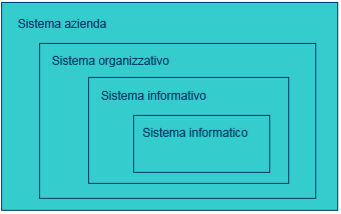
**Basi di dati**

**Organizzazione delle informazioni all’interno di un sistema aziendale:** L’organizzazione delle informazioni all’interno di un qualsiasi sistema aziendale è organizzata con uno schema a cipolla cosi rappresentato:

**All’interno dello schema vediamo:**

* **Sistema aziendale:** che rappresenta il nucleo dell’azienda o dell’organizzazione e svolge tutte le operazioni e i compiti assegnati a quest’ultima.
* **Sistema organizzativo:** è un insieme di risorse (persone, denaro, materiali, informazioni) e regole per lo svolgimento delle attività (processi) che permettono il raggiungimento dello scopo.
* **Sistema informativo:** fa parte del sistema organizzativo ed è atto a gestire e ad eseguire tutti quei processi che coinvolgono informazioni, esso è di solito adibito al supporto di altri sottosistemi e va quindi studiato nel contesto in cui si trova.
* **Sistema informatico:** è una parte del sistema informativo che si occupa della gestione delle informazioni tramite tecnologie informatiche.

**Database:** un database è una collezione di dati logicamente correlati organizzati per lo svolgimento di una specifica attività. Un database può avere un qualsiasi dimensione e complessità.

**Differenza tra:**

* **Dato:** Ciò che è immediatamente presente alla conoscenza, prima di ogni tipo di elaborazione. In informatica il dato è un elemento formato da simboli che ancora devono essere elaborati.
* **Informazione:** è un qualcosa che si ottiene dall’elaborazione di uno o più dati e che consente di ottenere conoscenza.

**Il DBMS:** il DBMS (Data Base Management System) è un sistema software in grado di gestire collezioni di dati, esso permette di:

* **Definire:** tipi di dati, strutture e vincoli.
* **Costruire:** memorizzare fisicamente i dati all’interno della memoria di massa.
* **Manipolare:** interrogare il database, aggiornarlo e generare report sui dati.

Ed è caratterizzato dalle seguenti proprietà:

* **Condivisibili:** nel senso che applicazioni e utenti diversi devono poter accedere, secondo opportune modalità, ai dati contenuti all’interno della base di dati. Ciò permette di ridurre la ridondanza dei dati e di conseguenza l’inconsistenza di quest’ultimi.
* **Persistenti:** in quanto i database non hanno un ciclo di vita limitato a quello dei singoli programmi che li utilizzano, ma devono perdurare anche una volta che il processo che necessitava di tali dati ha terminato l’esecuzione.

Garantendo al tempo stesso:

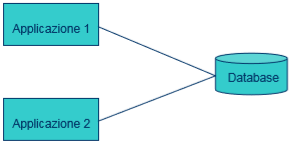
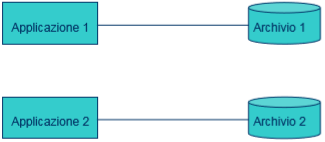
* **Affidabilità:** ovvero la capacità del sistema di conservare intatto il contenuto della base di dati, anche in caso di malfunzionamenti hardware e software. A questo scopo il DBMS fornisce funzionalità di backup e recovery, oltre a gestire in maniera controllata copie dei dati.
* **Privatezza:** ovvero la capacità del sistema di permettere solo a determinati utenti identificati tramite delle credenziali (nome utente, password) di svolgere determinate azioni all’interno della base di dati.
* **Efficienza:** ovvero la capacità del sistema di svolgere determinate operazioni con tempi e numero di risorse impiegati accettabili dall’utente finale.

**Transazioni:**

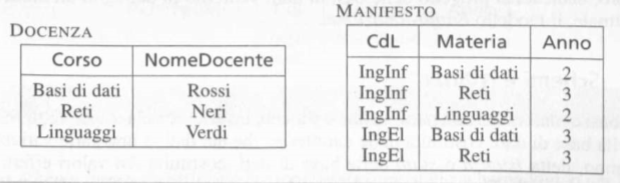
* **Per l’utente:** Programmi che si occupano di realizzare attività frequenti e predefinite, per esempio un versamento presso un ufficio bancario
* **Per il sistema:** Sequenza indivisibile di operazioni.

**DBMS vs File system:** la gestione di dati grandi e persistenti è possibile anche attraverso sistemi più semplici, gli ordinari file system che però non permettono una condivisione dei dati evoluta come i DBMS (o tutto o niente). Inoltre, nei programmi tradizionali ogni programma contiene al suo interno una descrizione del file stesso, mettendo così a rischio la consistenza dei dati, mentre il DBMS mette a disposizione una descrizione centralizzata dei dati che può essere utilizzata dai vari programmi.

**File system Database**



**Modelli di dati:** un modello di dati è un insieme di concetti utilizzati per organizzare in dati e descriverne la struttura in modo che essa risulti comprensibile ad ‘un elaboratore. Il modello oggi più utilizzato è il **modello relazionale** questo modello permette di descrivere i dati attraverso il costrutto della relazione, che consente di organizzare i dati attraverso una struttura fissa. Una relazioni in questo modello viene rappresentato tramite una tabella dove le colonne rappresentano gli attributi dei vari record, attributi che vengono utilizzati per unire tramite una relazione le varie tabelle.



Per esempio, l’attributo **materia -> corso** può essere usato per legare la tabella manifesto a quella docenza.

Esistono inoltre alti tipi di modelli:

* **Modello gerarchico:** basato su una struttura d’Albero
* **Modello reticolare:** basato sull’uso di grafi
* **Modello ad oggetti:** che estende alle basi di dati il paradigma della programmazione ad oggetti
* **Modello XML:** all’interno del quale i dati vengono presentati con una loro descrizione che non gli impone di sottostare ad ‘un determinato schema logico

**Concetto di modello logico e modello concettuale:**

* **Modello logico:** indica come i dati sono organizzati logicamente all’interno della base di dati (relazionale, logico, gerarchico…).
* **Modello concettuale:** è un modello usato per descrivere in dati in modo del tutto indipendente dal modello logico utilizzato, viene utilizzato nella fase di progettazione della base di dati per analizzare concettualmente la realtà studiata.

**Schemi e istanze:**

* **Schema:** Descrive la struttura di una relazione, per esempio la struttura della tabella **Docenza** nella figura precedente può essere cosi rappresentata:
  + **Docenza (corso, NomeDocente)**

Lo schema di una base dati non varia nel tempo. Inoltre, lo schema viene definito anche come la componente intensionale della base di dati (**concetto simile alle classi di java**).

* **Istanza:** sono i valori attuali presenti in una relazione, che possono variare nel tempo, riprendendo sempre l’esempio della tabella docenze, le istanze sono:
  + **Basi di dati Rossi**
  + **Reti Neri**
  + **Linguaggi verdi**

Inoltre, l’istanza vene anche definita come la parte estensionale della base di dati.

**Livelli di astrazione del DBMS:** Lo schema di una base di dati può essere ulteriormente divisa in tre sottoschemi:

* **Schema logico:** Che descrive l’interno della base di dati attraverso il modello logico adottato dal DBMS.
* **Schema interno o fisico:** Che descrive in maniera fisica la rappresentazione dello schema logico (una tabella diventa un file).
* **Schema esterno o interfaccia utente:** Che permette di descrivere una porzione della base di dati di interesse è inoltre la parte del database che si interfaccia con l’utente.









**Indipendenza dei dati:** l’architettura dei dati codi definita sopra garantisce l’indipendenza dei dati che è la principale proprietà del DBMS, essa può essere divisa in:

* **indipendenza fisica:** che consente di interagire con il DBMS in modo indipendente dalla struttura fisica dei dati, questo permette di modificare la struttura fisica del database senza influire sulla descrizione dei dati ad alto livello.
* **indipendenza logica:** che consente di apportare modifiche al livello esterno in modo indipendente dallo schema logico e dualmente consente di modificare lo schema logico in modo indipendente dallo schema esterno.

**Linguaggi per basi di dati:** All’interno del DBMS è possibile specificare operazioni di vario tipo, in particolare per quanto riguarda schemi e istanze esistono due tipi di linguaggi:

* **DDL (Data Definition Language):** utilizzato per la definizione degli schemi.
* **DML (Data Manipulation Language):** utilizzato per l’interrogazione e la manipolazione delle istanze all’interno della base di dati.

**Categorie di utenti:** Tra le tipologie di utenti che possono interagire con una base di dati troviamo:

* **L’amministratore della base di dati:** è la persona o il gruppo di persone che si occupa della progettazione, controllo e amministrazione della base di dati, inoltre l’amministratore è anche responsabile di controllare l’efficienza e l’affidabilità della base di dati.
* **Progettisti e programmatori:** è una persona o un gruppo di persone che definiscono e realizzano programmi che accedono alla base di dati tramite il linguaggio DML.
* **Utenti:** che utilizzano la base di dati per le loro attività vengono divisi in due macro-gruppi:
  + **Utenti casuali:** che effettuano richieste non standardizzate per tanto non ottimizzate
  + **Utenti finali o terminalisti:** che utilizzano programmi che realizzano attività predefinite e di frequenza elevata con poche eccezioni previste a priori.

**Vantaggi e svantaggi del DBMS:**

Vantaggi:

* I dati vengono considerati come una risorsa comune dell’organizzazione, con opportune forme di controllo.
* La base di dati fornisce un modello unificato e preciso della parte di mondo reale di interesse dell’organizzazione con la possibilità di essere standardizzato e arricchito nel tempo.
* Controllo centralizzato della base di dati con conseguente diminuzione degli errori.
* La condivisione permette di ridurre ridondanze e inconsistenze.
* L’indipendenza dei dati permette lo sviluppo di applicazioni flessibili e facilmente modificabili.

Svantaggi:

* Costi alti e alta complessità di realizzazione.
* Non scorporabilità delle funzionalità (potrebbe portare ad una perdita di efficienza) praticamente non si possono eliminare servizi.

**Modello relazionale:** Il modello relazionale si basa principalmente su due concetti:

* **Relazione:** Definizione che proviene dalla teoria degli insiemi.
* **Tabella:** Semplice concetto intuitivo.

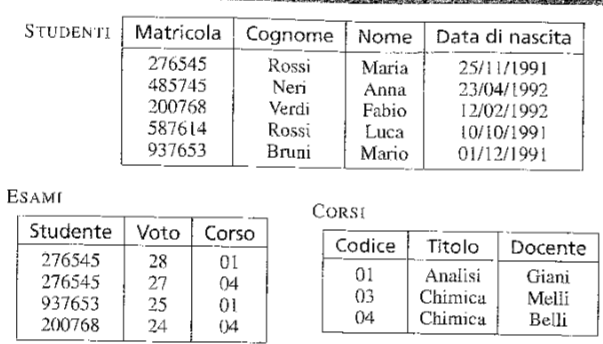
Ogni entità è caratterizzata da delle caratteristiche che la rendono unica, queste caratteristiche prendono il nome di **attributi** che hanno le seguenti proprietà:

* **Formato o dominio:** che indica il tipo di dato rappresentato (stringa, intero, data/ora).
* **Dimensione:** che indica la quantità massima di dati di un determinato tipo che possono essere contenuti all’interno dell’attributo
* **Opzionalità:** valore binario che indica se l’attributo deve essere in stanziato o meno al momento della creazione dell’ennupla (codice fiscale serve, soprannome non serve per forza).

L’insieme di queste tre proprietà danno vita ai **metadati** che sono di dati che si occupano di descrivere di dati. Gli attributi all’interno del modello permettono di realizzare **relazioni non posizionali**, esempio:

Nella prima tabella nel caso in cui le prime due colonne vengano scambiate il significato della tabella cambia. Nella seconda invece con l’inserimento degli attributi anche nel caso in cui le colonne vengano spostate il significato della tabella e quindi della relazione tra la Squadra di casa e la Squadra ospitata non cambia.

**Definizione di ennupla o tupla:** Dal punto di vista formale si può definire una tupla come una funziona che associa a ciascun attributo di un dato elemento Per un valore appartenente dal dominio dell’attributo. Da un punto di vista grafico una tupla non è altro che una riga di una tabella o relazione.

**Modello relazionare VS modello a puntatori.:**

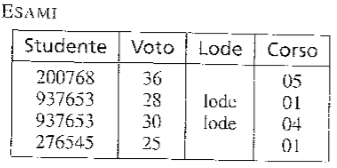
All’interno del modello relazionare le relazioni vengono create unendo logicamente determinati valori all’interzo delle relazioni, nell’esempio sopra la tabella esami collega, tramite i valori degli attributi studente e corso, le tre tabelle.

Questo tipo di collegamento porta due principali vantaggi rispetto al modello basato su puntatori e record:

* La rappresentazione logica dei dati non fa alcuno riferimento a quella fisica (in quanto non ci sono puntatori in gioco) possiamo cosi ottenere l’indipendenza fisica dei dati.
* Essendo tutta l’informazione contenuta nei contatori è più semplice trasferire i dati da una macchina all’altra e manipolarli.

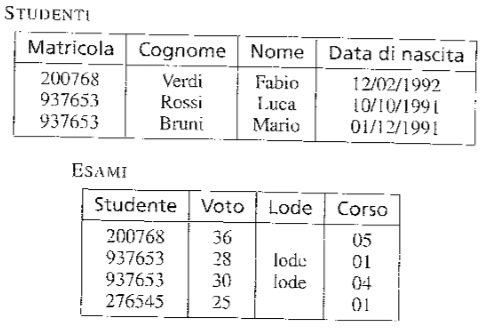
**Concetto di valore nullo:** All’interno del modello relazionare le tuple devono essere rappresentate tramite dati omogenei (ovvero tramite valore appartenenti al dominio dell’attributo in questione). Per mantenere tale proprietà all’interno delle relazioni è stato introdotto il concetto di **valore nullo** (**NULL**) un carattere speciale, completamente estraneo al dominio dell’attributo, utilizzato per rappresentare in modo semplice e chiaro un valore non ancora inizializzato o non disponibile al momento.

**Concetto di vincoli di integrità:** In alcuni cassi all’interno dei una base di dati alcune informazioni possono rilevarsi incorrette o imprecise, anche se dal punto di vista sintattico sono corrette, esempio:

Per evitare tali errori all’interno del modello relazionare è stato inserito il concetto di **vincolo di integrità** come proprietà che deve essere verificata dalle tuple per rappresentare informazioni corrette all’interno della base di dati.

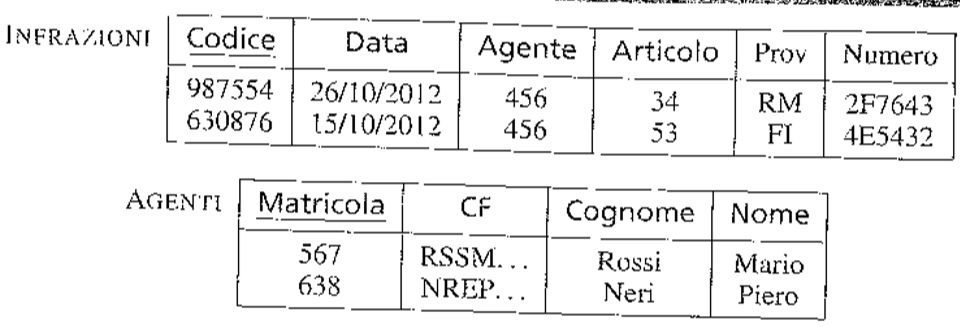
All’interno del modello relazionare esistono principalmente due tipi principali di vincoli di integrità:

* **Vincoli intrarelazionali:** se il loro soddisfacimento è definito rispetto a singole relazioni della base di dati (Lo dice il nome stesso, comunque è l’esempio sopra). In base al coinvolgimento delle tuple o dei valori degli attributi esistono due sottocategorie:
  + **Vincoli di dupla:** vincoli che vengono valutati sulle singole tuple (caso sopra)
  + **Vincoli sui valori o di dominio:** caso in cui i vincoli sono definiti sui singoli valori di un attributo (voto esame va da 18 a 30).
* **Vincoli interrelazionali:** quando il vincolo coinvolge più relazioni, esempio:



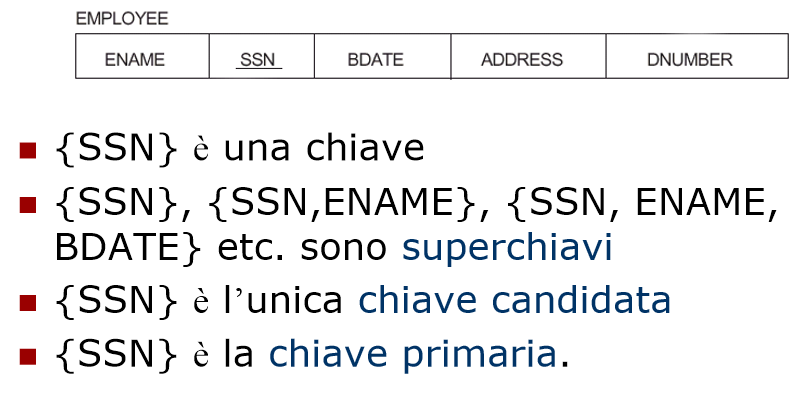
Un numero di matricola può comparire nella relazione esami sono se compare anche nella relazione studenti.

* **Vincoli referenziali:** Un vincolo referenziale si ha se, presi degli attributi X appartenenti a due relazioni R1 e R2, tali valori compaiono come chiave primaria nella relazione R2 senza comparire nella relazione R1, Esempio:

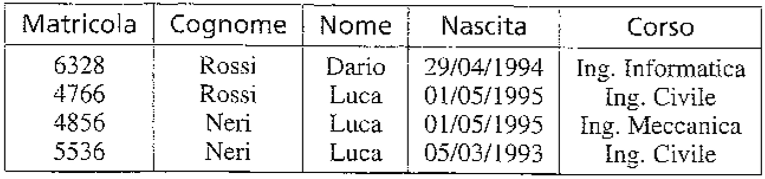


L’agente 456 non esiste nella tabella agenti.

**Chiavi:** dal punto di vista logico una chiave è un insieme di attributi che identifica in modo univoco una tupla, tuttavia e possibile fare un ulteriore è più approfondita riflessione:

* **Superchiave:** non è altro che un insieme di attributi che identificano in modo univoco una tupla
* **Chiave mimale:** non è altro che uno o più attributi che da soli bastano a indentificare una tupla (non c’è una Superchiave dentro questi attributi).

**Esempio:**

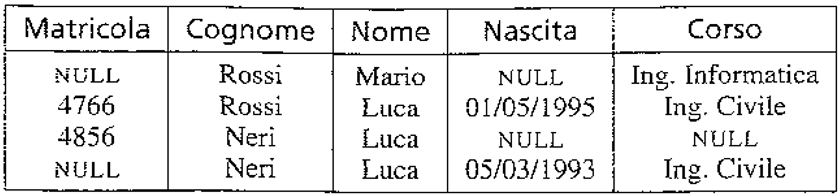
****

**Matricola, Cognome, Nome, Nascita, Corso:** è una superchiave in quanto permette di identificare una tupla in modo univoco, non è però una chiave minimale in quando l’insieme **Cognome, Nome, Nascita** è a sua volta una superchiave

**Cognome, nome, nascita:** Insieme formano una Superchiave in quanto nessuno di loro da solo è in grado di identificare in modo univoco una tupla, inoltre sono anche una chiave minale in quando nessuno dei tre attributi è in grado di identificare una tupla in modo univoco.

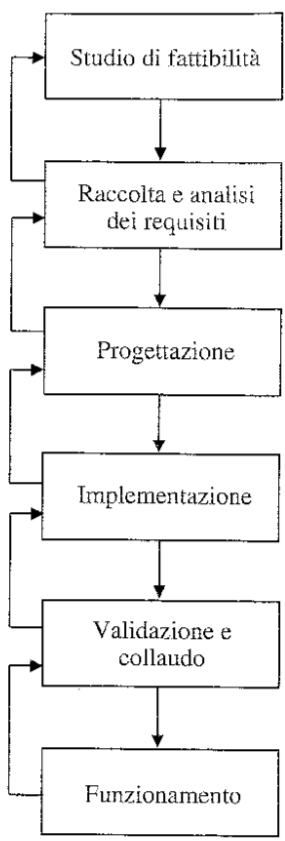
**Matricola:** è una chiave minimale in quanto da sola è in grado di identificare una tupla.

Praticamente una volta indentificate le **chiavi minimali** quella formata dal minor numero di attributi diventa la **Primary Key.**

**Valori nulli e chiavi:** Al fine di rendere ogni tupla univocamente identificabile all’interno della base di dati è fondamentale non esagerare con i valori nulli. 

In questo esempio vediamo come i valori nulli presenti sia nella **PK** che nell’altra chiame minima (Cognome, Nome, Nascita)impediscono di identificare in modo univoco i record.

**Progettazione concettuale:**

**Ciclo di vita dei sistemi informativi:** Il ciclo di vita di un sistema informativo, ovvero un sistema di gestire informazioni, si divide nelle seguenti parti: 

* **Studio di fattibilità:** Serve a stabilire il costo delle varie soluzioni del problema e a stabilire la priorità di realizzazione delle varie componenti.
* **Raccolta e analisi dei requisiti:** Consiste nell’individuazione e nello studio delle proprietà che il sistema dovrà avere. In questa fase è previsto un colloquio con gli utilizzatori finali, in modo da chiarire cosa il sistema dovrà fare e con quale frequenza.
* **Progettazione:** Questa fase consiste nella progettazione:
  + **Dei dati:** ovvero stabilire come i dati saranno organizzati all’interno del sistema.
  + **Delle applicazioni:** ovvero la progettazione dei programmi che andranno a interagire con i dati.
* **Implementazione:** Consiste nell’implementazione e nel popolamento della base di dati secondo le caratteristiche stabilite nella fase di progettazione, oltre che all’implementazione delle applicazioni precedentemente progettate.
* **Validità e collaudo:** Consiste nel verificare che il corretto funzionamento e la qualità del sistema in ogni situazione possibile.
* **Funzionamento:** In questa fase il sistema informativo diventa operativo. Se non ci sono problemi o guasti le uniche attività da eseguire sul sistema sono quelle di manutenzione e aggiornamento.

È importante notare che in una qualsiasi di queste cause se si verificano problemi imprevisti, si recede nel ciclo di vita (per esempio se non riusciamo a scrivere un’applicazione perché concettualmente errata si torna alla fase di progettazione, ovviamente più avanti si va più gli errori sono disastrosi).

**Metodologie di progettazione e basi di dati:** Prima di parlare delle metodologie utilizzate per la progettazione di basi di dati, è importante precisare quali siano le proprietà di una buona metodologia:

* **Decomposizione:** Deve dividere il processo di progettazione in una serie di passi finiti e indipendenti.
* **Strategie di scelta:** Che ci permettono di scegliere un tipo di soluzione da un’altra, all’interno dei vari passi.
* **Modelli di riferimento:** Che ci permettono di distinguere i dati in uscita e in ingresso dalle varie fasi.

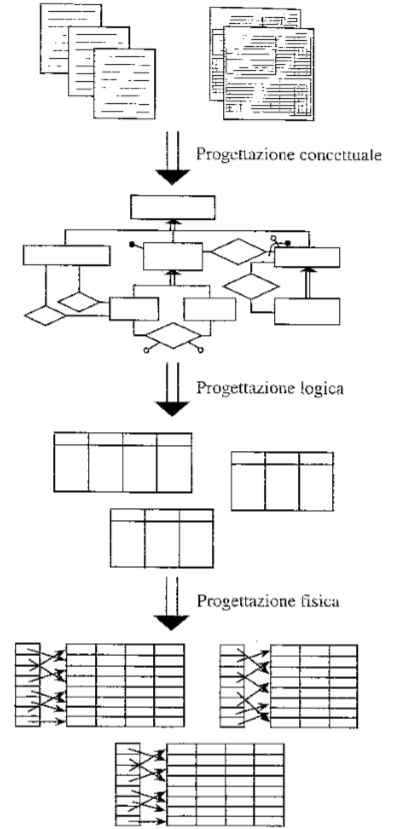
In modo da garantire le seguenti proprietà della soluzione:

* **Generalità:** Rispetto alle applicazioni e ai sistemi utilizzati per la progettazione.
* **Qualità del prodotto:** in termini di correttezza, completezza e efficienza rispetto alle risorse impiegate.
* **Facilità d’uso:** delle strategie e dei modelli di riferimento.

Per quanto riguarda la progettazione delle basi di dati negli anni si è perfezionata la seguente metodologia di progettazione:

* **Progettazione concettuale:** Il suo scopo è quello di rappresentare la realtà che si vuole analizzare sotto forma di specifiche formali indipendenti dal modello logico che si vuole utilizzare. L’output di questa fase è lo schema concettuale.
* **Progettazione logica:** Consiste nella traduzione dello schema concettuale in termini di modello logico adottato (modello relazione, ad albero ecc.…), in questa fase si fa anche uso di tecniche formali per verificare la qualità dello schema logico (come le regole di normalizzazione).
* **Progettazione fisica:** Si inseriscono all’interno dello schema logico i parametri che permettono la memorizzazione fisica dei dati (modello fisico dei dati).

Lo schema sotto mostra gli output delle varie fasi, in un modello entità-relazioni.

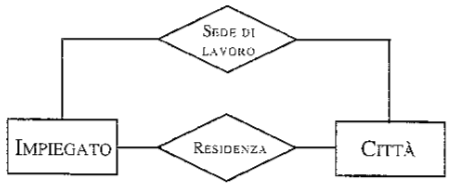
**Modello entità-relazione:** è un modello concettuale dei dati, che fornisce una serie di **costrutti**. Costrutti che vengono utilizzati per definire **schemi** utili a definire la struttura dell’**occorrenza** (ennupla)**.** Questo modello è più ad alto livello rispetto a quello relazionare.

**Costrutti del modello ER:**

**Entità:** Rappresenta classi di oggetti che godono delle stesse proprietà. Un’occorrenza di un’entità è l’oggetto che appartiene a quella classe di entità.

Esempio: 

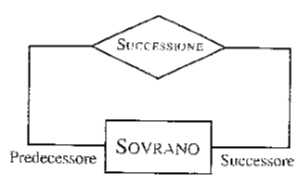
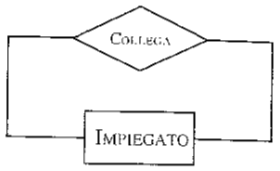
**Roma** è un’occorrenza dell’entità Città, mentre **Rossi** dell’entità impiegato.

**Relazione o associazione:** rappresentazione di legami logici tra due o più entità. Un esempio di relazione tra due entità è il seguente:

Che collega l’oggetto **impiegato** all’oggetto **città** tramite le relazioni: **sede di lavoro** e **residenza**. Dal punto di vista tabellare Sede di lavoro e Residenza sono tabelle dove ogni ennupla crea una relazione tra un’occorrenza di impiegato e una di città. È convenzione usare dei **sostantivi** per identificare le relazioni in modo da non creare dei versi in queste ultime (Sede di lavoro va bene, lavora in non va bene). Inoltre, è molto importante notare che le relazioni cosi definite non ammettono che al loro interno (nella tabella) ci siano occorrenze ripetute, aspetto che abbia importanti conseguenze.

Come nel caso in cui uno studente abbia ripetuto più volte lo stesso esame (non ci possono essere tot ennuple per ogni tot tentativo, per questo motivo serve un altro oggetto che chiameremmo esame).

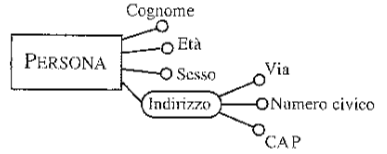
È anche possibile avere relazioni **ricorsive,** che coinvolgono un’entità e sé stessa, come in questi due esempi:



Nel caso **impiegato, collega** la relazione collega, associa due impiegati al medesimo compito. Mentre nel caso **successione, sovrano** la relazione successione indica quale sovrano succederà quello in carica. Infine, è importante notare che non esiste limite al numero di entità che può prendere parte ad ‘una relazione.

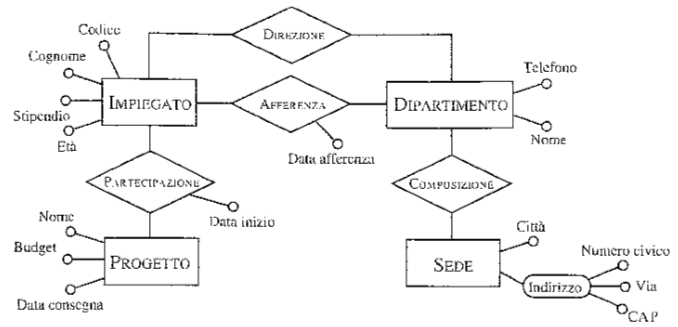
**Attributi:** Descrivono le proprietà delle entità che sono di interesse per la realtà che si sta osservando, ogni attributo altre al suo valore ha anche le seguenti proprietà:

* **Formato o dominio:** che indica il tipo di dato rappresentato (stringa, intero, data/ora).
* **Dimensione:** che indica la quantità massima di dati di un determinato tipo che possono essere contenuti all’interno dell’attributo
* **Opzionalità:** valore binario che indica se l’attributo deve essere in stanziato o meno al momento della creazione dell’ennupla (codice fiscale serve, soprannome non serve per forza).

Può essere utile alcune volte raggruppare diversi attributi sotto un singolo nome, per esempio gli attributi: **via, numero civico** e **CAP** possono essere raggruppati **nell’attributo** **composto** **indirizzo**

In questo esempio vengono mostrati con i pallini collegati direttamente a persona, gli **attributi**, mentre con i pallini collegati ad indirizzo e poi a persona un **attributo composto**.

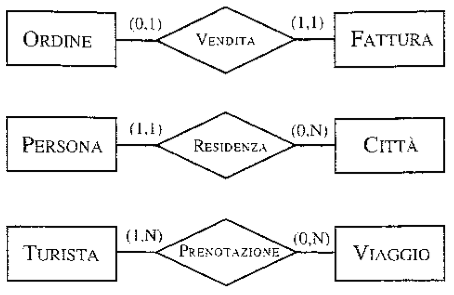
**Esempio di schema con costrutti base del modello ER:**

****

**Concetto di cardinalità delle relazioni:** Vengono specificate per ogni relazione che si va a creare e descrivono il numero massimo e minimo di occorrenze (tuple) che possono partecipare alla relazione.

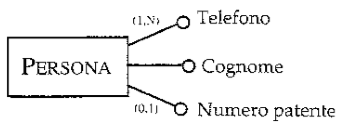
**Esempio:** Ad un impiegato possono essere assegnati minimo 1 incarico e massimo 5, mentre per ogni incarico ci possono essere 0 o al massimo 50 impiegati che ci lavorano sopra.

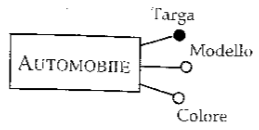
In linea di principio è sufficiente utilizzare solo tre valori: **zero, uno** e il simbolo **N.** In particolare:

Nel caso **0.1** diremmo che la partecipazione è **opzionale.**

Nel caso 1.1 diremmo che la partecipazione è **obbligatoria.**

**Cardinalità degli attributi:** Può essere specificata per gli attributi di un’entità o relazione e descrive il numero minimo e massimo di valori dell’attributo associato ad un’occorrenza (tupla) o per l’appunto ad una relazione.

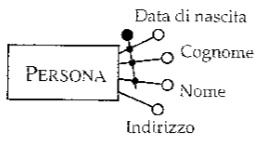
**Esempio:** Una persona può avere più numeri di telefono e al minimo ne ha uno. Una persona può avere una sola patente e non è detto che la possegga.

**Identificatori dell’entità:** Conosciuti anche come **PK** (Primary Key) e **SK** (Secondary Key) (solo quando si parla di attributi però attenzione). Sono dei **concetti** (Attributi o Entità) che permettono di indentificare in modo univoco un’occorrenza (tupla)

**Esempio:**

All’interno dell’oggetto automobile ogni occorrenza può essere identificata in modo univoco dal numero di targa.

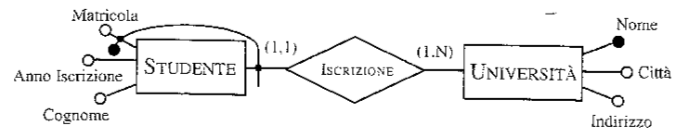
Alcune volte può capitare che un singolo attributo non sia in grado di indentificare in modo univoco un’occorrenza, in questo caso si utilizzano per tale scopo più attributi.

**Esempio:** 

All’interno dell’oggetto persona possono esiste individui con lo stesso nome, quindi per identificarli si utilizza la combinazione di tre attributi: nome, cognome e data di nascita.

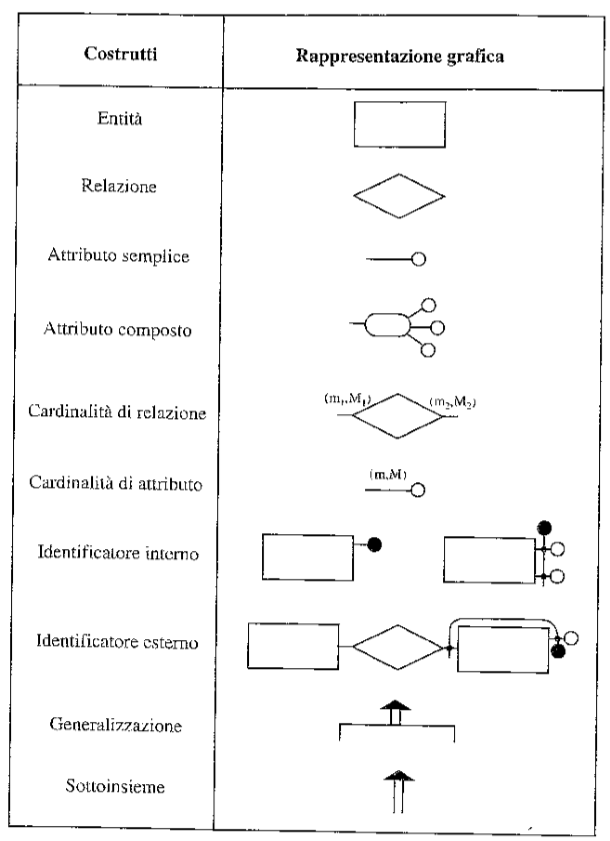
In altri casi può accadere che per identificare in modo univoco un’occorrenza sia necessario utilizzare un attributo presente in una relazione **Entità debole** (badando bene questa cosa si può fare solo se l’occorrenza che appartiene all’entità da identificare è coinvolta con cardinalità **1,1** alla relazione), in questo caso di parla di **Identificatore esterno**.

**Esempio:**

****

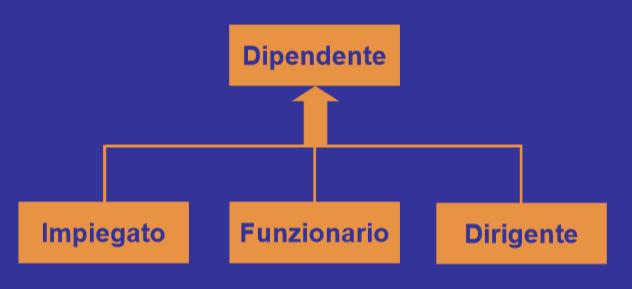
È possibile che due studenti possano avere matricola uguale se iscritti a università diverse, quindi per indentificarli in modo univoco è necessario considerare anche il nome dell’università d’appartenenza, contenuto nella relazione iscrizione.

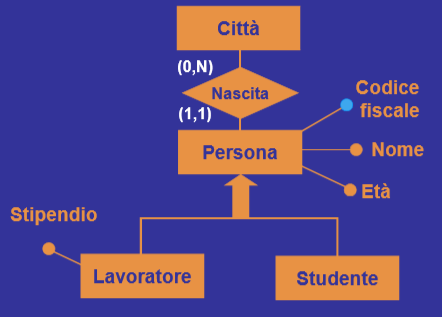
**Considerazioni su questi tipi di identificatori:**

* Possono **essere coinvolti più attributi** ognuno dei quali con cardinalità 1.1.
* Possono **essere coinvolte più entità** che facciano parte alla relazione (vale sempre che quella da indentificare partecipi alla relazione con cardinalità 1.1).
* Si possono **identificare entità a loro volte identificare esternamente**, senza creare però cicli.
* Ogni entità **deve avere un identificatore**, ma ne può avere **anche più di uno** che possono anche essere opzionali 0.1 l’importante che uno non lo sia.

**Generalizzazione:** All’interno del linguaggio UML è possibile generalizzare più entità in una sola.

**Esempio:**

L’entità **dipendente** è generalizzazione dell’entità **impiegato**, **funzionario** e **dirigente**. Entità che per **dipendente** vengono dette **specializzazioni**.



**Concetto di ereditarietà:**

È importante notare che le specializzazioni ereditano tutti gli attributi dell’entità che li generalizza, aggiungendone a loro volta degli altri (che le specializzano).

**Tipi di generalizzazione:** Esistono due tipi fondamentali di generalizzazioni:

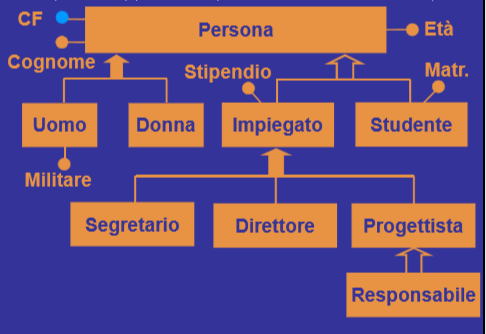
* **Totale:** Se ogni occorrenza (tupla) dell’entità genitore è occorrenza di almeno una delle entità figlie, altrimenti è **parziale**.
* **Esclusiva:** se ogni occorrenza (tupla) dell’entità genitore fa parte al massimo di un entità figlia, altrimenti è **sovrapposta.**

Si considerano (senza perdita di generalità) solo generalizzazioni esclusive e distinguiamo fra totali e parziali.

**Graficamente:**

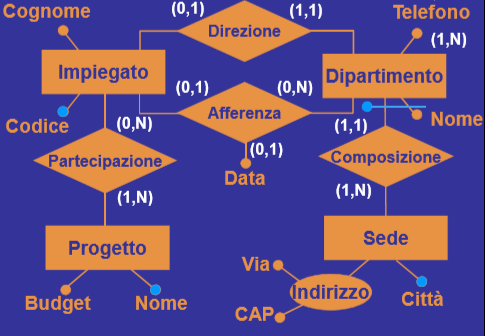
**Totale Parziale**

Se una generalizzazione ha non un’entità figli si parla di **Sottoinsieme.**

**Esempio:**

La generalizzazione impiegato, studente è **Esclusiva** perché non è detto che all’interno dell’insieme persona siano tutti impiegati o studente. Mentre la generalizzazione uomo, donna è **totale** in quanto tutte le persone saranno o uomo o donna.

**Documentazione associata agli schemi concettuali:** Insieme agli schemi concettuali vengono si solito associate le seguenti documentazioni:

* **Dizionari dei dati**: Descrivono brevemente le varie:
  + **Entità**
  + **Relazioni**
* **Vincoli di integrità non esprimibili.**

**Esempio:**

**Dizionario dei dati (entità):**

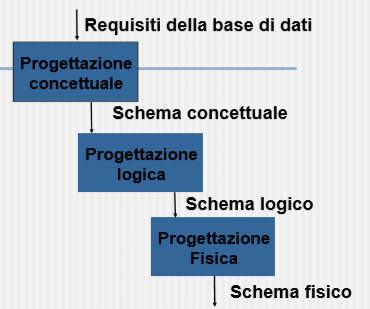


**Dizionario dei dati (associazioni):**

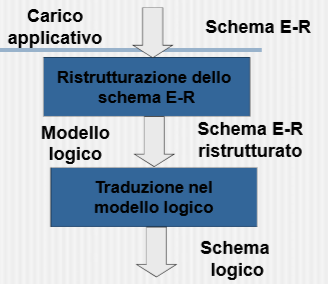


**Vincoli non esprimibili:**





**Ristrutturazione del modello ER:** All’atto del passaggio dalla progettazione concettuale (modello ER) a quello logica (modello relazionare) è opportuno effettuare delle valutazioni sulle prestazioni della base di dati, in modo da semplificare la traduzione e ottimizzare le prestazioni. Ovviamente le valutazioni che andiamo a fare sono relative (basate su delle medie) visto che non possiamo sapere con esatta certezza il carico di lavoro che la base di dati dovrà sopportare.

**Parametri:** I parametri utilizzati per valutare le prestazioni del DB sono:

* Il **numero di occorrenze** (tuple) che la base di dati dovrà gestire.
* Il **numero di accessi** alle tuple durante le operazioni.

**Principio di Pareto (80:20):** principio su cui si basa l’analisi delle prestazioni la quale afferma che un sistema utilizza l’80% delle sue risorse per elaborare il 20% delle operazioni più frequenti. Principio che sfruttiamo per calcolare gli accessi totali sul 20% delle informazioni più frequenti.

**Tavole di carico:** sono delle tavole utilizzate per tenere traccia e stimare le prestazioni del DB sono di tre tipi:

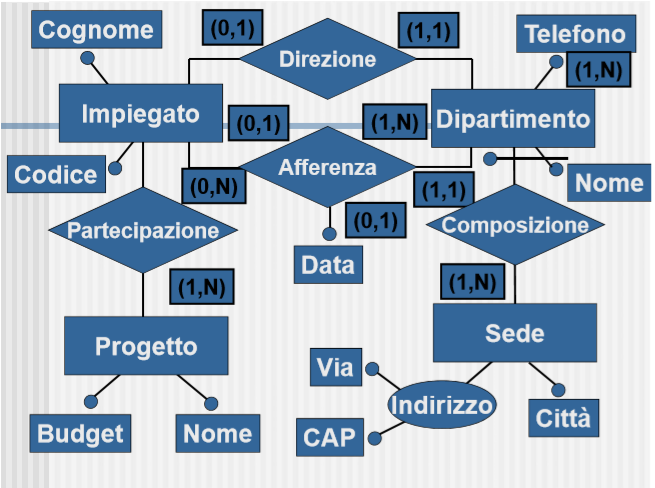
* **Tavola dei volumi:** contiene una stima del numero di occorrenze (tuple) per ogni entità e relazione.
* **Tavola delle operazioni:** riporta tipo e frequenza per ognuna del 20% delle operazioni più frequenti.
  + **Operazioni Interattive:** richiedono la presenza di un utente (quindi più lente).
  + **Operazioni batch:** non richiedono la presenza di un utente (quindi più veloci).
* **Tavola degli accessi:** riporta il numero di accessi in lettura e scrittura su entità e relazioni per il 20% delle operazioni più frequenti.

**Ottenimento dei dati:** è importante sottolineare che dati come:

* Frequenza di una determinata operazioni
* Numero di associazioni (tuple) previste

Sono tutte informazioni che ci vengono date dall’utente finale quindi impossibili da reperire in maniera autonoma.

**Esempio**

****

**Tavola dei volumi:**

****

**Tavola delle operazioni:**

****

**Tavola degli accessi:**

****

**Attività delle ristrutturazioni:**

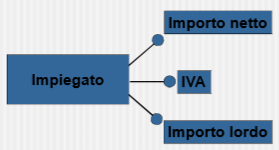
oltre alla valutazione delle prestazioni la ristrutturazione del modello ER include anche le seguenti operazioni:

* **Analisi delle ridondanze**
* **Eliminazione delle generalizzazioni**
* **Partizionamento/accorpamento di entità e relationship**
* **Scelta degli identificatori primari**

**Analisi delle ridondanze:** All’interno di un modello ER una ridondanza è un’informazione significativa e presente nel database, ma che potrebbe tranquillamente essere derivata da altre informazioni sempre contenute nella base di dati. In questa fase della ristrutturazione si decide se conviene conservare le ridondanze (perché offrono un aumento prestazionale) oppure eliminare (perché sprecano spazio o risorse).

**Vantaggi:**

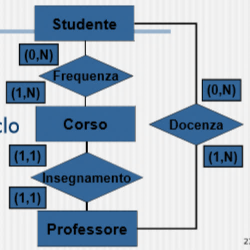
* Semplificazione delle interrogazioni.

**Svantaggi:**

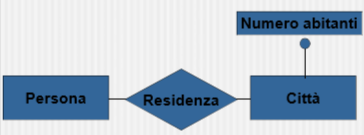
* Appesantimento degli aggiornamenti
* Maggiore occupazione di spazio

All’interno di un modello ER le ridondanze si presentano come:

* **Attributi derivabili:** 
  + Da altri attributi della stessa entità (o associazione)
  + Da attributi di altre entità (o associazioni)
* **Relazioni derivabili dalla composizione di altre relazioni in presenza di cicli.**



**Esempio di analisi di una ridondanza:** in questo esempio la ridondanza è data dall’attributo **numero di abitanti** che può essere ottenuto contando il numero di occorrenze presenti nella tabella persona. Ma ci



**Ipotesi di tavola dei volumi:**

Dal numero di tuple previste totalmente possiamo calcolare che per questo DB serviranno circa 600 byte a città per conservare il dato ridonante.

**Ipotesi di tavola operazioni:**

**Operazioni 1:** Memorizza una nuova persona e relativa città di residenza.

**Operazione 2:** Stampa i dati di una città incluso il numero di abitanti.

**Tavola degli accessi:**

| **Presenza di ridondanza** | **Assenza di ridondanza** |
| --- | --- |
| **Numero totale di accessi:**  Costi:   * **Op 1**: 1500 accessi in scrittura e 500 in lettura al giorno. * **Op 2**: 1 accesso in lettura   Aggiungiamo i doppi accessi per la scrittura  **Totale:** 3501 accessi al giorno e 600 byte di dato ridondante | **Numero totale di accessi:**  Costi:   * **Op 1**: 1000 accessi in scrittura * **Op 2**: 10000 accesso in lettura   Aggiungiamo i doppi accessi per la scrittura  **Totale:** 12000 accessi al giorno |

Possiamo tranquillamente dedurre che in questo particolare DB la ridondanza riduce di molto gli accessi ed ‘aumenta le prestazioni, per questo motivo è utile conservarla.

**Eliminazione delle generalizzazioni:** Il modello relazionare non può rappresentare direttamente le generalizzazioni (esso rappresenta i dati tramite entità e relazioni tabelle praticamente) per questo motivo si eliminano sostituendole con entità o relazioni. Per fare ciò esistono tre possibilità:

* **Accorpamento dei figli nel genitore:** gli attributi dei figli vengono inseriti nel padre, con un ulteriore attributo che ne caratterizza il tipo, questa soluzione si applica nel caso in cui:
  + Quando le operazioni non fanno molta distinzione tra le occorrenze e gli attributi delle stesse.

Così facendo avremmo uno spreco di memoria che però ci consente di ridurre gli accessi.

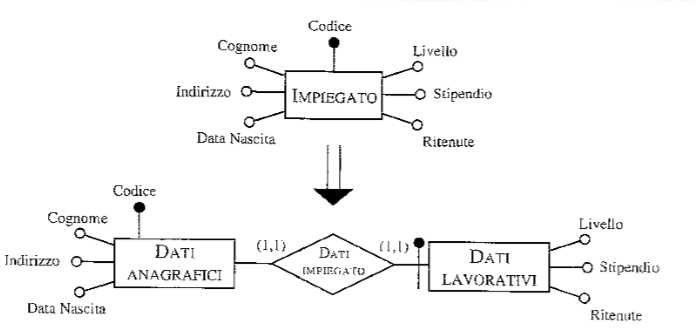
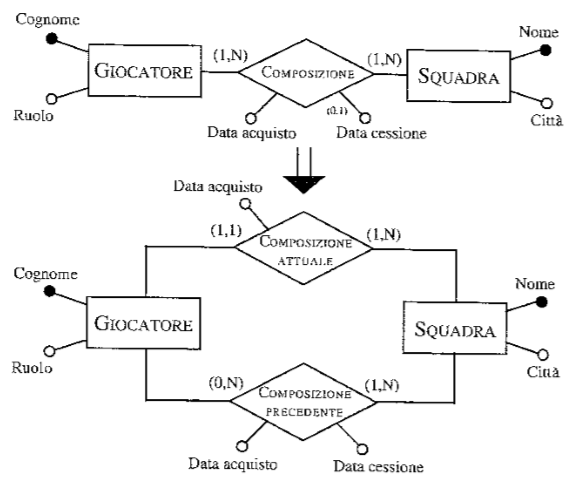
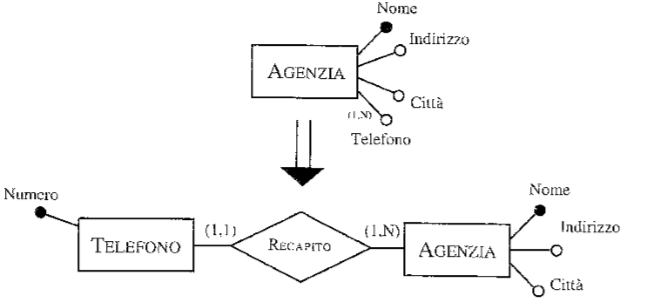
* **Accorpamento del genitore nei figli:** gli attributi del padre vengono trasferiti ai figli, questa soluzione è possibile solo se la generalizzazione è totale (altrimenti alcune tuple rimarrebbero escluse) si applica nel caso in cui:
  + Quando le operazioni fanno molta distinzione tra le occorrenze e gli attributi.

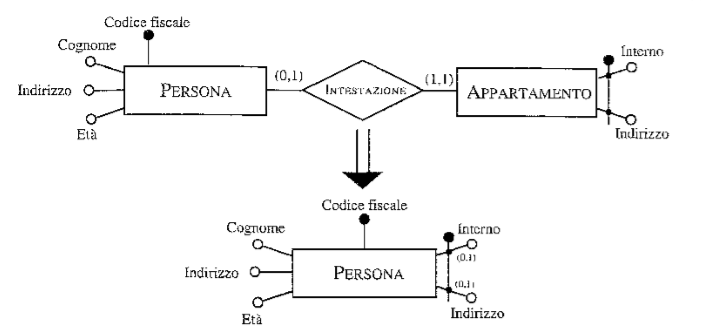
In questo caso abbiamo un notevole risparmio di memoria ma un aumento degli accessi.

* **Sostituzione della generalizzazione con relazioni:** la generalizzazione si trasforma in delle relazioni collegate con cardinalità 1,1 a quella genitore, in questo caso non c’è scambio di attributi. Viene inoltre aggiunto un vincolo di integrità che vieta alle tuple di appartenere ad ‘entrambe le relazioni questa soluzione si applica nel caso in cui:
  + Quando la generalizzazione non è totale (requisito opzionale) e quando le operazioni si riferiscono solo ad alcuni figli (c’è distinzione tra figli e genitori).

In questo caso abbiamo un risparmio di memoria ma un aumento di accessi.

**Partizionamento/accorpamento di entità a relazioni:** Il partizionamento o l’accoppiamento di un’entità a una relazione è una ristrutturazione che si effettua quando per una determinata entità vengono richiesto l’accesso solo ad una parte di dati appartenenti a quest’ultima. Esistono vari tipi di approcci:

* **Partizionamento verticale di entità:** Quando si suddividono gli attributi dell’entità. Questa soluzione è ottima perché genera delle entità con pochi attributi.
* **Partizionamento orizzontale di relazioni:** Quando suddividiamo un’entità in base alle sue occorrenze, per esempio: Potremmo dividere gli impiegati in analisti e venditori. Questo tipo di soluzione può portare ad ‘uno spreco di memoria e attributi. 
* **Eliminazione degli attributi multi-valore:** Visto che il modello relazionare non è in grado di tradurre attributi multi-valore, durante la ristrutturazione del modello ER è opportuno gestire come entità singole gli attributi multi-valore.

* **Accorpamento di entità/relazioni:** è l’operazione inversa del partizionamento, si realizza nel caso di relazioni di tipo 1 a 1 ed è utile attuarla nel caso in cui gli attributi di un’entità vengano usati quasi sempre da un altro esempio:

**Scelta degli identificatori principali:** è un passo fondamentale visto che il modello relazionare usa le chiavi per stabilire legami tra dati in relazioni diverse. Criteri per la scelta di una buona chiave primaria sono:

* **Assenza di Opzionalità:** non può giustamente essere un attributo opzionale.
* **Semplicità:** Deve essere un attributo semplice (meglio se un numero).
* **Utilizzo nelle operazioni più frequenti o importanti:**

Se nessun attributo soddisfa i requisiti precedenti si realizza un attributo specifico (il famoso ID).

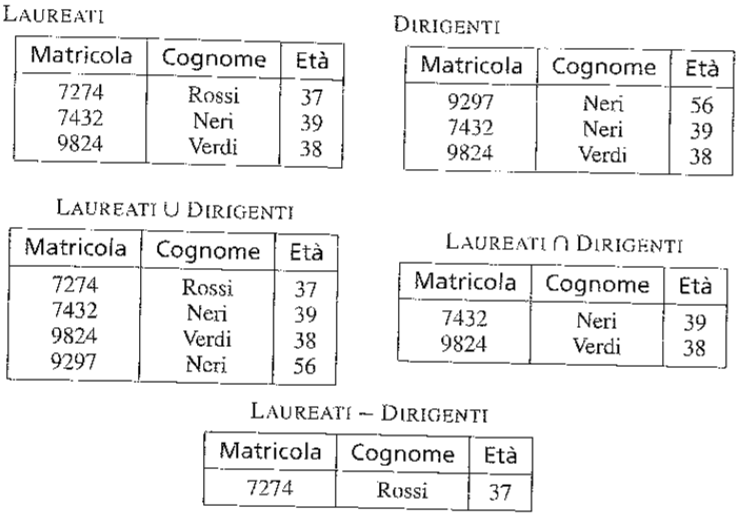
**Mapping (Modello ER to Modello Relazionare)** disponibile a questo [link](https://studentiunisait-my.sharepoint.com/personal/g_arienzo_studenti_unisa_it/Documents/Universit%C3%A0/Informatica%20Anno%202/Basi%20di%20dati/Kit%20di%20studio/Traduzione%20dal%20modello%20ER%20a%20quello%20relazionare.pdf).

**Algebra e calcolo relazionare applicato al modello relazionare:**

Il linguaggio SQL si basa su un pool di operazioni definite nell’algebra e nel calcolo relazionare. I principali operatori insiemistici sono:

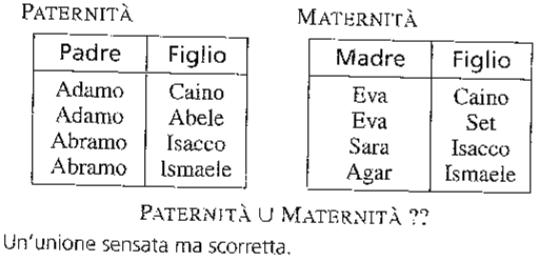
* **Unione:** Simbolo **U** è un operatore che prese due relazioni R1 e R2 crea una terza relazione R3 che contiene tutte le tuple di R1 e R2 (senza ripetizioni).
* **Intersezione:** Simbolo **∩** è un operatore che prese due relazioni R1 e R2 crea una terza relazione R3 che contiene le tuple che appartengono sia a R1 che a R2.
* **Differenza:** Simbolo **–** è un operatore che prese due relazioni R1 e R2 crea una terza relazione R3 che contiene le tuple di R1 senza le tuple contenute in R2.

**Esempio:**

****

Ovviamente per essere sensate tali operazioni vanno eseguite su relazioni che hanno attributi simili.

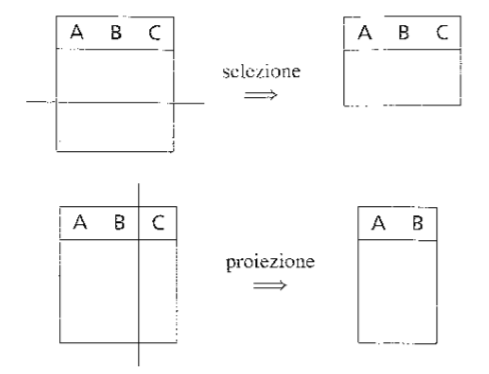
**Esempio:**



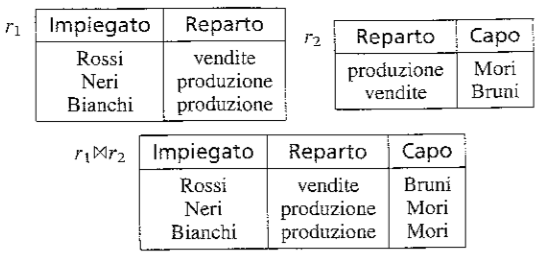
**Ridenominazione:**

Come abbiamo visto nell’esempio precedente è impossibile eseguire operazioni insiemistiche su insiemi (relazioni) che non godono delle stesse proprietà (attributi) per questo l’operatore di **ridenominazione** permette di“adattare” i nomi degli attributi in modo da rendere possibili le operazioni sopracitate.

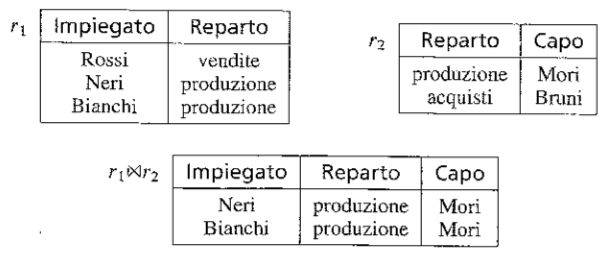
**Simbolo: 🡨**

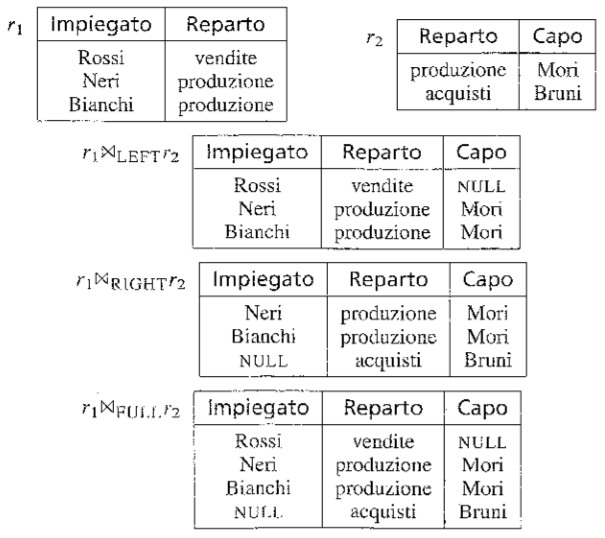
**Selezione e Proiezione:** Insieme al join questi due operatori sono i più tipici dell’algebra relazionare e ci permettono effettivamente di manipolare le relazioni. La selezione e le proiezioni sono due operatori “ortogonali”, in quanto la selezione ci permette di selezionare una tupla di una relazione, mentre la proiezione di proiettare tutti i valori di un determinato attributo (colonna):

**Join:** permette di correlare dati contenuti in relazioni diverse fra di loro, sulla base di valori uguali o attributi, esistono diversi tipi di join:

**Join naturale:** Come dice il nome è il tipo più naturale di join che unisce più relazioni tramite valori o attributi. Per esempio, Rossi viene unito con Mori che è il capo del reparto vendite.

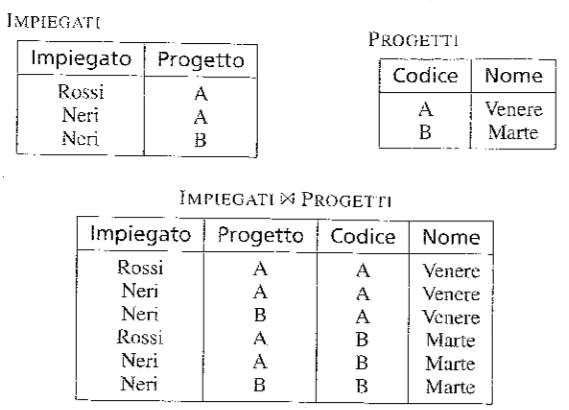
**Join completi e incompleti:** una join può essere definita in due modi:

* **Completa:** quando tutte le tuple di una delle relazioni in gioco fanno parte della relazione generata.
* **Incompleta:** quando alcune tuple non vengono inserite nella nuova relazione.

**Join esterni:** questo tipo di join, inserisce all’interno della nuova relazione anche le tuple che non trovano corrispondenza in una delle due relazioni iniziali, riempiendo dove necessario le tuple con valori nulli.

Da notare come **destro**, **sinistro** e **totale**, si riferiscono a quale tabella andrà presa per interno.

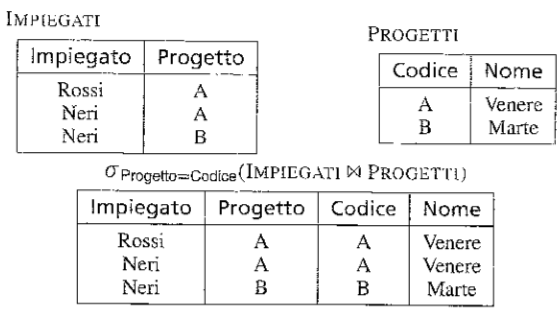
**Esempio:**

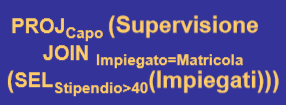
**Theta-join ed equi-join:** Osservando l’esempio sotto possiamo notare come il prodotto cartesiano ha di solito poco significato. Per questo all’interno di SQL sono stati definiti altri due tipi di join:

* **Theta-join:** che permette di inserire una selezione dopo aver eseguito un prodotto cartesiano. (è una join con una select all’interno)



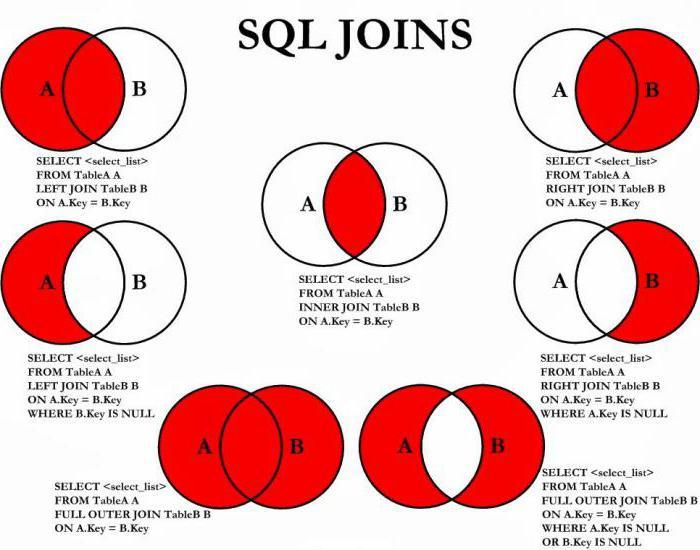
Software\_House **INNER JOIN** Giochi **ON** Software\_House.ID=GIochi.ID\_SoftwareH

* **Equi-join:** è un particolare tipo di Theta-join, in cui la condizione di selezione sia una congiunzione di atomi di uguaglianza, con un attributo della prima o della seconda relazione. **Esempio:**

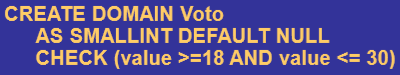
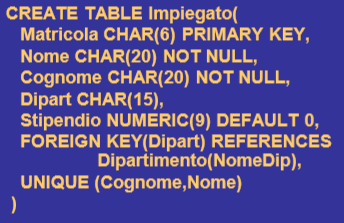


Software\_House INNER JOIN Giochi ON Software\_House.ID=GIochi.ID\_SoftwareH

**Algebra relazionare** [**qui**](https://studentiunisait-my.sharepoint.com/personal/g_arienzo_studenti_unisa_it/Documents/Universit%C3%A0/Informatica%20Anno%202/Basi%20di%20dati/Kit%20di%20studio/Lezione%206-Sintesi-Query-Algebra-Relazionale.pdf)

**Comandi SQL:** Structured Query Language**:**

**DDL:** Data Definition Language

**Create Table: Create Domain:**

**Not null:** Definisce campi obbligatori

**Default ‘manca qualcosa’:** visualizza la stringa fra apici in caso di mancato valore dell’attributo

Metodi per la creazione di chiavi esterne:

1. Dipartimento **Char(5) references** Dipartimento(codice),
2. **Foreign key (**Manager**) references** Impiegati (manager)

**On delete** set null

**On update** cascade

Metodi per l’unione di più tabelle:

1. **FROM** Nome\_tabella1 **INNER JOIN** Nome\_tabella2 **ON** Nome\_tabella1.Attributo = Nome\_tabella2.Attributo
2. **WHERE** Nome\_tabella1.Attributo = Nome\_tabella2.Attributo

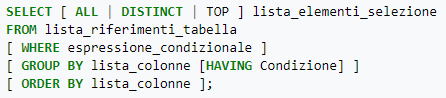
| **ALTER TABLE** nome\_tabella  **ADD** nome\_attributo tipo\_attributo; | Aggiunge un attributo all’interno di una tabella già creata |
| --- | --- |
| **ALTER TABLE** nome\_tabella  **DROP** nome\_attributo | Elimina un attributo all’interno di una tabella |

**DML:**

| **INSERT INTO** nome\_tabella  (elenco attributi che si vuole valorizzare)  **VALUES** (elenco valori da inserire nello stesso ordine dell’elenco precedente)**;** | Permette di inserire nuple all’interno di una tabella |
| --- | --- |
| **UPDATE** nome\_tabella  **SET** nome\_attributo **=** valore da assegnare  **WHERE** eventuale condizione**;** | Permette di modificare il valore di uno specifico attributo |
| **DELETE FROM** nome\_tabella  **WHERE** eventuale condizione; | Permette di eliminare un nupla all’interno della tabella |

**DQL:** Data Query Language

**Select:**

****

**QUERY LANGUAGE**

| **SELECT** nome\_attributo/i  **FROM** nome\_tabella  **WHERE** eventuale\_condizione; | | Permette di selezionare un dato attributo/i in una determinata tabella con una determinata condizione. |
| --- | --- | --- |
| **SELECT DISTINCT** | | Seleziona solo una volta i valori presenti più di una volta nella tabella |
| **SELECT** nome\_attributo **AS** nome\_colonna | | Assegna un nome alla colonna |
| **SELECT** nome\_attributo/i  **FROM** nome\_tabella  **WHERE** nome\_attributo = [valore\_input]; | | Selezione nuple che soddisfano un determinato valore dato in input |
| **SELECT TOP** numero nome\_attributo | | Permette di selezionare un tot di righe (numero) partendo dall’alto |
| **FUNZIONI DI AGGREGAZIONE** | | |
| **SELECT COUNT ( )** | | Conta il numero di volte in cui l’attributo in parentesi compare |
| **SELECT SUM ( )** | | Restituisce la somma dei valori presenti nella colonna |
| **SELECT AVG ( )** | | Calcola la media dei valori in una colonna |
| **SELECT MIN ( ) MAX ( )** | | Restituisce il valore minimo o il valore massimo presente nella colonna (funziona anche su valori non numerici) |
| **ORDINAMENTI E RAGGRUPPAMENTI** | | |
| **ORDER BY** nome\_attributo | | Ordina la colonna in modo ascendente |
| **ORDER BY** nome\_attributo **DESC** | | Ordina la colonna in modo discente |
| **GROUP BY** nome\_attributo  **HAVING** eventuale\_condizione | | Raggruppa le nuple per un determinato attributo |
| **CONDIZIONI DI RICERCA** | | |
| **WHERE** attributo **BETWEEN** valore\_1 **AND** valore\_2 | | Compreso fra valore 1 e valore 2 |
| **WHERE** attributo **IN (**valore\_1, valore\_2**)** | | Presente nel dominio fra parentesi |
| **WHERE** attributo **IS NOT NULL** | | Valore dell’attributo non nullo |
| **WHERE** attributo **LIKE** | | |
| **LIKE ‘**xyz**\*’** | Ricerca le stringhe che iniziano con xyz | |
| **LIKE ’\***xyz**’** | Ricerca le stringhe che terminano con xyz | |
| **LIKE ’\***xyz**\*’** | Ricerca le stringhe che contengono xyz | |
| **LIKE ‘\_**xyz**’** | Ricerca le stringhe di 4 caratteri che finiscono con xyz | |

| **Tabella a doppia entrata** | |
| --- | --- |
| **TRANSFORM** Count (Impiegati.ID) **AS** Nome\_colonna  **SELECT** impiegati.residenza  **FROM** impiegati  **GROUP BY** impiegati.residenza  **PIVOT** impiegati.dipartimento; | Restituisce il conteggio degli impiegati per dipartimento (Colonne) , divisi per la città di provenienza (Righe). |

**DCL**

| **GRANT** comando  **ON** nome\_tabella  **TO** nome\_utente\_1, nome\_utente\_2; | Garantisce un determinato servizio ad un determinato utente su una determinata tabella |
| --- | --- |
| **REVOKE** comando  **ON** nome\_tabella  **TO** nome\_utente\_1, nome\_utente\_2; | Revoca un determinato servizio ad un determinato utente su una determinata tabella |

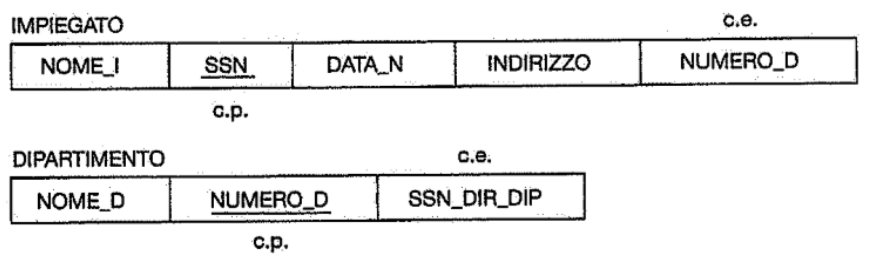
**Normalizzazione di Schemi di Basi di Dati Relazionali**

**10.1:** Per verificare la “bontà” di uno schema relazionare esistono quattro misure informali:

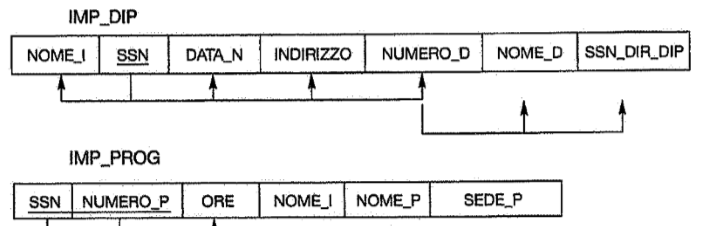
* **Semantica degli attributi**
* **Riduzione dei valori ridondanti nelle tuple**
* **Riduzione del numero di valori nulli nelle tuple**
* **Impossibilità di generare tuple spurie**

**10.1.1 Semantica degli attributi:** la semantica degli attributi è un concetto molto simile alla coerenza delle classi nella programmazione ad oggetti, praticamente ogni relazione deve mantenere attributi che descrivono sé stessa.

**Linea guida 1:** Si progetti ogni schema di relazione in modo tale che sia semplice spiegarne il significato. Non si uniscano attributi proveniente da più tipi di entità e tipi di associazione in un’unica relazione.

Esempio di entità semanticamente chiare:

Esempio di entità non del tutto chiara (contiene attributi che non la riguardano):



**10.1.2 Informazioni ridondanti nelle tuple e anomalie di aggiornamento:** come abbiamo visto in precedenza, mantenere attributi ridondanti può sia essere uno svantaggio che un vantaggio, in particolare una relazione che contiene attributi ridondanti oltre a sprecare spazio potrebbe anche incorrere nel problema delle **anomalie di aggiornamento**, esse possono essere classificate in tre categorie:

* **Anomalie di inserimento:** possono essere divide in due tipi:
  + Necessita di inserire valori in diverse relazioni per inserire una tupla in una specifica relazione (Esempio **estetista/appuntamento**)
  + Impossibilità di inserire alcune tuple in alcune relazioni se mancano tuple a cui collegarle (Esempio **stanza/macchinario**).
* **Anomalie di cancellazione:** Se si cancella una tupla a cui sono logicamente collegate altre tuple si può incorrere in delle anomalie (Esempio non posso cancellare una **stanza** perché poi il **macchinario** rimane scoperto).
* **Anomalie di modifica:** Se si aggiornano i valori di una tuple bisogna aggiornali in tutta la base di dati altrimenti si creano problemi di incoerenza dei dati.

**Linea guida 2:** si progettino gli schemi relazioni di base in modo che nella relazioni non siano presenti anomali di inserimento, cancellazione o modifica. Nel caso in cui siano presenti delle anomali è bene rilevarle e aggiornare i programmi di conseguenza.

**10.1.3 Valori nulli nelle tuple:** è possibili che in alcuni schemi vengano raggruppati un gran numero di attributi a formare una grossa relazione, in questi casi può capitare che molti di questi attributi non vengano utilizzati e che gli venga assegnato un valore nullo, in questi casi si incorre in inulti sprechi di spazio e problemi. Inoltre i valori nulli possono avere i seguenti significati

* **L’attributo non è pertinente per questa tupla**.
* **Il valore** dell’attributo per questa tupla **è sconosciuto**.
* **Il valore è noto ma assente**, cioè non è ancora stato memorizzato.

**Linea guida 3:** Per quanto possibile, si eviti di porre in una relazione di base attributi in cui i valori non possono essere frequentemente nulli. Se i valori nulli sono inevitabili, ci si assicuri che essi si presentino solo in casi eccezionali e che non riguardino una maggioranza di tuple nella relazione.

**10.1.4 Generazione di tuple spurie:** con il termine tuple spurie si fa riferimento a quelle tuple che dopo una join non vengono adeguatamente “collegate”.

**Linea guida 4:** Si progettino schemi di relazioni in modo che essi possano essere riuniti, tramite JOIN, con condizione di uguaglianze su attributi che sono o chiavi primarie o chiavi esterne in modo che non vengano generate tuple spurie. Non si abbiano relazioni che contengono attributi di accoppiamento diversi dalle combinazioni chiave esterna chiave primaria. Se tali relazioni sono inevitabili non si effettui su di esse un operazione di join sulla base di questi attributi, in modo da non produrre tuple spurie.

**10.2 Dipendenza funzionali:** è un vincolo tra due insiemi di attributi del database, in particolare Supponiamo che lo schema di db relazionale abbia n attributi **A1, A2, …, An** e che l'intero database sia descritto da uno schema di relazione universale **R={A1, A2, …, An}.**

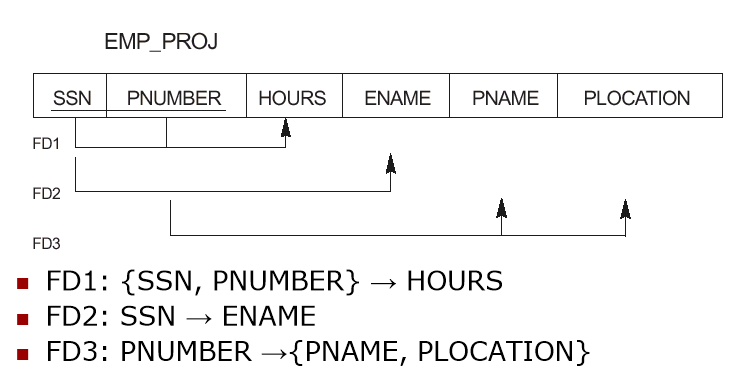
Una dipendenza funzionale, denotata da **X -> Y**, tra due insiemi di attributi **X** e **Y** che sono sottoinsiemi di R, **specifica un vincolo sulle possibili tuple che possono formare una istanza di relazione r di R.**

Ciò significa che:

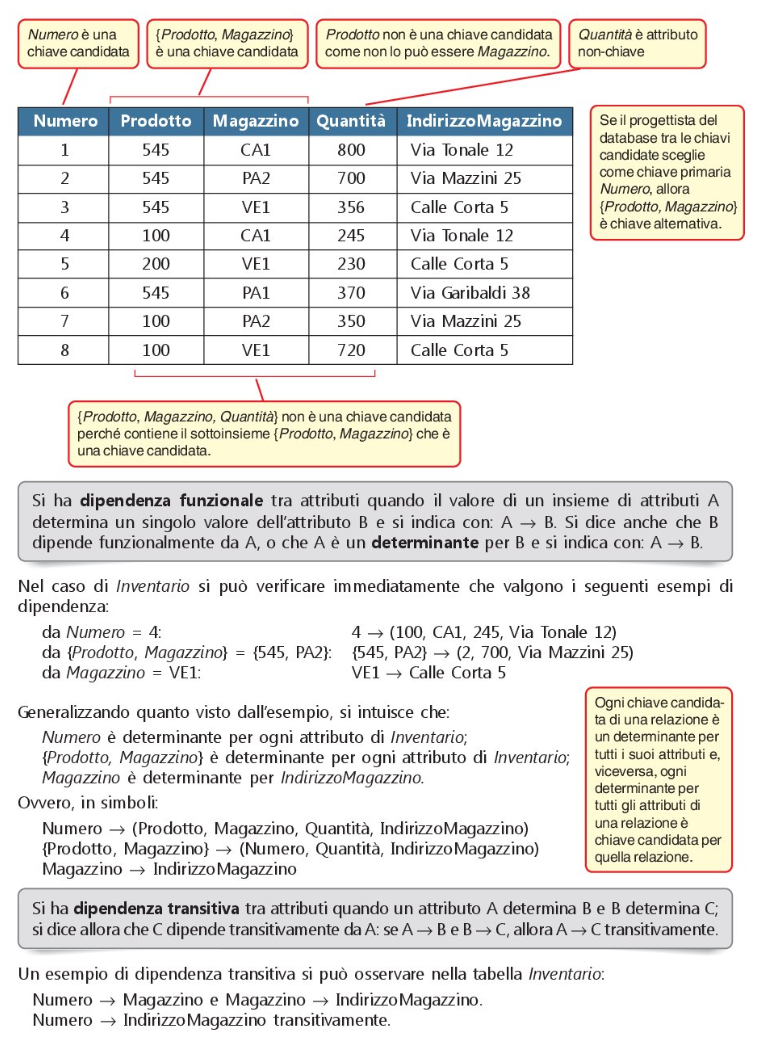
* I valori della componente **Y** di una tupla di **r** **dipendono** da (o sono **determinati** da) i valori della componente **X**.
* E i valori della componente **X** di una tupla **determinano univocamente** (o **funzionalmente**) i valori della componente **Y**.

Cioè esiste una dipendenza funzionale da X a Y:

* **Y** è funzionalmente dipendente da **X**
* **X** è la parte sinistra della FD
* **Y** è la parte destra della FD

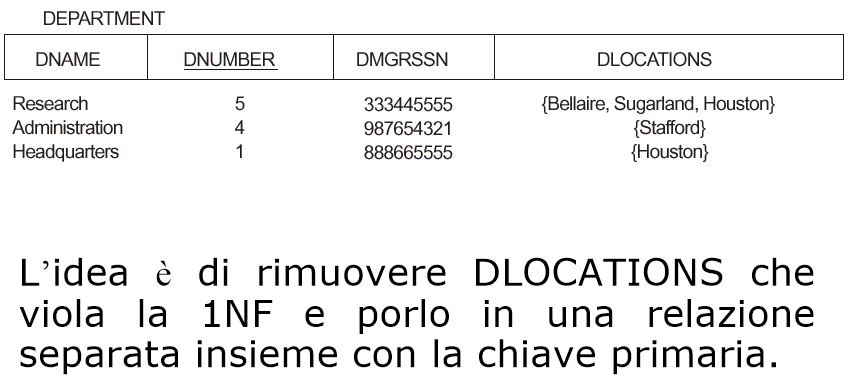
Esempio:

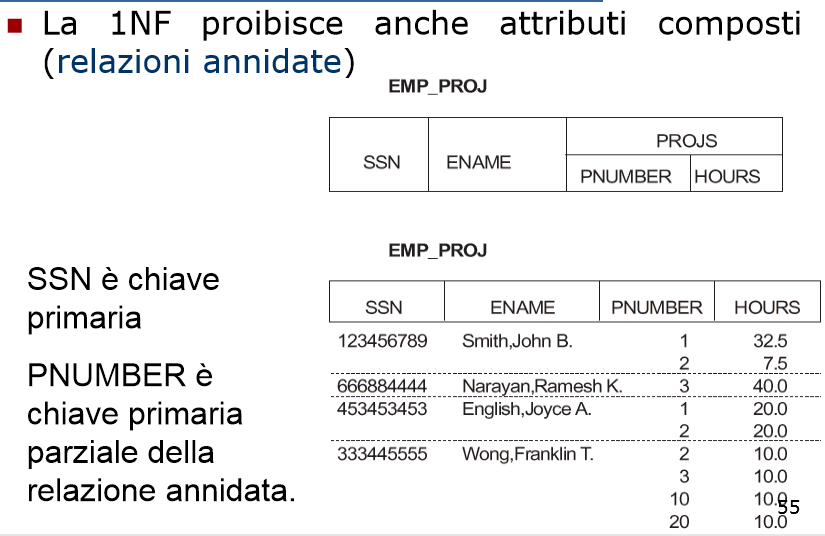
È importante notare che una dipendenza funzionale è una proprietà dello schema relazionale, pertanto non può essere creata in modo automatico ma deve essere definita da chi conosce il significato degli attributi

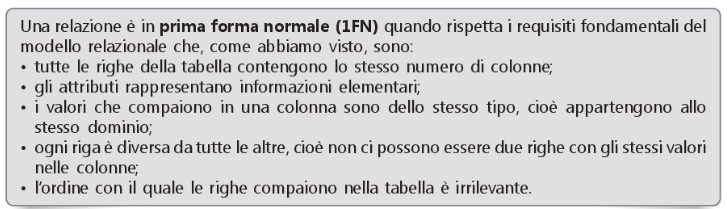
**Normalizzazione dei dati:** può essere vista come un processo che consente di decomporre schemi di relazione non ottimali in schemi più piccoli, tali da garantire la mancanza di Anomalie di Aggiornamento, esistono varie forme di normalizzazione:

* **Prima forma normale:** è stata definita per non consentire attributi multi-valore, composti e loro combinazioni. Gli unici valori consentiti da 1NF sono valori **atomici** (o **indivisibili**).

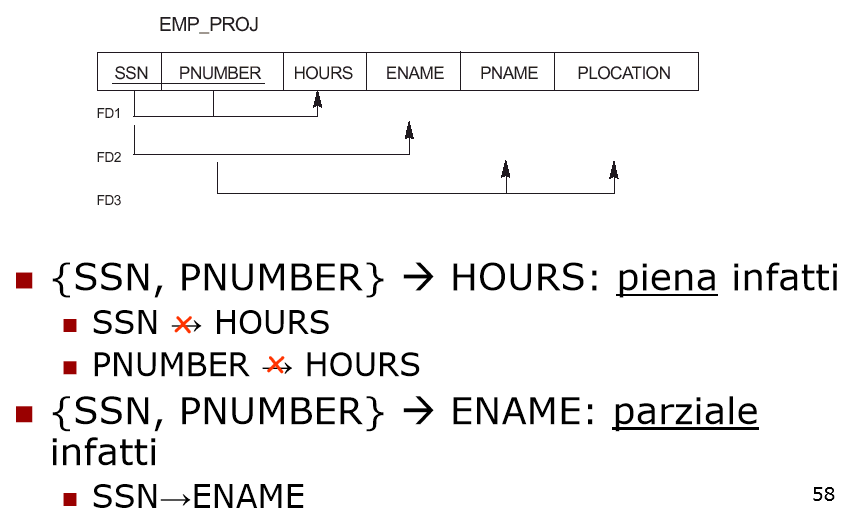
**Esempi:**

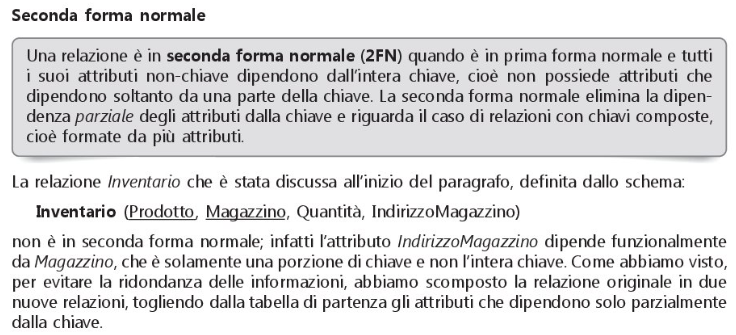
****

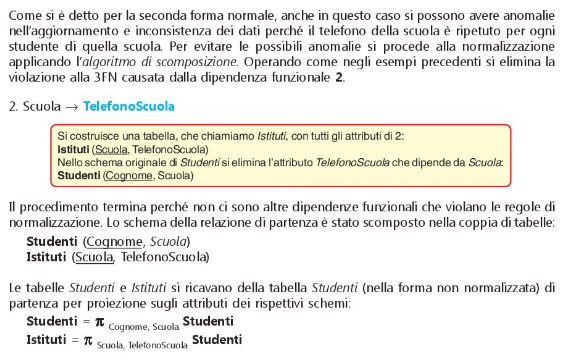
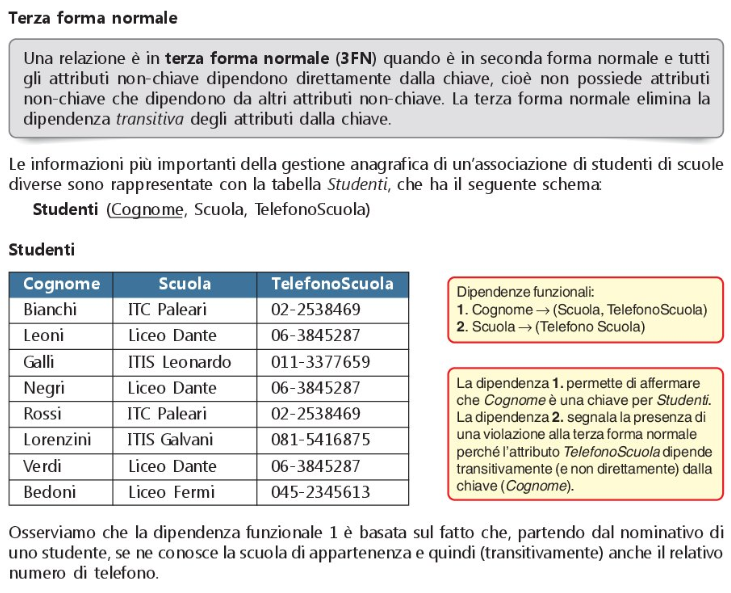


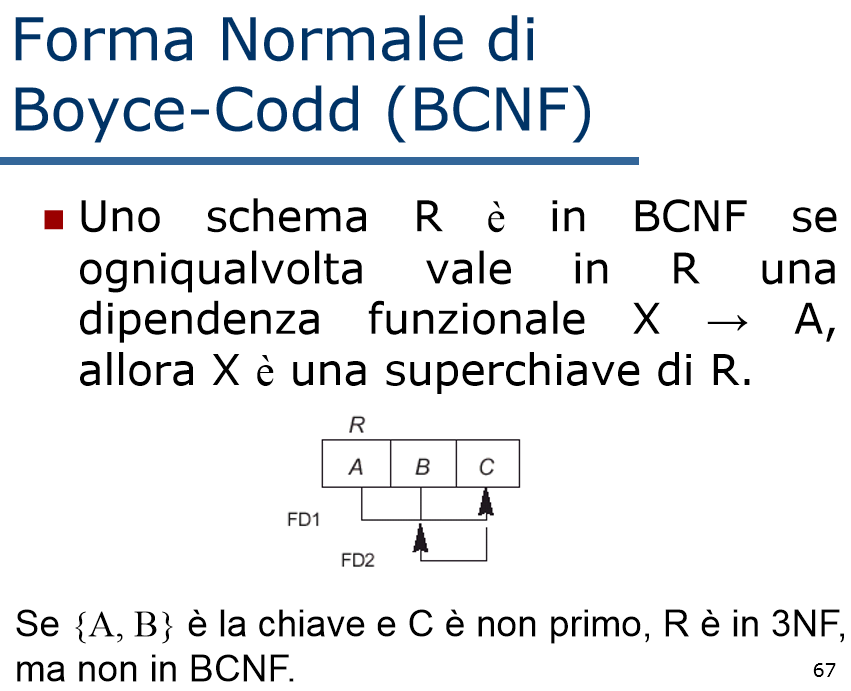
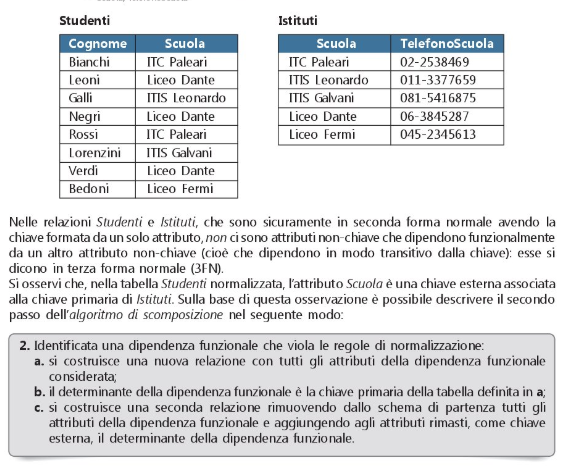


* **Seconda forma normale:** è basata sul concetto di **dipendenza funzionale piena** ovvero un tipo di dipendenza funzionale in cui la rimozione di qualche attributo **A** da **X**, implica che al dipendenza non vale più, esempio:



Quindi uno schema di relazione R è in 2NF se ogni attributo non primo A in R ha una dipendenza funzionale piena dalla chiave primaria di R





**Riassumendo:**

1. **Prima regola di normalizzazione:**
   1. Tutte le righe contengono lo stesso numero di colonne;
   2. Attributi rappresentano informazioni elementari;
   3. I valori in ogni colonna appartengono allo stesso dominio;
   4. Ogni nupla è diversa da tutte le altre;
   5. L’ordine delle righe è irrilevante.
2. **Seconda regola di normalizzazione:**
   1. Prima verificata;
   2. Tutti gli attributi dipendono interamente dalla chiave;
3. **Terza regola di normalizzazione:**
   1. Seconda verificata;
   2. Non ci devono essere attributi che dipendono da altri attributi che a loro volta dipendono dalla chiave (dipendenza transitiva).

**Progettazione fisica:**

Le basi di dati devono essere memorizzate in memoria secondaria, sostanzialmente per due motivi:

* **Dimensioni**
* **Persistenza**: devono perdurare anche quando non sono utilizzate

I dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale, tra le principali strutture di dati utilizzate per la memorizzazione di basi di dati troviamo:

| Struttura | Implementata in |
| --- | --- |
| Sequenziale | Memoria principale |
| Calcolate (funzione hash) | Memoria principale e talvolta secondaria |
| Ad albero | Memoria secondaria o principale |

**Strutture sequenziali:**

Le operazioni che possono essere eseguite su questo tipo di struttura sono fortemente influenzate dallo stato di quest’ultima (ordinato, non ordinato):

* **Inserimento:** In caso di una struttura non ordinata il nuovo elemento viene semplicemente posto in ultima posizione. Nel caso di una struttura ordinata bisogna individuare la posizione dove andare a posizionare il nuovo elemento, inoltre occorre anche traslare tutti gli elementi successivi, operazione che può rivelarsi molto complessa. Per semplificare questa operazione di utilizzano delle aree di **Overflow** che non sono altro che delle aree vuote all’interno dell’archivio (possono essere **Distribuite** o **Concentrate**).
* **Aggiornamento e Cancellazione:** Nel caso di una struttura ad accesso sequenziale bisogna utilizzare un archivio d’appoggio. Per un archivio ad accesso diretto invece basta individuare l’elemento e modificarlo. Per la cancellazione è inoltre possibile inserire un campo flag che se non è vero rende l’elemento irraggiungibile.
* **Ricerca:** La ricerca è nella maggior parte dei casi l’operazione più importante da eseguire, la sua efficacia a prescindere da metodo usato varia molto a seconda di come sono strutturati i dati all’interno della struttura, nel caso di una struttura organizzata in modo sequenziale ad accesso sequenziale abbiamo:
  + Nel caso di una struttura non ordinato
    - In caso di successo abbiamo (N+1) /2 accessi
    - In caso di insuccesso abbiamo N accessi
  + Nel caso di una struttura ordinato
    - In caso di successo abbiamo (N+1) /2 accessi
    - In caso di insuccesso abbiamo (N+1) /2 accessi

**L’organizzazione Hash**

Questo tipo di organizzazione si basa sull’utilizzo di una funzione che trasforma la chiave primaria di record in un numero intero che rappresenta l’indirizzo logico detto **indirizzo Hash.** Tale organizzazione dei dati permette un accesso diretto molto efficiente.

**La funzione Hash** è la funzione che si occupa di generare dalla chiave K l’indirizzo logico X (H(k)=x) che deve essere calcolato ogni volta che si vuole accedere ad un elemento. Le caratteristiche della funzione Hash sono le seguenti:

* **Deve essere facilmente calcolabile** in quanto viene eseguita molto spesso
* Dato una chiave K **deve produrre sempre** lo stesso indirizzo X
* **Deve generare indirizzi uniformemente distribuiti nell’archivio**
* **Deve generare indirizzi casualmente distribuiti nell’ambito dell’archivio**
* **Evitare che ci siano indirizzi mai generati**

L’ultima caratteristica è che la funzione Hash **deve generare meno collisioni possibili e che deve essere in grado di gestire nel modo più efficiente quest’ultime**, può capitare infatti vista la quasi impossibilità di creare funzioni Hash che non producano mai indirizzi simili fra di loro, che all’inserimento di due chiavi diverse fra di loro la funzione restituisca lo stesso indirizzo logico creando quindi una collisione.

**Gestione delle collisioni:**

le tecniche di gestione delle collisioni sono funzioni che permettono di indentificare un indirizzo libero per la chiave K nel caso in quest’ultima abbia generato una collisione

le principali tecniche di gestione delle collisioni sono:

* **Scansione lineare o indirizzamento aperto**: una volta individuata una collisione si inizia a scansionare l’archivio con un passo unitario (per esempio p=3) fino d’individuare una posizione libera dove inserire il record. Questo metodo ha però un problema esso infatti crea delle **Agglomerazioni** (clustering) ovvero va ad occupare le aree libere dell’archivio con i record che hanno creato delle collisioni aumentando la possibilità che quest’ultime si verifichino.
* Questo problema si risolve utilizzando un altro metodo detto **Scansione non lineare** o **metodo pseudo-random**, questo metodo utilizza una seconda funzione Hash detta funzione di **ReHash** che genera un nuovo indirizzo per il record andato in collisione.
* Un altro metodo consiste nell’inserire l’elemento che genera la collisione in una coda contenente tutti gli elementi destinati a quell’indirizzo, per poi procedere alle ricerca, lettura e inserimento con metodi standard.

**Strutture ad ’albero:**

**Concetto di indice:** un indice è una struttura ausiliaria che permette di identificare in modo univoco un elemento di un insieme, sono esempi di indice:

* L’indice analitico di un libro
* Tabella delle pagine in una memoria paginata
* ID all’interno di una tabella

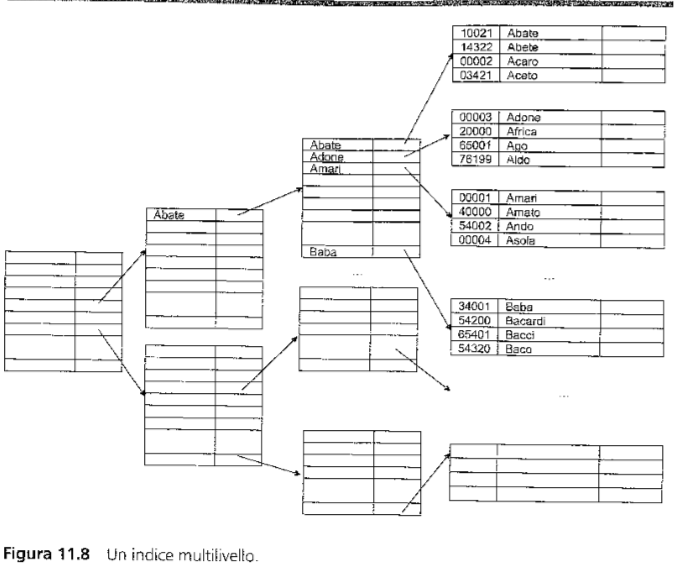
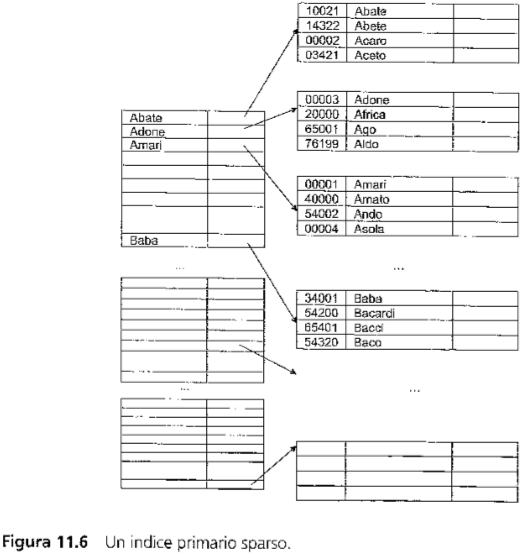
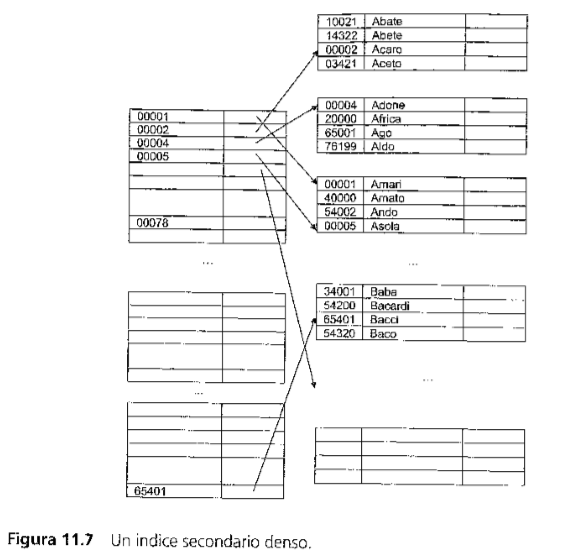
Un indice può essere di due tipi:

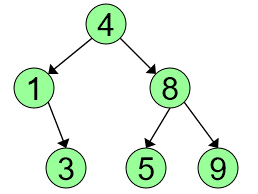
* **Denso:** se contiene tutti i valori della chiave e quindi identifica in modo univoco tutto il file
* **Sparso:** se non contiene tutti i valori della chiave e quindi identifica in modo sparso il file

In particolare diremmo che un indice si dice:

* **primario:** se si fa riferimento ad un campo su cui è basato l’ordinamento della memorizzazione. Può essere sia **sparso** che **denso** (esempio indice generale).
* **secondario:** se si fa riferimento ad un campo diverso da quello su cui è basata la memorizzazione. È denso (esempio indice analitico).

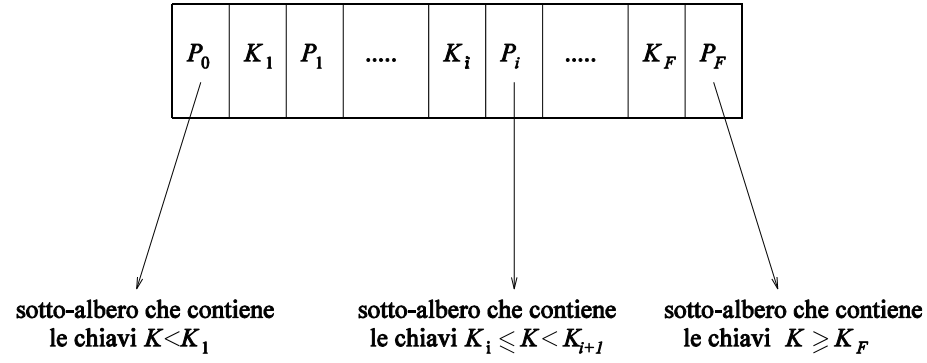
**Indice multilivello:** E un indice su cui è stato create un altro indice in modo da effettuare la ricerca solo su alcuni “blocchi” del file.

**Esempi:**

**Un problema degli indici:** sono basati su strutture ordinate e quindi sono poco flessibili in presenza di elevata dinamicità. Per questo motivo all’interno dei moderni DB si utilizzano principalmente strutture ad albero bilanciato (cioè efficienti anche in presenza di aggiornamenti), ogni albero è caratterizzato da:

* **Nodo radice**
* **Diversi nodi intermedi**
* **Nodi foglia**

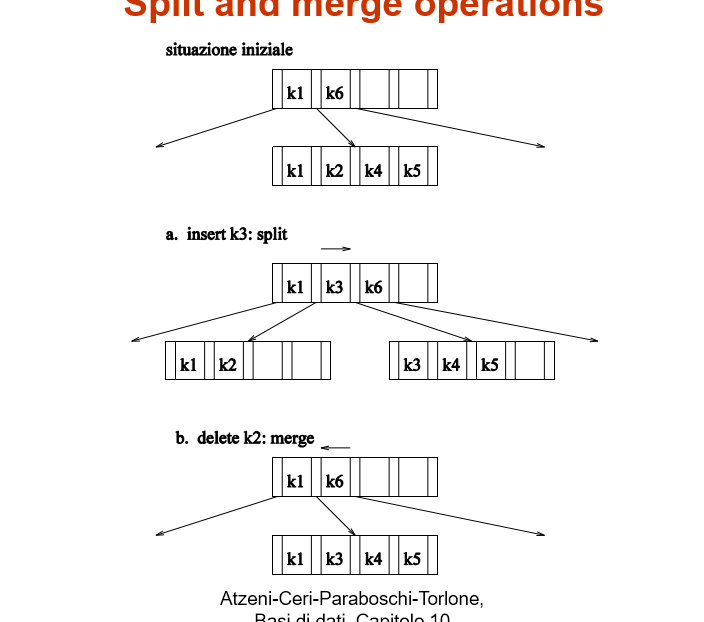
Una caratteristica importante di questi alberi è che devono essere bilanciati, ovvero la lunghezza del cammino che collega un nodo indice ad ‘un qualsiasi nodo foglia deve essere costante.

La struttura tipica di un nodo non foglia è la seguente:

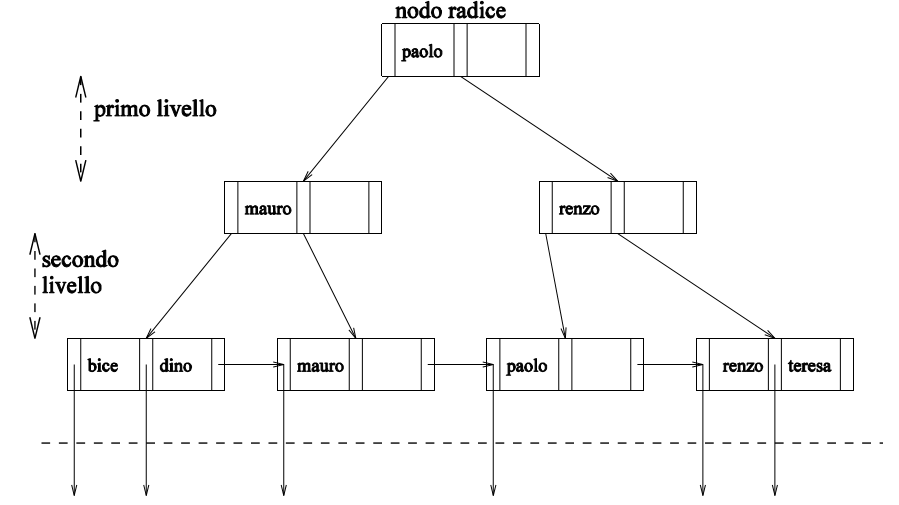
**Operazioni sugli alberi:**

* **Ricerca:** in questo tipo di struttura la ricerca di un valore **V** viene eseguita “seguendo” i puntatori, in particolare:
  + Se **V<k1** si segue il puntatore **P0**
  + Se **V>kF** si segue **PF** e cosi via

Una volta trovata la foglia giusta la ricerca può continuare in due modi a seconda di come l’albero è impostato:

* + Nel caso di **indice primario** nel senso stretto del termine le foglie contengono anche i dati, quindi abbiamo fatto
  + Nel caso di **indice secondario** le foglie contengono solitamente un puntatore ai blocchi della base di dati
* **Inserimento:** se nel blocco foglie è disponibile uno spazio, il nuovo valore viene aggiunto senza problemi, in caso contrario è necessario eseguire un’operazione di **split**, per mantenere l’albero bilanciato.
* **Eliminazione:** operazione molto semplice, inoltre per mantenere un buon grado di riempimento (tra il 50 e il 70%) si effettua un operazione di **merge.**

**Alberi B e B+:** Esistono sostanzialmente due tipi di alberi:

* **B+-tree:** le chiavi compaiono tutte nelle foglie (e quindi quelle nei nodi intermedi sono comunque ripetute nelle foglie), le foglie sono collegate in una lista. Tali alberi sono ottimi per le ricerche su intervalli, per questo motivo sono molto usati nei DBMS**.** Esistono due tipi di B+ tree:
  + **Primario**: le ennuple possono essere contenute nelle foglie.
  + **Secondario**: le foglie contengono puntatori alle ennuple**.**
* **B-tree:** Le chiavi che compaiono nei nodi intermedi non sono ripetute nelle foglie inoltre anche i nodi intermedi contengono ennuple (se primari) o puntatori (se secondari).

