BASE DI DATI

Corso A

Autore
Giuseppe Acocella
2025/26
https://github.com/Peenguino

Ultima Compilazione - October 5, 2025

Contents

1 Introduzione

I database sono insiemi di dati omogenei gestiti in collezioni. Alla base di questo definiamo tabelle, i cui campi possono fare riferimento ad altre tabelle del database. Il topic di studio del corso è quello dei **database**, ma in questa introduzione definiamo un confronto con i **Data Warehouse**, per evidenziarne le **differenze**, anche non essendo parte del programma del corso.

1.1 OLTP vs OLAP

Mettiamo a confronto i due tipi di **ambiti applicativi**:

1. **OLTP - Database (transazionale)**:
   (a) Utilizzo comune.
   (b) Molti users.
   (c) Dati analitici e relazionali.
   (d) Relazioni statiche.
   (e) Una query altera solitamente pochi record della tabella.
   (f) Mirato all'utilizzo da parte delle applicazioni.
   (g) Aggiornamenti frequenti.
   (h) Visione dei dati correnti.
   (i) Pensato per le transazioni.
2. **OLAP - Data Warehouse (analitico)**:
   (a) Pochi utenti esperti.
   (b) Dati multidimensionali.
   (c) Relazioni dinamiche.
   (d) Una query altera molti record della tabella.
   (e) Mirato ai soggetti.
   (f) Aggiornamenti rari ma massivi.
   (g) Visione dei dati storica.
   (h) Pensato per l'analisi di trend.

Per questa motivazione, se dei dati presenti in un database, dovessero servire per un analisi di trend andrebbe effettuata un operazione abbastanza complessa di estrazione e preparazione per l'immagazzinamento nel Data Warehouse. Entrambi (DB e DW) seguono una politica **schema first**, ossia viene prima definito uno schema (insieme di campi) su cui verrà basata la successiva popolazione della collezione di informazioni.

Big Data Un esempio di collezione di dati che **non segue** una politica **schema first**, basandosi infatti sulle proprietà di volume, varietà e velocità non possono mantenere la rigidità impostata da uno schema. Solitamente sono quindi associati a sistemi NoSQL o approcci Data Lake.

1.2 Data Base Management Systems - DBMS

Un DBMS (sistema per basi di dati) è un sistema centralizzato o distribuito che offre opportuni linguaggi per:

1. Definire lo **schema** di una DB.
2. Scegliere le **strutture dati** a **supporto** della DB.
3. **Memorizzare** dati seguendo i vincoli definiti dai schemi del DB.
4. Recuperare e modificare dati del DB tramite interrogazioni (**query**).

Solitamente si pone tra i programmi applicativi e l'effettivo database per permettere l'interazione vincolata tra i due.

Dati gestiti dai DBMS Solitamente in un DB sono contenuti:

1. **Metadati**: Descrivono permessi, applicazioni, parametri quantitativi sui dati effettivi. Seguono uno schema definito dal DBMS stesso.
2. **Dati**: Rappresentazioni di fatti conformi alle definizioni degli schemi del DB.
   (a) Sono organizzati in **insiemi** strutturati ed **omogenei**, tra i quali sono definite delle **relazioni**.
   (b) Sono accessibili tramite **transazioni**, operazioni atomiche che non hanno **mai effetti parziali**.
   (c) Sono protetti da accessi non autorizzati e preservati da possibili malfunzionamenti.
   (d) Sono utilizzabili in maniera concorrente da più utenti.

DBMS a Modello Relazionale Il modello relazionale è il più comune tra i DBMS commerciali e si basa sull'astrazione della **relazione**, ossia la **tabella** vista come un insieme di record con campi ben definiti. Questo ci permette di poter creare tabelle ed interrogarle con un linguaggio ad alto livello.

Funzionalità dei DBMS Elenchiamo quindi le proprietà garantite da un DBMS:

1. Linguaggio per la definizione di un DB.
2. Linguaggio per l'uso dei dati nel DB.
3. Meccanismi di controllo del DB.
4. Strumenti per la gestione admin del DB.
5. Strumenti per lo sviluppo delle app che richiedono dati dal DB.

1.2.1 Livelli di Vista Dati dei DBMS

Per garantire le proprietà di **indipendenza fisica** e **logica** dei dati è stato proposto l'approccio di tre livelli di descrizione dei dati.

1. **Indipendenza Fisica**: Le applicazioni che utilizzano il DB **non** devono essere modificati in seguito a modifiche dell'organizzazione fisica dei dati nel DB.
2. **Indipendenza Logica**: Le applicazioni che utilizzano il DB **non** devono essere modificati in seguito a modifiche dello schema logico del DB.

Gli effettivi livelli di vista sono invece:

1. **Livello Fisico**: Gestione effettiva dell'immagazzinamento dei dati nel DB, ad esempio in questo livello si scelgono le strutture dati e gli algoritmi utilizzati dal DBMS per navigare tra i dati.
2. **Livello Logico**: Descrizione della struttura degli insiemi di dati e delle relazioni tra di loro, astraendo completamente dalla loro gestione fisica.
3. **Livello Vista Logica**: Sottoinsieme del livello logico esposto alle applicazioni esterne.

1.2.2 Meccanismi di Controllo Dati e Transazioni dei DBMS

I DBMS cercano di garantire queste proprietà sui dati immagazzinati in un DB:

1. **Integrità**: Mantenimento delle proprietà definite dallo schema.
2. **Sicurezza**: Protezione dei dati da usi non autorizzati.
3. **Affidabilità**: Protezione in caso di malfunzionamenti hardware/software.

Transazioni - Operazioni Atomiche Una transazione è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea secondo queste proprietà:

1. **Atomicità**: Le transazioni terminate prematuramente sono trattate come se non fossero mai iniziate. I loro effetti sul DB sono nulli.
2. **Persistenza**: Le transazioni terminate con successo sono permanenti, ossia non alterabili neanche da malfunzionamenti.
3. **Serializzabilità**: L'esecuzione concorrente di più transazioni è vista come un esecuzione seriale di transazioni.

Riepilogo Pro e Contro Utilizzo DBMS Elenchiamo rapidamente i pro e i contro di questo approccio:

1. **Pro**: Indipendenza dei dati, recupero efficiente dei dati, integrità e sicurezza, accessi interattivi e concorrenti, amministrazione e riduzione dei tempi di sviluppo delle applicazioni.
2. **Contro**: Necessaria la definizione di uno schema, gestiscono solo dati strutturati ed omogenei, ottimizzati per app OLTP e non per OLAP.

2 Progettazione di una Base Dati

La nascita dei database è causata da alcune problematiche presenti in sistemi di gestione informazioni più datati. Un classico esempio di problematica è quella della **ridondanza logica**, ossia un informazione ripetuta più volte tra vari record. In queste casistiche si preferisce astrarre e fare riferimento solo una volta ad un dato.

2.1 Attori e Fasi della Progettazione di DB

Elenchiamo attori e fasi della progettazione di una base dati.

Attori della Progettazione Elenchiamoli:

1. **Committente**: Azienda che commissiona la creazione di una base dati, per una propria necessità.
2. **Consulente**: Progettisti del DB.
3. **Utente**: Chi usufruirà del DB, solitamente un dipendente del committente.
4. **DB Administrator**: Amministratore del DB.

Fasi della Progettazione Si definiscono fasi specifiche della progettazione di un DB:

1. **Specifica Requisiti Committente**: Il committente deve definire le proprie necessità ed il consulente deve raccogliere le informazioni.
2. **Progettazione Concettuale**: Realizzazione di uno schema concettuale orientato agli oggetti che deve essere osservabile ed approvato dal committente. In questa fase si astrae completamente da dettagli tecnici d'implementazione/ottimizzazione proprio perchè deve risultare semplice al committente e non deve causare ridondanza logica.
3. **Progettazione Logica**: Concretizzazione della progettazione concettuale tramite linguaggi relazionali.
4. **Progettazione Fisica**: Allocazione fisica delle tabelle generate dal linguaggio relazionale della progettazione logica.

Progettazione come Modellazione Spesso il committente stesso non riesce a rendere esplicite tutte le sue necessità, di conseguenza è compito del consulente capire a fondo i comportamenti ed i dettagli necessari alla modellazione logica.

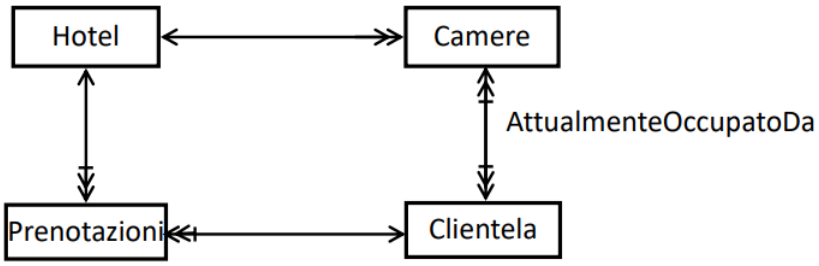
2.2 Progettazione Concettuale

Questo linguaggio si basa su 3 operatori diversi:

1. **Classi (aka Collezioni)**: Ad esempio la classe Persone di entità persona. Formalmente una classe modella un insieme di entità omogenee. Queste possono essere entità fisiche, avvenimenti o **modelli (progetti)** di entità.

2. **Associazioni**: Insieme di fatti binari, ad esempio associazione di un proprietario ad un auto.

3. **Sottoclassi**: Sottoinsieme di una classe, come Studenti può esserlo di Persone.



2.2.1 Associazioni e Cardinalità

Formalmente le associazioni sono insiemi di coppie, quindi delle relazioni. Il tipo di freccetta che indica l'associazione è detta cardinalità e corrisponde informalmente alla domanda:

"Per ogni elemento della classe A quanti della classe B?"

Chiaramente va fatto sulla stessa direzione per entrambi i versi. Questo permette di risolvere le ambiguità causate dalla terminologia comune che è detta:

"Uno a Molti" oppure "Molti a Molti"

che in qualche modo genera ambiguità perchè è come se si tenesse in conto solo di un verso. E' fondamentale ricordarsi quindi che la caratterizzazione della cardinalità di un associazione va fatta in entrambi i versi.

Graficamente quindi avremo queste possibilità alle estremità delle associazioni:

1. **Freccia Singola**: Per ogni elemento della classe di partenza è presente un elemento nella classe d'arrivo.

2. **Freccia Doppia**: Per ogni elemento della classe di partenza sono presenti più elementi nella classe d'arrivo.

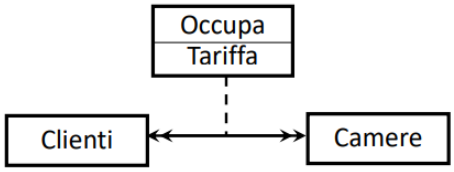
3. **Trattino**: Nessun limite inferiore, di conseguenza per ogni elemento della classe di partenza può anche non essere presente alcun elemento nella classe d'arrivo.

Esistono diverse notazioni grafiche, ma queste dispense fanno riferimento a quelle utilizzate durante gli esercizi del corso, quindi questa sarà la notazione comune di riferimento.

2.2.2 Associazioni Ternarie/con Attributi e Reificazione

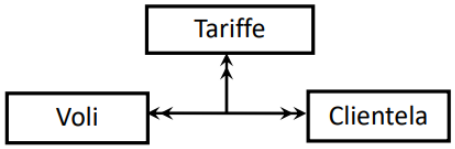
A volte le associazioni potrebbero complicarsi perchè:

1. Potrebbero avere la **necessità** di avere degli **attributi**. Ad esempio in un caso di associazione tra Clienti e Stanze, la tariffa potrebbe non essere nè attributo di Clienti e nè di Stanze. In questo caso si assegna un **attributo** all'**associazione**.



2. Potrei invece immaginare l'attributo come **vera** e **propria entità** di una specifica classe Tariffe. In quel caso non starei semplicemente dando un attributo

   all'associazione ma starei componendo un **associazione ternaria**.



Reificazione Si preferisce, nei casi illustrati sopra, semplificare la gestione dell'associazione tramite processo di reificazione, ossia la creazione di una classe di supporto aggiuntiva che permetta la gestione regolare delle associazioni viste prima. Si illustrano le reificazioni delle associazioni viste sopra:

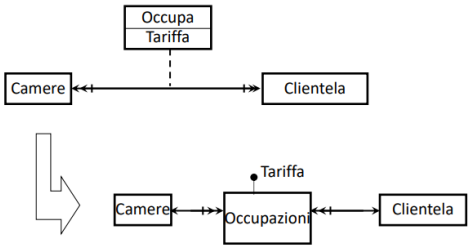


Figure 1: Reificazione di Associazione con Attributo

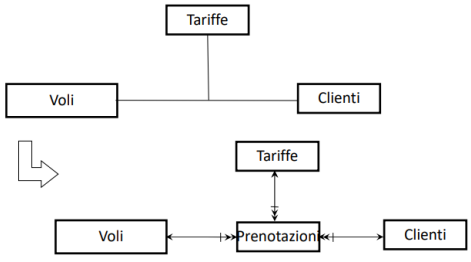


Figure 2: Reificazione di Associazione Ternaria

Un dettaglio da notare è la **cardinalità** sulla nuova classe di supporto, infatti in **direzione** della **nuova classe** sarà presente un associazione di **uno a molti**.

2.2.3 Sottoclassi e Tipologie

Una **sottoclasse** è un sottoinsieme di elementi di una classe, per i quali prevediamo di raccogliere ulteriori informazioni.

$$\text{Studenti} \subseteq \text{Persone}$$

$$\text{Libri Rari} \subseteq \text{Libri}$$

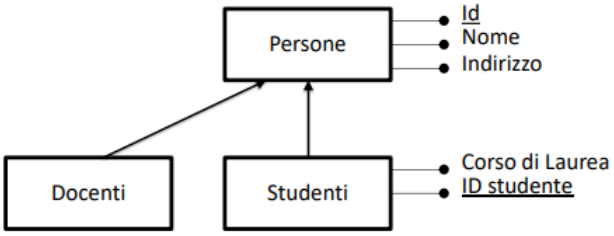
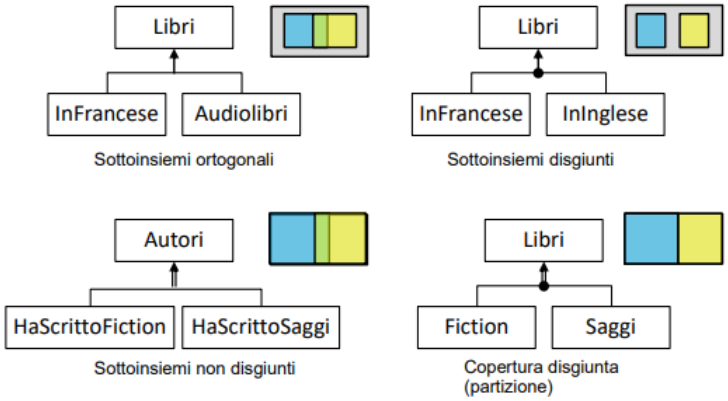


Figure 3: Notazione grafica di riferimento

Esistono varie tipologie di sottoclassi:



2.3 Progettazione Logica

Il **tipo** di **schema logico** presentato in questo corso è lo **schema relazionale**.

Basata su **schemi relazionali** detti informalmente **tabelle** di valori elementari con chiavi primarie:

Impiegati			
IdImpiegato	Nome	Stipendio	IdReparto*
232	Lucia	1200	Y1
143	Luigi	1500	X2

In questo contesto **non esiste differenza** tra **attributi** ed **associazioni** dato che le chiavi esterne permetteranno il **riferimento** ad **altre tabelle**.

Reparti	
IdReparto	Budget
Y1	100000
X2	200000

Solitamente quindi si definisce **uno dei campi** della **tabella** come **chiave primaria**, ossia che **identifica** univocamente **la riga**. Invece il campo che permette la dereferenziazione univoca di una riga in un altra tabella è detta **chiave esterna**, che corrisponde alla **chiave primaria** della **tabella esterna**.

Chiave e Superchiave Minimale Definiamo **superchiave** qualsiasi **insieme di attributi** che **non** può **ripetersi** in **righe diverse**. Si definisce **chiave** una **superchiave minimale**, ossia un insieme di attributi a cui non possiamo rimuovere alcun attributo. Sarà quindi scelta del progettista scegliere una **chiave primaria tra** tutte le **chiavi possibili**.

2.4 Da Schema Concettuale a Schema Logico - Fasi

Elenchiamo e descriviamo le fasi necessarie per il passaggio da schema **concettuale** a **logico**.

1. Aggiungere una **chiave primaria** artificiale ad ogni collezione che ne ha bisogno.
   (a) Una chiave deve essere **immutabile**, **muta**, ossia non deve portare con se alcuna informazione, ed **invisibile** agli utenti.
2. Tradurre le **associazioni** e le **inclusioni** in **chiavi esterne**.
   (a) Una associazione $1 - N$ diventano **chiavi esterne**.
   (b) Una associazione $M - N$ diventano **tabelle** con due chiavi esterne che puntano alle tabelle tra cui esiste l'associazione.
3. Tradurre gli **attributi multivalore** in **tabelle**.
4. **Appiattire** gli **attributi complessi**, ossia da struct a lista di parametri semplici.