Appunti Corso di Database: Introduzione

Basato sulle slide del Prof. Danilo Montesi

17 maggio 2025

Indice

1 Informazioni Chiave per il Corso

Per avere successo in questo corso, il professore sottolinea l'importanza di:

- **Studio Autonomo:** Concentrarsi sui concetti fondamentali.
- **Esperienza Personale:** Cercare di collegare i concetti a proprie esperienze pregresse.
- **Esercizi:** Svolgere regolarmente gli esercizi proposti.
- **Progetto/Esercitazioni Pratiche:**
  - Sviluppare un progetto.
  - OPPURE partecipare attivamente agli esercizi durante le lezioni.
  - Utilizzare strumenti concreti come: `DB2`, `SQLServer`, `Oracle`, `PostgreSQL`, `MySQL`, `MS Access`, ecc. (`PostgreSQL` e `MySQL` sono ottimi punti di partenza se già conosci SQL).

2 Concetti Fondamentali

2.1 Cos'è un Database

- **Definizione Generale:** Un insieme organizzato di dati che supporta lo svolgimento di attività specifiche (per un'istituzione, un'azienda, un ufficio, una persona).
  - *Esempio pratico:* La tua rubrica telefonica è un piccolo database personale. L'anagrafe comunale è un database istituzionale.
- **Definizione Specifica (nel contesto del corso):** Un insieme di dati gestito da un **DBMS** (DataBase Management System).

2.2 Sistema Informativo vs. Sistema IT

- **Sistema Informativo:** La componente di un'organizzazione che gestisce le informazioni rilevanti per raggiungere gli obiettivi aziendali. Può esistere anche senza computer (pensa agli archivi cartacei di secoli fa). Include:
  - Raccolta e acquisizione informazioni.
  - Memorizzazione e conservazione.
  - Elaborazione, trasformazione, produzione.
  - Distribuzione, comunicazione, scambio.
- **Sistema IT (Information Technology):** La parte *automatizzata* del sistema informativo che utilizza tecnologie informatiche.
  - *Gerarchia:* Impresa $>$ Organizzazione $>$ Sistema Informativo $>$ Sistema IT. Un database è una componente chiave del Sistema IT.

2.3 Informazione vs. Dati

- **Informazione:** Fatti forniti o appresi su qualcosa o qualcuno. Ha un significato intrinseco.
  - *Esempio:* 'La lezione di Basi di Dati è alle 9:45 in aula N2 con il Prof. Rossi.'
- **Dati:** Rappresentazioni grezze dell'informazione, spesso codificate. Un punto di partenza fisso per un'operazione. I dati, da soli, possono non avere significato senza un'interpretazione.
  - *Esempio (dal cartello stradale):* Il dato `"8-17"` sul cartello. Da solo non significa nulla. Diventa informazione ('Divieto di sosta dalle 8 alle 17') quando interpretato nel contesto del cartello, del giorno ('Mon-Fri'), e delle regole stradali.
  - *Esempio pratico (MongoDB/JSON):*

```
{ "nome": "Mario", "cognome": "Rossi", "eta": 30 }
```

L'informazione è che c'è una persona di nome Mario Rossi di 30 anni.

- **Processo:** L'informazione viene organizzata, codificata e memorizzata sotto forma di dati.

2.4 Perché i Dati sono Importanti?

- È difficile rappresentare precisamente informazioni e conoscenze complesse.
- I **dati sono una risorsa strategica** perché sono più stabili rispetto ad altre rappresentazioni (processi di business, tecnologie, ruoli umani).
  - *Esempio:* I dati anagrafici di una persona o i dati di un conto bancario rimangono fondamentalmente gli stessi anche se cambiano le procedure dell'ufficio anagrafe, il software della banca o le persone che ci lavorano.

2.5 DBMS: DataBase Management System

Un software progettato per creare, gestire e interrogare database. Deve gestire collezioni di dati che sono:

- **Grandi (Big):** Tipicamente più grandi della memoria principale (RAM) del sistema. Si parla di Terabyte, miliardi di record.
- **Persistenti (Persistent):** La loro durata di vita è indipendente dai processi che li utilizzano. I dati sopravvivono allo spegnimento del computer o alla chiusura dell'applicazione.
  - *Esempio pratico:* Quando la tua app Node.js scrive su un database PostgreSQL usando Prisma, quei dati rimangono nel DB anche se riavvii il server Node.js.
- **Condivisi (Shared):** Accessibili e utilizzabili da multiple applicazioni e utenti, spesso contemporaneamente.
  - Questo porta a problemi di:
    - **Ridondanza:** Stessi dati duplicati in più posti.
    - **Inconsistenza:** Se i dati duplicati non vengono aggiornati ovunque, si creano discrepanze.
  - Un DB centralizzato riduce questi problemi.

Un DBMS deve inoltre garantire:

- **Privacy/Sicurezza (Privacy):** Controllo degli accessi. Chi può vedere/modificare cosa?
  - *Esempio SQL:*

```
GRANT SELECT ON tabella_clienti TO 'utente_marketing';
```

- **Affidabilità (Reliability):** Tolleranza ai guasti (hardware o software). I dati devono essere preservati. La tecnica cruciale è la gestione delle **Transazioni**.
- **Efficienza (Efficiency):** Uso ottimale delle risorse (memoria, tempo di CPU, I/O disco) per rispondere rapidamente alle richieste.
  - *Esempio pratico:* L'uso di indici (`CREATE INDEX`) su colonne frequentemente interrogate accelera enormemente le ricerche.
- **Efficacia (Effectiveness):** Fornire funzionalità potenti e flessibili che migliorino le attività degli utenti.

Esempi di DBMS: Oracle, SQLServer, DB2, **MySQL, PostgreSQL** (Relazionali/SQL), MS Access, SQLite (embedded), BigQuery (cloud data warehouse), **MongoDB** (NoSQL documentale).

2.6 Transazioni

Una transazione è una sequenza di operazioni sul database che viene trattata come una singola unità logica di lavoro. Deve avere le proprietà **ACID** (anche se non esplicitamente nominate come acronimo, i concetti ci sono):

- **Atomicità (Atomic):** Le operazioni all'interno di una transazione vengono eseguite *tutte o nessuna*. Se una qualsiasi operazione fallisce, l'intera transazione viene annullata (rollback) e il database torna allo stato precedente.
  - *Esempio classico:* Trasferimento di denaro da un conto A a un conto B. Deve avvenire sia il prelievo da A SIA il deposito su B. Se uno dei due fallisce, nessuno dei due deve avere effetto.
- **Coerenza (Consistency):** Una transazione porta il database da uno stato coerente a un altro stato coerente. Rispetta tutti i vincoli definiti.
- **Isolamento (Isolation, riferito a "Concurrent"):** Le transazioni concorrenti (eseguite contemporaneamente da più utenti/processi) non devono interferire tra loro. Ogni transazione deve apparire come se fosse l'unica in esecuzione.
  - *Esempio:* Se due utenti tentano di prenotare l'ultimo posto disponibile su un volo, il sistema deve garantire che solo uno ci riesca, evitando doppie prenotazioni.
- **Durabilità (Durability, riferito a "Permanent"):** Una volta che una transazione è stata confermata (**commit**), i suoi effetti sono permanenti e sopravvivono a guasti del sistema (es. crash del server, mancanza di corrente).
  - *Esempio SQL:* Dopo un `COMMIT;`, i dati sono scritti in modo sicuro.

Punti di vista sulla transazione:

- **Utente:** Un'operazione di business completa (es. 'registra nuovo cliente', 'paga fattura').
- **Sistema:** Una sequenza indivisibile di operazioni che garantisce affidabilità.

2.7 DBMS vs. File System

- **File System:** Gestisce file e cartelle. L'accesso ai dati è 'grezzo' (tutto il file o niente).
  - *Esempio:* Leggere un file CSV riga per riga.
- **DBMS:** Estende le funzionalità del file system offrendo:
  - Accesso granulare ai dati (righe specifiche, colonne, filtri complessi).
  - Linguaggi di interrogazione potenti (SQL).
  - Controllo della concorrenza, gestione delle transazioni, sicurezza, ecc.
  - Indipendenza dei dati (vedi sotto).

3 Architettura e Modelli dei Dati

3.1 Evoluzione della Gestione dei Dati

- **Anni '70:** Applicazioni gestivano i propri file. Logica dei dati e logica applicativa mescolate. Alta ridondanza.
- **Anni '80:** Primi DBMS. Separazione tra dati e applicazioni. Nascono le 'tabelle dati'.
- **Anni '90 (Comportamento Procedurale):** Introduzione di logica condivisa (procedure) all'interno del DBMS.
  - **Stored Procedures:** Blocchi di codice (spesso SQL esteso) memorizzati nel DB ed eseguibili.
  - **Trigger:** Procedure speciali eseguite automaticamente dal DBMS in risposta a determinati eventi (INSERT, UPDATE, DELETE su una tabella).

* *Esempio SQL (concettuale):*

```
CREATE TRIGGER aggiorna_quantita_magazzino
AFTER INSERT ON ordini
FOR EACH ROW
BEGIN
UPDATE prodotti
SET quantita = quantita - NEW.quantita_ordinata
WHERE id = NEW.prodotto_id;
END;
```

- **Anni 2000 (Web):** Architetture a più livelli (Client con Javascript, Server Applicativo con Java/Node.js, DBMS).
- **Anni 2010 (Mobile):** Simile all'architettura web, ma con client mobile.

3.2 Descrizione dei Dati e Data Model

- **Senza DBMS:** Ogni software descrive internamente la struttura dei file che processa. Molteplici descrizioni possono portare a inconsistenza.
- **Con DBMS:** Esiste un **catalogo** o **dizionario dei dati** (una porzione del database stesso) che contiene una descrizione centralizzata dei dati (lo schema). Questa descrizione è condivisa tra tutte le applicazioni.
- **Data Model:** Un insieme di costrutti usati per organizzare e descrivere il comportamento dei dati.
  - Componente cruciale: fornire **struttura** ai dati (tramite 'costruttori di tipo').
  - Il **modello relazionale** (usato da SQL) fornisce il costruttore 'relazione' (tabella), che permette di definire insiemi di record omogenei.

3.3 Schema e Istanza

- **Schema (Intensionale):** La descrizione della struttura del database. È (relativamente) invariante nel tempo. Definisce 'come sono fatti' i dati.
  - *Esempio SQL:*

```
CREATE TABLE Utenti (
ID INT PRIMARY KEY,
Nome VARCHAR(100),
Email VARCHAR(100) UNIQUE
);
```

    definisce lo schema della tabella `Utenti`. Include nomi delle colonne, tipi di dato, chiavi, vincoli. Corrisponde agli 'headers' delle tabelle.
- **Istanza (Estensionale):** I valori effettivi contenuti nel database in un dato momento. Cambia rapidamente con le operazioni di inserimento, modifica, cancellazione. Corrisponde al 'corpo' delle tabelle (le righe).

3.4 Livelli di Modellazione

- **Modellazione Concettuale:**
  - Rappresenta i dati in modo indipendente da sistemi specifici.
  - Descrive concetti del mondo reale.
  - Usata nella fase preparatoria di un progetto (analisi dei requisiti).
  - Il linguaggio più diffuso è **Entity-Relationship (E-R)**.

    - *Esempio:* In un sistema universitario, identifichiamo entità come 'Studente', 'Corso', 'Docente' e relazioni come 'Studente SI ISCRIVE A Corso', 'Docente TIENE Corso'.

- **Modelli Logici:**
  - Usati dai DBMS per memorizzare e organizzare i dati.
  - Sono indipendenti dalla rappresentazione fisica.
  - Esempi: **Relazionale (SQL)**, gerarchico (antenato di JSON/XML), a oggetti, XML.
    - *Esempio:* Il diagramma E-R viene tradotto in uno schema di tabelle relazionali (`CREATE TABLE...`).

3.5 Architettura di un DBMS

Architettura standard ANSI/SPARC a tre livelli per garantire l'**indipendenza dei dati**:

1. **Schema Interno (o Fisico):**
   - Descrive come i dati sono fisicamente memorizzati (file, indici, puntatori).
   - È il livello più basso, nascosto agli utenti e alla maggior parte degli sviluppatori.
   - *Esempio:* Il DBMS decide di memorizzare una tabella in un certo file su disco e di creare un B-Tree index su una colonna.
2. **Schema Logico (o Concettuale, ma qui 'logico' è il termine usato per il modello del DBMS):**
   - Descrive la struttura dell'intero database usando un modello logico (es. il modello relazionale).
   - È la visione completa di tutte le tabelle, le relazioni tra esse, i vincoli, ecc.
   - È il livello a cui lavorano i DBA e gli sviluppatori backend (es. quando definisci le tabelle con `CREATE TABLE` o usi Prisma per definire i tuoi model).
3. **Schema Esterno (o Viste):**
   - Descrive una porzione del database per specifici utenti o applicazioni.
   - Può essere una 'vista' parziale dei dati (alcune colonne, alcune righe filtrate) o una combinazione di dati da più tabelle.
   - *Esempio SQL:*

     ```sql
     CREATE VIEW StudentiMarketing AS
     SELECT Matricola, Nome, Cognome
     FROM Studenti
     WHERE CorsoLaurea = 'Marketing';
     ```

     Questa vista mostra solo alcuni dati degli studenti di marketing, nascondendo altre informazioni o altri studenti.

3.6 Indipendenza dei Dati

È la capacità di modificare la definizione dello schema a un livello senza influenzare la definizione dello schema al livello superiore. È una conseguenza diretta dell'architettura a livelli.

- **Indipendenza Fisica dei Dati:**
  - La rappresentazione logica ed esterna è indipendente da quella fisica.
  - Si possono fare modifiche allo schema interno (es. cambiare algoritmi di accesso, aggiungere indici, spostare file su dischi diversi) senza dover modificare lo schema logico o le applicazioni che interrogano il DB.
  - *Esempio:* Il DBA aggiunge un indice a una tabella per migliorare le prestazioni. Le query SQL delle applicazioni continuano a funzionare come prima, senza modifiche.
- **Indipendenza Logica dei Dati:**

- Lo schema esterno (viste) è indipendente dallo schema logico.
- Si possono fare modifiche allo schema logico (es. aggiungere una colonna a una tabella, dividere una tabella in due mantenendo la possibilità di ricostruire l'originale) senza dover modificare le applicazioni che usano le viste, purché le viste possano ancora essere derivate dal nuovo schema logico.
- È più difficile da ottenere pienamente rispetto a quella fisica.
- *Esempio:* Se una tabella `Dipendenti` viene divisa in `DipendentiInfoPersonali` e `DipendentiInfoLavorative`, una vista `DipendentiCompleto` che fa il join delle due nuove tabelle può permettere alle vecchie applicazioni di funzionare senza modifiche.

3.7 Linguaggi per Database

I DBMS offrono diverse modalità di interazione:

- **Linguaggi testuali 'interattivi' (es. SQL):** L'utente scrive query direttamente in un client.
- **Statement SQL 'iniettati' in un linguaggio ospite (Host Language):** Comandi SQL incorporati in linguaggi di programmazione come Java, C, Python, Node.js.
  - *Esempio Node.js (con `pg` driver per PostgreSQL):*

```
const { rows } = await client.query('SELECT * FROM users WHERE id = $1', [userId]);
```

  - ORM come Prisma astraggono ulteriormente questo, permettendo di scrivere:

```
prisma.user.findUnique({ where: { id: userId } });
```

- **Linguaggi Ad Hoc (es. PL/SQL di Oracle, T-SQL di SQL Server):** Estensioni procedurali di SQL specifiche del DBMS.
- **Interfacce Utente Grafiche (GUI):** Strumenti visuali per interagire con il DB (es. pgAdmin, MySQL Workbench, MongoDB Compass).

3.8 DDL e DML (Separazione dati da software)

Due categorie principali di comandi SQL:

- **DDL (Data Definition Language):**
  - Usato per definire e modificare gli **schemi** (logico, esterno, fisico) e altre operazioni sulla struttura.
  - Comandi: `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`, `CREATE INDEX`, `CREATE VIEW`, `DROP VIEW`.
  - *Esempio:*

```
CREATE TABLE hours (
course CHAR(20),
teacher CHAR(20),
room CHAR(4),
hour CHAR(5)
);
```

- **DML (Data Manipulation Language):**
  - Usato per interrogare e aggiornare le **istanze** (i dati effettivi) del database.
  - Comandi: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

4 Attori e Ruoli

Diverse figure interagiscono con i database:

- **Progettisti e sviluppatori di DBMS:** Creano il software DBMS stesso.
- **Progettisti e sviluppatori di Database:** Disegnano lo schema del database per una specifica applicazione, scrivono query, stored procedure.
- **Amministratori di Database (DBA):**
  - Responsabili del controllo e della gestione centralizzata del database.
  - Si occupano di: efficienza (tuning prestazioni), affidabilità (backup, recovery), sicurezza (gestione permessi), installazione, aggiornamenti.
  - Spesso progettano anche il database, tranne in progetti molto complessi.
- **Progettisti e sviluppatori di applicazioni end-user:** Creano le applicazioni (web, mobile, desktop) che usano il database.
- **Utenti:**
  - **Utenti finali (operatori terminali):** Eseguono operazioni predefinite (transazioni di business, es. un cassiere in un supermercato).
  - **Utenti occasionali/casual:** Eseguono operazioni non definite a priori, usando linguaggi interattivi (es. un analista che esplora i dati con SQL).

5 Vantaggi e Svantaggi dei DBMS

5.1 Pro (Vantaggi)

- Dati come risorsa condivisa, modellano l'ambiente reale.
- Gestione centralizzata dei dati, standardizzabile e scalabile.
- Fornisce servizi integrati (query, sicurezza, backup, recovery).
- Riduce ridondanze e inconsistente.
- **Indipendenza dei dati:** Supporta lo sviluppo e la gestione delle applicazioni software (le app non devono preoccuparsi di come i dati sono memorizzati fisicamente).

5.2 Contro (Svantaggi)

- I prodotti DBMS (specialmente quelli commerciali enterprise) possono essere costosi, così come l'adozione di tali soluzioni (richiede personale specializzato).
- Le funzionalità integrate possono a volte ridurre l'efficienza specifica per compiti molto particolari, rispetto a soluzioni custom altamente ottimizzate (ma questo è un caso limite).