Introduzione

Le Basi di Dati e i loro utenti

Cos’è una base di dati?

- Una **base di dati** è una raccolta organizzata (strutturata) di informazioni che devono essere gestite per un periodo di tempo lungo, anche di molti anni

- Più precisamente, si può dire che una base di dati è una raccolta di informazioni gestita da un *database management system* (DBMS)

Esempi di Basi di Dati

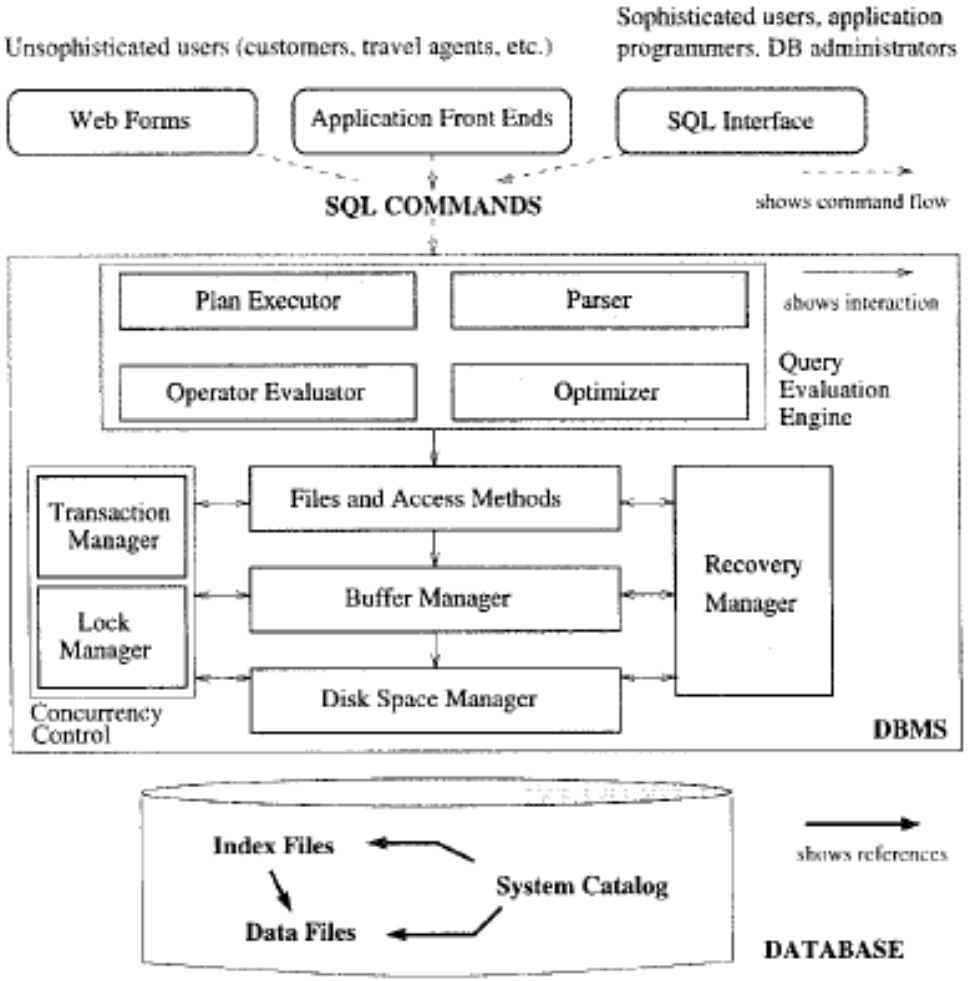
- I dati relativi agli studenti, ai corsi e agli esami dell'Università di Trento
- Il catalogo dei prodotti di un marketplace elettronico
- I Points of Interest (POI) su una mappa (es. Google Maps)
- Il Pubblico Registro Automobilistico (PRA) dell'Automobile Club Italiano (ACI)
- I pagamenti delle tasse da parte dei contribuenti di un Comune
- Gli orari dei voli e le prenotazioni dei passeggeri di una compagnia aerea
- ….

DBMS e Database System

- **Database Management System (DBMS):**
  - Pacchetto software progettato per gestire basi di dati informatizzate (vedi sotto)

- **Database System:**
  - E' l'insieme costituito dal software del DBMS insieme ai dati memorizzati nella base di dati.
  - Talvolta nella definizione sono incluse anche le applicazioni software esterne che interagiscono con il DBMS per eseguire i propri compiti.

Architettura di un Database System



Cosa deve fornire un DBMS?

- Un *data definition language* (DDL), ovvero un linguaggio che permetta di definire schemi per strutturare i dati
- Un *data manipulation language* (DML o anche *query language*), per modificare i dati ed estrarli dalla base di dati
- Supportare la *memorizzazione* di grandi quantità di dati
- Gestire la *robustezza* dei dati e il loro recupero in caso di fallimenti, errori o azioni malevole
- Controllare l'accesso ai dati di più utenti contemporaneamente (concorrenza), evitando interazioni indesiderate tra utenti (*isolation*) e modifiche incomplete dei dati (*atomicity*).

Interazione tra applicazioni e basi di dati

- I programmi applicativi interagiscono con la base di dati generando:
  - Interrogazioni (*queries*): accesso a differenti porzioni di dati e formulazione della risposta
  - Transazioni (*transactions*): sequenze atomiche di azioni (letture / scritture) sulla base di dati
- Le applicazioni non devono consentire l'accesso ai dati ad utenti non autorizzati

Sistemi di Basi di Dati relazionali

- Un enorme impulso allo sviluppo delle Basi di Dati è venuto dal lavoro di Ted Codd (1970)
- Codd ha introdotto l'idea di mostrare agli utenti una visione logica dei dati, indipendente dalla struttura fisica
- Le interrogazioni vengono espresse attraverso un linguaggio di alto livello, che utilizza il modello logico e non quello fisico dei dati

Indipendenza dei dati

- **Separazione tra programmi e dati:**
  - Questa proprietà è spesso chiamata **indipendenza programmi-dati**
  - Permette di modificare le strutture dati e l'organizzazione fisica dei dati senza dover cambiare l'accesso dei programmi alla base di dati via DBMS
- **Vantaggi**:
  - Nei sistemi basati su file, la struttura dei file di dati è contenuta nei programmi che devono accedere ai dati
  - Se cambia la struttura del file di dati, è probabile che debbano essere modificati i programmi che vi accedono
  - Questo rende molto **difficile** e **costosa** la manutenzione nel tempo del software del sistema

Indipendenza logica e fisica dei dati

Indipendenza Logica dei Dati:

- **Definizione**: La capacità di modificare lo schema logico del database senza dover alterare lo schema esterno o le applicazioni che utilizzano il database.
- **Vantaggi**:
  - consente di aggiungere, rimuovere o modificare campi e tabelle senza influire sulle applicazioni esistenti
  - migliora la flessibilità e la gestione dell'evoluzione del database
- **Esempio**: Aggiungere un nuovo attributo a una tabella senza modificare il codice delle applicazioni che accedono a quella tabella.

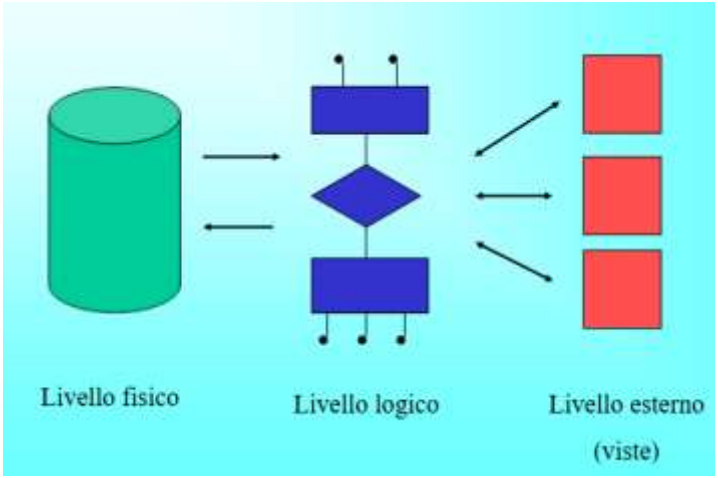
Indipendenza logica e fisica dei dati

Indipendenza Fisica dei Dati:

- **Definizione**: La capacità di modificare lo schema fisico del database senza influenzare lo schema logico o le applicazioni.
- **Vantaggi**:
  - Permette di ottimizzare le prestazioni (ad es. cambiando la struttura di memorizzazione o gli indici) senza alterare la logica delle applicazioni.
  - Migliora la scalabilità e la gestione delle risorse hardware.
- **Esempio**: Spostare un database su un nuovo hardware o ristrutturare l'archiviazione dei dati senza cambiare la modalità di accesso dei dati per l'utente finale

Livelli di astrazione dei dati

- **Schema logico**: modello che rappresenta i dati nei termini del modello proprio del DBMS (per esempio, il modello relazionale nei RDBMS)
- **Schema fisico**: fornisce dettagli su come le relazioni descritte dallo schema logico sono memorizzate su memoria secondaria. Per esempio: file di record ordinati / non ordinati, creazione di indici (di che tipo), ecc.
- **Schema esterno**: permette di creare una o più **viste** specifiche per velocizzare / facilitare / autorizzare l'accesso a insiemi di dati a specifici utenti o gruppi di utenti



Fonte immagine: https://slideplayer.it/slide/551201/

Livelli di astrazione dei dati: esempio

- Schema logico per un DB universitario

Students(*sid:* `string`, *name:* `string`, *login:* `string`,
age: `integer`, *gpa:* `real`)
Faculty(*fid:* `string`, *fname:* `string`, *sal:* `real`)
Courses(*cid:* `string`, *cname:* `string`, *credits:* `integer`)
Rooms(*rno:* `integer`, *address:* `string`, *capacity:* `integer`)
Enrolled(*sid:* `string`, *cid:* `string`, *grade:* `string`)
Teaches(*fid:* `string`, *cid:* `string`)
Meets_In(*cid:* `string`, *rno:* `integer`, *time:* `string`)

- Schema fisico: utilizzare un file non ordinato di record e creare un indice B+Tree sulla prima colonna di STUDENTS e COURSES
- Esempio di schema esterno: creare una relazione che mostri tutti i corsi con il relativo docente e numero di iscritti:

Courseinfo(*cid:* `string`, *fname:* `string`, *enrollment:* `integer`)

Perché usare una base di dati?

- Indipendenza (logica e fisica) dei dati (già vista)
- Accesso concorrente ai dati
- *Crash recovery*

… e poi:

- Integrità dei dati e sicurezza
- Amministrazione dei dati
- Riduzione dei tempi di sviluppo di applicazioni

Accesso concorrente ai dati e *crash recovery*

- **Condivisione dei dati e transazioni multi-utente**
  - Permette a gruppi di utenti di leggere e scrivere contemporaneamente (**concurrent users**) sulla stessa base di dati
  - Il **controllo della concorrenza** nel DBMS garantisce che ogni transazione sia eseguita correttamente o annullata
- Il sottosistema di ***recovery*** assicura che ogni transazione correttamente completata sia memorizzata in modo permanente nella base di dati
  - Questo permette di eseguire centinaia di transazioni concorrenti al secondo mantenendo validità e coerenza dei dati – si parla di **OLTP** (*Online Transaction Processing*)

Digressione: OLTP vs. OLAP

OLTP (Online Transaction Processing):

- **Finalità**: Gestire e facilitare le transazioni quotidiane di un'azienda (e.g., ordini, pagamenti).
- **Caratteristiche**:
  - **Alta frequenza** di transazioni con operazioni brevi e semplici (insert, update, delete).
  - **Data integrity** e coerenza sono fondamentali.
  - **Schema di database** normalizzato per minimizzare la ridondanza.
  - **Risposta rapida** alle query per garantire l'efficienza delle operazioni.
- **Esempio**: Sistemi di gestione di e-commerce, banche, punti vendita.

Digressione: OLTP vs. OLAP - II

OLAP (Online Analytical Processing):

- **Finalità**: Supportare l'analisi complessa dei dati per il business intelligence e il reporting.
- **Caratteristiche**:
  - **Bassa frequenza** di operazioni, ma con query complesse e di lunga durata.
  - Ottimizzato per **query ad-hoc** e analisi multidimensionali.
  - **Schema di database** denormalizzato (es. *star schema*) per migliorare le performance delle query.
  - **Aggregazione e storicizzazione** dei dati per analisi tendenziali e strategiche.
- **Esempio**: Sistemi di reporting aziendale, dashboard di business intelligence.

Sistemi per OLTP e OLAP

- **Software per OLTP:**
  - **Database Relazionali (RDBMS)**. Esempi: MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, IBM DB2, Oracle Database.
  - **Software ERP (Enterprise Resource Planning)**. Esempi: SAP, Oracle, Panthera, TeamSystem, ….
- **Software per OLAP:**
  - **Data Warehousing Platforms.** Esempi: Amazon Redshift, Google BigQuery, Snowflake, Microsoft Azure Synapse Analytics
  - **Business Intelligence Tools.** Esempi: Tableau, Power BI, Qlik Sense, SAP BusinessObjects

Utenti di una Base di Dati

- Si possono dividere in due gruppi principali:
  - Quelli che utilizzano e controllano il contenuto della base di dati, uniti a coloro che progettano e sviluppano applicazioni sulla base di dati (**utilizzatori**)
  - Quelli che progettano e sviluppano il software del DBMS con tutti i suoi sottosistemi (**sviluppatori**)

Tipi di utilizzatori

- **Database administrator [Amministratore]:**
  - E' la figura responsabile dell'autorizzazione di accesso alla base di dati, del coordinamento e del monitoraggio del suo utilizzo, dell'acquisto di risorse software e hardware, del monitoraggio dell'efficienza operativa del sistema.

- **Database Designer [Progettista]:**
  - E' la figura responsabile della definizione del contenuto, dello schema, dei vincoli e delle transazioni che possono essere eseguite sulla BD. Comunicano con gli utenti per raccogliere le loro necessità.

Tipi di utilizzatori

- **End-users [Utenti finali]:** Utilizzano i dati mediante query e report. Alcuni possono aggiornare i contenuti della base di dati. Possono essere suddivisi in:
  - **Casual [Occasionali]**: fanno accesso alla base di dati solo occasionalmente, quando necessario. Ad esempio, top manager che accedono in lettura a dati fortemente aggregati
  - **Naïve** or Parametric [non-esperti]: costituiscono la maggior parte degli utenti finali.
    - Tipicamente utilizzano funzioni predefinite (le cosiddette "*canned transactions*") per interagire con la base di dati
    - Esempi: cassieri di banca, impiegati di segreteria, utenti dei social media, utenti delle applicazioni web, ecc.

Tipi di utilizzatori

- **Sophisticated [Esperti]:**
  - Questo Gruppo include business analysts, data scientists, ingegneri e tutti quello che possiedono conoscenza approfondita sul funzionamente del sistema
  - Spesso utilizzano strumenti software che lavorano a un livello molto vicino a quello della base di dati
- **Stand-alone [Indipendenti]:**
  - Per lo più utenti che creano una propria base di dati utilizzando applicazioni pronte all'uso
  - Un esempio potrebbe essere il commerciale di un'azienda che si crea una base di dati con i propri contatti

Quando è meglio NON usare un DBMS

- Cosa può inibire l'adozione di una soluzione con DBMS:
  - Alti investimenti iniziali e necessità di acquistare ulteriore hardware da dedicare al funzionamento della base di dati
  - Costi aggiuntivi per fornire sicurezza, controllo di concorrenza, recovery, funzioni di integrità
- Quando un DBMS potrebbe NON essere necessario:
  - Se la base di dati e l'applicazione sono molto semplici, ben definite e non soggette a continue modifiche
  - Se non è richiesto l'accesso di molteplici utenti
- Quando adottare un DBMS può essere impossibile:
  - In molti casi di *embedded systems*, dove possono mancare adeguate capacità di memorizzazione per un DBMS *general purpose*

Quando è meglio NON usare un DBMS

- In caso di requisiti di tempo reale molto stringenti (es., sistemi di switch telefonico)
- Quando il modello dei dati del DBMS non è abbastanza espressivo per gestire la complessità dei dati da memorizzare (es., basi di dati genomici e proteici)
- Se gli utenti hanno bisogno di operazioni speciali non supportate dallo specifico DBMS (es., GIS e sistemi basati sulla localizzazione)