Riassunto primo compitino
Basi Di Dati

1. Discutere il concetto di indipendenza dei dati e spiegare perché il modello relazionale è più adatto di altri (ad es. gerarchico e reticolare) per ottenerla.

Il modello relazionale definisce tipi attraverso il costruttore *relazione,* che organizza i dati secondo **record a struttura fissa**, rappresentabili attraverso tabelle. (modello tabellare dei dati)
Il concetto di **indipendenza dei dati** consiste nel fatto che gli utenti hanno la possibilità di operare ad un livello astratto, ossia indipendentemente dai dettagli realizzativi del DBMS.

- *Indipendenza fisica:*
  I. Consente di mantenere inalterata la struttura logica dei dati al variare della realizzazione fisica del sistema.
  II. Consente di utilizzare basi di dati su piattaforme diverse, o di distribuire una base di dati su più macchine. (Portabilità)
- *Indipendenza logica:*
  I. Rende indipendente lo schema esterno da quello logico, consentendo di inserire nuove viste senza alterarlo, o di alterarlo mantenendo inalterate le viste definite in precedenza.
  II. Permette di accedere al DB in modo indipendente da struttura logica dei dati (per esempio, tabelle)

Il modello relazionale nato agli inizi degli anni '70 da Codd è stato creato con la finalità di realizzare l'indipendenza dei dati e attualmente risulta essere il modello più utilizzato.
E' stato teorizzato per separare il più possibile il livello logico dal livello fisico della descrizione dei dati; in questo modo si evita all'utente di dover conoscere l'effettiva gestione fisica dei dati da parte del SO. Esso si basa sul concetto di **relazione** e di **tabella.**
Inoltre, i riferimenti fra dati in relazioni diverse avvengono attraverso la corrispondenza dei valori con cui nelle tuple, che sono logicamente collegate, si instanziano domini corrispondenti.

Relazione:

I. rappresentazione di un'entità complessa tramite attributi;
II. **insieme di record omogenei**, cioè definiti sugli stessi campi.
III. Una relazione matematica su due insiemi D1 e D2 è un sottoinsieme di D1 x D2.
IV. **Relazione**: concetto mutuato dalla definizione di relazione matematica della teoria degli insiemi, come sottoinsieme del prodotto cartesiano fra n insiemi. Nel modello relazionale corrisponde ad una struttura dati tabellare.

Proprietà :

Ogni n-pla è internamente ordinata: l'i-esimo valore proviene dall'i-esimo dominio (struttura posizionale)
Non esiste ordinamento intrinseco fra le n-ple, per la natura insiemistica della relazione.
Non sono ammesse n-ple uguali (ogni elemento di un insieme è unico).

Conseguenze :

Lo scambio fra righe di una tabella non modifica la relazione.
Lo scambio fra colonne di una tabella può portare alla sua inconsistenza con lo schema.

Come ogni campo di un record è associato ad un nome, così si associa ad ogni colonna della relazione un attributo. Ogni attributo ha un suo dominio su cui è definito.
Una *tupla* è un insieme di valori, uno per attributo, ordinati secondo lo schema della relazione e definiti ciascuno su un proprio dominio.
Una relazione è una serie di tuple definite sul dominio della relazione (insieme ordinato dei domini dei singoli attributi).

Vantaggi dell'approccio basato su valori

- Si inseriscono nella base di dati solo valori significativi per l'applicazione (i puntatori sono dati aggiuntivi relativi alla sola implementazione).
- Il trasferimento dei dati da un ambiente ad un altro è più semplice (i puntatori hanno validità solo locale)
- la rappresentazione logica dei dati non fa riferimento a quella fisica e quindi si ottiene l'indipendenza dei dati

Gli altri modelli (gerarchico, reticolare) utilizzano puntatori per le corrispondenze e sono *basati su record e puntatori*, perciò sono meno adatti all'indipendenza dei dati.

2. Descrivere le caratteristiche dei 3 tipi di schema (concettuale, logico e fisico) attraverso i quali può essere descritta una base di dati.

Schemi

In ogni base di dati si possono distinguere:

- lo **schema**, sostanzialmente invariante nel tempo, che ne descrive la struttura (aspetto intensionale)
  - ➢ nell'esempio, le intestazioni delle tabelle
- le **istanze**, cioè i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale)
  - ➢ nell'esempio, il "contenuto" di ciascuna tabella

Lo schema di una base di dati è la parte dichiarativa ed invariante della base di dati e ne definisce la struttura. Nel modello relazionale lo schema di una relazione è paragonabile alla definizione del prototipo di una funzione in C.
Gli schemi possono operare a diversi livelli di astrazione

1. **Schema logico**
   descrive la struttura dell'intera base di dati mediante il modello logico adottato dal DBMS (reticolare, gerarchico, relazionale)
2. **Schema interno o fisico**
   implementa lo schema logico per mezzo di strutture fisiche di memorizzazione (file sequenziali con o senza indici)
3. **Schema esterno o concettuale**
   descrive la struttura di una porzione della base di dati attraverso il modello logico, riflettendo il punto di vista di una classe di utenti. Generalmente è realizzato per mezzo di *viste*, relazioni derivate da quelle che costituiscono lo schema logico.

3. Discutere i principali tipi di vincoli interrelazionali e intrarelazionali

Non tutte le combinazioni possibili di valori dei domini su cui è definita una relazione sono accettabili. Alcuni valori possono essere incompatibili con altri all'interno della stessa relazione alcuni valori possono essere incompatibili con i valori di un'altra relazione. Alcuni attributi possono assumere valori in un certo intervallo e alcuni attributi devono essere diversi in ogni tupla della stessa relazione.
Es.

valori dell'attributo Matricola in una relazione del tipo: Studenti(Matricola, Cognome, Nome, DataNascita)

I vincoli di integrità sono condizioni, espresse come *predicati logici*, che sono inserite nella base di dati per garantirne la consistenza. Ogni istanza della base di dati *deve soddisfare* i vincoli di integrità (il predicato corrispondente al vincolo deve assumere valore vero in ogni istante).
Una istanza che soddisfi tutti i vincoli è detta *corretta* (o *lecita* o *ammissibile*).

I vincoli possono essere principalmente di due tipi:

I. Intrarelazionale, se coinvolge attributi della stessa relazione
   - Vincoli di tupla: possono essere valutati su ciascuna tupla indipendentemente dalle altre;
   - Vincoli di dominio: sono definiti su singoli valori.

II. Interrelazionale, se coinvolge più relazioni

Vincoli Intrarelazionali

I più semplici vincoli intrarelazionali predefiniti sono:

- **not null** indica che il valore nullo non è ammesso su uno specifico attributo. Quindi richiede che sia inserito un valore, salvo che non sia già definito un valore di default. SQL non distingue i diversi tipi di valore nullo. Se è necessario farlo vanno definiti opportuni domini.
- **unique** (*Attributo* {, *Attributo*}) indica che l'insieme di attributi deve essere una superchiave per la tabella.
- **primary key** (*Attributo* {, *Attributo*}) definisce la chiave primaria. Può essere una sola. Tutti gli attributi sono not null.

Vincoli Interrelazionali

Vincolo di integrità referenziale:

In SQL si usa il vincolo di **foreign key** (*chiave esterna*) per creare un legame fra i valori di un attributo della tabella corrente (*interna*) e un attributo di un'altra tabella (*esterna*). Si impone che per ogni riga della tabella interna il valore dell'attributo sia presente nel corrispondente attributo della tabella esterna. L'attributo della tabella esterna deve essere **unique.**

Se l'attributo è unico allora si usa **references.** Altrimenti si usa **foreign key.**

Es.

```
create table Impiegato (
        Matricola     character(6) primary key,   Nome        character(20) not null,
        Cognome       character(20) not null,     Stipendio   numeric(9) default 0,
        Dipart        character(15)
                      references Dipartimento(NomeDip),
              unique (Cognome, Nome),
        foreign key(Nome, Cognome)
                      references Anagrafica(Nome,Cognome))
```

Per i vincoli visti l'inserimento di un valore che li viola viene semplicemente impedito.

In caso di vincoli di integrità referenziale, quando la violazione avviene per un cambiamento apportato alla tabella esterna, si hanno diverse possibili reazioni associabili al comando di aggiornamento che causa l'inconsistenza.

Politiche di reazione:

1° Caso: **Modifica** (comando **update**):

I. **cascade** il nuovo valore dell'attributo della tabella esterna viene riportato su tutte le corrispondenti righe della tabella interna.
II. **set null** all'attributo referente viene assegnato il valore nullo.
III. **set default** all'attributo referente viene assegnato il valore di default.
IV. **no action** la modifica non viene consentita.

2° Caso: **Cancellazione** (comando **delete**):

I. **cascade** tutte le corrispondenti righe della tabella interna vengono cancellate.
II. **set null** all'attributo referente viene assegnato il valore nullo.
III. **set default** all'attributo referente viene assegnato il valore di default.
IV. **no action** la cancellazione non viene consentita.

Per applicare una delle politiche:

on < delete | update >

< cascade | set null | set default | no action > subito dopo la specifica del riferimento.

4. Discutere i concetti di: superchiave, chiave, chiave primaria e chiave esterna. Mostrare con un esempio come tali vincoli possono essere espressi in SQL.

Una *chiave* è un insieme *minimale* di attributi utilizzato per identificare univocamente le tuple di una relazione.

Definizioni:

- Un insieme di attributi K è ***superchiave*** per una relazione *r* se *r* non contiene due tuple t1 e t2 tali che t1[K] = t2[K]
- Un insieme di attributi K è ***chiave*** per *r* se è superchiave minimale, cioè se non esiste un'altra superchiave K' che sia sottoinsieme di K o, comunque, di dimensioni inferiori.
- Una chiave è ***primaria*** se si impone che non contenga valori nulli.

Una chiave è tale se soddisfa la definizione per tutte le possibili tuple appartenenti alla relazione, e non solo per quelle che effettivamente appaiono come istanze della relazione stessa. Quindi la chiave è legata allo schema della relazione e non ai valori effettivamente assunti dalle istanze dello schema.
Ogni relazione, per definizione, possiede una chiave. Infatti, poiché una relazione non ammette due tuple uguali, l'intero insieme X su cui la relazione è definita è sicuramente superchiave per essa.
Di solito la chiave primaria compare sottolineata nello schema di una relazione.

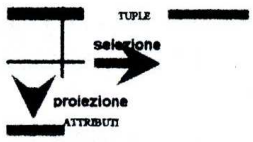
In alcuni casi (corrispondenze fra relazioni) è necessario che i valori degli attributi di una relazione R_1 si trovino anche in attributi corrispondenti di un'altra relazione R_2.
Un *vincolo di integrità referenziale* (o *foreign key*) fra un insieme di attributi X di R_1 e un'altra relazione R_2 è soddisfatto se i valori su X di ciascuna tupla di R_1 compaiono come valori della chiave (primaria) di R_2.

5. Descrivere in breve l'operatore *selezione e proiezione* in algebra relazionale e come vengono espressi in SQL, evidenziando eventuali differenze di comportamento.

Le relazioni sono insiemi e quindi è naturale estendere ad esse le operazioni relative.
Tuttavia le relazioni sono insiemi di tuple omogenee e quindi ha senso definire ed applicare tali operatori solo a tuple definite sugli stessi attributi.
(Es. l'unione fra due relazioni su tuple non omogenee **non è** una relazione).
Le operazioni di selezione e di proiezione si applicano ad una relazione e ne restituiscono una porzione.
Possono essere considerate ortogonali o complementari, in quanto una opera sulle righe e l'altra sulle colonne.

Selezione

La *selezione* produce una nuova relazione definita sugli stessi attributi, contenente solamente le tuple di una relazione che soddisfano una specifica *condizione di selezione*.



Si indica con $\sigma_F(r)$ o $SEL_F(r)$ dove:
F è una condizione da verificare
r è la relazione a cui la selezione è applicata.
Quindi $\sigma_F(r)$ produce una relazione, avente lo stesso schema di r, contenente tutte le tuple per le quali F è vera.
La sintassi della selezione è:

- SEL $_{Condizione}$ (*Operando*)
  - *Condizione*: espressione booleana (come quelle dei vincoli di tupla)
  - il risultato contiene le tuple dell'operando che soddisfano la condizione

F è una *formula proposizionale* su X, cioè una formula ottenuta combinando con i simboli $\wedge$ (*and*) $\vee$ (*or*) $\neg$ (*not*) espressioni del tipo $A \theta B$ o $A \theta c$
dove :

- θ è un operatore di confronto ($\leq, <, =, >, \geq$)
- A e B sono attributi di X su cui i confronto abbia senso
- *c* è una costante tale che il confronto con A sia definito

Per riferirsi ai valori nulli esistono forme apposite di condizioni: IS NULL, IS NOT NULL.
Nel linguaggio SQL l'operatore di selezione viene espresso attraverso la clausola WHERE.
Nel risultato vengono visualizzate solo quelle tuple che appartengono al prodotto cartesiano delle tabelle in 'ListaTabelle' e che soddisfano la condizione della clausola where.
Es.

```
select ListaAttributi
from ListaTabelle
[ where Condizione ]
```

La clausola where ammette come argomento una condizione logica.
Gli operatori ammessi per i predicati semplici (confronto attributo-costante o attributo-espressione) sono: **=, <>, <, >, <=, >=.**
I predicati semplici possono essere modificati tramite gli operatori logici: **and, or, not.**
not ha precedenza su **and** e **or**, ma non è definita la precedenza fra **and** e **or.** Quando si coordinano più predicati con **and** e **or** è bene esplicitare le precedenze con le parentesi.

Proiezione

Dati una relazione $r(X)$ e un sottoinsieme Y di X, la proiezione di r su Y si indica con $\Pi_Y(r)$ o $\mathbf{PROJ}_Y(r)$ ed è l'insieme di tuple su Y ottenute dalle tuple di r considerando solo i valori su Y.
$\Pi_Y(r) = \{ t[Y] \mid t \in r \}$
Una proiezione ha un numero di tuple *minore o uguale* rispetto alla relazione r cui è applicata. Il numero di tuple è uguale se e solo se Y è superchiave per r. La proiezione produce un risultato definito su un insieme limitato di attributi, cui contribuiscono tutte le tuple.
La sintassi della proiezione è:

- PROJ $_{\text{ListaAttributi}}$ (Operando)

Il risultato contiene le tuple che si ottengono restringendo tutte le tuple dell'operando agli attributi nella lista (ed eliminando gli eventuali duplicati).
In SQL l'operatore di proiezione viene espresso attraverso l'operatore SELECT.
Es.

```
select ListaAttributi
       from tabella
```

6. Ridefinire gli operatori booleani AND,NOT,OR per gestire una logica a 3 valori: VERO, FALSO, NULL (Con NULL che rappresenta il concetto di indeterminato).

not	
F	V
U	U
V	F

and	V	U	F
V	V	U	F
U	U	U	F
F	F	F	F

or	V	U	F
V	V	V	V
U	V	U	U
F	V	U	F

7. Introdurre brevemente il concetto di vista.

Una vista può essere considerata una relazione derivata, ossia una relazioni il cui contenuto è funzione del contenuto di altre relazioni (definito per mezzo di interrogazioni).
Esistono due principali tipi di viste:

- **Relazioni Virtuali (Viste)**
  Relazioni definite mediante funzioni o espressioni del linguaggio di interrogazione, non memorizzate ma utilizzabili come se lo fossero. Devono essere ricalcolate tutte le volte.
- **Viste materializzate**
  Relazioni virtuali effettivamente memorizzate nella base di dati.
  Immediatamente disponibili ma critiche per il mantenimento dell'allineamento con le relazioni da cui derivano. Non sono supportate dai DBMS.

Le viste forniscono alcuni importanti vantaggi:

- Permettono di mostrare a un utente le sole componenti della base di dati che interessano.
- Espressioni molto complesse possono essere definite come viste.
- Sicurezza: è possibile definire diritti di accesso anche relativamente ad una vista (e quindi ad una particolare porzione della base di dati).
- In caso di ristrutturazione della base di dati, le "vecchie" relazioni possono essere di nuovo ricavate mediante viste, consentendo l'uso di applicazioni che fanno riferimento al vecchio schema.

Le viste hanno però il grave difetto di permettere pochi aggiornamenti su di esse.
In SQL:
Le viste sono tabelle virtuali ricavate da informazioni contenute in altre tabelle.
Nella definizione possono essere contenute anche altre viste purché non vi siano dipendenze ricorsive o immediate (una vista non può dipendere da se stessa), né transitive.

create view *NomeVista* **[**(*ListaAttributi*)**]** **as** *SelectSQL*
[with [local | cascaded] check option]
La query SQL deve restituire un numero di attributi pari a quelli contenuti nello schema e l'ordine degli attributi nella target list deve rispettare quello dello schema.
Su certe viste è possibile fare modifiche che alterano le tabelle che le compongono. Ci sono problemi se la vista è definita tramite un join. SQL permette la modifica di una vista solo se una sola riga di ciascuna tabella di base corrisponde a una riga della vista. Di solito si richiede che sia definita su una sola tabella e/o che contenga almeno una chiave primaria.
check option richiede che si facciano modifiche solo sulle righe della vista e che le righe continuino ad appartenere alla vista dopo le modifiche.
Se una vista è definita in termini di altre viste l'opzione **local** o **cascaded** (default) specifica se il controllo debba coinvolgere solo la vista più esterna o se deve essere propagato a tutti i livelli.
Quindi se è specificata la **check option** ogni comando di aggiornamento per essere propagato non deve eliminare righe dalla vista. Le viste consentono anche di sostituire interrogazioni nidificate o di creare interrogazioni altrimenti impossibili da definire.

8. Descrivere in quali modi può essere realizzato un inner join in algebra relazionale e in SQL senza usare esplicitamente l'operatore join.

Se non si desidera utilizzare l'operatore join in SQL basta porre la condizione del join nella clausola WHERE. Es.

```
select nome,cognome,corso
from studenti,esami
where studenti.matricola = esami.matstud
```

```
select nome,cognome,corso
from studenti join esami on studenti.matricola = esami.matstud
```

Le due scritture sono identiche. Basta effettuare una proiezione e una selezione.
Proiezione: select ----- Selezione: where → JOIN = $\text{PROJ}_{\text{nome, cognome, corso,}}$ ($\text{SEL}_{\text{matricola = matstud}}$)

9. Descrivere i diversi tipi di join in algebra relazionale illustrando con un esempio come possono essere tradotti in SQL.

Combinando selezione e proiezione, si possono estrarre informazioni da **una sola** relazione
Non si possono però correlare informazioni presenti in relazioni diverse.
Il **join** è l'operatore più interessante (potente) dell'algebra relazionale in quanto permette di correlare dati presenti in relazioni diverse. E' l'operatore più caratteristico dell'algebra relazionale, che evidenzia la proprietà del modello relazionale di essere basato su valori. Non ha un corrispettivo nella teoria degli insiemi. L'operatore di join (naturale) correla dati contenuti in relazioni diverse. Il suo risultato è una relazione definita sull'unione degli insiemi di attributi degli operandi, le cui tuple sono ottenute combinando le tuple degli operandi **che hanno valori uguali** su attributi comuni (con lo stesso nome).
Il **join naturale** $r_1 \lozenge r_2$ di $r_1(X_1)$ e $r_2(X_2)$ è una relazione definita su $X_1 \cup X_2$ (X_1X_2):
$r_1 \lozenge r_2 = \{ t \text{ su } X_1X_2 \mid t[X_1] \in r_1 \text{ e } t[X_2] \in r_2\}$
Il grado della relazione ottenuta è minore o uguale al grado della somma dei gradi delle due relazioni in quanto gli attributi omonimi compaiono una sola volta.

NOTA BENE:
Se $X_1 \cap X_2$ è vuoto il join naturale equivale al *prodotto cartesiano* fra le relazioni.
Se $X_1 = X_2$ il join naturale equivale all'intersezione fra le relazioni.
Se ciascuna tupla di ciascuno degli operandi contribuisce ad almeno una tupla del risultato il join si dice *completo.* Se per alcune tuple non è verificata la corrispondenza e non contribuiscono al risultato, le tuple si dicono ***dangling.***

CASI LIMITE:
1. join vuoto, in cui nessuna tupla degli operandi è combinabile;
2. join in cui ciascuna delle tuple di un operando è combinabile con tutte le tuple dell'altro. In questo caso la cardinalità della relazione risultante è pari al prodotto della cardinalità degli operandi. E' ovviamente un caso limite, che si verifica solo se esistono attributi comuni che assumono lo stesso valore per ogni tupla di entrambi gli operandi (quindi se l'attributo, di fatto, *non contiene informazione significativa*).

Infine, bisogna ricordare che l'operatore join gode della proprietà commutativa e associativa.
Se si devono correlare attributi con nome diverso (cioè $X_1 \cap X_2$ è vuoto) è possibile fare il ***theta-join,*** definito come un prodotto cartesiano seguito da una selezione:
$r_1 \lozenge_F r_2 = \sigma_F (r_1 \lozenge r_2)$, dove F è la condizione di selezione.
Se F è una condizione di uguaglianza fra un attributo della prima relazione e uno della seconda, allora siamo in presenza di un ***equi-join.***
Sono importanti formalmente:
il join naturale è basato sui *nomi* degli attributi
equi-join e theta-join sono basati sui *valori*

Join Esterni

Il join naturale tralascia le tuple in cui non vi è corrispondenza fra gli attributi legati dal join. Si definiscono allora altri tipi di join, che fanno sì che anche quelle tuple vengano considerate, inserendo valori nulli dove non vi sia corrispondenza.

1. Join sinistro
Contribuiscono tutte le tuple del primo operando eventualmente estese con valori nulli.

2. Join destro
Contribuiscono tutte le tuple del secondo operando eventualmente estese con valori nulli.

3. Join completo
Contribuiscono tutte le tuple del primo e del secondo operando eventualmente estese con valori nulli.

10. Descrivere i diversi vincoli di tupla.

Un vincolo di tupla è un vincolo che può essere valutato su ciascuna tupla indipendentemente dalle altre. Un vincolo definito con riferimento a singoli valori viene detto vincolo su valori o vincolo di dominio in quanto impone una restrizione sul dominio dell'attributo. Possono essere definiti attraverso operatori booleani.

Per esempio:

- Vincolo di dominio: (Voto ≥ 18) AND (Voto ≤ 30)
- Vincolo su più attributi (NOT(Lode = "vero")) OR (Voto = 30)
- Pagamenti(Data, Importo, Ritenute, Netto) Netto = Importo - Ritenute

11. Descrivere a cosa serve il comando SQL CREATE DOMAIN e portare un esempio di uso del comando stesso.

Come nei linguaggi ad alto livello (es. C) è possibile definire nuovi domini (tipi di dati) a partire da quelli predefiniti, anche se il costruttore è più limitato.

create domain *NomeDominio* **as** *TipodiDato*
[*ValDefault*]
[*Vincolo*]

Non si possono creare array o strutture poiché il modello relazionale richiede che gli attributi siano definiti su un dominio elementare. E' però possibile associare dei vincoli ad un dominio definito dall'utente. Se si modifica la definizione di un dominio, la modifica si propaga a tutte le tabelle.

Es.

```
CREATE DOMAIN Voto
        AS SMALLINT DEFAULT NULL
        CHECK ( value >=18 AND value <= 30 )
```

12. Descrivere brevemente il concetto di valore nullo e gli effetti che la sua presenza possono avere su una base di dati.

Le tuple che compongono la base di dati devono essere omogenee. Quindi ad ogni attributo deve essere associato un valore in ogni tupla. Non sempre questo è possibile.

Non conviene (anche se è un espediente di uso comune) usare valori del dominio normalmente non utilizzati (0, stringa nulla, "99", ...), come spesso accade nella programmazione.

Nel modello relazionale è definito un valore convenzionale, detto ***valore nullo***, che indica la non disponibilità dell'informazione. E' comune a tutti i domini e garantisce l'integrità di ciascuna tupla.

Il valore nullo può rappresentare 3 tipi di informazione:

- sconosciuta
- inesistente
- indeterminata (nei DBMS disponibili sul mercato si considera in genere questo caso) .

In SQL, la gestione del valore nullo avviene attraverso una logica a due o tre valori. Le condizioni sui valori nulli possono essere definite attraverso i predicati **is null** e **is not null.**

Es.

```
update Insegnamento
set corso = "Analisi"
where corso IS NULL
```

All'interno di una query, quindi, il valore nullo viene considerato come un valore particolare, comune a tutti i domini, identificabile appunto attraverso i predicati **is null** e **is not null.**

13. Discutere il concetto di efficienza di uno schema di base di dati ed i relativi criteri di ottimizzazione.

Un Database Management System (**DBMS**) è un sistema software che si interpone fra le applicazioni e la memoria di massa in cui si trovano collezioni di dati, per consentire la gestione in modo indipendente dalle applicazioni stesse.
Normalmente le applicazioni accedono a dati locali gestendoli attraverso file, che sono proprietà delle applicazioni stesse.
Le finalità dei DBMS è di estendere le funzionalità del file system, offrendo:

- Nuove modalità di accesso ai dati
- Condivisione dei dati
- Gestione più sofisticata dei file

Le basi di dati gestite dai DBMS sono collezioni dati:

- **Grandi**: possono avere notevoli dimensioni (fino a migliaia di Gbyte) e devono quindi necessariamente risiedere in memoria secondaria;
- **Condivise**: applicazioni e utenti diversi devono poter accedere ai dati
- **Persistenti**: il tempo di vita dei dati va oltre la durata dell'esecuzione delle singole applicazioni

Caratteristiche di un DBMS:

1. ***AFFIDABILITA'***:

E' necessario che un DBMS garantisca di poter mantenere intatto il suo contenuto, anche in caso di malfunzionamento. L'integrità dei dati è affidata a procedure di backup (salvataggio) e recovery (recupero) dei dati, e alla loro duplicazione nei casi più critici.

2. ***PRIVATEZZA DEI DATI:***

Ogni utente, abilitato a utilizzare la base di dati attraverso una procedura di riconoscimento, può accedere ad insiemi limitati di dati e compiere solo certe operazioni su di essi.

3. ***EFFICIENZA***:

Un DBMS deve operare e fornire risposte agli utenti in tempi accettabili, utilizzando una quantità il più possibile limitata di risorse. L'efficienza di un DBMS dipende essenzialmente dalle tecniche utilizzate per la sua implementazione e dalla buona progettazione della base di dati. Si misura in termini di *tempo di esecuzione* (tempo di risposta) e *spazio di memoria* (principale e secondaria) occupato.

4. ***EFFICACIA***:

Capacità di un DBMS di rendere produttive le attività degli utenti cioè di consentire la realizzazione di basi di dati che risolvano in modo efficace i problemi degli utenti. Non esistono criteri oggettivi per valutarla.
I DBMS possono essere classificati in base al modello dei dati che utilizzano. Un modello dei dati è costituito dai concetti sulla base dei quali i dati sono strutturati e codificati.

Modelli dei dati

- Relazionale: il più diffuso, basato su un modello tabellare dei dati
- Gerarchico: usato nei primi DBMS e tuttora usato, basato su strutture ad albero
- Reticolare: estensione del modello gerarchico, basato su grafi
- A oggetti: estensione del modello relazionale basato sui paradigmi di programmazione ad oggetti.

Questi modelli sono detti modelli logici, in quanto, seppure astratti, riflettono la struttura con cui i dati sono organizzati. I modelli concettuali si collocano ad un livello di astrazione superiore, si svincolano dalla presentazione dei dati e rappresentano solo concetti del mondo reale.