**Database relazionali**: contenitore di dati organizzati e strutturati

|---> Proprietà ACID Atomicità Coerenza Isolamento Durabilità

|---> Operazioni CRUD Create Read Update Delete

**DBMS**: software che permette di creare, gestire, etc. i dati all'interno del database.

è l'interfaccia tra utente e database

ES: MySQL, Oracle, SQLite, MariaDB.

**🎯 Compiti principali di un DBMS**

1. **🗃️ Gestione dei dati**
   * Organizza i dati in tabelle, righe e colonne (nel caso dei DB relazionali).
   * Permette di **creare, leggere, aggiornare e cancellare** dati (le famose operazioni **CRUD**).
2. **🛡️ Controllo dell'accesso**
   * Gestisce **utenti, ruoli e permessi** per proteggere i dati da accessi non autorizzati.
3. **🧩 Integrità dei dati**
   * Applica **vincoli** (es. chiavi primarie, vincoli di integrità referenziale) per assicurare che i dati siano **coerenti e validi**.
4. **🔄 Gestione delle transazioni**
   * Garantisce che le operazioni sui dati siano eseguite in modo **completo e affidabile** (rispettando le regole **ACID**: Atomicità, Coerenza, Isolamento, Durabilità).
5. **📚 Recupero da errori e backup**
   * Salva copie di sicurezza e consente il **ripristino** dei dati in caso di errore o crash.
6. **📊 Ottimizzazione delle prestazioni**
   * Usa **indici, cache, e piani di esecuzione delle query** per rendere più veloce l’accesso ai dati.
7. **📈 Concorrenza**
   * Permette a **più utenti** di accedere contemporaneamente ai dati, evitando conflitti o corruzione.

**L’applicazione WEB** è più interattiva mentre il **sito web** non molto (solo lettura e navigazione)

**Vincolo di Integrità referenziale:** regola che vieta di creare dei record che hanno come foreign key un elemento non esistente nella tabella puntata.

**Modello concettuale:** è il primo passo della creazione di un database seguito da modello logico e poi da quello fisico. Un modello per rappresentarlo è quello entità/associazioni formato da entità attributi e relazioni.

**Engine:** È il motore di archiviazione interno all’interno di un DBMS che si occupa di come i dati vengono fisicamente memorizzati, organizzati e gestiti su disco e in memoria.

**Normalizzazione**: Processo lossless che permette di organizzare i dati in modo efficiente evitando ridondanze e anomalie (errori di inserimento).

Ci sono tre forme importanti:

1. Ogni campo deve essere atomico, ogni tupla deve essere diversa, i valori che appartengono ad una colonna devono essere dello stesso dominio.
2. Deve essere in 1NF, ogni campo non chiave deve dipendere da tutta la chiave e non solo da una parte in caso dovesse essere una PPK
3. Deve essere in 2NF, tutti gli attributi devono dipendere direttamente dalla PK (Dipendenza funzionale) e non da altri attributi (Dipendenza transitiva)

**Anomalie:**

1. 📝 Anomalia di Inserimento

Si verifica quando non puoi inserire un dato perché manca un altro dato obbligatorio, anche se non è rilevante in quel momento.

(ESEMPIO DEL MANAGER CON ID AZIENDA E AZIENDA CON ID MANAGER)

2. ✏️ Anomalia di Aggiornamento

Succede quando devi modificare lo stesso dato in più punti, e rischi che non siano tutti aggiornati allo stesso modo.

Esempio:  
Se l’email di un cliente è salvata in più righe (perché ha più ordini), devi aggiornarla ovunque. Se dimentichi una riga, i dati diventano incoerenti.

3. 🗑️ Anomalia di Cancellazione

Avviene quando cancellando un dato ne perdi anche un altro che non volevi eliminare.

Esempio:  
Cancellando l’ultima prenotazione di un cliente, perdi anche le informazioni sul cliente (se è tutto nella stessa tabella).

**Tupla:** modelloconcettuale

**Riga:** modellologico

**Record:** Database (modello fisico)

**NoSQL**: Un database NoSQL è un sistema che memorizza e gestisce i dati in modi diversi rispetto ai tradizionali sistemi relazionali. Non richiede uno schema fisso e supporta strutture di dati più flessibili, come documenti, coppie chiave-valore, grafi o colonne larghe. Questo approccio consente di adattarsi rapidamente a cambiamenti nei dati e nelle esigenze applicative.

**⚙️ Caratteristiche principali**

* **Flessibilità dello schema**: Possibilità di modificare la struttura dei dati senza interruzioni.
* **Scalabilità orizzontale**: Espansione aggiungendo più server, ideale per gestire grandi volumi di dati distribuiti.
* **Alta disponibilità e tolleranza ai guasti**: Replica dei dati su più nodi per garantire continuità del servizio.
* **Prestazioni elevate**: Ottimizzazione per operazioni rapide e gestione efficiente dei dati.

Anche i database relazionali possono essere scalati orizzontalmente ma per farlo occorre l’uso di strumenti e architetture complesse siccome rispettano le proprietà ACID e hanno relazioni molto complesse.

|--> Proprietà BASE:

-Basically Available (Disponibile di base): il sistema risponde sempre (anche in caso di parziale degrado)

-Soft state (Stato morbido): lo stato del sistema può cambiare nel tempo, anche senza input (per esempio, dati non ancora replicati ovunque)

-Eventual Consistency (Consistenza eventuale): alla fine tutti i nodi saranno coerenti, ma nel breve termine possono esserci differenze

**📊 Tabella: Confronto tra modelli NoSQL**

| **Modello** | **Esempi** | **✅ Pro** | **❌ Contro** | **🧰 Usi Tipici** |
| --- | --- | --- | --- | --- |
| **Document Store** | MongoDB, CouchDB | - Flessibilità dello schema - Facile da usare - Supporta query complesse | - Query JOIN limitate - Inconsistenza potenziale | CMS, e-commerce, app mobile, logging |
| **Key-Value** | Redis, DynamoDB | - Estremamente veloce - Semplice da scalare - Ottimo per caching | - Nessun supporto nativo per query complesse - Schema rigido | Cache, sessioni utente, ranking, token |
| **Wide-Column** | Cassandra, Bigtable | - Ottimizzato per scritture massive - Alta scalabilità - Alta disponibilità | - Complesso da configurare - Meno adatto a query relazionali | IoT, log di eventi, time-series, data warehousing |
| **Graph** | Neo4j, ArangoDB | - Eccellente per relazioni complesse - Navigazione rapida tra nodi | - Poco adatto a grandi volumi non relazionati - Curva di apprendimento più alta | Social network, raccomandazioni, frodi, genealogy |

**Performance-schema:** database di sistema che permette di raccogliere statistiche sulle prestazioni e sul comportamento del server MySQL. (ES. query più lente)

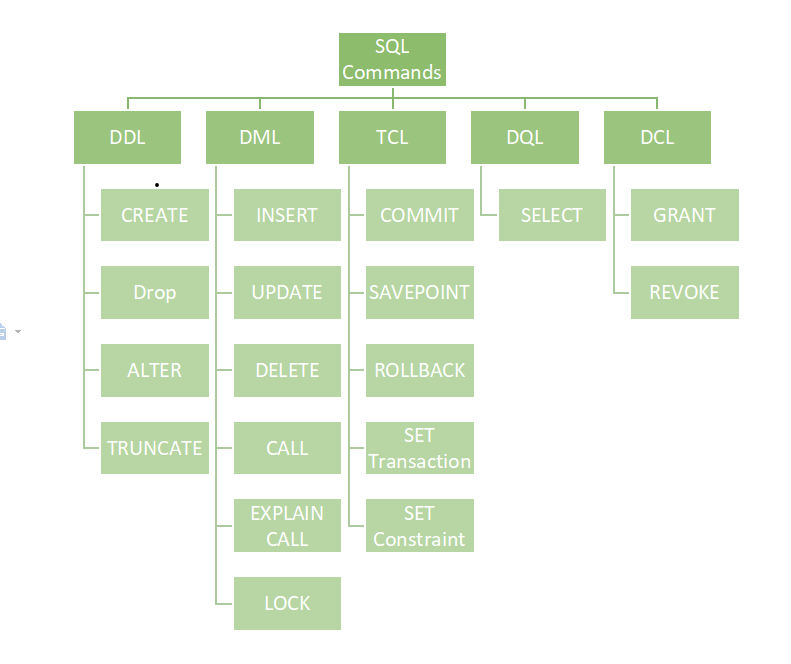
**Information-schema:** è un meta-database che contiene informazioni sulle strutture del database

**SQL**: Sta per Structured Query Language, si dice che è standard perché definito tale dall’ ISO (International Organization for Standardization) e che ha specifiche comuni per tutti i sistemi di database che lo implementano. Tutti i principali DBMS che lo usano SQL dove ciascuno segue lo standard base e applica le proprie estensioni.

Basato sul modello relazionale quindi su tabelle dove ogni record ha una chiave univoca

Famiglie di comandi:

* - Il Data Query Language è responsabile della lettura, o interrogazione, dei dati di un database. In SQL, questo corrisponde al comando SELECT.
* - Il Data Manipulation Language è responsabile dell'aggiunta, della modifica o della cancellazione di dati da un database. In SQL, corrisponde ai comandi INSERT, UPDATE, DELETE, CALL e LOCK, tra gli altri.
* - Il Data Definition Language è responsabile della definizione della struttura dei dati in un database. In SQL, questo corrisponde alla manipolazione delle tabelle attraverso i comandi CREATE TABLE, ALTER TABLE, DROP TABLE e TRUNCATE TABLE, tra gli altri.
* - Il Data Control Language è responsabile delle attività amministrative di controllo del database stesso, in particolare la concessione e la revoca dei permessi agli utenti. In SQL, questo corrisponde ai comandi GRANT, REVOKE, e DENY, tra gli altri.
* - Il Transaction Control Language è responsabile della gestione delle transazioni. In SQL, questo corrisponde ai comandi COMMIT, SAVEPOINT, ROLLBACK, tra gli altri. Rispettano le proprietà ACID.



**APPROFONDIMENTO DCL:**

DCL sta per Data Control Language ed è un sottoinsieme del linguaggio SQL che serve a gestire i permessi e la sicurezza dei dati in un database.

In pratica, con i comandi DCL puoi:

* Controllare chi può accedere ai dati
* Definire quali operazioni possono fare (leggere, scrivere, modificare, cancellare)
* Gestire i privilegi degli utenti e ruoli

Esempi:

GRANT SELECT, INSERT ON clienti TO mario;

REVOKE INSERT ON clienti FROM mario;

Si possono anche creare dei ruoli così che non si debba garantire permessi o revocarne altri a mano per ogni singolo utente ma bensì creare un ruolo con determinati permessi e assegnarlo poi ai vari utenti.

Esempio:

CREATE ROLE manager;

GRANT SELECT, INSERT, UPDATE ON clienti TO manager;

GRANT manager TO mario;

**FUNZIONI DI AGGREGAZIONE:**

Le funzioni di aggregazione in sql permettono di fare calcoli in run-time in modo tale da avere un riscontro con i dati presenti nel database.

Esistono 6 funzioni di aggregazione:

-AVG() ->Calcola la media dei valori presenti nel campo esplicitato all'interno delle parentesi

-SUM() ->Calcola la somma dei valori presenti nel campo esplicitato all'interno delle parentesi

-MIN() ->Ritorna il valore minimo tra quelli presenti nella tabella specificata tra parentesi

-MAX() ->Ritorna il valore massimo tra quelli presenti nella tabella specificata tra parentesi

-COUNT() ->Conta quanti valori ci sono in base a che colonna scriviamo tra parentesi

->Se tra parentesi metto l'asterisco conta anche i valori nulli

->Se ra parentesi metto un campo non conta i NULL

-STDEV() ->Deviazione standard

Il Group By permette di raggruppare i valori che servono a calcolare una delle funzioni sopraelencate in base a una tabella che scriviamo dopo: ad esempio se dovessimo fare la media dei voti nel group by scriveremo "NomeStudente" così da avere la media per ogni studente. In pratica tutte le colonne che inseriamo nel SELECT che non siano funzioni di aggregazione vanno messe anche dopo il GROUP BY

**QUERY ANNIDATE:** Sono delle query che permettono di fare delle ricerche in base a dati dei quali non siamo in possesso ma sono ricavabili da un’altra query.

ES.

SELECT nome, città

FROM clienti

WHERE id\_cliente IN ( //Nella WHERE posso mettere più attributi se sono ritornati

SELECT id\_cliente

FROM ordini

WHERE importo > 1000

);

-> IN: la query interna ritorna più valori e controllo se quello che sto cercando è in quella lista

-> EXISTS: controlla se la subquery ritorna almeno una riga (non va specificata la colonna dopo il WHERE)

-> =: la subquery ritorna un solo valore

-> > ALL: controlla che ciò che stiamo cercando abbia il campo x maggiore a tutti quelli ritornati dalla subquery

-> > ANY: controlla che ciò che stiamo cercando abbia il campo x maggiore a ad almeno uno di quelli ritornati dalla subquery

**ENUM:** è un tipo di dato speciale che ti permette di definire un campo con un insieme fisso di valori predefiniti. È molto utile quando un campo deve accettare solo certi valori specifici, come ad esempio "attivo", "inattivo", "sospeso". Con l’enum posso definire io il dominio.

**Viste:** una vista è una query salvata con un nome, che si può usare come se fosse una tabella.

**📌** A cosa servono le viste?

| Vantaggio | Descrizione |
| --- | --- |
| ✅ Leggibilità | Semplifica query complesse |
| ✅ Sicurezza | Puoi mostrare solo certi dati (es. nascondere colonne sensibili) |
| ✅ Riutilizzabilità | Evita di riscrivere la stessa query ogni volta |
| ✅ Astrazione | Nasconde la struttura interna delle tabelle reali |

**Join:** commando utilizzato per unire record provenienti da due o più righe.

**TRANSAZIONI:** Una transazione è un’unita di lavoro composta da una o più operazioni SQL che vengono eseguite insieme come un unico blocco rispettando le proprietà ACID

**🔧 Comandi principali**

| **Comando** | **Funzione** |
| --- | --- |
| START TRANSACTION | Inizia una transazione |
| COMMIT | Conferma tutte le operazioni |
| ROLLBACK | Annulla tutte le operazioni |
| SAVEPOINT <nomecheckpoint> | Crea un punto intermedio nella transazione |
| ROLLBACK TO <nomecheckpoint> | Torna a uno specifico savepoint |

Sono utilizzate quando ad esempio devo prelevare soldi da un conto e trasferirli in un altro. Se la prima operazione dovesse andare a buon fine mentre la seconda no creerebbe non pochi problemi. Usando una transazione o il blocco di operazioni va tutto a buon fine oppure si ritorna a prima che la prima operazione sia compiuta.

Esempio:

START TRANSACTION;

UPDATE conti SET saldo = saldo - 100 WHERE nome = 'Mario';

UPDATE conti SET saldo = saldo + 100 WHERE nome = 'Luca';

COMMIT;

E se qualcosa va storto invio semplicemente il comando “ROLLBACK”

**BLOB:** sta per Binary Large Object, è un tipo di dato utilizzato nei database per memorizzare grandi quantità di dati binari (non leggibili come testo) come immagini, video o audio.

È meglio utilizzare i blob quando si ha da salvare pochi file e si vuole semplificare il backup altrimenti, se si vogliono delle performance migliori e salvare molti file è più conveniente salvare nel file system i file e scrivere solo il percorso nel DB oppure usare degli storage esterni come Google Cloud Storage. Il file binario è estraibile tramite tool oppure codici (es. python).

File binario: b'\xff\xd8\xff\xe0\x00...'

**GDPR:** GDPR sta per General Data Protection Regulation ed è il Regolamento Europeo 2016/679 sulla protezione dei dati personali. Ricade su tutte le aziende (EU e non) che trattano dati di clienti provenienti dall’Europa. Il suo obbiettivo è quello di proteggere i dati personali e che vengano trattati in modo sicuro. Tra questi dati troviamo: Nome, Cognome, Codice fiscale, Indirizzo IP, Dati sanitari, Foto, etc.

**⚙️ Principi fondamentali del GDPR**

1. **Liceità, correttezza e trasparenza**
2. **Limitazione della finalità** → i dati vanno usati solo per lo scopo dichiarato
3. **Minimizzazione dei dati** → raccogli solo i dati strettamente necessari
4. **Esattezza** → i dati devono essere aggiornati
5. **Limitazione della conservazione** → non tenere i dati più a lungo del necessario
6. **Integrità e riservatezza** → protezione da accessi non autorizzati
7. **Responsabilità del titolare** → chi gestisce i dati deve dimostrare di rispettare il GDPR

**🔐 Diritti dell’interessato**

Chiunque ha diritto a:

* Accedere ai propri dati
* Chiedere la rettifica o la cancellazione ("**diritto all’oblio**")
* Limitare o opporsi al trattamento
* Portabilità dei dati (trasferimento ad altro servizio)
* Essere informato in caso di violazione dei dati (**data breach**)

Le sanzioni per le aziende possono arrivare fino a 20 milioni di euro oppure il 4% del fatturato annuale a seconda di quale di questi due sia maggiore.

Ogni utente che ha a che fare con un’azienda che ricade in queste caratteristiche deve dare il consenso anche al trattamento dei dati. Il trattamento deve essere Libero, Specifico, Informato e Inequivocabile.

Esempio: in un sito web, prima di raccogliere dati per marketing o cookie non tecnici, devi chiedere esplicitamente il consenso.

**PASSAGGIO DATI TRA CLIENT E SERVER:**

**🛠️ Quando usare cosa?**

| **Metodo** | **Quando** **si** **usa** |
| --- | --- |
| GET | Recuperare dati, pagine, immagini |
| POST | Inviare form, registrazione, login |
| PUT/PATCH | Aggiornare profili, articoli, ecc. |
| DELETE | Eliminare post, utenti, file |
| AJAX | Fare tutto questo senza ricaricare la pagina |

Ajax è una tecnica (non un metodo http) usata nel browser per inviare dati senza ricaricare la pagina, si può usare con qualsiasi metodo http e viene inviata tramite linguaggi come javascript (con la funzione fetch())

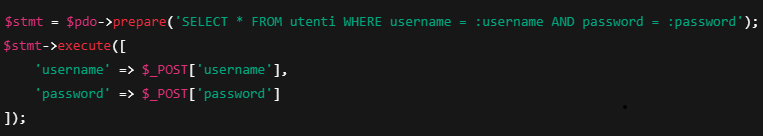
Il GET ha bisogno della ricarica della pagina e inoltra i dati nell’URL il che non rinde sicuro il passaggio di dati particolari.

Il POST ha bisogno della ricarica della pagina e i dati vanno nel body della richiesta il che rende tutto più sicuro.

**MINACCE:**

SQL Injection: Questo attacco sfrutta la mancata validazione dell'input dell'utente per inserire comandi SQL malevoli, permettendo all'attaccante di visualizzare, modificare o eliminare dati nel database, eseguire operazioni non autorizzate, come l'accesso a credenziali sensibili, compromettere l'intero sistema, inclusi i privilegi di amministratore.

Soluzione: utilizzo di query parametrizzate, i dati inseriti dall’utente non vengono subito inseriti nella query ma vengono messi in dei placeholder, sarà poi compito del database riempirli in modo sicuro.



Directory Traversal: Consente all'attaccante di accedere a file sensibili al di fuori della directory prevista, come /etc/passwd, sfruttando la scarsa validazione dei percorsi dei file.

Soluzione: Validare e sanificare rigorosamente i percorsi dei file, evitando l'uso di input dell'utente per determinare i percorsi.

Cross-Site Scripting (XSS): L'attaccante inserisce script JavaScript dannosi in un'applicazione web, che vengono poi eseguiti nel browser dell'utente. Questo può portare a furto di cookie e credenziali, manipolazione dell'interfaccia utente, reindirizzamenti verso siti malevoli.

Prevenzione: Sanificare e validare tutti gli input dell'utente, e implementare politiche di Content Security Policy (CSP).

Buffer overflow: molto rari al giorno d’oggi siccome le applicazioni web moderne utilizzano funzioni di input sicure eliminando quelle che erano considerate pericolose e la gestione della memoria è gestita automaticamente e non affidata al developer.

**CLOUD COMPUTING**: significa usare risorse informatiche (come server, storage, database, rete, software) fornite tramite internet, anziché avere tutto in locale su un computer o server privato.

In pratica, il cloud ti permette di affittare potenza di calcolo o spazio di archiviazione da un provider come Amazon Web Services (AWS) o Microsoft Azure.

**🔧 Tipi di servizi cloud**

**1. IaaS – Infrastructure as a Service**

Tu affitti infrastruttura virtuale (server, rete, storage). Devi gestire tu i sistemi operativi, le app, ecc.

* Esempi: AWS EC2, Google Compute Engine, Azure Virtual Machines

**2. PaaS – Platform as a Service**

Il provider gestisce tutta l'infrastruttura e ti fornisce una piattaforma pronta per eseguire le tue app.

* Esempi: Heroku, Google App Engine, Azure App Service

**3. SaaS – Software as a Service**

Usi direttamente applicazioni via web. Non devi installare nulla, paghi per usarle.

* Esempi: Gmail, Google Drive, Microsoft 365, Dropbox

**🏗️ Modelli di distribuzione**

| **Modello** | **Descrizione** |
| --- | --- |
| ☁️ Public Cloud | Risorse condivise su internet |
| 🏠 Private Cloud | Cloud dedicato all'interno di un'organizzazione |
| 🔁 Hybrid Cloud | Combinazione di cloud pubblici e privati |
| 🌍 Multi-cloud | Uso di più provider cloud contemporaneamente |

**✅ Vantaggi del Cloud**

* **Scalabilità**: puoi aumentare o ridurre le risorse su richiesta
* **Flessibilità**: accesso da ovunque tramite internet
* **Costi ridotti**: paghi solo ciò che usi (modello *pay-as-you-go*)
* **Affidabilità**: backup automatici, failover, distribuzione geografica
* **Collaborazione**: più persone possono lavorare sugli stessi dati/app

**⚠️ Svantaggi/Rischi**

* **Sicurezza e privacy**: i tuoi dati sono su server di terzi
* **Dipendenza dal provider** (lock-in)
* **Connettività**: senza internet, niente cloud
* **Costi imprevisti**: se non monitori bene l’uso, potresti ricevere fatture alte

**API**: sta per Application Programming Interface — cioè Interfaccia di Programmazione di un'Applicazione. È un insieme di regole e protocolli che permettono a due applicazioni o sistemi di comunicare tra loro.

**🛠️ Tipi di API più comuni**

**1. Web API / REST API**

* Basate su HTTP/HTTPS
* Usano metodi come GET, POST, PUT, DELETE
* Comunicano tramite formato **JSON** o **XML**
* Esempio: chiamata per ottenere dati da un database remoto

**2. SOAP API**

* Più "vecchie" e formali
* Usano il formato XML e protocolli specifici
* Spesso usate in contesti aziendali (es. banche)

**3. GraphQL**

* Alternativa moderna a REST
* Il client specifica **esattamente quali dati vuole**
* Risposte più ottimizzate, ma più complesso da gestire

DOVE SI USANO LE API

| **Settore** | **Esempio** |
| --- | --- |
| Siti web | Chiamate al server per caricare dati senza ricaricare la pagina (AJAX) |
| App mobili | App meteo che usa un’API per ottenere i dati attuali |
| Pagamenti | Integrazione con API come PayPal o Stripe |
| Social | Twitter API per leggere o scrivere tweet da altre app |
| Cloud | API AWS per creare istanze, bucket, ecc. |
|  |  |

🔑 Token API: cos'è e come funziona

Un token API è una chiave segreta (una stringa alfanumerica lunga) che identifica chi sta facendo la richiesta. Serve per:

1. Autenticare l’utente o l’app
2. Autorizzare specifiche azioni
3. Monitorare e limitare l’uso (quota, billing, sicurezza)

Le API hanno vari piani di pagamento in base a quale stiamo utilizzando, alcune hanno un numero di richieste mensile gratuito dopo il quale si paga (Free Tier), alcune hanno una tariffa fissa (Abbonamento fisso), alcune vanno pagate in base all’utilizzo effettivo (Pay-as-you-go) e altre se si superano i limiti prestabiliti si paga extra (Overage fees).