NoSQL e gestione dei dati non relazionali

Obiettivi formativi

Al termine di questa unità, gli studenti saranno in grado di:

- Comprendere i concetti fondamentali, le motivazioni e le differenze architetturali tra sistemi di database relazionali e non relazionali (NoSQL), inclusi vantaggi e limiti negli ambienti di gestione dei dati moderni.
- Classificare e descrivere le principali categorie di database NoSQL (archivi chiave-valore, documentali, a famiglie di colonne e a grafo), individuando casi d'uso appropriati e tecnologie rappresentative per ciascuna categoria.
- Comprendere e valutare le soluzioni NoSQL nelle architetture software, soprattutto nel contesto dei big data, del cloud computing e dell'analisi in tempo reale, a sostegno di strategie di gestione dei dati efficienti e flessibili.

1. Introduzione: sistemi informativi ambientali ad alta intensità di dati

1.1. L'esplosione dei dati ambientali

Negli ultimi due decenni, il monitoraggio ambientale ha subito una profonda trasformazione tecnologica. Oggi i sistemi ecologici vengono osservati mediante fitte reti di sensori IoT, costellazioni satellitari, droni autonomi, database sulla biodiversità, modelli climatici e piattaforme di scienza partecipativa. Questi sistemi producono continuamente enormi quantità di dati eterogenei sulle condizioni atmosferiche, sui cicli idrologici, sull'uso del suolo, sulla biodiversità e sugli indicatori di salute degli ecosistemi.

A differenza dei dataset ambientali tradizionali, raccolti periodicamente ed elaborati offline, le attuali infrastrutture di monitoraggio producono flussi di dati caratterizzati da velocità, volume e varietà elevati. Una sola rete regionale per la qualità dell'aria può generare milioni di misurazioni con marcatura temporale ogni mese. Le piattaforme di osservazione satellitare forniscono quotidianamente terabyte di dati raster. Portali sulla biodiversità come GBIF ospitano centinaia di milioni di registrazioni di presenza delle specie, corredate da metadati spaziali e temporali.

Questa trasformazione ha dato origine a una nuova categoria di sistemi: i sistemi informativi ambientali ad alta intensità di dati. Tali sistemi devono:

- integrare dati eterogenei, strutturati e non strutturati;
- operare in ambienti geograficamente distribuiti;
- supportare analisi quasi in tempo reale;
- scalare in modo elastico;
- rimanere resilienti anche in presenza di eventi ambientali imprevedibili.

Le architetture di dati tradizionali non erano state progettate per un tale livello di dinamicità.

1.2. Dal monitoraggio ambientale alle piattaforme di dati distribuite

Le moderne piattaforme ambientali gestiscono dataset che combinano diverse tipologie di dati, come illustrato nella Tabella 1.

Tabella 1 – Tipologie di dati ambientali

Tipo di dato	Natura strutturale	Esempi
Serie temporali	Sequenziale, ad alta frequenza	Temperatura, umidità, PM2.5, umidità del suolo
Spaziale	Geospaziale	Coordinate, aree protette, poligoni di copertura del suolo
Basato su eventi	Semistrutturato	Rilevamento della fauna selvatica, allerte alluvione
Metadati	Documentale	Specifiche dei sensori, registri di calibrazione
Dati relazionali	Struttura a grafo	Relazioni specie-habitat, reti di bacini idrografici
Testuale	Non strutturato	Note di campo, relazioni, annotazioni

La coesistenza di queste forme di dati introduce diverse sfide architetturali:

- gli schemi rigidi diventano poco pratici;
- gli schemi devono evolvere frequentemente;

- le query relazionali che richiedono molti JOIN perdono prestazioni su larga scala;
- la scalabilità verticale diventa economicamente inefficiente;
- la replica distribuita è complessa nei sistemi RDBMS tradizionali.

Con il passaggio del monitoraggio ambientale verso architetture continue, in tempo reale e geograficamente distribuite, emerge la necessità di modelli di database flessibili e scalabili orizzontalmente.

1.3. Perché le architetture relazionali possono diventare limitanti

I sistemi di gestione di database relazionali (RDBMS) offrono eccellenti garanzie di coerenza transazionale e si adattano a contesti con schemi ben definiti. Tuttavia, incontrano i seguenti limiti.

1. **Rigidità dello schema.** I sistemi di sensori ambientali evolvono continuamente con l'introduzione di nuovi parametri, come l'indice UV, la salinità dell'acqua o la concentrazione di microplastiche. Gli schemi richiedono quindi modifiche frequenti e ripetute operazioni ALTER TABLE possono diventare problematiche per la stabilità e le prestazioni del sistema.
2. **Query con molti JOIN.** Le query ambientali multidimensionali richiedono spesso di combinare dati dei sensori, metadati dei siti, livelli spaziali, registrazioni di specie e altro ancora. Le operazioni JOIN su milioni di righe possono quindi incidere notevolmente sulle prestazioni.
3. **Limiti della scalabilità orizzontale.** Tradizionalmente, la crescita dei sistemi relazionali si basa sul potenziamento verticale dell'hardware, una strategia costosa e soggetta a limiti fisici.
4. **Inefficienze nella gestione di dati geospaziali e grafi.** Sebbene esistano estensioni relazionali, come PostGIS, queste introducono ulteriore complessità e non sempre scalano efficientemente in architetture distribuite.
5. **Difficoltà nell'acquisizione di flussi continui.** Scritture ad alta frequenza possono sovraccaricare i meccanismi di blocco transazionale.

Questi limiti non significano che i database relazionali siano obsoleti, ma che **i sistemi ambientali di big data richiedono paradigmi architetturali complementari o alternativi.**

È in questo contesto che diventano importanti NoSQL e la gestione dei dati non relazionali.

I sistemi NoSQL segnano il passaggio da una progettazione guidata dallo schema a una modellazione guidata dall'applicazione. Invece di chiedersi «Come normalizzare i dati?», ci si chiede «Come verranno acceduti, interrogati e distribuiti i dati?». Questo cambiamento comporta diversi principi fondamentali:

- Schemi flessibili o dinamici
- Denormalizzazione come strategia per migliorare le prestazioni
- Scalabilità orizzontale come ipotesi di base
- Compromessi nella coerenza distribuita
- Partizionamento e replica dei dati previsti fin dalla progettazione

Nei sistemi ambientali, in cui le query riguardano spesso intervalli temporali recenti, regioni geografiche o categorie di ecosistemi, progettare in funzione delle modalità di accesso diventa più importante della normalizzazione rigorosa.

2. Fondamenti di NoSQL e delle architetture di dati distribuite

2.1. Fondamenti teorici

L'emergere delle tecnologie NoSQL riflette un profondo cambiamento nel modo di concepire e realizzare i sistemi di dati distribuiti. Mentre i database relazionali sono stati progettati intorno a schemi predefiniti e principi rigorosi di normalizzazione, i moderni sistemi informativi ambientali operano sempre più spesso in ambienti dinamici, eterogenei e geograficamente distribuiti, nei quali tale rigidità può risultare limitante. In questi contesti il database non è più soltanto un archivio strutturato, ma un componente adattivo di un'architettura distribuita più ampia.

Nonostante le differenze, i database NoSQL condividono alcune caratteristiche distintive:

- *Flessibilità dello schema:* i sistemi NoSQL sono generalmente privi di uno schema rigido oppure consentono uno schema opzionale. Le strutture possono evolvere senza definizioni preventive delle tabelle. Ciò favorisce lo sviluppo agile e l'iterazione rapida: è possibile memorizzare nuovi campi o oggetti annidati senza migrazioni del database.
- *Scalabilità orizzontale:* anziché potenziare una singola macchina, i sistemi NoSQL distribuiscono i dati su più server o nodi. Questa progettazione consente di incrementare la capacità aggiungendo nodi.
- *Alta disponibilità e tolleranza ai guasti*: molti database NoSQL replicano i dati su diversi nodi, spesso mediante algoritmi di consenso distribuito o meccanismi di replica basati su quorum. Ciò consente di proseguire le operazioni anche in caso di guasto di alcuni nodi.
- *Modelli di coerenza rilassati:* negli ambienti distribuiti, garantire una coerenza forte può aumentare la latenza e ridurre la disponibilità. Numerosi sistemi NoSQL adottano il modello

BASE (Basically Available, Soft State, Eventually Consistent), che attenua alcuni requisiti ACID per favorire scalabilità e tolleranza ai guasti.

- *Ottimizzazione per casi d'uso specifici:* invece di adottare un unico modello relazionale generalista, le tecnologie NoSQL sono ottimizzate per determinati tipi di dati e modalità di accesso, come caching, archiviazione di documenti, attraversamento di grafi o elaborazioni analitiche.
- *Architettura distribuita:* i sistemi NoSQL sono progettati come cluster di server che condividono il carico di lavoro. Il partizionamento (sharding) distribuisce i dati, mentre la replica favorisce durabilità e tolleranza ai guasti.

Uno dei fondamenti teorici più influenti dei database NoSQL è il teorema CAP, proposto da Eric Brewer. Esso afferma che in un sistema distribuito non è possibile garantire contemporaneamente tutte e tre le seguenti proprietà:

- ***Coerenza (C**onsistency**):*** tutti i nodi vedono gli stessi dati nello stesso momento.
- ***Disponibilità (A**vailability**):*** ogni richiesta riceve una risposta, anche se basata su dati non aggiornati.
- ***Tolleranza alle partizioni (P**artition Tolerance**):*** il sistema continua a funzionare nonostante partizioni o guasti della rete.

Un sistema NoSQL deve affrontare compromessi tra queste proprietà: i sistemi CA (rari in ambienti distribuiti) privilegiano coerenza e disponibilità; i sistemi CP (ad esempio HBase e MongoDB in determinate configurazioni) privilegiano coerenza e tolleranza alle partizioni; i sistemi AP (ad esempio Cassandra e CouchDB) privilegiano disponibilità e tolleranza alle partizioni, consentendo la coerenza eventuale.

Per favorire scalabilità e resilienza, i database NoSQL seguono spesso la filosofia BASE, che comprende:

- ***Basically Available (disponibilità di base***): il sistema garantisce la disponibilità nella maggior parte dei casi.
- ***Soft State (stato flessibile):*** i dati possono cambiare nel tempo anche senza interventi degli utenti, per effetto della replica.
- ***Eventually Consistent (coerenza eventuale***): in assenza di nuovi aggiornamenti, i dati diventeranno coerenti tra i nodi.

Questo modello permette di ottenere un throughput elevato e tolleranza ai guasti, soprattutto nelle infrastrutture cloud e distribuite.

2.2. Principali modelli di dati NoSQL

I database NoSQL vengono normalmente suddivisi in quattro grandi categorie, ciascuna progettata per specifiche modalità di memorizzazione e accesso:

1. Archivi chiave-valore
2. Database orientati ai documenti
3. Database a famiglie di colonne
4. Database a grafo.

Gli archivi chiave-valore rappresentano la forma più semplice di sistemi NoSQL. Ogni record comprende una chiave univoca associata a un valore, che può essere un oggetto binario arbitrario, un documento JSON o una struttura serializzata. Il database non interpreta la struttura interna del valore, ma lo recupera tramite la chiave. Questa semplicità architetturale consente letture e scritture estremamente rapide, senza JOIN, indicizzazioni complesse o vincoli relazionali. Le caratteristiche principali sono quindi l'elevata velocità e le limitate possibilità di interrogazione***.***

Esempi di archivi chiave-valore sono Redis, Amazon DynamoDB e Riak KV. Redis è un archivio di strutture dati in memoria, con supporto alla persistenza e alla messaggistica pub/sub. Amazon DynamoDB è un servizio cloud completamente gestito, ispirato all'architettura Dynamo. Riak KV è un database chiave-valore distribuito incentrato sulla tolleranza ai guasti.

Nei sistemi di monitoraggio ambientale, gli archivi chiave-valore sono particolarmente utili quando la bassa latenza è più importante delle interrogazioni complesse. Per esempio, una dashboard in tempo reale mostra spesso la lettura più recente di ciascun sensore. Anziché scorrere l'intera serie temporale, il sistema può memorizzare l'ultimo valore sotto una chiave deterministica.

Consideriamo il seguente esempio con Redis:

```
SET sensor_1001_last_reading '{"temperature": 12.6, "humidity": 78.5}'
```

In questa istruzione, la chiave sensor_1001_last_reading identifica univocamente il valore memorizzato, una stringa JSON con le ultime misurazioni ambientali. Quando la dashboard richiede l'ultima lettura, esegue:

```
GET sensor_1001_last_reading
```

Il database restituisce immediatamente il valore memorizzato senza filtri né aggregazioni. L'efficienza deriva dall'archiviazione in memoria e dall'indicizzazione basata su hash di Redis, che permettono recuperi in tempo costante.

I sistemi chiave-valore vengono utilizzati anche per contatori, ad esempio per registrare il numero di rilevamenti di specie in un breve intervallo temporale. Per esempio:

```
INCR site_001_detection_counter
```

Questo comando incrementa atomicamente il contatore associato alla chiave. L'atomicità assicura la correttezza anche quando più processi aggiornano il contatore contemporaneamente, un requisito essenziale nei sistemi ecologici ad alta frequenza di eventi.

Pur offrendo grande velocità e semplicità, gli archivi chiave-valore hanno capacità analitiche limitate. Non supportano efficientemente query come «trovare tutti i sensori con temperatura superiore a 30 °C», poiché i valori sono opachi al sistema salvo la presenza di ulteriori livelli di indicizzazione. Sono pertanto particolarmente adatti come componenti complementari di architetture che impiegano più tecnologie di database.

I database orientati ai documenti archiviano i dati in formati semistrutturati e flessibili, come JSON, BSON o XML. Ogni documento racchiude le informazioni su un'entità, inclusi oggetti e array annidati. Diversamente dagli archivi chiave-valore, questi database comprendono la struttura interna dei record e consentono indicizzazione e interrogazioni sugli attributi annidati.

Tra i casi d'uso figurano sistemi di gestione dei contenuti, cataloghi e-commerce, profili utente e registri di eventi. MongoDB è uno dei database documentali più diffusi, con schemi flessibili e interrogazioni espressive. CouchDB utilizza JSON per i documenti e HTTP per le API, privilegiando replica e funzionamento offline. ArangoDB è un database multimodello per documenti, chiave-valore e grafi. Le caratteristiche principali dei database documentali sono:

- uno schema flessibile che favorisce agilità e iterazioni rapide;
- linguaggi di interrogazione ricchi e indicizzazione anche di campi annidati;
- scalabilità orizzontale tramite sharding e insiemi di repliche.

Nelle applicazioni ambientali, i database documentali sono adatti alle osservazioni ecologiche semistrutturate, la cui struttura può variare tra siti o evolvere nel tempo. Una stazione può misurare temperatura e umidità, mentre un'altra registra anche salinità e radiazione UV.

Consideriamo il seguente esempio MongoDB:

```
db.sensor_readings.insertOne({
```

```
    sensor_id: "sensor_1001",
    site_id: "wetland_A",
    timestamp: ISODate("2025-10-15T09:12:00Z"),
    temperature: 12.6,
    humidity: 78.5,
    soil_moisture: 0.21
});
```

Questo documento rappresenta una singola osservazione ambientale. MongoDB la memorizza senza richiedere uno schema predefinito. Se compare un nuovo parametro del sensore, come la pressione atmosferica, è sufficiente aggiungerlo:

```
db.sensor_readings.insertOne({
    sensor_id: "sensor_1002",
    site_id: "wetland_A",
    timestamp: ISODate("2025-10-15T09:15:00Z"),
    temperature: 13.1,
    humidity: 76.4,
    air_pressure: 1012.3
});
```

Non serve modificare lo schema. Questa flessibilità è fondamentale nei sistemi ecologici, nei quali i parametri di monitoraggio possono cambiare con l'evoluzione degli obiettivi di ricerca.

I database documentali consentono anche di indicizzare campi specifici:

```
db.sensor_readings.createIndex({ site_id: 1, timestamp: -1 });
```

Questo indice composto permette di recuperare efficientemente le misurazioni più recenti di un sito. Il primo campo (site_id) organizza logicamente i dati per località, mentre il secondo (timestamp) consente l'ordinamento cronologico. Quando viene eseguita una query come:

```
db.sensor_readings.find({ site_id: "wetland_A" }).sort({ timestamp: -1
}).limit(10);
```

MongoDB utilizza l'indice per restituire rapidamente i record più recenti senza esaminare l'intera collezione.

Il modello documentale combina flessibilità ed espressività delle interrogazioni, ed è uno dei paradigmi NoSQL più diffusi nei sistemi ambientali.

I database a famiglie di colonne organizzano i dati in gruppi di colonne correlate per consentire accessi efficienti. Tra gli utilizzi rientrano l'archiviazione di serie temporali, l'aggregazione di log ed eventi e l'analisi su larga scala. Apache Cassandra è altamente disponibile, scalabile e tollerante ai guasti. HBase è un'implementazione open source di Bigtable,

costruita sopra HDFS di Hadoop. ScyllaDB è una reimplementazione ad alte prestazioni di Cassandra in C++. Le caratteristiche principali sono:

- ottime prestazioni in scrittura e scalabilità;
- livelli di coerenza configurabili;
- ottimizzazione per dati a colonne ampie (wide-column);
- idoneità ai carichi di lavoro analitici.

Questo modello è particolarmente efficiente per i dati ambientali in serie temporali, in cui le misurazioni vengono registrate sequenzialmente e spesso interrogate per intervallo temporale.

Consideriamo la definizione della seguente tabella Cassandra:

```
CREATE TABLE ecology.sensor_data (
    site_id text,
    day text,
    timestamp timestamp,
    temperature float,
    humidity float,
    soil_moisture float,
    PRIMARY KEY ((site_id, day), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

La chiave primaria comprende una chiave di partizione (site_id, day) e una colonna di clustering timestamp. La chiave di partizione determina la distribuzione dei dati tra i nodi del cluster. Tutte le misurazioni di uno specifico sito e giorno sono memorizzate insieme, consentendo interrogazioni efficienti per i riepiloghi giornalieri.

La colonna di clustering definisce l'ordine fisico delle righe all'interno di una partizione. Poiché i timestamp sono ordinati in senso decrescente, recuperare le letture più recenti è efficiente:

```
SELECT *
FROM ecology.sensor_data
WHERE site_id = 'wetland_A'
AND day = '2025-10-15'
LIMIT 5;
```

La query non esamina l'intero dataset: Cassandra accede direttamente alla partizione pertinente e recupera gli ultimi record sfruttando l'ordine di clustering.

I database a famiglie di colonne eccellono nell'acquisizione di dati ad alta intensità. Quando migliaia di sensori trasmettono letture ogni secondo, Cassandra distribuisce le scritture sui nodi mediante hashing coerente, evitando colli di bottiglia. Risulta quindi adatto alle infrastrutture di monitoraggio ambientale su scala nazionale.

I database a grafo utilizzano nodi, archi e proprietà per rappresentare e memorizzare i dati. Sono progettati per esplorare relazioni e pattern tra entità. Tra i casi d'uso figurano reti sociali e sistemi di raccomandazione, rilevamento di frodi, analisi di reti, knowledge graph e modelli semantici.

Neo4j è tra i database a grafo più diffusi e utilizza il linguaggio Cypher. Amazon Neptune è un servizio gestito che supporta grafi a proprietà e RDF. OrientDB combina modelli documentali e a grafo. Le caratteristiche più rilevanti sono:

- ottimizzazione per l'attraversamento di relazioni complesse;
- linguaggi potenti per la ricerca di percorsi e il riconoscimento di pattern;
- adeguatezza ai dati fortemente connessi, in cui le relazioni sono importanti quanto le entità.

I database a grafo modellano i dati come nodi collegati da relazioni (archi); entrambi possono possedere proprietà. Questa rappresentazione è naturale per i sistemi ecologici, nei quali le interazioni tra entità sono spesso importanti quanto le entità stesse.

Per esempio, le specie vivono in determinati siti, i siti appartengono a regioni e le regioni possono fare parte di reti di aree protette. Rappresentare queste interazioni in tabelle relazionali richiede numerose operazioni JOIN, che possono diventare onerose all'aumentare della profondità delle relazioni.

In Neo4j è possibile modellare esplicitamente queste relazioni:

```
MERGE (s:Species {name: "Anas platyrhynchos"})
MERGE (site:Site {id: "wetland_A"})
MERGE (region:Region {name: "Northern Basin"})
MERGE (s)-[:OBSERVED_AT]->(site)
MERGE (site)-[:LOCATED_IN]->(region);
```

Ogni istruzione MERGE garantisce che il nodo o la relazione esista senza creare duplicati. Le relazioni esplicite OBSERVED_AT e LOCATED_IN consentono interrogazioni intuitive di attraversamento.

Per calcolare la biodiversità per regione si può eseguire la seguente query:

```
Per calcolare la biodiversità per regione, è possibile eseguire la
seguente query:
RETURN region.name, COUNT(DISTINCT s) AS biodiversity
ORDER BY biodiversity DESC;
```

Questa query attraversa la struttura del grafo anziché eseguire JOIN relazionali. Poiché i database a grafo memorizzano direttamente le adiacenze, tali operazioni possono essere efficienti anche nelle reti complesse.

I modelli a grafo sono particolarmente utili per:

- modellare le reti trofiche;
- analizzare le reti migratorie;
- valutare la connettività degli habitat;
- studiare le interazioni tra ecosistemi.

Il loro punto di forza è l'analisi delle relazioni piuttosto che l'aggregazione massiva di dati numerici.

2.3. Database NoSQL e database relazionali a confronto

A differenza dei database relazionali, che richiedono normalizzazione e definizioni rigorose degli schemi, i database NoSQL favoriscono progettazioni denormalizzate e guidate dall'applicazione. La modellazione dipende fortemente dalle modalità di accesso ai dati. I principi fondamentali sono:

- *Progettare per le query,* non soltanto per l'integrità dei dati: il modello deve riflettere le operazioni di lettura e scrittura più frequenti.
- *Adottare la denormalizzazione:* memorizzare dati ridondanti per ridurre i JOIN complessi e migliorare le prestazioni.
- *Utilizzare aggregati:* raggruppare dati correlati (per esempio i dettagli di un ordine) in un unico documento o in una famiglia di colonne.
- *Incorporare oppure referenziare:* scegliere tra incorporare dati correlati (letture più rapide) o utilizzare riferimenti (maggiore normalizzazione).

Mentre SQL fornisce un linguaggio di interrogazione unitario, i sistemi NoSQL adottano meccanismi diversi:

1. Ricerche per chiave negli archivi chiave-valore.
2. Interrogazioni simili a JSON nei database documentali.
3. CQL (Cassandra Query Language) negli archivi a colonne.
4. Linguaggi di attraversamento dei grafi, come Cypher o Gremlin.

Per ottimizzare le query si impiegano indici hash, B-tree e per la ricerca testuale. Alcuni sistemi, come MongoDB, supportano indici composti, geospaziali e testuali; altri, come Cassandra,

utilizzano chiavi di partizione e colonne di clustering. Le architetture moderne adottano spesso la persistenza poliglotta, cioè più tecnologie di database nella stessa applicazione: Redis per cache e sessioni, MongoDB per contenuti e profili, Cassandra per eventi ad alta frequenza e Neo4j per le relazioni. L'integrazione avviene mediante API, pipeline di dati o code di messaggi. Provider cloud come AWS e Azure offrono soluzioni NoSQL gestite, quali DynamoDB e Cosmos DB. Nonostante i vantaggi, NoSQL comporta nuove sfide:

- complessità della gestione della coerenza nei sistemi distribuiti;
- compromessi di modellazione che richiedono una comprensione approfondita delle modalità di accesso;
- assenza di standardizzazione fra linguaggi di interrogazione e API;
- gestione della sicurezza e delle transazioni diversa dalle convenzioni RDBMS.

Le tendenze emergenti mirano a superare queste difficoltà. I database NewSQL, come CockroachDB e Google Spanner, combinano la scalabilità di NoSQL con transazioni ACID. I database multimodello, come ArangoDB e OrientDB, consentono di gestire documenti, grafi e coppie chiave-valore nello stesso motore. Il futuro della gestione non relazionale comprende architetture ibride che si adattano ai carichi di lavoro e integrano ottimizzazione delle query con l'IA, sharding automatizzato e interoperabilità fra modelli. Le tecnologie NoSQL hanno trasformato la gestione dei dati in ambienti distribuiti e ad alte prestazioni, offrendo flessibilità, scalabilità e adattabilità. Comprenderne principi, compromessi e applicazioni è essenziale per gli sviluppatori e i progettisti di dati.

2.4. Database NoSQL nel contesto ambientale

La progettazione relazionale tradizionale definisce tabelle, relazioni e vincoli prima dell'implementazione dell'applicazione. I sistemi NoSQL, invece, adottano una modellazione guidata dall'applicazione: si analizza come i dati verranno acceduti, interrogati e distribuiti. Questo è particolarmente importante nei sistemi ambientali, dove le query riguardano spesso misurazioni recenti, specifiche regioni geografiche o indicatori aggregati degli ecosistemi. La progettazione per tali accessi richiede flessibilità strutturale e distributiva.

Consideriamo, per esempio, un semplice schema relazionale per il monitoraggio ambientale:

```
CREATE TABLE sensor_readings (
    sensor_id VARCHAR(50),
    measurement_time TIMESTAMP,
    temperature FLOAT,
    humidity FLOAT,
    soil_moisture FLOAT,
```

```
    PRIMARY KEY (sensor_id, measurement_time)
);
```

Sebbene questo schema sembri inizialmente sufficiente, l'introduzione di nuove variabili ambientali crea difficoltà. L'aggiunta di attributi richiede modifiche allo schema:

```
ALTER TABLE sensor_readings
ADD air_pressure FLOAT;
```

Nelle infrastrutture di monitoraggio in rapida evoluzione, i sensori vengono aggiornati frequentemente e modifiche ripetute dello schema possono risultare onerose e problematiche.

Un database documentale come MongoDB consente invece di memorizzare lo stesso record senza imporre uno schema predefinito:

```
db.sensor_readings.insertOne({
    sensor_id: "sensor_1001",
    timestamp: ISODate("2025-10-15T09:12:00Z"),
    temperature: 12.6,
    humidity: 78.5,
    soil_moisture: 0.21,
    air_pressure: 1012.3
});
```

La struttura è flessibile: se compare un nuovo parametro, lo si aggiunge ai documenti futuri senza modificare quelli esistenti. È uno dei vantaggi fondamentali di NoSQL nei sistemi ambientali dinamici.

Oltre alla flessibilità dello schema, un'altra distinzione centrale è la scalabilità distribuita. Le piattaforme ambientali raccolgono spesso dati da centinaia o migliaia di sensori dislocati in aree geografiche diverse. Un database centralizzato può diventare un collo di bottiglia. I sistemi NoSQL distribuiti affrontano il problema ripartendo i dati su più nodi. In Cassandra, per esempio, la distribuzione avviene tramite chiavi di partizione:

```
CREATE TABLE sensor_data (
    site_id text,
    day text,
    timestamp timestamp,
    temperature float,
    humidity float,
    PRIMARY KEY ((site_id, day), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

La chiave di partizione composta (site_id, day) raggruppa le misurazioni giornaliere di ciascun sito, distribuendole tra i nodi del cluster. All'aumentare dei siti monitorati si possono aggiungere nuovi nodi senza modificare lo schema, realizzando una scalabilità orizzontale.

I sistemi distribuiti comportano però dei compromessi. Il teorema CAP mostra che, in presenza di partizioni della rete, non si possono garantire simultaneamente una coerenza rigorosa e la piena disponibilità. Molti sistemi ambientali privilegiano la disponibilità, soprattutto durante le emergenze. Per esempio, gli insiemi di repliche MongoDB possono essere configurati per garantire ridondanza:

```
rs.initiate({
   _id: "replicaSet1",
   members: [
      { _id: 0, host: "node1:27017" },
      { _id: 1, host: "node2:27017" },
      { _id: 2, host: "node3:27017" }
   ]
});
```

In questa configurazione, se il nodo primario si guasta, un nodo secondario viene promosso automaticamente, consentendo al sistema di continuare a funzionare. Durante il passaggio possono verificarsi incoerenze temporanee, ma il servizio resta operativo: una condizione essenziale per il monitoraggio ecologico in tempo reale.

I diversi modelli NoSQL offrono strutture adatte a dati ambientali distinti. I database documentali gestiscono osservazioni ecologiche semistrutturate, gli archivi chiave-valore forniscono cache velocissime per dashboard in tempo reale, i database a famiglie di colonne supportano serie temporali ad alta frequenza e i database a grafo rappresentano relazioni ecologiche. In Neo4j, per esempio, specie e habitat possono essere collegati esplicitamente:

```
MERGE (s:Species {name: "Anas platyrhynchos"})
MERGE (site:Site {id: "site_001"})
MERGE (s)-[:OBSERVED_AT]->(site);
```

Questa rappresentazione consente interrogazioni di attraversamento complesse, ad esempio per identificare le specie osservate negli habitat collegati:

```
MATCH (s:Species)-[:OBSERVED_AT]->(site:Site)
RETURN site.id, COUNT(DISTINCT s) AS biodiversity
ORDER BY biodiversity DESC;
```

Operazioni basate sui grafi possono diventare difficili da realizzare efficientemente nei sistemi relazionali tradizionali quando aumenta la profondità delle relazioni.

In pratica, le moderne piattaforme ambientali raramente adottano un solo paradigma di database. Utilizzano invece la persistenza poliglotta, combinando sistemi specializzati in un'architettura unificata. Per esempio, Kafka può gestire l'acquisizione in tempo reale, Cassandra memorizzare le misurazioni, MongoDB gestire i metadati e Neo4j analizzare le relazioni. L'integrazione tra componenti avviene tramite API o pipeline di streaming anziché tramite JOIN relazionali monolitici.

Questa strategia permette a ciascun modello di operare secondo i propri punti di forza, mantenendo la coerenza complessiva del sistema. La distribuzione delle responsabilità tra motori specializzati migliora scalabilità, tolleranza ai guasti e capacità di adattamento ai nuovi requisiti ambientali.

In definitiva, i fondamenti di NoSQL e delle architetture distribuite riflettono un'evoluzione più ampia della gestione dei dati ambientali. Invece di imporre schemi centralizzati e rigidi, i sistemi moderni adottano modelli distribuiti, scalabili e flessibili, adatti alle reti di sensori ad alta frequenza e ai dataset ecologici eterogenei. Comprendere questi principi è essenziale per progettare piattaforme resilienti per il monitoraggio in tempo reale, l'analisi di lungo periodo e decisioni ambientali sostenibili.

3. Caso di studio: piattaforma per il monitoraggio in tempo reale della biodiversità e dell'ambiente

3.1. Quadro concettuale e obiettivi

Questo caso di studio illustra come un'architettura NoSQL distribuita possa supportare non solo la memorizzazione scalabile dei dati ambientali, ma anche un'interpretazione ecologica strutturata. L'obiettivo è progettare e realizzare una piattaforma di monitoraggio ambientale e della biodiversità in tempo reale, capace di trasformare osservazioni ambientali eterogenee in regimi ecologici identificabili.

Il sistema integra dati di sensori ambientali ad alta frequenza, serie temporali meteorologiche e registrazioni della presenza di specie. Queste fonti differiscono per struttura, risoluzione temporale e significato semantico. La sfida principale consiste quindi nell'armonizzarle in un flusso analitico unitario, preservando scalabilità, flessibilità e interpretabilità.

Esaminiamo la logica di una pipeline analitica che procede dall'acquisizione e memorizzazione distribuite fino alla costruzione delle caratteristiche e all'identificazione non supervisionata dei regimi. L'obiettivo finale è riconoscere stati ecologici strutturali (stabile, transitorio e critico) a partire da indicatori ambientali e di biodiversità integrati.

3.2. Domini dei dati e caratteristiche strutturali

La piattaforma di monitoraggio integra tre domini principali di dati.

Il primo comprende le misurazioni ambientali basate su IoT. Umidità del suolo, temperatura superficiale e precipitazioni vengono registrate con risoluzione al minuto in più zone umide. Si tratta di serie temporali di grande volume, caratterizzate da un rigoroso ordinamento temporale.

Il secondo dominio comprende dati meteorologici recuperati ogni ora da un'API esterna, che descrivono le condizioni atmosferiche influenti sui fenomeni di stress idrologico.

Il terzo comprende le registrazioni di presenza di specie importate da archivi di osservazioni naturalistiche. Sono documenti semistrutturati con identificativi delle specie, coordinate geografiche, marcature temporali e metadati contestuali.

Ogni dominio presenta proprietà strutturali e frequenze di aggiornamento differenti, come illustrato nella Tabella 1.

Tabella 1. Caratteristiche strutturali e ruoli ecologici dei domini di dati integrati

Dominio dei dati	**Struttura dei dati**	**Risoluzione temporale**	**Ruolo ecologico**
Sensori ambientali IoT	Serie temporali	Al minuto	Stress idrologico e termico locale
API meteorologica	Serie temporali	Oraria	Forzante climatica
Registrazioni di biodiversità	Documenti semistrutturati	Basato su eventi	Risposta biologica
Species-Site Relationships	Graph structure	Event-driven	Ecological connectivity

Questa eterogeneità richiede una strategia di persistenza poliglotta, nella quale i diversi modelli NoSQL vengono scelti in base alle esigenze strutturali e analitiche.

3.3. Architettura distribuita e flusso dei dati

Consideriamo l'architettura distribuita e il flusso complessivo dei dati rappresentati schematicamente nella Figura 1.

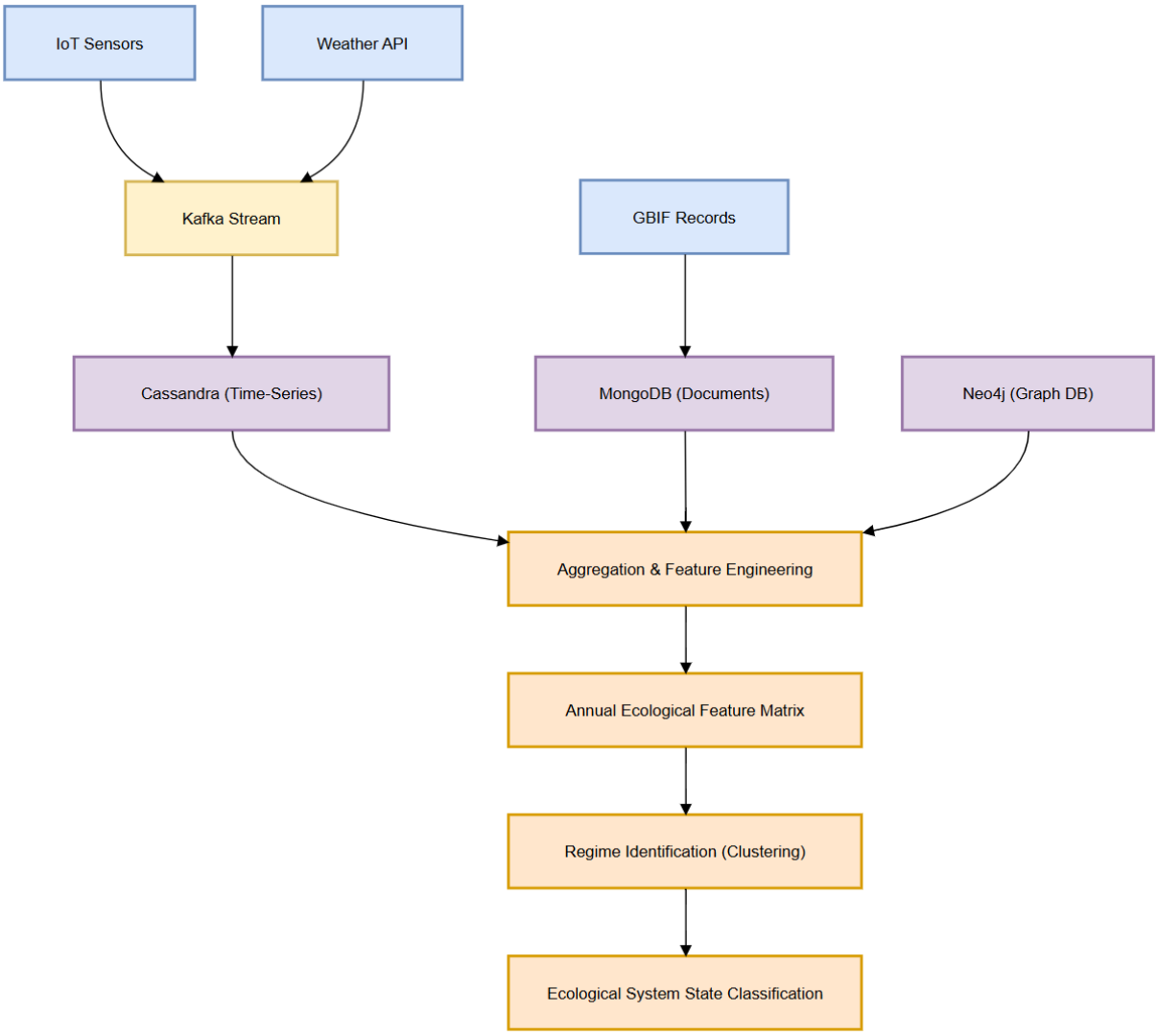

Figura 1. Architettura della piattaforma e flusso dei dati

L'architettura separa acquisizione, memorizzazione specializzata, integrazione e modellazione analitica in livelli distinti ma interconnessi.

I flussi IoT e meteorologici vengono inizialmente gestiti da un *livello di streaming basato su Kafka*. Ciò disaccoppia i produttori di dati dai sistemi di memorizzazione e migliora la resilienza, garantendo una persistenza temporanea durante eventuali interruzioni.

I dati ambientali in serie temporali vengono memorizzati in *Cassandra.* La sua architettura basata su partizioni supporta scalabilità orizzontale e un'elevata capacità di scrittura, risultando adatta all'acquisizione continua di misurazioni al minuto.

Le registrazioni della presenza delle specie sono archiviate in MongoDB. Il modello documentale consente di rappresentare flessibilmente attributi delle specie in evoluzione e supporta indici geospaziali per interrogazioni spaziali.

Le relazioni ecologiche tra specie, siti e regioni sono modellate in Neo4j. La rappresentazione a grafo consente interrogazioni efficienti di attraversamento che descrivono la connettività degli ecosistemi.

Questa architettura a livelli garantisce che ogni sistema di memorizzazione svolga attività coerenti con i propri punti di forza, contribuendo a un risultato analitico comune.

3.4. Progettazione della memorizzazione e aggregazione degli indicatori

Come spiegato in precedenza, le misurazioni ambientali in serie temporali vengono memorizzate in Cassandra. È possibile utilizzare una chiave di partizione composta:

```
CREATE TABLE ecology.environmental_readings (
  site_id text,
  day text,
  timestamp timestamp,
  temperature float,
  precipitation float,
  soil_moisture float,
  PRIMARY KEY ((site_id, day), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

La chiave di partizione (site_id, day) distribuisce i dati tra i nodi, mantenendo insieme le misurazioni giornaliere di ciascun sito. La colonna di clustering timestamp garantisce l'ordine cronologico all'interno di ogni partizione.

Per ottenere indicatori ambientali annuali, le query di aggregazione condensano i dati al minuto in variabili strutturate di stress:

```
SELECT
  site_id,
  EXTRACT(YEAR FROM timestamp) AS year,
  AVG(temperature) AS avg_temp_year,
  COUNT(CASE WHEN precipitation < 1 THEN 1 END) AS dry_days_year,
```

```
  COUNT(CASE WHEN soil_moisture < 0.20 THEN 1 END) AS drought_events
FROM ecology.environmental_readings
GROUP BY site_id, EXTRACT(YEAR FROM timestamp);
```

Questi indicatori quantificano lo stress termico, il deficit di precipitazioni e la frequenza dello stress idrologico.

Le registrazioni di presenza della biodiversità vengono memorizzate in MongoDB come documenti JSON flessibili:

```
db.species_occurrences.insertOne({
  species: "Rana temporaria",
  site_id: "wetland_A",
  eventDate: ISODate("2023-05-14"),
  coordinates: {
    type: "Point",
    coordinates: [30.5234, 50.4501]
  }
});
```

La ricchezza di specie per sito e anno viene calcolata tramite una pipeline di aggregazione:

```
db.species_occurrences.aggregate([
  {
    $group: {
      _id: { site_id: "$site_id", year: { $year: "$eventDate" }},
      species_set: { $addToSet: "$species" }
    }
  },
  {
    $project: {
      site_id: "$_id.site_id",
      year: "$_id.year",
      species_richness: { $size: "$species_set" }
    }
  }
])
```

Questo processo produce indicatori della risposta della biodiversità, allineati alle variabili di stress ambientale.

3.5. Costruzione della matrice annuale delle caratteristiche ecologiche

Gli indicatori ambientali e di biodiversità aggregati vengono uniti in una rappresentazione analitica comune: la matrice annuale delle caratteristiche ecologiche (Tabella 2). Ogni riga corrisponde a un’osservazione sito-anno e ogni colonna a un attributo ecologico derivato.

Tabella 2. Matrice annuale delle caratteristiche ecologiche

Sito	Anno	Temperatura media	Giorni secchi	Eventi di siccità	Ricchezza di specie
Zona umida A	2018	14.2	210	2	58
Zona umida A	2019	15.6	242	5	47
Zona umida A	2020	17.1	275	9	32

La matrice integra stress ambientale e risposta della biodiversità nella stessa unità osservativa. Prima del clustering, le caratteristiche vengono standardizzate per garantire contributi equilibrati:

```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(
    inputCol="raw_features",
    outputCol="scaled_features",
    withStd=True,
    withMean=True
)
```

Questa standardizzazione impedisce a una singola variabile di dominare il clustering a causa delle differenze di scala.

3.6. Identificazione e analisi dei regimi ecologici

Alla matrice standardizzata viene applicato un clustering non supervisionato per identificare i regimi ecologici strutturali.

```
kmeans = KMeans(k=3, seed=1)
model = kmeans.fit(data)
predictions = model.transform(data)
```

I centroidi dei cluster rivelano tre regimi ecologici (Tabella 3).

Tabella 3. Regimi ecologici

Cluster	Temperatura media	Eventi di siccità	Ricchezza di specie	Interpretazione del regime

1	Moderata	Basso	Alto	Stabile
2	Elevata	Medio	Moderata	Transitorio
3	Alto	Frequenti	Basso	Stress critico

La distribuzione temporale dei cluster mostra i passaggi tra regimi (Tabella 4).

Tabella 4. Cluster per anno

Anno	**Cluster**	**Anno**	**Cluster**	**Anno**	**Cluster**
2015	1	2018	2	2021	3
2016	1	2019	3	2022	3
2017	2	2020	3	2023	3

La matrice di transizione conferma la dipendenza dal percorso (Tabella 5).

Tabella 5. Matrice di transizione

Da \ A	**Cluster 1**	**Cluster 2**	**Cluster 3**
Cluster 1	1	1	0
Cluster 2	0	1	1
Cluster 3	0	0	4

Una volta che l'ecosistema entra nel regime critico, la persistenza aumenta, suggerendo una resilienza ridotta. Questa informazione può essere cruciale per le decisioni di gestione ecologica.

Il caso di studio mostra quindi che i sistemi NoSQL non sono semplici soluzioni di archiviazione scalabili. Combinati con aggregazioni strutturate e costruzione delle caratteristiche, consentono una modellazione ecologica multidimensionale. L'integrazione di indicatori di stress ambientale e misure di risposta della biodiversità rivela regimi ecologici strutturali che riflettono trasformazioni ambientali persistenti.

Mediante memorizzazione distribuita specializzata, costruzione integrata delle caratteristiche e identificazione dei regimi, la piattaforma trasforma i dati ambientali grezzi in stati interpretabili del sistema. Questi stati forniscono informazioni operative per la gestione ecologica, la valutazione dei rischi e il rilevamento precoce di situazioni critiche.

Esercitazioni pratiche

Attività 1

1. Comporre un insieme eterogeneo di dati ambientali ed ecologici utilizzando le fonti indicate nella tabella seguente.

Dataset	Descrizione	Collegamento
GBIF (Global Biodiversity Information Facility)	Un vasto archivio di presenze di specie e osservazioni della biodiversità	https://www.gbif.org/
European Environment Agency – DataHub	Dataset ambientali ed ecologici di tutta Europa	https://www.eea.europa.eu/en/datahub
UKCEH (UK Centre for Ecology & Hydrology)	Dataset di monitoraggio ambientale; serie temporali e metadati spaziali	https://www.ceh.ac.uk/data
NEON (National Ecological Observatory Network)	Open-science ecological data resources	https://libguides.cmich.edu/lifesciencedata/ecological

2. Analizzare le caratteristiche del dataset ottenuto.

3. Raccomandare un'architettura di memorizzazione NoSQL per gestire il dataset.

Risultato richiesto: una relazione sui risultati.

Criteri di valutazione: completezza dei risultati e chiarezza dell'interpretazione.

Attività 2

Immagina di collaborare con un'agenzia ambientale che monitora la presenza delle specie, parametri degli habitat (temperatura, umidità, umidità del suolo) e osservazioni basate su eventi (per esempio attivazioni di fototrappole e allarmi dei sensori). Proponi un insieme di tecnologie NoSQL per progettare un sistema software di monitoraggio che acquisisca flussi di dati da sensori e fototrappole in molti luoghi, supporti interrogazioni flessibili e analizzi i parametri ecologici.

Risultato richiesto: una relazione sui risultati.

Criteri di valutazione: completezza dei risultati e chiarezza dell'interpretazione.

Verifica dell'apprendimento

1. Risolvi il cruciverba.

Il teorema *CAP* afferma che in un sistema di dati distribuito non è possibile garantire contemporaneamente tutte e tre le proprietà seguenti.

C																	
A																	
P									—								

2. Rispondi alle domande a scelta multipla.

1. Quale delle seguenti affermazioni descrive meglio la principale motivazione dell'uso dei database NoSQL?

A. Imporre rigorosamente i vincoli di integrità relazionale

B. Gestire efficientemente dati non strutturati o semistrutturati su larga scala

C. Semplificare la progettazione degli schemi relazionali

D. Eliminare la necessità dei sistemi di gestione di database

Risposta corretta: B

2. Quale dei seguenti NON è un tipo di database NoSQL?

A. Archivio chiave-valore

B. Archivio documentale

C. Archivio a famiglie di colonne

D. Archivio di tabelle relazionali

Risposta corretta: D

3. In un database NoSQL orientato ai documenti, i dati vengono generalmente memorizzati in:

A. File CSV

B. Documenti simili a JSON

C. Record di lunghezza fissa

D. Tabelle con chiavi esterne

Risposta corretta: B

4. Quale dei seguenti database NoSQL è un esempio di archivio chiave-valore?

A. MongoDB

B. Cassandra

C. Redis

D. Neo4j

Risposta corretta: C

5. Quale modello NoSQL è più adatto a rappresentare dati di rete o basati sulle relazioni, come le interazioni tra specie e habitat?

A. Archivio a famiglie di colonne

B. Archivio chiave-valore

C. Database a grafo

D. Archivio documentale

Risposta corretta: C

6. Qual è il vantaggio principale della scalabilità orizzontale nei sistemi NoSQL?

A. Riduce la ridondanza normalizzando tutte le tabelle.

B. Permette di aumentare la capacità aggiungendo nodi, anziché potenziare l'hardware esistente.

C. Garantisce la piena conformità ACID nei sistemi distribuiti.

D. Converte automaticamente i dati non strutturati in formato relazionale.

Risposta corretta: B

7. Quale componente di un ecosistema NoSQL si occupa di acquisire flussi continui di dati?

A. Motore di indicizzazione

B. Elaboratore di query

C. Livello di acquisizione dati / streaming

D. Dashboard di reportistica

Risposta corretta: C

8. Nelle applicazioni di monitoraggio ecologico, un database per serie temporali è particolarmente utile per:

A. Gestire dati con relazioni complesse

B. Memorizzare letture sequenziali dei sensori e dati ambientali con marcature temporali

C. Memorizzare testo non strutturato

D. Eseguire ricerche testuali complete nei documenti

Risposta corretta: B

9. Quale delle seguenti tecnologie viene comunemente utilizzata per lo streaming e le pipeline di dati in tempo reale?

A. Apache Kafka

B. Microsoft Access

C. Oracle RDBMS

D. SQLite

Risposta corretta: A

10. Quale affermazione descrive meglio un database documentale come MongoDB?

A. Richiede schemi predefiniti e JOIN tra tabelle.

B. Memorizza e recupera i dati come documenti flessibili e annidati.

C. È ottimizzato per query relazionali complesse in SQL.

D. Non può gestire strutture di dati gerarchiche o annidate.

Risposta corretta: B

Criteri di valutazione: punteggio del quiz (1 punto per domanda, totale: 10 punti).

Riferimenti bibliografici e letture di approfondimento

1. Chodorow, K. (2019). MongoDB: The Definitive Guide (3rd ed.). O'Reilly Media.
2. Redmond, E., & Wilson, J. R. (2012). Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Pragmatic Bookshelf.
3. Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications (ICPCA), 363–366. IEEE.
4. Gajendran, S., & Cook, J. (2021). Learning NoSQL: A Hands-On Guide to Big Data Management Systems. Packt Publishing.
5. Strauch, C. (2011). NoSQL Databases. Stuttgart Media University.
6. Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional.

7. Tanenbaum, A. S., & van Steen, M. (2017). Distributed Systems: Principles and Paradigms (2nd ed.). Pearson.