PROGETTAZIONE DI BASI DI DATI

La progettazione di basi di dati è il processo di strutturazione e organizzazione dei dati in modo da permettere un accesso efficiente, la gestione corretta e la sicurezza delle informazioni. Una buona progettazione di una base di dati è fondamentale per garantire la coerenza, l'affidabilità e la velocità nelle operazioni di inserimento, aggiornamento, eliminazione e ricerca dei dati.

Cos'è una base di dati?

Una base di dati è un insieme di dati organizzati in modo da poter essere facilmente consultati, gestiti e aggiornati. i dati sono memorizzati in un supporto di memoria permanente e sono progettati in modo che più utenti possano accedervi contemporaneamente. i database sono creati e gestiti dai database management system (dbms). Un dbms è un insieme di strumenti software in grado di gestire grandi moli di dati strutturati, condivisi e permanenti, che possono essere aggiornati e interrogati. un dbms assicura, inoltre, affidabilità e riservatezza. Una base di dati è, quindi, una collezione di dati gestita da un dbms

il dbms è quindi un software (o anche una collezione di software) che permette di creare dei database, inserire dei dati, modificarli ed effettuare ricerche su di essi.

• GESTIONE DEL DATABASE: un dbms permette la creazione di un database, l'inserimento, la modifica, la cancellazione e la ricerca dei dati. ci si può inoltre interfacciare con un dbms utilizzando linguaggi di programmazione o applicazioni che adottano protocolli e linguaggi standard.

• PERSISTENZA E CONSISTENZA DEI DATI: un dbms conserva i dati in una memoria permanente e ne permette il recupero in caso di malfunzionamenti tramite alcune funzionalità di backup e ripristino. assicura inoltre che i dati siano consistenti tra loro. un'operazione di modifica ha successo se tutti i dati a cui fa riferimento vengono modificati, in tal caso si dice che l'operazione è atomica (cioè, indivisibile).

• PRIVACY E SICUREZZA: solo gli utenti autorizzati (tramite, di solito, username e password) possono accedere ai dati e solo a quelli per cui hanno autorizzazione.

• INTEGRITÀ DEI DATI: in un dbms vi è un utente speciale, il dba (database administrator), che è autorizzato ad apportare modifiche alle strutture contenenti i dati e può stabilire dei vincoli su di essi; per esempio imporre che, per l'età, sia inserito un numero intero maggiore o uguale a o. il dbms garantisce allora che tali vincoli siano sempre rispettati, in modo che non sia possibile inserire valori non validi. come vedremo in seguito, l'integrità va vista in relazione a che cosa è stato definito tramite i vincoli.

• SCALABILITÀ: al contrario di quanto avviene utilizzando file di testo o fogli di calcolo, le performance rimangono pressoché invariate a prescindere dalla quantità dei dati, siano essi pochi o moltissimi.

• SUPPORTO ALLE TRANSAZIONI: una transazione è una sequenza di operazioni correlate tra loro che vengono eseguite atomicamente; cioè, nel caso che anche solo una di tali operazioni fallisca, l'intera sequenza viene annullata e il database torna allo stato precedente al suo inizio. solo

se tutte le operazioni della sequenza hanno successo il database viene modificato e questo preserva l'integrità e la consistenza dei dati.

-• GESTIONE DEI METADATI: i metadati sono le informazioni che descrivono i dati, per esempio i nomi delle tabelle e dei campi, che vengono anch'esse salvate nel database in speciali tabelle accessibili all'amministratore.

Esistono diversi tipi di **database**, ognuno con caratteristiche specifiche per diversi tipi di applicazioni. Ecco le principali categorie:

DATABASE RELAZIONALI (RDBMS)

Un **database relazionale** è un tipo di database che organizza i dati in tabelle (dette **relazioni**) composte da righe (**tuple**) e colonne (**attributi**). È basato sul **modello relazionale** proposto da Edgar F. Codd nel 1970 e utilizza il linguaggio SQL (*Structured Query Language*) per la gestione e l'interrogazione dei dati.

Caratteristiche principali:

Struttura a tabelle: I dati sono organizzati in tabelle con righe e colonne, dove ogni riga rappresenta un'istanza e ogni colonna un attributo.
Chiavi primarie e chiavi esterne: Ogni tabella ha una **chiave primaria** (Primary Key, PK) che identifica in modo univoco ogni riga. Le **chiavi esterne** (Foreign Key, FK) collegano le tabelle tra loro, stabilendo relazioni tra i dati.
Integrità referenziale : Assicura la coerenza dei dati tra le tabelle collegate tramite chiavi esterne, impedendo operazioni che violerebbero i vincoli di relazione.
Indipendenza logica e fisica dei dati : La struttura logica del database è separata dalla sua implementazione fisica, consentendo modifiche senza impattare l'applicazione.
Linguaggio SQL : il database relazionale viene gestito e interrogato tramite SQL, che permette operazioni di inserimento, aggiornamento, eliminazione e selezione dei dati.
Atomicità, Coerenza, Isolamento, Durabilità (ACID): I sistemi di database relazionali garantiscono la correttezza delle transazioni attraverso queste quattro proprietà:

- **Atomicità**: ogni transazione è eseguita completamente o annullata.
- **Coerenza**: il database rimane in uno stato valido prima e dopo una transazione.
- **Isolamento**: le transazioni concorrenti non interferiscono tra loro.
- **Durabilità**: una transazione completata resta salvata nel database anche in caso di guasti.

Normalizzazione dei dati: Processo che elimina ridondanze e incongruenze nei dati suddividendoli in più tabelle collegate.
Sicurezza e controllo degli accessi : Gestione degli utenti con permessi differenziati per garantire riservatezza e integrità dei dati.

Esempi di **DBMS relazionali** più diffusi:

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server
- SQLite

DATABASE GERARCHICI

Un **database gerarchico** è un tipo di database che organizza i dati in una **struttura ad albero**, in cui ogni **record padre** può avere più **record figli**, ma ogni figlio ha un solo padre. Questa struttura è simile a una **cartella con sottocartelle** in un sistema operativo.

Caratteristiche Principali

1. Struttura ad Albero
- I dati sono organizzati in una gerarchia con **un nodo radice** e più livelli di sotto-nodi.
- Ogni **padre** può avere più **figli**, ma ogni **figlio ha un solo padre** (relazione **uno-a-molti, 1:N**).

2. Relazioni Uno-a-Molti (1:N)
- Un singolo nodo **padre** può avere più nodi **figlio**.
- **Non supporta direttamente le relazioni molti-a-molti (N:N)**, a meno che i dati non vengano duplicati.

3. Percorso di Accesso Rigido
- Per accedere a un nodo figlio, bisogna attraversare la gerarchia dall'alto verso il basso.
- Questo rende il recupero veloce se la struttura è ben definita**,** ma inefficiente per ricerche trasversali**.**

4. Efficienza nelle Ricerche Predefinite
- Molto efficiente se le query seguono la gerarchia naturale.
- Non adatto se i dati devono essere recuperati con ricerche trasversali (es. trovare tutti i dipendenti indipendentemente dal reparto).

5. Difficoltà di Modifica e Aggiornamento
- Cambiare la struttura è complicato**:** Se un nodo deve essere spostato, può essere necessario riorganizzare tutto il database. Duplicazione dei dati per rappresentare relazioni complesse.

6. Maggiore Sicurezza e Controllo
- Essendo basato su una struttura gerarchica fissa, è più semplice controllare gli accessi ai vari livelli.

Oggi, i database gerarchici sono stati quasi completamente sostituiti dai database relazionali (SQL) che offrono maggiore flessibilità.

DATABASE RETICOLARI

Un **database reticolare** è un tipo di database che organizza i dati in una **struttura a grafo**, dove ogni record può avere più collegamenti con altri record, permettendo relazioni **molti-a-molti (N:N)** tra i dati. È stato sviluppato negli anni '60 come evoluzione del **modello gerarchico** per superarne le limitazioni e fu standardizzato dal **CODASYL (Conference on Data Systems Languages)**

Caratteristiche principali

1. Struttura a Grafo
- I dati sono organizzati in una **rete di nodi e archi**, simile a un grafo.
- Ogni record può avere più connessioni con altri record, permettendo relazioni complesse.
- I collegamenti tra i record sono rappresentati da **puntatori diretti**, rendendo la navigazione più efficiente rispetto ai database gerarchici.

2. Relazioni Molti-a-Molti (N:N)
- A differenza del modello gerarchico (che supporta solo **uno-a-molti, 1:N**), un database reticolare può rappresentare **molti-a-molti (N:N)** senza bisogno di duplicare i dati.

- Questo lo rende adatto per situazioni in cui gli elementi sono fortemente interconnessi, come in un sistema di gestione universitaria in cui uno studente può essere iscritto a più corsi e un corso può avere più studenti**.**

3. Accesso Diretto ai Dati

- Utilizza **puntatori fisici** per collegare i record, rendendo la ricerca e l'accesso ai dati molto rapido.
- A differenza del modello relazionale, non richiede complesse query SQL con join**,** ma permette l'accesso diretto ai record correlati.

4. Complessità nella Gestione

- L'uso di **puntatori fisici** rende i database reticolari più complessi da gestire rispetto ai database relazionali.
- Ogni modifica alla struttura della rete può richiedere una riprogettazione degli archi e dei nodi.

5. Maggiore Flessibilità rispetto ai Database Gerarchici

- A differenza del modello gerarchico, dove ogni nodo ha un solo padre, qui un nodo può essere collegato a più nodi genitori e figli**.**
- Questo rende il database più flessibile e riduce la duplicazione dei dati.

DATABASE A OGGETTI

Un **database a oggetti** è un tipo di database che unisce i concetti della **programmazione orientata agli oggetti (OOP)** con la gestione dei dati.

- Gli elementi archiviati non sono solo dati **strutturati in tabelle** (come nei database relazionali), ma anche **oggetti**, che includono attributi e metodi.
- Ogni oggetto appartiene a una **classe** e può avere relazioni con altri oggetti, supportando concetti come **ereditarietà, incapsulamento e polimorfismo**.

Caratteristiche Principali

1. Struttura Basata su Oggetti

- I dati sono memorizzati sotto forma di **oggetti** invece di tabelle.
- Ogni oggetto è un'**istanza di una classe**, che definisce i suoi attributi e metodi.
- Le classi possono ereditare caratteristiche da altre classi (**ereditarietà**).

2. Supporto per la Persistenza degli Oggetti

- Un oggetto salvato nel database mantiene **lo stato e i comportamenti** definiti nella sua classe.
- Può essere recuperato e usato senza bisogno di convertirlo in un formato diverso.

3. Relazioni tra Oggetti

- Gli oggetti possono avere **associazioni** dirette tra loro.
- Non sono necessarie **join complesse** come nei database relazionali, poiché gli oggetti sono già collegati tramite riferimenti.

4. Incapsulamento e Metodi negli Oggetti

- Ogni oggetto può avere **metodi** che eseguono operazioni sui dati, riducendo la necessità di scrivere codice SQL separato.
- Gli utenti possono interagire con gli oggetti senza preoccuparsi della loro struttura interna

5. Supporto per Ereditarietà e Polimorfismo

- Le classi possono **ereditare** attributi e metodi da altre classi, evitando ridondanza di dati.
- Un oggetto può essere trattato come un'istanza della sua classe padre (**polimorfismo**).

6. Maggiore Complessità rispetto ai Database Relazionali

- La gestione delle relazioni tra oggetti può diventare complessa per strutture molto grandi.
- Meno strumenti standardizzati rispetto ai database SQL.

Database XML

Un **database XML** è un tipo di database progettato per archiviare, gestire e interrogare dati strutturati in **formato XML (eXtensible Markup Language)**.

- XML è un formato **gerarchico** e **autodescrittivo**, ideale per rappresentare dati con strutture flessibili.
- I database XML vengono usati in applicazioni che richiedono **scambi di dati tra sistemi diversi**, come web services, documenti elettronici e configurazioni software.

Caratteristiche Principali

1. Struttura Basata su XML

- I dati sono archiviati in documenti XML invece che in tabelle relazionali.
- Ogni documento ha una struttura gerarchica con **nodi annidati**.
- Gli elementi XML possono avere **attributi** e **valori testuali**.

Qui i dati sono organizzati in una **struttura gerarchica**, dove **università** è il nodo principale, e ogni **studente** è un nodo figlio.

2. Gerarchia e Nesting

- La struttura ad albero permette di **annidare i dati**, rappresentando facilmente relazioni padre-figlio.
- **Non sono necessarie tabelle separate** come nei database relazionali.

3. Flessibilità nella Struttura dei Dati

- XML non richiede una **struttura fissa** come le tabelle SQL.
- È possibile aggiungere nuovi campi senza modificare l'intero schema.
- Ideale per dati con **strutture variabili** o **non omogenei**.

4. Supporto per Query XML (XPath, XQuery)

- I database XML usano **linguaggi specifici** per interrogare i dati, come:
  - **XPath** → per navigare tra i nodi XML.
  - **XQuery** → per interrogare e trasformare dati XML.

5. Archiviazione Nativa o Relazionale

I database XML possono essere:

1. **Nativi XML** → memorizzano i dati **direttamente in formato XML**, ottimizzando le ricerche.
2. **XML su Database Relazionali** → memorizzano dati XML in colonne di tipo XML all'interno di un database SQL.

6. Indipendenza dalla Piattaforma e Interoperabilità

- XML è **universale** e leggibile da qualsiasi sistema.
- Facilita lo **scambio di dati tra applicazioni diverse** (es. tra un database e un'app web).
- Usato nei **web services (SOAP, REST)** per trasmettere informazioni tra server.