliori delle altre

Sviluppo agile

Per rimanere competitive nell'economia digitale, le aziende di oggi devono innovare a un ritmo più veloce che mai. Tale innovazione, che si concentra sullo sviluppo di moderne applicazioni Web, IoT e per dispositivi mobili, espone gli sviluppatori a un'enorme pressione. Per queste applicazioni, la velocità è fondamentale quanto l'agilità: la loro rapidità di evoluzione è decisamente più alta delle applicazioni legacy come i sistemi ERP. I database relazionali richiedono una struttura piatta di dati relativamente ristretta e non rispondono in modo ottimale alle modifiche frequenti al modello di dati. Questa caratteristica non soddisfa le esigenze delle moderne applicazioni e dei requisiti aziendali che evolvono rapidamente.

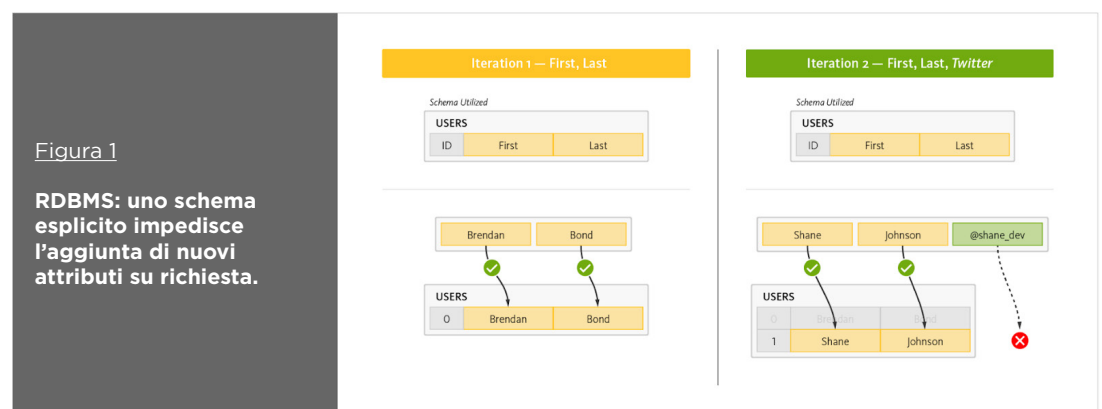
Tutte le piattaforme agili offrono la flessibilità necessaria per uno sviluppo più rapido e semplice delle applicazioni. Alcuni di questi vantaggi riguardano la modalità di gestione dei dati della piattaforma, altri riguardano le modalità di interazione con il database disponibili per le applicazioni.

Requisiti dello schema NoSQL adattabile

Uno dei principi fondamentali dello sviluppo agile è l'adattabilità ai mutevoli requisiti delle applicazioni: ogni volta che i requisiti cambiano, cambia anche il modello di dati. Per i database relazionali questo costituisce un problema, in quanto il modello di dati è fisso e definito da uno schema statico. Pertanto, per modificare il modello di dati, gli sviluppatori sono obbligati a modificare lo schema o, peggio ancora, a richiedere una "modifica dello schema" agli amministratori del database. Tutto ciò rallenta o arresta lo sviluppo: non solo perché il processo è manuale e richiede molto tempo, ma anche perché influisce su altre applicazioni e servizi.

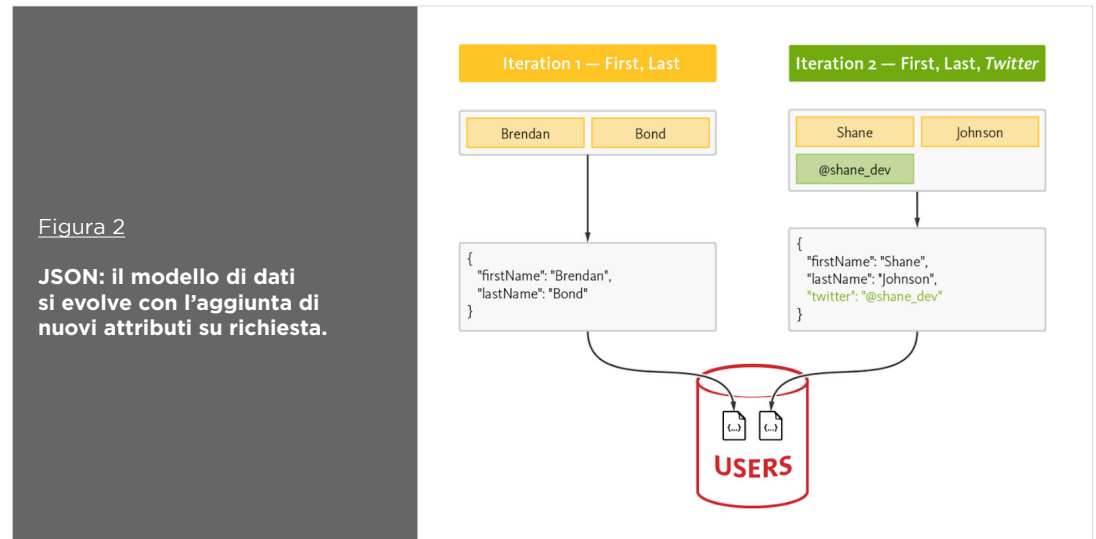
Figura 1

RDBMS: uno schema esplicito impedisce l'aggiunta di nuovi attributi su richiesta.



Flessibilità per sviluppare con maggiore velocità

A confronto, un database NoSQL a documenti supporta completamente lo sviluppo agile, in quanto non presenta uno schema e non definisce in modo statico come devono essere modellati i dati. Al contrario, fa riferimento alle applicazioni e ai servizi, e di conseguenza agli sviluppatori, per definire come i dati devono essere modellati. Con i sistemi NoSQL, il modello di dati viene definito dal modello dell'applicazione. Applicazioni e servizi modellano i dati come oggetti.



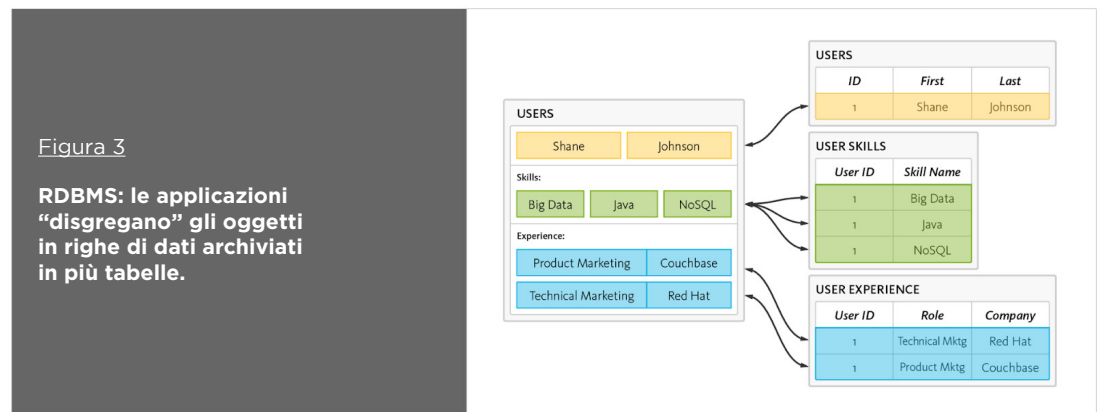
Semplicità per uno sviluppo più facile

Applicazioni e servizi modellano i dati come oggetti (ad es. il profilo del dipendente), i dati con più valori come raccolte (ad es. i ruoli) e i dati correlati come oggetti nidificati o raccolte (ad es. relation-manager). Al contrario, i database relazionali modellano i dati in forma di tabelle composte da righe e colonne e i dati correlati ad essi come righe all'interno di tabelle diverse. Il problema dei database relazionali è il fatto che i dati vengono letti e scritti scomponendo (o disgregando) e ricomponendo gli oggetti. Si tratta del cosiddetto "impedance mismatch", ovvero dell'incompatibilità tra sistemi orientati agli oggetti e sistemi relazionali.

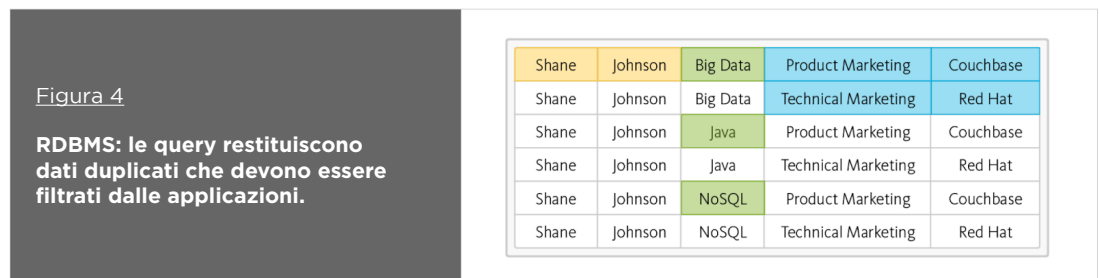
Una soluzione al problema la offrono i framework ORM (Object-Relational Mapping) ma nel fare ciò introducono inefficienze nel migliore dei casi e problemi in quelli peggiori.

Prendiamo in considerazione un'applicazione per la gestione dei curriculum. Essa interagisce con i curriculum come oggetto degli oggetti utente. Contiene un array per le competenze e una raccolta per le posizioni. Tuttavia, la scrittura di un curriculum in un database relazionale richiede all'applicazione di "disgregare" l'oggetto utente.

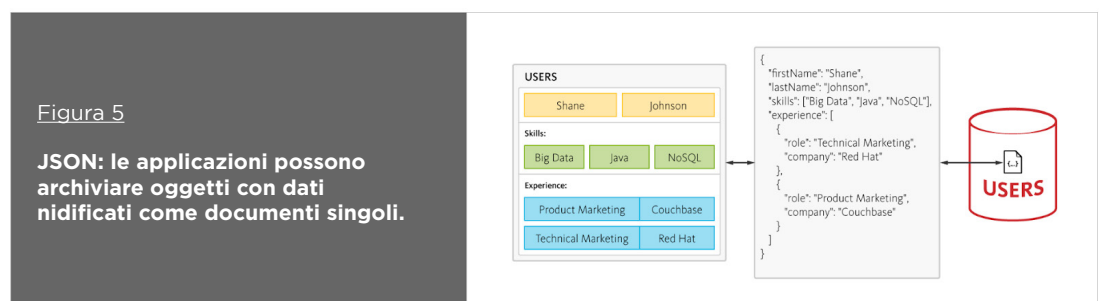
Per archiviare il curriculum l'applicazione dovrebbe inserire sei righe in tre tabelle, come mostrato nella **Figura 3**.



Per la lettura di questo profilo, tuttavia, l'applicazione dovrebbe leggere sei righe da tre tabelle, come illustrato nella **Figura 4**.



Contrariamente all'approccio relazionale, un database NoSQL orientato ai documenti legge e scrive i dati formattati in JSON, lo standard per il consumo e la produzione di dati per le applicazioni Web, i dispositivi mobili e le applicazioni IoT. Ciò non solo elimina l'impedance mismatch ma elimina anche il sovraccarico dei framework ORM e semplifica lo sviluppo delle applicazioni, in quanto consente di leggere e scrivere gli oggetti senza "disgregarli". Questo vuol dire che un singolo oggetto può essere letto o scritto come un singolo documento, come illustrato nella **Figura 5**.



Raggruppare i documenti per facilitare l'accesso

Anziché usare set predefiniti di schemi per differenziare le tabelle tra loro, i database NoSQL sfruttano un concetto, come quello dei bucket (in altre parole, un contenitore), che funge da area di conservazione generale per tutti i documenti. Un database può avere più bucket logici e denominati per vari scopi.

All'interno di tali bucket vi sono ulteriori raggruppamenti logici gerarchici che possono essere limitati a particolari utenti o ruoli. Questi prendono il nome di raccolte e/o ambiti e permettono di assegnare un nome ai sottoinsiemi di documenti presenti in un bucket. Grazie a questa flessibilità che aiuta a separare i dati di un utente o di un'applicazione dagli altri, gli sviluppatori non devono compilare un codice proprietario di sicurezza e affidabilità, bensì possono lasciar svolgere l'operazione al database sottostante.

Eseguire query con SQL

Gli sviluppatori di applicazioni abituati a eseguire query con SQL possono continuare a usare lo stesso linguaggio sulle piattaforme NoSQL, e lo possono fare anche con dati salvati come documenti JSON! Couchbase, ad esempio, fornisce un linguaggio query basato su SQL, noto come N1QL, che restituisce risultati in JSON come insiemi di righe e componenti dei sotto-documenti ove appropriato. Ciò contrasta con la vasta maggioranza degli altri database NoSQL (come MongoDB™) che non utilizzano SQL e forzano gli sviluppatori ad imparare un nuovo linguaggio.

In N1QL (in altre parole SQL per JSON) sono supportate istruzioni standard tra cui le sintassi SELECT .. FROM .. e WHERE. N1QL supporta anche l'aggregazione, l'ordinamento e le unioni (GROUP BY .. SORT BY .. LEFT OUTER/INNER JOIN). Sono supportate le query partizionate, CTE e persino array nidificati. Le prestazioni delle query possono essere modificate con indici composti, parziali, inclusivi e molto altro.

Per gli esperti di SQL, il passaggio a N1QL richiede solo delle piccole modifiche e il funzionamento di tutte le query di base è predefinito.

SQL	N1QL
<pre>SELECT p.FirstName + ' ' + p.LastName AS Name, d.City FROM AdventureWorks2016.Person. Person AS p INNER JOIN AdventureWorks2016. HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person. Address AS a INNER JOIN AdventureWorks2016. Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;</pre>	<pre>SELECT p.FirstName ' ' p.LastName AS Name, d.City FROM AdventureWorks2016.Person. Person AS p INNER JOIN AdventureWorks2016. HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person. Address AS a INNER JOIN AdventureWorks2016. Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;</pre>



Transazioni ACID nei database NoSQL

I database NoSQL sono utilizzati come sistemi operazionali anche in casi in cui vi sia la necessità di supportare le applicazioni con transazioni. Quando, in un database relazionale, si rende flat un'entità contenuta in più tabelle separate, è necessaria una transazione per quasi ogni aggiornamento. Con i database NoSQL non è, in genere, necessario rendere flat l'entità, ed è anche possibile contenerla in un unico documento. Gli aggiornamenti di un documento singolo sono quindi minori e allo stesso tempo non richiedono una transazione. Tuttavia, alcuni aggiornamenti potrebbero estendersi a più documenti e richiedere un controllo per assicurarsi che la transazione avvenga completamente o non avvenga affatto.

Questo è il motivo per cui i database NoSQL come Couchbase supportano le transazioni.

```
Transactions transactions = Transactions.create(cluster,
    TransactionConfigBuilder.create()
        .durabilityLevel(TransactionDurabilityLevel.PERSIST_TO_MAJORITY)
        .logOnFailure(true, Event.Severity.WARN)
        .build());

TransactionResult result = transactions.run((ctx) -> {
    // Insertar un documento:
    ctx.insert(collection, "doc-a", JsonObject.create());

    // Ottenere i documenti:
    // Usare ctx.getOptional se il documento potrebbe non esistere
    Optional<TransactionGetResult> docOpt =
        ctx.getOptional(collection, "doc-a");

    // Usare ctx.get se il documento esiste, e la transazione
    // non andrà a buon fine in caso contrario
    TransactionGetResult docA = ctx.get(collection, "doc-a");

    // Sostituire un documento:
    TransactionGetResult docB = ctx.get(collection, "doc-b");
    JsonObject content = docB.contentAs(JsonObject.class);
    content.put("transactions", "are awesome");
    ctx.replace(docB, content);

    // Rimuovere un documento:
    TransactionGetResult docC = ctx.get(collection, "doc-c");
    ctx.remove(docC);

    ctx.commit();
});
```



La combinazione di transazioni e possibilità di utilizzo del linguaggio SQL estende enormemente il numero di casi d'uso in cui è possibile considerare un database NoSQL. In passato, l'impossibilità di eseguire join o transazioni, significava che i database NoSQL erano scelti solo per quei casi in cui vi erano esigenze di scalabilità o di gestione di grandi volumi. Tutto questo oggi è cambiato: SQL e transazioni sono disponibili nel mondo NoSQL e ciò significa che si può utilizzare database NoSQL anche per casi in cui i RDBMS tradizionali necessitano di maggiore flessibilità e potenza.

Un'unica fonte di dati, più metodi di accesso

I database NoSQL operano repository primario di informazioni. Questo vuol dire che è possibile immettere i dati in un'applicazione ma accedervi in diversi modi a seconda del caso d'uso. Ad esempio, gli sviluppatori possono usare le chiamate dirette alle API per accedere a specifici documenti utilizzando una chiave o tramite una query SQL che restituisce più righe di dati in una risposta in formato JSON.

A seconda del database sono disponibili altri metodi di accesso, inclusi i sistemi di ricerca full-text che consentono di inviare richieste di ricerca in linguaggio naturale. Le richieste possono essere effettuate per corrispondenze complete o parziali ("fuzzy"), per intervalli geografici o per ricerche con caratteri jolly. La risposta include un documento JSON contenente gli elenchi degli ID dei documenti corrispondenti, le informazioni contestuali e un punteggio di pertinenza.

I sistemi di ricerca full-text spesso sono separati dai database. Tuttavia, i database NoSQL come Couchbase integrano tali sistemi nel sistema sottostante, consentendo ai responsabili di semplificare l'architettura complessiva.

Anche l'analisi dei Big Data è possibile, usando dei sottosistemi complementari che processano volumi più estesi di dati storici. Sfruttando le avanzate capacità di indicizzazione e query, basate su un linguaggio noto come SQL++, è possibile effettuare analisi più avanzate nello stesso database senza necessità di un sistema OLAP esterno.

Poiché tutti i dati sono archiviati e indicizzati all'interno di un unico prodotto database, gli sviluppatori possono collegarsi a un unico sistema e trasmettere tutte le richieste pertinenti.

Operare su qualsiasi scala

I database che supportano le applicazioni Web, IoT e per dispositivi mobili devono avere la capacità di operare su qualsiasi scala. Sebbene sia possibile scalare un database relazionale come Oracle (usando, ad esempio, Oracle RAC), l'operazione è in genere complessa, costosa e non totalmente affidabile. Con Oracle, ad esempio, la scalabilità orizzontale con la tecnologia RAC richiede numerosi componenti e crea un "single point of failure" che mette a repentaglio la disponibilità.

A confronto, un database NoSQL distribuito, progettato con un'architettura scalabile e nessun single point of failure, offre dei interessanti vantaggi operativi.



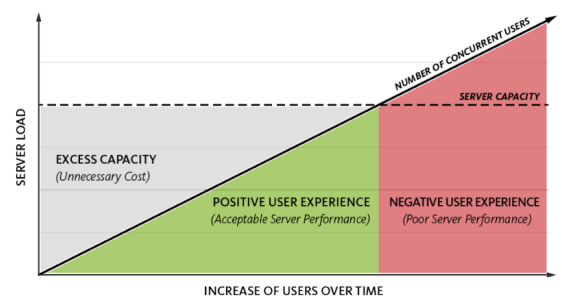
Elasticità per prestazioni su larga scala

Applicazioni e servizi devono supportare un numero di utenti e dati in costante crescita: centinaia, migliaia o milioni di utenti, gigabyte e terabyte di dati operativi. Al contempo, necessitano di scalabilità per mantenere alte le prestazioni e l'efficienza.

Il database deve essere scalabile in termini di letture, scritture e archiviazione. Si tratta di un problema per i database relazionali con scalabilità limitata, ossia che possono essere scalati solo aggiungendo più processori, memoria e capacità di archiviazione a un singolo server fisico. Come risultato, la capacità di scalare in modo efficiente e su richiesta costituisce una sfida. Diventa sempre più costosa, in quanto le aziende devono acquistare server sempre più grandi per ospitare più utenti e più dati. Inoltre, pone il rischio di incorrere in tempi di inattività se il database deve essere messo offline per l'esecuzione degli aggiornamenti dell'hardware.

Figura 6

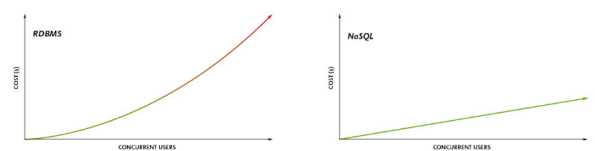
RDBMS: il server è troppo grande o troppo piccolo e questo comporta costi inutili o prestazioni scadenti.



Un database NoSQL distribuito, al contrario, sfrutta l'hardware di largo consumo per la **scalabilità orizzontale**, consentendo di aggiungere più risorse semplicemente aggiungendo più server a un cluster. La scalabilità orizzontale consente alle aziende di scalare con maggiore efficienza (a) installando una quantità di hardware non superiore a quello richiesto per soddisfare il carico attuale; (b) sfruttando hardware e/o un'infrastruttura cloud meno costosi e (c) offrendo scalabilità su richiesta senza tempi di inattività.

Figura 7

NoSQL: aggiunge server meno costosi su richiesta, in modo che le risorse hardware corrispondano al carico dell'applicazione.



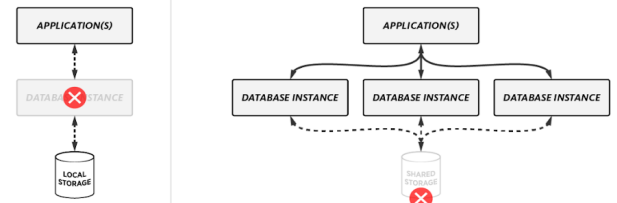
Distribuendo lettura, scrittura e archiviazione in un cluster di nodi, i database NoSQL possono operare su qualsiasi scala. Inoltre, sono progettati per essere facili da configurare e installare e per gestire cluster di piccole e/o grandi dimensioni.

Disponibilità per una distribuzione globale sempre attiva

Con il crescere del numero di interazioni utente tramite il Web e le applicazioni mobile, la “availability” diventa un punto importante (se non cruciale). Queste applicazioni mission-critical devono essere disponibili 24 ore su 24, 7 giorni su 7, senza alcuna eccezione. Offrire una disponibilità 24/7 è una sfida per i database relazionali distribuiti in un singolo server fisico o basati sul clustering con archiviazione condivisa. Se, in caso di errore con distribuzione sul singolo server o di errore nell’archiviazione condivisa con distribuzione in cluster, il database non è più disponibile, le applicazioni si interrompono e il l’esperienza utente ne risente.

Figura 8

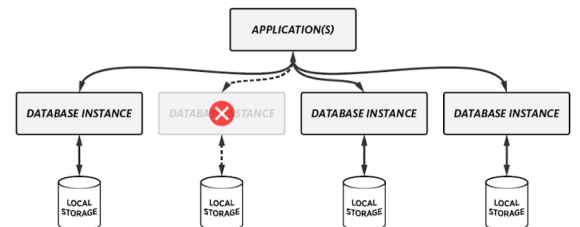
RDBMS: il guasto di un server o di un dispositivo di archiviazione provoca il blocco dell’intero database.



Contrariamente alla tecnologia relazionale, i database NoSQL partizionano e distribuiscono i dati in più istanze di database senza alcuna risorsa condivisa. In aggiunta, i dati possono essere replicati in una o più istanze per un’elevata disponibilità (replica inter-cluster) e in diverse aree geografiche. Mentre i database relazionali come Oracle richiedono un software separato per la replica, ad esempio Oracle Active Data Guard, lo stesso non vale per i database NoSQL: qui il software è integrato e la replica automatica. Inoltre, il failover automatico garantisce l’esecuzione delle operazioni di scrittura e lettura da parte del database anche se un nodo fallisce, grazie all’invio delle richieste a un nodo diverso.

Figura 9

NoSQL: se un’istanza genera un errore, l’applicazione può inviare richieste a un’altra istanza.

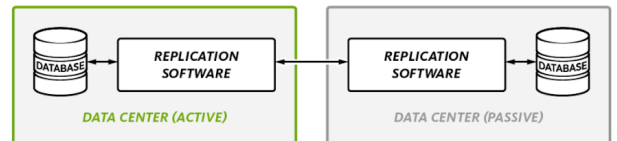


Il comportamento de gli utenti spinge oggi le aziende a supportare più canali fisici, online e su dispositivi mobili in più regioni e spesso in più Paesi. Se la distribuzione di un database in più data center aumenta la disponibilità e supporta il ripristino di emergenza, presenta anche il vantaggio di aumentare le prestazioni. Tutte le operazioni di lettura e scrittura possono essere eseguite nel data center più vicino, riducendo di conseguenza la latenza.

Garantire la disponibilità globale è difficile per i database relazionali in cui sono richiesti componenti aggiuntivi separati (che aumentano la complessità) o in cui la replica tra più data center può essere usata solo per il failover, poiché è attivo un solo data center alla volta. Oracle, ad esempio, richiede Oracle GoldenGate. Durante la replica tra data center, le applicazioni create sui database relazionali possono subire una riduzione delle prestazioni o rilevare che i data center non sono sincronizzati.

Figura 10

RDBMS: richiede un software distinto per replicare i dati in altri data center.

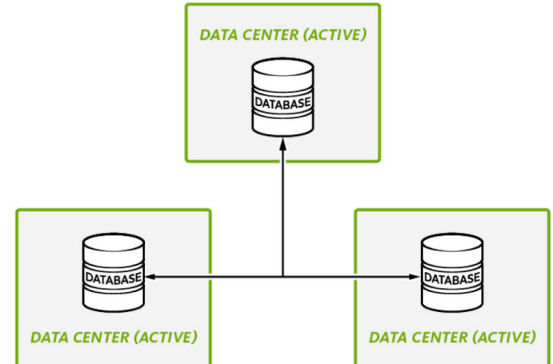


Un database NoSQL distribuito include la replica integrata tra data center, senza necessità di software separati. In aggiunta, alcuni includono la replica bidirezionale consentendo la completa distribuzione active-active in più data center. Questo consente di distribuire il database in più Paesi o regioni fornendo l'accesso ai dati locali alle applicazioni locali e ai relativi utenti.

La distribuzione in più data center non solo migliora le prestazioni, ma consente anche il failover immediato tramite i router dell'hardware. Le applicazioni non devono attendere che il database rilevi l'errore ed esegua il suo failover.

Figura 11

NoSQL: la replica tra i data center è completamente integrata e può essere bidirezionale.



I database NoSQL sono più indicati per i requisiti dell'economia digitale

Con il passaggio delle aziende all'economia digitale — reso possibile dal cloud, dai dispositivi mobili, dai social media e dalle tecnologie dei Big Data — sviluppatori e team delle operazioni devono sviluppare e mantenere applicazioni Web, IoT e per dispositivi mobili a ritmi sempre più serrati e su una scala sempre più vasta. NoSQL è sempre più la tecnologia di database preferita per alimentare le moderne applicazioni Web, IoT e per dispositivi mobili.

Centinaia di aziende della classifica Global 2000, oltre a decine di migliaia di imprese più piccole e start-up, hanno adottato soluzioni NoSQL. Molti hanno iniziato a usare NoSQL con una cache open source, un modello di verifica o una piccola applicazione, per poi estenderne l'uso ad applicazioni mirate mission-critical. Ora è la base per lo sviluppo di tutte le applicazioni. Oggi, il database NoSQL Couchbase serve migliaia di questi tipi di clienti.

Con NoSQL le aziende migliorano le proprie capacità di sviluppo agile e operatività su qualsiasi scala e offrono le prestazioni e la disponibilità richieste per soddisfare le esigenze dell'economia digitale.

Informazioni su Couchbase

Couchbase offre un database di classe enterprise, compatibile con la distribuzione su più cloud e su dispositivi periferici che garantisce le solide funzionalità necessarie alle applicazioni business-critical su una piattaforma altamente scalabile e disponibile. Diversamente da altri database NoSQL, come database distribuito cloud-native, Couchbase può essere eseguito in ambienti moderni e dinamici oltre che in qualsiasi cloud, gestito dal cliente o completamente gestito as-a-service. Couchbase è progettato in base a standard aperti e abbina il meglio della tecnologia NoSQL con la potenza e la notorietà del linguaggio SQL, per semplificare il passaggio da database relazionali e mainframe ad ambienti distribuiti on-premise o di cloud hosting.

Couchbase è ormai presente nel nostro quotidiano: i nostri clienti includono leader di settore come Amadeus, American Express, Carrefour, Cisco, Comcast/Sky, Disney, eBay, LinkedIn, Marriott, Tesco, Tommy Hilfiger, United, Verizon e centinaia di altri marchi molto conosciuti.

Per maggiori informazioni, visita il sito www.couchbase.com.

© 2021 Couchbase. All rights reserved.

