1 Introduzione a basi di dati

Un sistema che deve occuparsi di dati e la loro rispettiva gestione viene definito **sistema organizzativo**. Tale sistema è composto da *risorse* e *regole* che permettono di svolgere attività (o processi) allo scopo di compiere un obiettivo.
La parte che si occupa della raccolta, archiviazione ed elaborazione delle informazioni è chiamato **sistema informativo** che svolge processi informatici ed esiste indipendentemente da automatizzazioni informatiche. La variante automatizzata è conosciuta come **sistema informatico**, l'insieme di informazioni è detto **basi di dati**. Una base di dati è di grandi dimensioni, ha vita indipendentemente dai programmi che la usano, essendo una risorsa condivisa in un sistema e rappresenta le informazioni sotto forma di simboli, essendo un'informazione complessa da salvare. Idealmente 2 archivi separati per contenere tipologie di informazioni potrebbe risultare la migliore opzione, ma potrebbero presentarsi problemi di incoerenza tra le informazioni. Una base di dati invece, grazie all'integrazione e la condivisione di dati, ovvia il problema. Una definizione più precisa è:

"Una base di dati è un insieme di dati grandi, organizzati e gestiti da un DBMS"

1.1 Database Managment System

Un **DBMS** o DataBase Managment System, è un sistema che permette di gestire grandi basi di dati, offrendo dei servizi potenti e funzionali. All'interno del DBMS esistono dei dati specifici utilizzati dai programmi che formano il **dizionario**, che contiene la definizione centralizzata dei dati nel database.
I pro nell'utilizzo di un DBMS associato ad un db sono:

- Economia a scala;
- Servizi integrati;
- Indipendenza dei dati;

Mentre i contro:

- Costo dei prodotti e delle transazioni (a livello di memoria);
- Le funzioni non possono venire scorporate, con conseguente perdita di prestazioni;

I servizi offerti sopracitati sono:

1. **Privatezza**: non tutti i dati sono accessibili da tutti gli utenti. Esistono diversi meccanismi di autorizzazioni a seconda dell'utente che richiede l'accesso;
2. **Efficienza**: lo sfruttamento delle risorse è ottimizzato, ma i DBMS con troppe funzioni non scorporabili potrebbero comunque avere problemi prestazionali in alcuni casi;
3. **Efficacia**: le funzioni offerte sono potenti ed efficaci, ma il DBMS deve essere ben strutturato;
4. **Affidabilità**: la base di dati è resistente a problemi hardware e software, grazie a backups e recovery functions.

1.2 Transazione

Una transazione è una serie di istruzioni da eseguire sulla base di dati e garantisce l'affidabilità essendo **indivisibile**, **corretta** e **definitiva**.
La prima proprietà è garantita grazie all'**atomicità** dell'operazione, ovvero: data una transazione e le varie istruzioni che la compongono, questa deve essere eseguita per intero o altrimenti viene annullata. Un **controllo di concorrenza** permette di verificare la *correttezza* della transazione: se più transazioni richiedono di essere eseguite insieme, si è in grado di serializzare le operazioni per farle una dopo l'altra. Infine è tenuta traccia di ogni operazione, con il suo risultato, andata a buon fine anche in presenza di guasti o problemi di correttezza.
Un **architettura distribuita** caratterizza i DBMS e permette di mettere in comunicazione più macchine tra di loro tramite una rete. L'**architettura client-server** o **architettura two-tier** è la più diffusa e la più semplice: è costituita da un server e da uno o più client.
Il **server** svolge un ruolo reattivo, ovvero si attiva solo al ricevimento di richieste ed il suo compito è quello di offrire servizi al richiedente. Il client quindi svolge un ruolo attivo e chiede i servizi al server. Esistono 2 tipi di client:

- *Thin client*, dove la struttura logica dell'applicazione risiede nel server;
- *Thick client*, dove la struttura logica dell'applicazione risiede nel client. Ciò richiede una fiducia nel client essendo lui ad inviare dati al server ed inoltre soffre di non scalabilità, quindi può comprende pochi client nel sistema complessivo.

Un'altra architettura importante è la **three-tier**, che basandosi sul concetto della two-tier aggiunge un ulteriore server di intermezzo tra db e client che gestisce la logica dell'applicazione. I vantaggi di questa architettura sono:

1. Possibilità di comprendere sistemi *eterogenei* nello schema;
2. Tutti i client sono di tipo thin;
3. Scalabilità dei client;

1.3 Un accenno ai Big Data

Per **Big Data** si intende l'insieme di tutti i piccoli dati raccolti in grandi quantità. Caratterizzati da la **regola delle 4 V**, ovvero:

- *Volume*, dovuto alla grande quantità dei dati;
- *Velocità*, data dalla generazione ed analisi dei dati in un breve lasso di tempo;
- *Varietà* dei dati, essendo ormai di tipo non strutturato (file, immagini, testo, ...);
- *Veridicità* grazie alla quale si è in grado di estrarre informazioni veritiere anche da dati parzialmente completi o imprecisi (**data quality**);

La scienza che si occupa dell'analisi dei dati è la **Data Science**, un ibrido tra statistica ed informatica e comprende diversi passi intermedi:

1. **Data cleaning**

   Costruzione di una raccolta di dati di un sufficiente livello qualitativo;
2. **Data Integration**

   Costruzione di una raccolta di dati integrata a diverse fonti;
3. **Data Mining**

   Processo di estrazione di informazioni utili dai dati;
4. **Metodi predittivi**

   Metodi che permetto di predire i dati di situazioni future da dati di situazioni passate;
5. **Machine Learning**

   Classificazione predittiva dei dati, in particolare il **deep learning** cerca di imitare il comportamento del cervello umano per classificare i dati.

Essendo i Big Data in continua crescita, lo sono di conseguenza anche i Big Users e ciò ha portato le grandi aziende ad organizzarsi per poter archiviare tutti i dati nei database. Si parla di **scalare i db**, orizzontalmente (se si vogliono usare dei db standard) e verticalemente (se si intende aggiornare i server con altri più preformanti).

2 Il modello logico relazionale

Per visualizzare gli schemi di un database si ricorre ad una vista astratta dei dati, che ci permette di capire facilmente su cosa stiamo lavorando o cosa stiamo creando. Il **modello logico relazionale** si basa su delle relazioni tra tabelle, dove è presente uno **schema** che rimane solitamente invariato una volta creato e rappresenta la struttura del db e tante **istanze** i cui valori possono cambiare rapidamente. Ci sono inoltre diversi linguaggi per comunicare con una base di dati:

- D.D.L per la modifica degli schemi
- D.M.L per l'interrogazione e l'aggiornamento delle istanze, a cui si appoggia un altro linguaggio conosciuto come **sql**.

Il modello relazione si basa su relazioni matematiche del tipo:

$$Partite \subseteq string\ X\ string\ X\ int\ X\ int$$

Dove ogni elemento contenuto in partite è detto **dominio**. Ciascun dominio ha un proprio tipo ed assume un ruolo diverso a seconda della posizione in cui si trova. Si parla dunque di **struttura posizionale**.

$D_1, ..., D_n :=$ Insieme di domini, per ogni dominio esiste un **attributo** che ne descrive il ruolo
D_1 x ... x $D_n :=$ **prodotto cartesiano** di domini, insieme di tutte le n-uple $d_i \in D_i$, $i = 1, .., n$

Una **relazione** è un sottoinsieme del prodotto cartesiano:
$r \subseteq D_1$ x D_2

Nel modello relazionale non hanno importanza gli ordinamenti che possono esserci tra colonne (domini) e righe (record o insieme di n-uple).
I **vantaggi** di tale modello sono:

1. Indipendenza delle strutture fisiche
2. Portabilità dei dati tra sistemi diversi
3. Sia l'utente che il programmatore vedono gli stessi dati
4. Gli eventuali puntatori presenti a livello fisico sono invisibili a livello logico

Si identifica uno **schema di relazione** (tabella) con R$(A_1, ..., A_n)$ quindi una relazione (R) con un insieme di attributi (A), mentre uno **schema di una base di dati** come una relazione di relazioni con R$(R_1(A_1), ..., R_n(A_n))$.