Database

🗄️ L'Universo dei Database: SQL vs. NoSQL

Un **Database Management System (DBMS)** è un software progettato per creare, gestire, interrogare e mantenere i database. Agisce come un intermediario tra l'utente (o l'applicazione) e il database fisico, garantendo sicurezza, coerenza e astrazione dalla complessità hardware.

La scelta del DBMS è una delle decisioni architetturali più importanti nello sviluppo di un'applicazione. Storicamente, il mondo era dominato da un unico paradigma: quello relazionale. Oggi, il panorama è molto più variegato, diviso principalmente in due grandi famiglie: **SQL** e **NoSQL**.

🏛️ Il Mondo Relazionale (SQL): L'Ordine e la Struttura

I database relazionali (RDBMS) sono il paradigma dominante da decenni. Si basano sul modello relazionale, un modello matematico rigoroso introdotto da Edgar F. Codd nel 1970.

La Filosofia di Base

Il principio cardine è la **struttura**. I dati sono organizzati in **tabelle** (o relazioni), che sono insiemi di **righe** (o tuple) e **colonne** (o attributi). Le relazioni tra le diverse tabelle sono esplicitamente definite tramite l'uso di chiavi, garantendo coerenza e integrità.

Concetti Chiave

- **Schema Rigido (Schema-on-Write)**: La struttura della tabella (colonne, tipi di dato, vincoli) deve essere definita **prima** di inserire qualsiasi dato. Se si vuole modificare la struttura, è necessaria un'operazione di `ALTER TABLE`.
- **Tabelle**: Raccolte di dati omogenei. Esempio: una tabella `Studenti`.
- **Righe (Tuple)**: Rappresentano una singola istanza di un'entità. Esempio: la riga dello studente "Mario Rossi".
- **Colonne (Attributi)**: Rappresentano le proprietà di un'entità. Esempio: `Matricola`, `Nome`, `Cognome`.
- **Chiave Primaria (Primary Key)**: Un attributo (o un insieme di attributi) che identifica univocamente ogni riga della tabella. Non può essere `NULL`.
- **Chiave Esterna (Foreign Key)**: Un attributo in una tabella che corrisponde alla chiave primaria di un'altra tabella, creando così un legame logico (integrità referenziale).
- **Normalizzazione**: Il processo di progettazione per ridurre la ridondanza e migliorare l'integrità dei dati, come visto in precedenza (1NF, 2NF, 3NF, BCNF).

Il Linguaggio: SQL (Structured Query Language)

SQL è il linguaggio standard de facto per interagire con i database relazionali. È un linguaggio dichiarativo: l'utente specifica *cosa* vuole ottenere, non *come* il DBMS debba ottenerlo.

Il Modello di Coerenza: ACID

I database relazionali sono costruiti attorno alle proprietà **ACID**, che garantiscono l'affidabilità delle transazioni anche in caso di errori o crash.

- **Atomicity**: Le transazioni sono "tutto o niente".
- **Consistency**: I dati sono sempre in uno stato coerente e valido.
- **Isolation**: Le transazioni concorrenti non si influenzano a vicenda.
- **Durability**: I dati confermati (`COMMIT`) sono permanenti.

Scalabilità: Verticale (Scale-Up)

Tradizionalmente, per gestire più carico, un RDBMS viene scalato **verticalmente**: si aumenta la potenza della singola macchina (più CPU, più RAM, dischi più veloci). Questo approccio è potente ma ha un limite fisico e può diventare molto costoso.

Punti di Forza (Perché scegliere SQL?)

1. **Integrità dei Dati Garantita**: Grazie a schemi rigidi, tipi di dato e vincoli (PK, FK), è molto difficile inserire dati "sporchi" o incoerenti.
2. **Transazioni ACID**: Fondamentali per sistemi critici come quelli finanziari, e-commerce, gestionali.

3. **Potenza di Interrogazione**: SQL è un linguaggio maturo, potente e standardizzato che permette query complesse, aggregazioni e join tra più tabelle.
4. **Ecosistema Maturo**: Anni di sviluppo hanno prodotto strumenti, community e documentazione vastissimi.

Punti di Debolezza (Quando SQL è un limite?)

1. **Scarsa Flessibilità**: Lo schema rigido rende difficile gestire dati non strutturati o in rapida evoluzione.
2. **Scalabilità Orizzontale Complessa**: Distribuire un database relazionale su più macchine (sharding) è complesso da implementare e gestire.
3. **Object-Relational Impedance Mismatch**: La difficoltà nel mappare gli oggetti di un linguaggio di programmazione (come classi in Java o Python) alle tabelle relazionali.

Esempi di RDBMS: `MySQL`, `MariaDB` (un fork open-source di MySQL), `PostgreSQL` (noto per la sua estensibilità e aderenza agli standard SQL), `Oracle Database`, `Microsoft SQL Server`, `SQLite` (per applicazioni embedded e mobile).

🚀 Il Mondo Non Relazionale (NoSQL): La Flessibilità e la Scala

Il termine **NoSQL** (spesso interpretato come "Not Only SQL") si riferisce a una vasta famiglia di database che non seguono il modello relazionale. Sono emersi per rispondere alle esigenze delle applicazioni web moderne: gestione di enormi volumi di dati (Big Data), alta velocità di scrittura/lettura e gestione di dati non strutturati.

La Filosofia di Base

Il principio cardine è la **flessibilità** e la **scalabilità orizzontale**. I dati non devono necessariamente conformarsi a uno schema predefinito, e il sistema è progettato fin dall'inizio per essere distribuito su un cluster di macchine commodity (economiche).

Le Categorie Principali di NoSQL

NoSQL non è un'unica tecnologia, ma un insieme di approcci diversi:

1. Document Databases

- **Modello**: I dati sono memorizzati in **documenti**, tipicamente in formati come **JSON** o **BSON** (Binary JSON). Un documento è una struttura dati complessa e annidata (contiene oggetti, array, ecc.).
- **Analogia Relazionale**: Un documento è simile a una riga, ma può avere una struttura completamente diversa da un altro documento nella stessa "collezione" (l'equivalente di una tabella).
- **Uso Tipico**: Content management, profili utente, cataloghi di prodotti.
- **Esempio di DBMS**: `MongoDB`, `CouchDB`.

```
// Esempio di documento Utente in MongoDB
{
  "_id": " ObjectId('...') ",
  "username": "mrossi",
  "email": "mario.rossi@email.com",
  "indirizzi": [
    { "tipo": "casa", "via": "Via Roma 1", "citta": "Milano" },
    { "tipo": "lavoro", "via": "Via Garibaldi 2", "citta": "Milano" }
  ],
  "interessi": ["sport", "tecnologia", "viaggi"]
}
```

2. Key-Value Stores

- **Modello**: Il più semplice. I dati sono una collezione di coppie **chiave-valore**. La chiave è unica e il valore può essere qualsiasi cosa (una stringa, un JSON, un'immagine). Il database non conosce la struttura del valore.
- **Analogia**: Un gigantesco dizionario o hash map.
- **Uso Tipico**: Caching, gestione delle sessioni utente, dati in tempo reale.
- **Esempio di DBMS**: `Redis` (estremamente veloce, in memoria), `Amazon DynamoDB`.

3. Column-Family Stores

- **Modello**: I dati sono organizzati in righe e colonne, ma a differenza dei database relazionali, le colonne non sono fisse per tutte le righe. Le colonne sono raggruppate in "famiglie di colonne".
- **Analogia**: Una tabella con righe "sparse", dove ogni riga può avere un insieme diverso di colonne.
- **Uso Tipico**: Big Data analytics, sistemi con altissima intensità di scrittura (logging, IoT).
- **Esempio di DBMS**: `Apache Cassandra`, `HBase`.

4. Graph Databases

- **Modello**: Progettati specificamente per gestire dati le cui **relazioni sono importanti tanto quanto i dati stessi**. Utilizzano una struttura a grafo con **nodi** (entità), **archi** (relazioni) e **proprietà** (attributi di nodi e archi).
- **Uso Tipico**: Social network, motori di raccomandazione, sistemi di rilevamento frodi.
- **Esempio di DBMS**: `Neo4j`, `Amazon Neptune`.

Il Modello di Coerenza: BASE e il Teorema CAP

I database NoSQL, essendo distribuiti, devono affrontare il **Teorema CAP**, che afferma che un sistema distribuito può garantire al massimo due delle seguenti tre proprietà:

- **Consistency (Coerenza)**: Tutti i nodi vedono gli stessi dati nello stesso momento.
- **Availability (Disponibilità)**: Ogni richiesta riceve una risposta (non un errore).
- **Partition Tolerance (Tolleranza alle Partizioni)**: Il sistema continua a funzionare anche se la comunicazione tra i nodi si interrompe.

Poiché la tolleranza alle partizioni (P) è un requisito non negoziabile su Internet, la scelta è tra C e A.

- I sistemi **SQL** tradizionali scelgono **CP**: sacrificano la disponibilità per garantire la coerenza.
- Molti sistemi **NoSQL** scelgono **AP**: sacrificano la coerenza forte per garantire la massima disponibilità.

Questo porta al modello **BASE**:

- **Basically Available**: Il sistema è sostanzialmente sempre disponibile.
- **Soft State**: Lo stato del sistema può cambiare nel tempo, anche senza input.
- **Eventual Consistency (Coerenza Eventuale)**: Il sistema raggiungerà uno stato coerente "prima o poi", una volta che gli aggiornamenti si saranno propagati a tutti i nodi.

Scalabilità: Orizzontale (Scale-Out)

I database NoSQL sono progettati per scalare **orizzontalmente**: per gestire più carico, si aggiungono nuove macchine al cluster (sharding e replica). Questo approccio è più economico e virtualmente illimitato.

⚖️ SQL vs. NoSQL: Tabella Comparativa

Criterio	SQL (Relazionale)	NoSQL (Non Relazionale)
Modello Dati	Tabelle con schema rigido e predefinito.	Vario: Documenti, Key-Value, Colonne, Grafi. Schema flessibile o inesistente.
Coerenza	Forte coerenza (modello **ACID**).	Coerenza eventuale (modello **BASE**). La coerenza forte è configurabile.
Scalabilità	Principalmente **verticale** (scale-up).	Principalmente **orizzontale** (scale-out).
Linguaggio Query	**SQL** (linguaggio standardizzato e potente).	Nessuno standard. Ogni DB ha la sua API o un suo linguaggio di query (es. MQL).
Integrità Dati	Molto alta, imposta a livello di database.	Generalmente gestita a livello di applicazione.
Casi d'Uso Tipici	Sistemi transazionali (banche, ERP), data warehousing.	Big Data, applicazioni real-time, content management, social media, IoT.
Esempi	MySQL, PostgreSQL, Oracle DB.	MongoDB, Redis, Cassandra, Neo4j.

Conclusione: Quale Scegliere? "It Depends."

Non esiste un "vincitore" assoluto. La scelta dipende interamente dal problema che si sta cercando di risolvere.

- **Scegli SQL se**:
  - I tuoi dati sono altamente strutturati e le relazioni sono complesse e importanti.
  - L'integrità dei dati e la coerenza transazionale (ACID) sono requisiti non negoziabili.

    - Hai bisogno di un linguaggio di query potente e standard per analisi complesse.
- **Scegli NoSQL se**:
    - I tuoi dati sono non strutturati, semi-strutturati o cambiano frequentemente.
    - La priorità assoluta è la scalabilità orizzontale e l'alta disponibilità.
    - Hai bisogno di performance estreme in lettura/scrittura per specifici pattern di accesso.
    - La coerenza forte immediata non è un requisito stringente.

Oggi, molte architetture complesse adottano un approccio ibrido, noto come **Polyglot Persistence**, utilizzando il database giusto per il lavoro giusto. Ad esempio, un'app di e-commerce potrebbe usare:

- Un **database SQL** (es. PostgreSQL) per gestire gli account utente, gli ordini e le transazioni finanziarie (dove ACID è fondamentale).
- Un **database a documenti** (es. MongoDB) per il catalogo prodotti (flessibile e facile da evolvere).
- Un **database key-value** (es. Redis) per la cache e la gestione delle sessioni utente.
- Un **database a grafo** (es. Neo4j) per il motore di raccomandazione ("chi ha comprato questo ha comprato anche...").