Definizione

Un database è un **sistema organizzato** per la **raccolta**, la **memorizzazione** e la **gestione** di dati in modo che possano essere facilmente **recuperati** e **aggiornati**. I dati in un database possono essere di diversi tipi, come numeri, testo, immagini, eccetera.

Applicazioni

1. **Sistemi di Gestione Aziendale (ERP):** I database sono ampiamente utilizzati nelle aziende per gestire informazioni finanziarie, risorse umane, inventario e altre attività aziendali.
2. **Siti Web e Applicazioni Online:** Quando visitate un sito web, ad esempio un negozio online, i dati sui prodotti, gli utenti, gli ordini vengono memorizzati in un database. Anche molte applicazioni che usiamo quotidianamente sul nostro telefono, come quelle di social media, utilizzano database per gestire dati utente.
3. **Sistemi di Prenotazione:** Se avete mai prenotato un volo, un hotel o un tavolo in un ristorante online, avete interagito con un database. Questi sistemi mantengono traccia delle disponibilità, delle prenotazioni e delle transazioni.
4. **Sistemi di Controllo del Traffico:** Nei sistemi di controllo del traffico stradale o ferroviario, i database vengono utilizzati per gestire informazioni sui veicoli, le strade, gli orari di transito, ecc.
5. **Settore della Sanità:** I database sono utilizzati per la gestione delle cartelle cliniche, delle informazioni sui pazienti, delle prescrizioni e altro ancora nel settore sanitario.

Parole chiave

- persistenza e durabilità
- efficienza
- concorrenza (più operazioni simultanee)
- accessibilità e sicurezza

Cosa NON è un DB:

- Excel
- File system
- Dati in memoria
- Documenti cartacei

Vocabolario base

- DBMS: Database management system (il server, software)
- Database: raccolta di dati specifici per una applicazione
- un DBMS può gestire più Database
- un server fisico può contenere un DBMS, ma...
- un DBMS può risiedere su più server fisici contemporaneamente (cluster)

Database relazionali (SQL)

Questi sono basati sul modello relazionale e utilizzano tabelle per organizzare i dati. SQL (Structured Query Language) è il linguaggio principale utilizzato per interrogare e gestire questi database.

Un database relazionale descrive le relazioni tra i diversi tipi di dati tipi di dati

- Cattura idee come quelle definite nelle regole di Affinità e di Raccolta
- Consente al software di rispondere alle interrogazioni su di esse

Esempi più noti

1. **MySQL:** Open-source, veloce, affidabile e ampiamente utilizzato. Supporta molte piattaforme e ha una vasta comunità di sviluppatori.
2. **Oracle Database:** Molto potente e scalabile, utilizzato principalmente in ambienti aziendali critici. Offre funzionalità avanzate e una gestione avanzata della sicurezza.
3. **Microsoft SQL Server:** Sviluppato da Microsoft, offre integrazione con altre tecnologie Microsoft. Ha una vasta gamma di funzionalità aziendali e di business intelligence.
4. **PostgreSQL:** Open-source, estensibile e conforme agli standard. È noto per il supporto avanzato delle funzioni SQL e la gestione della concorrenza.
5. **SQLite:** Leggero e incorporabile, è spesso utilizzato in applicazioni mobili e sistemi embedded. Non richiede un server separato.
6. **IBM Db2:** Ampiamente utilizzato in ambienti aziendali, offre funzionalità avanzate come la gestione delle analisi e l'integrazione con tecnologie IBM.
7. **SQLite:** Database leggero incorporato in molte applicazioni, senza la necessità di un processo server separato. Adatto per dispositivi embedded e applicazioni locali.

Concetti chiave

SQL

- Linguaggio di interrogazione strutturato
- Una query è il risultato di una richiesta di informazioni o di modifiche al database.
- SQL è il linguaggio che la maggior parte dei database relazionali (il tipo più comune di database) utilizza per l'interazione con i dati.

ACID

- Atomicity – transactions are either all or none (commit/rollback)
- Consistency – only valid data is saved
- Isolation – transactions should not affect each other
- Durability – written data will not be lost

CRUD

- Creare, leggere, aggiornare, cancellare (Create, read, update, delete)
- Le 4 principali operazioni eseguite sul database.

Database Schema

Rappresentazione logica della struttura di un database. **Definisce come i dati sono organizzati** e **relazionati** e **accessibili** all'interno del database. Lo schema del database fornisce una mappa o una descrizione dettagliata di come le **tabelle**, gli **attributi** e le **relazioni** sono strutturati all'interno del sistema di gestione del database.

Le principali componenti di uno schema di database includono:

1. **Tabelle:** Le tabelle rappresentano le entità principali del database, come ad esempio utenti, prodotti, ordini, ecc. Ogni tabella ha colonne che rappresentano gli attributi o le caratteristiche di quella entità.
2. **Attributi:** Gli attributi sono le caratteristiche specifiche associate alle colonne di una tabella. Ad esempio, in una tabella "Utenti", gli attributi potrebbero includere "Nome", "Cognome", "Indirizzo", ecc.
3. **Relazioni:** Lo schema del database definisce le relazioni tra le tabelle. Ad esempio, potrebbe esserci una relazione tra una tabella "Ordini" e una tabella "Prodotti" per indicare quali prodotti sono inclusi in ciascun ordine.
4. **Vincoli:** Gli schemi di database possono includere vincoli che specificano regole o restrizioni sui dati. Ad esempio, un vincolo di chiave primaria assicura che ogni riga in una tabella sia unica in base a un attributo specifico.
5. **Indici:** Gli indici possono essere definiti nello schema per ottimizzare le prestazioni delle query. Gli indici accelerano il processo di ricerca dei dati all'interno delle tabelle.
6. **Procedura di accesso ai dati:** In alcuni casi, gli schemi di database possono includere procedure o funzioni che definiscono come accedere e manipolare i dati all'interno del database.

Tabelle

Le tabelle (formalmente chiamate "**relazioni**") sono gli elementi costitutivi dei database relazionali.

- Le tabelle memorizzano i dati in **2D**, dove ogni riga riflette un'**istanza** di un **record** e ogni colonna riflette un aspetto degli **attributi** di tutte le istanze.
- Le righe possono anche essere chiamate "**tuple**".
- Le colonne possono anche essere chiamate "**campi**".

> **Ex**:
> Una tabella "studenti" può contenere (id studente, nome, cognome, classe, scuola, indirizzo di casa, ...), e ogni riga può rappresentare le informazioni di uno studente e ogni colonna della tabella rappresenta un'informazione per tutti gli studenti. E questo è chiamato "relazione".

Le istanze (righe) non sono necessariamente ordinate

> **Tutti i DB (relazionali) hanno tabelle, ma le tabelle non sono DB (excell)**

Entità

Una entità è un costrutto complesso che descrive un s/oggetto tramite una lista finita di sue proprietà. Le informazioni che compongono una entità possono essere sparsi su diverse tabelle

> **Ex:**
> Ho una tabella per gli ordini ed una per i singoli clienti.
> L'entità che definisce l'ordine fatto da un cliente si ottiene combinando i dati delle tabelle degli ordini + quella dei clienti

- Uniche: non posso avere due righe uguali per tutti gli attributi
- gli attributi devono esser "atomici" (non scomponibili) quanto più possibile

Chiavi (keys) e Vincoli

- Chiave primaria: campo (colonna) della tabella per cui volutamente tutti i valori sono diversi,
  - solitamente è designato a priori (colonna id)
  - può essere anche ottenuta come combinazione di campi
- Chiave esterna (foreign key) chiave primaria di un'altra tabella che viene referenziata (copiata) in un campo chiave per creare un **legame di stretta correlazione**
- Vincoli : Condizione imposta "by design" sui possibili valori che potrà assumere un campo in se o rispetto ad altri. Ex: controllo di sintassi (codice fiscale, email), limiti sui valori ammissibili (coordinate)

Indici

La principale funzione degli indici è accelerare il processo di ricerca e recupero dei dati, riducendo il numero di righe da esaminare durante l'esecuzione delle query. Gli indici sono una componente critica del tuning delle prestazioni dei database.

- velocizzano moltissimo la lettura e la ricerca (select / where / join / order by / union...)
- rallentano (di poco) la scrittura
- alleggeriscono moltissimo il carico computazionale della macchina

> **Aneddoto**:
> Il programma di un noto prodotto di fatturazione in alcuni casi poteva impiegare circa un giorno (24h) nella generazione di report e analisi complessi per clienti di grosse dimensioni. Questo a causa della mancanza degli indici nel DB.
> Introdotti gli indici lo stesso lavoro sulla stessa macchina è stato portato a termine in circa 3 minuti

Viste

Sono **tabelle virtuali** derivate da una o più tabelle di base. A differenza delle tabelle fisiche, le viste non contengono dati effettivi, ma rappresentano una "visione" dei dati presenti nelle tabelle di base. Le viste sono create per semplificare la complessità delle query, fornire una struttura logica personalizzata e limitare l'accesso ai dati solo alle informazioni rilevanti.

Le viste sono read-only (ovviamente...)

Esempio:
Supponiamo di avere una tabella `Utenti` con colonne `Nome`, `Cognome` e `DataNascita`. Vogliamo creare una 'tabella virtuale' che mostri solo il nome completo degli utenti maggiorenni (ex: nati dopo il 2005).
Non ha senso creare una tabella apposta solo per questo quindi creo una vista

```
CREATE VIEW Vista_Utenti_Maggiorenni AS SELECT CONCAT(Nome, ' ', Cognome) AS
NomeCompleto FROM Utenti WHERE DataNascita > '2005-01-01';
```

Successivamente potrò fare tutte le operazioni di lettura ed incrocio dei dati avendo come presupposto certo che i miei utenti sono maggiorenni

Vantaggi delle Viste:

1. **Astrazione dei Dettagli:** Le viste consentono di nascondere la complessità sottostante delle query e della struttura delle tabelle, facilitando l'accesso ai dati.
2. **Riusabilità delle Query:** Una vista ben progettata può essere utilizzata in più query, migliorando la riusabilità del codice.
3. **Controllo degli Accessi:** Le viste possono essere utilizzate per limitare l'accesso ai dati solo alle informazioni necessarie, proteggendo i dati sensibili.
4. **Semplificazione delle Query:** Le viste possono semplificare le query complesse, creando una struttura logica personalizzata che si adatta alle esigenze specifiche dell'applicazione.
5. **Aggiornamento Automatico:** Quando le tabelle di base cambiano, i dati nelle viste si aggiornano automaticamente, garantendo coerenza senza dover modificare le query.

Operazioni SQL Fondamentali

Queste sono solo alcune delle query più comuni, e ci sono molte altre funzionalità avanzate disponibili in MySQL.

1. **SELECT: Recuperare Dati da una Tabella**

```
SELECT * FROM nome_tabella;
```

Questa query restituisce tutti i dati presenti nella tabella specificata. Puoi sostituire `*` con i nomi specifici delle colonne se vuoi recuperare solo alcune informazioni.

2. **WHERE: Filtrare i Dati**

```
SELECT * FROM nome_tabella WHERE condizione;
```

Utilizzando il `WHERE`, puoi filtrare i risultati in base a una condizione specifica. Ad esempio, `WHERE nome = 'Mario'` restituirà solo le righe dove il campo "nome" è uguale a 'Mario'.

3. **ORDER BY: Ordinare i Risultati**

```
SELECT * FROM nome_tabella ORDER BY nome_colonna ASC|DESC;
```

Questa query ordina i risultati in base al valore di una specifica colonna in ordine ascendente (`ASC`) o discendente (`DESC`).

4. **INSERT: Inserire Dati in una Tabella**

```
INSERT INTO nome_tabella (colonna1, colonna2) VALUES ('valore1', 'valore2');
```

Utilizza questa query per inserire nuovi dati in una tabella specificando i valori per le colonne interessate.

5. **UPDATE: Aggiornare Dati in una Tabella**

```
UPDATE nome_tabella SET colonna = 'nuovo_valore' WHERE condizione;
```

Con questa query, puoi aggiornare i dati esistenti in una tabella specificando la colonna da modificare e la condizione per selezionare le righe da aggiornare.

6. **DELETE: Eliminare Dati da una Tabella**

```
DELETE FROM nome_tabella WHERE condizione;
```

Questa query elimina i dati da una tabella in base a una condizione specifica.

7. **GROUP BY: Raggruppare Dati**

```
SELECT colonna, COUNT(*) FROM nome_tabella GROUP BY colonna;
```

Utilizzando `GROUP BY`, puoi raggruppare i dati in base ai valori di una specifica colonna e applicare funzioni di aggregazione come `COUNT`, `SUM`, ecc.

8. **JOIN: Unire Dati da Diverse Tabelle**

```
SELECT * FROM tabella1 JOIN tabella2 ON tabella1.colonna = tabella2.colonna;
```

Con la clausola `JOIN`, puoi combinare i dati da due o più tabelle in base a una relazione definita tra le colonne.

Progettazione dei Databasse

Normalizzazione

L'obiettivo principale della normalizzazione è organizzare i dati in modo **efficiente**, evitando la **duplicazione** e garantendo che le informazioni siano **coerenti** (integrità referenziale).

Vantaggi della Normalizzazione:

1. **Riduzione della Ridondanza:** Elimina la duplicazione inutile dei dati, riducendo lo spazio di archiviazione e migliorando l'efficienza.

2. **Miglioramento dell'Integrità:** Minimizza il rischio di anomalie nei dati, garantendo che le informazioni siano coerenti e aggiornate.
3. **Facilità di Manutenzione:** Semplifica la gestione e la manutenzione dei dati, facilitando l'aggiunta, la modifica o l'eliminazione delle informazioni.
4. **Miglioramento delle Prestazioni:** In alcuni casi, la normalizzazione può migliorare le prestazioni delle query, poiché le tabelle più piccole possono essere interrogate più rapidamente rispetto a tabelle più grandi e complesse.
5. Aumentano l'aderenza ai principi **ACID**

Il processo di normalizzazione segue una serie di regole standardizzate chiamate **forme normali**.

Prima Forma Normale (1NF):

Una tabella è nella Prima Forma Normale se:

- Tutte le colonne contengono valori atomici (non suddivisibili).
- Non ci sono valori duplicati nella stessa colonna.

Seconda Forma Normale (2NF):

Una tabella è nella Seconda Forma Normale se:

- È già nella 1NF.
- Ogni colonna non chiave dipende completamente dalla chiave primaria (eliminazione delle dipendenze parziali).

Terza Forma Normale (3NF):

Una tabella è nella Terza Forma Normale se:

- È già nella 2NF.
- Non ci sono dipendenze transitive: nessuna colonna non chiave dipende da un'altra colonna non chiave.

Esempio

Supponiamo di avere una tabella `Ordini` con colonne `NumeroOrdine`, `Prodotto`, `Quantità`, `PrezzoUnitario`.

Ordini			
NumeroOrdine	Prodotto	Quantità	PrezzoUnitario
1	AS	4	12€
1	BC	2	9€
2	ZZ	1	5€

Questa tabella potrebbe non essere in 3NF perché il prezzo unitario può dipendere dal `Prodotto`, non solo dalla chiave primaria.

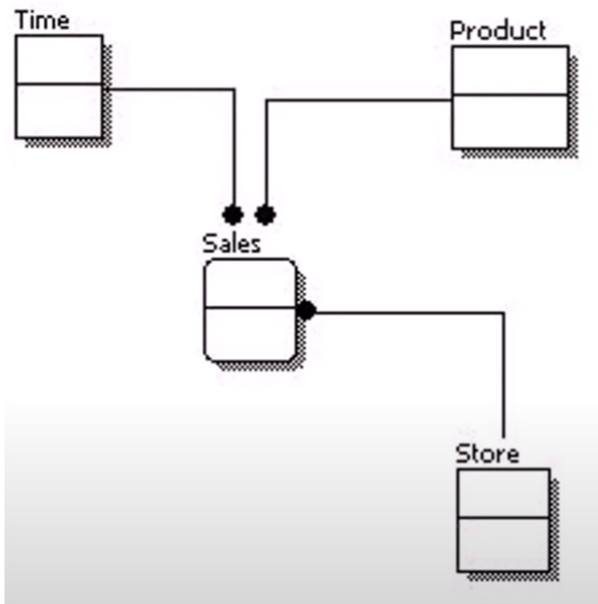
Una forma normalizzata potrebbe separare le informazioni sui prodotti in una tabella separata `Prodotti` con la chiave primaria `Prodotto`, e la tabella `Ordini` avrebbe solo la chiave primaria `NumeroOrdine` e `Prodotto` come chiave esterna.

Data model: concettuale, logico, fisico

La modellizazione del DB può passare attraverso tre fasi, una più specifica della precedente.

Concettuale

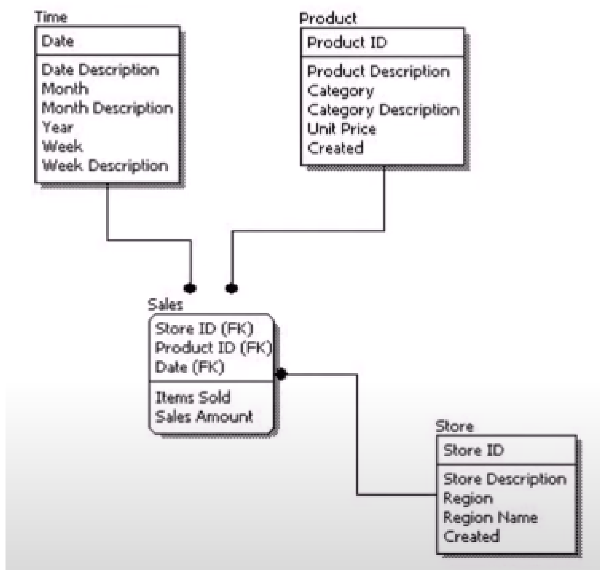
- molto ad alto livello (astratto)
- facile da capire
- facile da modificare
- rappresenta le entità e il fatto che c'è una relazione tra esse (senza specificare quale)
- carta e penna (no software)



Logico

Più specifico del precedente, identifica i campi di ogni entità e separa le chiavi dagli altri campi

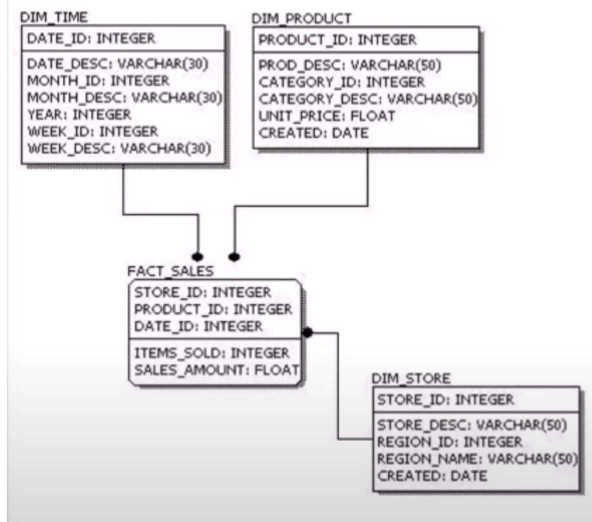
- introduce i campi/attributi
- identifica le chiavi della tabella interne e esterne (foreign key)
- nomi dei campi esplicativi
- agnostico rispetto al DBMS che verrà usato
- ci sono software apposta per farlo



Fisico

Non si intende "tangibile". Aggiunge molti dettagli specifici dell'implementazione

- le entità sono tabelle, gli attributi colonne
- usa nomi permessi dalla grammatica del DBMS
- più complesso, meno modificabile (siamo in fase di sviluppo !!)
- introduce tipi di dati e loro struttura



Database NoSQL

Non seguono il modello relazionale e sono adatti per situazioni in cui la struttura dei dati può cambiare frequentemente. Sono utilizzati in scenari come il web scalabile e le applicazioni con grandi volumi di dati non strutturati.

Perchè? : punti deboli dei DBMS relazionali classici

- Incapacità di gestire volumi di dati molto grandi (ordine di grandezza di petabyte)

- Impossibilità di gestire carichi estremi (migliaia di query al secondo e oltre)
- Il modello relazionale non è adatto all'archiviazione e all'interrogazione di alcuni tipi di dati (ad es. di alcuni tipi di dati (dati gerarchici, dati debolmente strutturati, dati semi-strutturati)
- Le proprietà **ACID** comportano gravi sovraccarichi in termini di latenza, accesso al disco, tempo della accesso al disco, tempo di CPU (lock, logging, ecc.)
- Prestazioni limitate dagli accessi al disco

In una parola : **performance**

- **Desiderata**:
  - modello di dati diverso,
  - scalabilità trasparente, prestazioni estreme
  - scalabilità orizzontale
    - aggiungo nodi al sistema per gestire volumi crescenti di dati e richieste.
  - architettura distribuita
    - la scalabilità orizzontale mi permette di utilizzare più server anche geograficamente distanti tra loro
- **Caratteristiche abbandonate**:
  - forte controllo della concorrenza e coerenza
  - query e modello dati (eventualmente) complesse o fortemente relazionali/interdipendenti

La scelta tra SQL e NoSQL dipende dalle esigenze specifiche del progetto, dalla struttura dei dati e dai requisiti di prestazioni. Inoltre, alcune applicazioni possono trarre vantaggio dall'uso di entrambi i tipi di database in scenari diversi all'interno dello stesso sistema (approccio chiamato poliglotta).

> **Non c'è quello "meglio". Dipende**

Key-Value Store

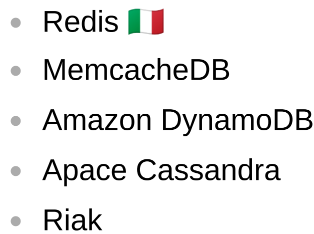
I database NoSQL di tipo "Key-Value Store" (archivio chiave-valore) sono una categoria di database NoSQL che memorizzano i dati in una struttura chiave-valore. Questo modello è molto semplice:

- ogni elemento dei dati è associato a una **chiave univoca**, e
- il valore è il dato stesso.
- Non ci sono schemi o strutture di dati complesse
  - ogni coppia chiave-valore può avere una struttura diversa
- l'accesso ai dati avviene tramite la chiave
- la velocità di accesso prevale sulla d

Utilizzo e Applicazioni

- Cache
- Gestione Sessioni
- Configurazioni

Esempi

- Redis 
- MemcacheDB
- Amazon DynamoDB
- Apace Cassandra
- Riak

Document DB

In un sistema di document store, i dati sono organizzati sotto forma di **documenti**, generalmente in formati come **JSON** o **BSON** (una rappresentazione binaria di JSON).
Ogni documento è un insieme di coppie chiave-valore, e il modello è spesso denominato "schemaless", poiché non richiede uno schema fisso e rigido.

- Il database è inteso come **collection** di documenti.
- Ogni documento è un insieme di chiavi e valori.
- Può contenere strutture annidate complesse.
- Ogni documento in una collezione può avere un formato diverso.
- Le operazioni di lettura e scrittura generalmente coinvolgono un documento completo.
- supportano query complesse e possono creare indici su attributi specifici dei documenti per migliorare le prestazioni delle query.

Utilizzo e applicazioni

- spesso utilizzati per gestire contenuti web dinamici e strutturati, come blog, pagine web o articoli
- per memorizzare configurazioni dinamiche, impostazioni utente personalizzate e preferenze
- adatti per memorizzare oggetti complessi con strutture annidate
- Archiviazione di Dati JSON-Like

Esempi

- Mongo
- Couch
- RavenDB

Esercizio NoSQL con Redis

Evidenziamo le differenze tra SQL e NoSQL
Il NoSQL è veloce ma non permette di gestire strutture complesse. Si compensa "sparpagliando i dati" (in modo ordinato) e utilizzando le istruzioni che permettono di operare sui valori, anche se potrei dover fare più query in sequenza.

Tabella semplice in SQL

user id	name	email	password	count
1	john	jogn@gmail.com	a/6sa_3gj	13

Per replicare la tabella in NoSQL (ex: Redis) devo conoscere prima quali dati userò come chiavi di ricerca e sparpagliare i dati in modo che mi permettano di trovare i valori richiest.
In questo caso, per la login, userò o l'ID o la mail (in caso di login). Con questi parametri posso costruire tutte le chiavi di ricerca necessarie.

key	value
user:1:email	jogn@gmail.com
user:1:data	{name: 'john', password: 'a/6sa_3gj ', count: 13}
email:jogn@gmail.com	1

Altre tipologie "esotiche" (specifiche)

Queste tipologie di database sono progettate per soddisfare esigenze specifiche e spesso si specializzano in un aspetto particolare della gestione dei dati. La scelta del database dipende sempre dalle specifiche esigenze dell'applicazione che stai sviluppando.
Ecco alcune categorie di database che vanno oltre la tradizionale distinzione tra SQL e NoSQL:

1. **Database a Grafo:**
   - **Esempi:** Neo4j, ArangoDB.
   - **Caratteristiche:** Ottimizzati per gestire relazioni complesse tra dati. Ideali per rappresentare reti sociali, mappe concettuali e grafi di relazioni.
2. **Database Temporali:**
   - **Esempi:** TimeScaleDB.
   - **Caratteristiche:** Progettati per gestire dati temporali e serie temporali. Adatti per applicazioni che richiedono l'analisi di dati nel tempo, come sensori, log e metriche di monitoraggio.
3. **Database Geospaziali:**
   - **Esempi:** MongoDB con supporto geospaziale, PostGIS.
   - **Caratteristiche:** Ottimizzati per gestire dati che includono componenti geografiche. Utile per applicazioni di mappatura, tracciamento GPS e analisi geospaziali.
4. **Database a Memoria:**
   - **Esempi:** Redis, Memcached.
   - **Caratteristiche:** Conservano i dati in memoria principale per garantire accessi veloci. Adatti per applicazioni in cui la velocità di accesso ai dati è fondamentale, come cache e gestione di sessioni.
5. **Database a Linguaggio Naturale:**
   - **Esempi:** Amazon QLDB.
   - **Caratteristiche:** Progettati per immagazzinare dati strutturati in un formato che può essere interrogato utilizzando il linguaggio naturale. Adatti per applicazioni che

richiedono una facile ricerca e accesso a dati utilizzando frasi umane.

6. **Database Blockchain:**
    - **Esempi:** BigchainDB.
    - **Caratteristiche:** Utilizzano la tecnologia blockchain per garantire l'immutabilità dei dati e la tracciabilità delle modifiche. Adatti per applicazioni che richiedono una registrazione sicura e trasparente delle transazioni.

> **NB: Le Blockchain sono parenti dei DB ma non sono database e i database non sono blockchain**

7. **Database Multimodali:**
    - **Esempi:** OrientDB, ArangoDB.
    - **Caratteristiche:** Supportano più modelli di dati, come il modello a grafo, documento e chiave-valore all'interno della stessa piattaforma. Adatti per applicazioni complesse con requisiti diversificati.
8. **Database Quantum:**
    - **Esempi:** IBM Quantum Hummingbird.
    - **Caratteristiche:** Sfruttano i principi della computazione quantistica per l'elaborazione dei dati. Ancora in fase di sviluppo, ma potrebbero offrire prestazioni rivoluzionarie in futuro.