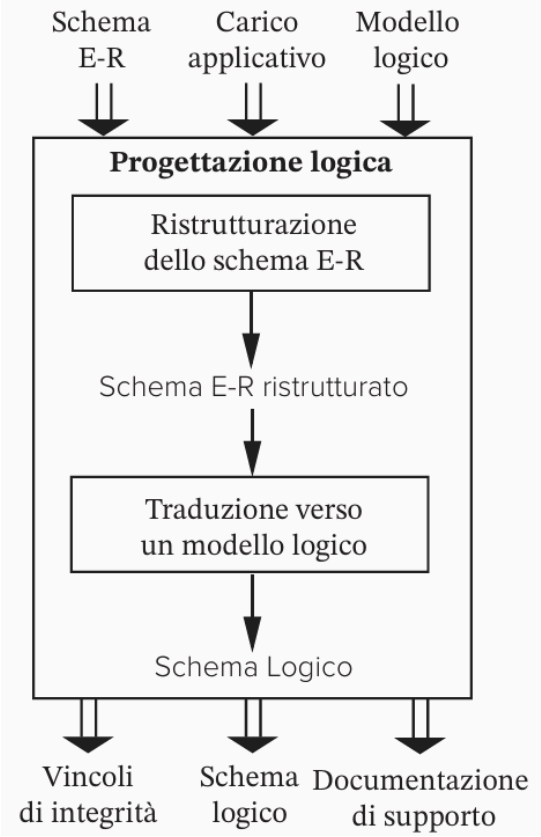
Logical Design

Le attività principali della progettazione logica sono:

- *ristrutturazione dello schema Entità-Relazione*: si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva;
- *traduzione verso il modello logico*: fa riferimento a uno specifico modello logico e può includere un'ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso.
  I dati di ingresso sono lo schema concettuale e il carico applicativo previsto, ovvero la dimensione dei dati e le caratteristiche delle operazioni. Il risultato è uno schema E-R ristrutturato. Questo schema e il modello logico scelto sono i dati di ingresso della seconda fase che produce lo schema logico; in questa fase si fanno verifiche della qualità dello schema ed eventuali ulteriori ottimizzazioni. I prodotti finali della progettazione logica sono lo schema logico finale, i vincoli di integrità definiti su di esso e la relativa documentazione.

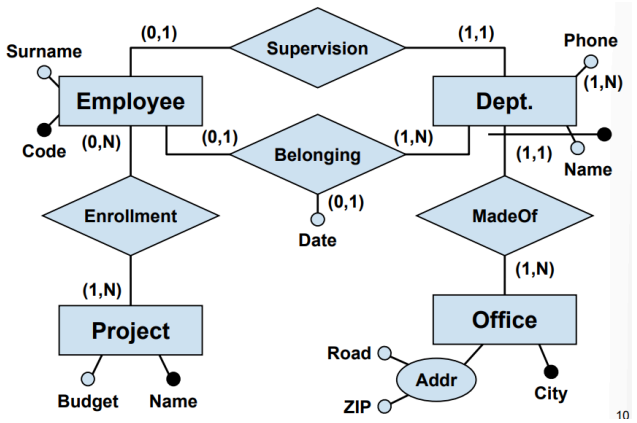


Analisi delle prestazioni

Uno schema E-R può essere modificato per ottimizzare alcuni *indici di prestazione*, chiamati cosi perché non sono valutabili in maniera precisa in sede di progettazione logica in quanto sono dipendenti anche da parametri fisici, ecc..

- *costo di un'operazione*: valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a un'operazione sulla base di dati;
- *occupazione di memoria*: valutato in termini dello spazio di memoria necessario per memorizzare i dati descritti dallo schema.
  Per lo studio di questi parametri serve conoscere:
- *volume dei dati*:

    - numero di occorrenze di ogni entità e associazione dello schema;
    - dimensioni di ciascun attributo (di entità o associazione).
- *caratteristiche delle operazioni*:
  - tipo dell'operazione;
  - frequenza (numero medio di esecuzioni in un certo intervallo di tempo);
  - dati coinvolti.

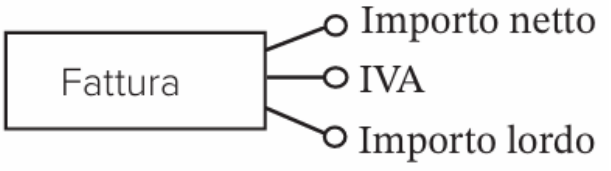


Name	Type	Size
Office	E	10
Dept.	E	80
Employee	E	2'000
Project	E	500
MadeOf	R	80
Belonging	R	1'900
Supervision	R	80
Enrollment	R	6'000

Ristrutturazione di schemi E-R

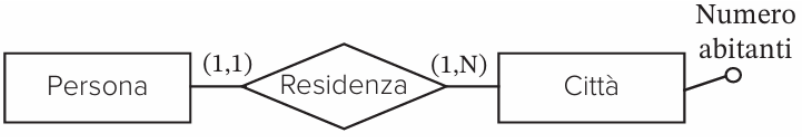
Suddivisa in:

- **analisi delle ridondanze**:
  - si decide se eliminare o mantenere eventuali ridondanze (:= presenza di un dato che può essere derivato da altri dati);
  - casi più frequenti:
    - attributi derivabili da altri attributi della stessa entità (o associazione):



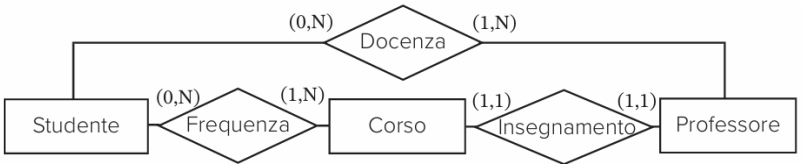
      uno degli attributi è deducibile dagli altri attraverso un'operazione di somma o differenza.
    - attributi derivabili da attributi di altre entità (o associazioni), di solito attraverso funzioni aggregative:

      importo totale è derivabile attraverso l'associazione *composizione* dall'attributo *prezzo*, sommando i prezzi dei prodotti di un acquisto.
    - attributi derivabili da operazioni di conteggio di occorrenze:



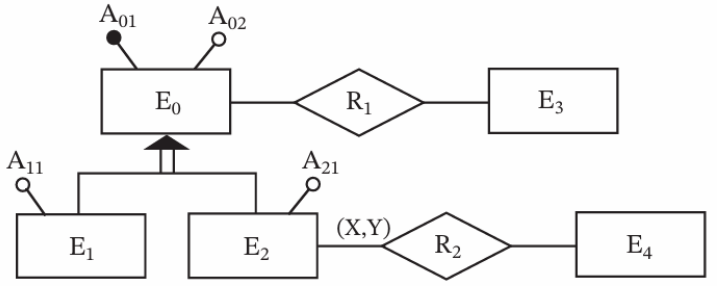
      numero abitanti può essere derivato contando le occorrenze dell'associazione *Residenza* a cui tale città partecipa.

- associazioni derivabili dalla composizione di altre associazioni in presenza di cicli:

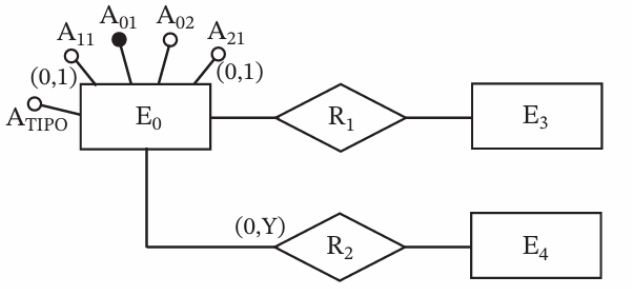


  l'associazione *Docenza* può essere derivata dalle associazioni *Frequenza* e *Insegnamento*.
- La presenza di un dato derivato presenta:
  - vantaggio: riduzione degli accessi necessari per calcolare il dato derivato;
  - svantaggio: maggiore occupazione di memoria e la necessità di effettuare operazioni aggiuntive per mantenere il dato derivato aggiornato.

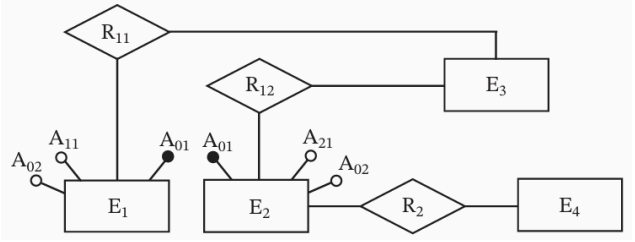
- **eliminazione delle generalizzazioni**:



  - i sistemi per la gestione delle basi di dati non consentono di rappresentare direttamente le generalizzazioni, perciò è necessario rappresentarle mediante entità o associazioni, per farlo esistono tre alternative:
    1. *accorpamento delle figlie della generalizzazione nel genitore*:
       le entità E_1 ed E_2 vengono eliminate e le loro proprietà vengono aggiunte all'entità genitore E_0. Inoltre viene aggiunto un attributo che serve a distinguere il "tipo" di un'occorrenza di E_0, ovvero se apparteneva a E_1 o a E_2 nel caso di generalizzazione totale, o a nessuna di esse nel caso di generalizzazione parziale.

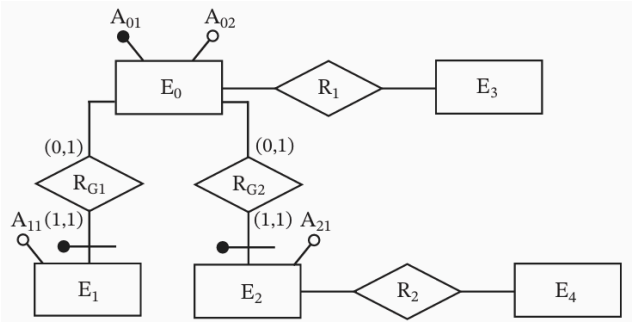


    2. *accorpamento del genitore delle generalizzazione nelle figlie*:
       l'entità genitore E_0 viene eliminata, i suoi attributi, il suo identificatore e le relazioni a cui partecipava vengono ereditate dalle entità figlie.

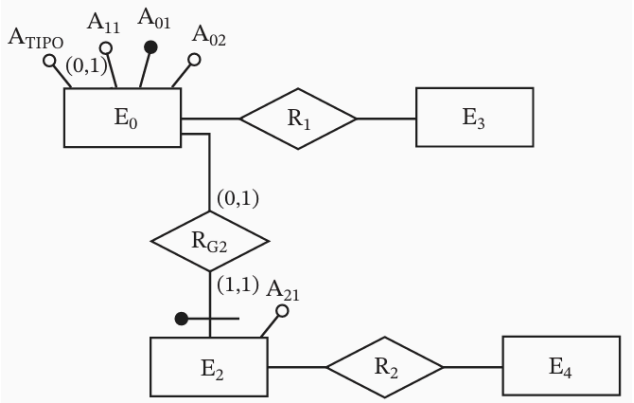


    3. *sostituzione della generalizzazione con associazioni*:
       la generalizzazione si trasforma in due associazioni uno a uno che legano l'entità genitore con le entità figlie. Non ci sono trasferimenti di attributi o associazioni e le entità figlie sono identificate esternamente dall'entità genitore. Vengono aggiunti dei vincoli:

ogni occorrenza di E_0 non può partecipare contemporaneamente a R_{G1} e R_{G2}; se la generalizzazione è totale ogni occorrenza di E_0 deve partecipare obbligatoriamente a una delle due relazioni.

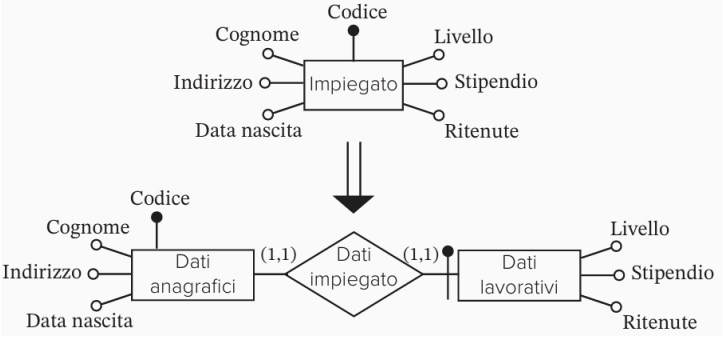


- la scelta tra le varie soluzioni possibili deve essere fatta considerando vantaggi e svantaggi relativamente all'occupazione di memoria e al costo delle operazioni:
  - *scelta 1*:
    - conveniente quando le operazioni non fanno distinzione tra le occorrenze e tra gli attributi di E_0, E_1 e E_2;
    - spreco di memoria per l'eventuale presenza di valori nulli;
    - ci assicura un numero minore di accessi rispetto alle altre soluzioni;
  - *scelta 2*:
    - è possibile solo se la generalizzazione è totale, altrimenti le occorrenze dell'entità genitore che non sono occorrenze delle entità figlie non sarebbero rappresentate;
    - E' conveniente quando ci sono operazioni che si riferiscono solo a una delle due entità figlie;
    - risparmio di memoria rispetto alla scelta (1), perché gli attributi non assumono mai valori nulli;
    - riduzione degli accessi rispetto alla scelta (3) perché non si deve visitare l'entità genitore per accedere agli attributi dei figli.
  - *scelta 3*:
    - è conveniente quando la generalizzazione non è totale e ci sono operazioni che si riferiscono solo a occorrenze di una delle entità figlie o dell'entità genitore;
    - risparmio di memoria rispetto alla scelta (1), per lo stesso motivo di prima;
    - incremento degli accessi per mantenere la consistenza delle occorrenze rispetto ai vincoli introdotti.
- le alternative viste non sono le uniche ammesse, talvolta è possibile utilizzare una combinazione delle tre soluzioni.

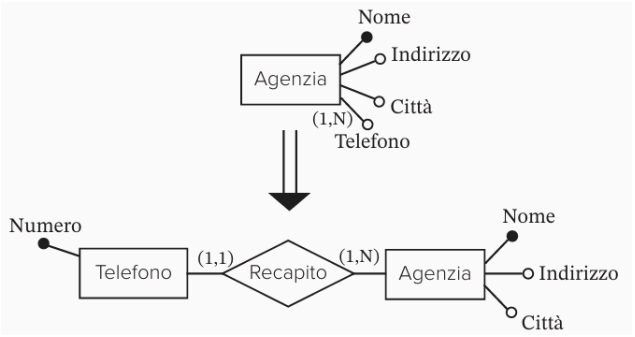


- **partizionamento/accorpamento di entità e associazioni**:

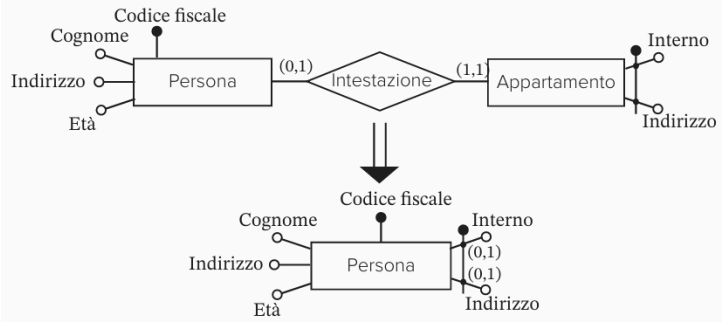
- entità e associazioni possono essere partizionati o accorpati in base a questo principio: gli accessi si riducono
  - separando attributi di uno stesso concetto che vengono acceduti da operazioni diverse;
  - raggruppando attributi di concetti diversi che vengono acceduti dalle stesse operazioni;
- *partizionamenti di entità*:
  - *decomposizione verticale*: si suddivide il concetto operando sui suoi attributi;
    - generano entità con pochi attributi che possono essere tradotte in strutture logiche sulle quali con un solo accesso è possibile recuperare molti dati.



  - *decomposizione orizzontale*: la suddivisione avviene sulle occorrenze dell'entità, può convenire decomporre l'entità in due entità distinte, corrisponde all'introduzione di una generalizzazione a livello logico.
    - effetto collaterale: dover duplicare le associazioni a cui l'entità originaria partecipava;
- *eliminazione di attributi multivalore*:
  - come le generalizzazioni, non sono rappresentabili nel modello relazionale;
  - la ristrutturazione avviene reificando l'attributo multivalore:

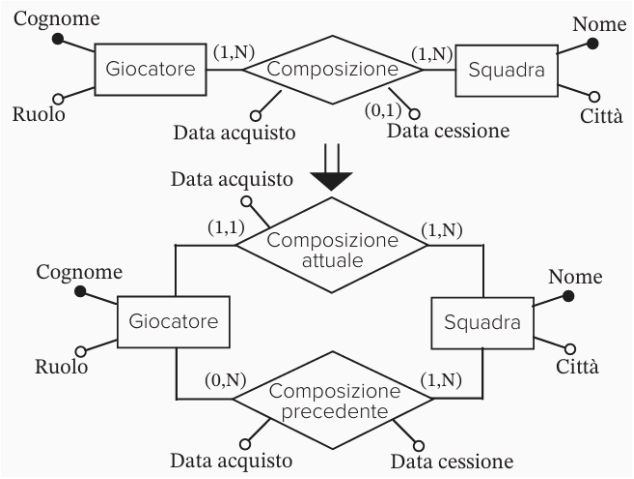


- *accorpamento di entità*:
  - operazione inversa del partizionamento;



  - effetto collaterale: possibile presenza di valori nulli;
  - generalmente di effettuano su associazioni di tipo uno a uno, raramente su associazioni uno a molti, mai su associazioni molti a molti (in quest'ultimo caso generano ridondanze).

- il concetto di partizionamento e accorpamento di entità si può applicare anche sulle associazioni:
  - in alcuni casi può essere utile decomporre un'associazione tra due entità in due (o più) associazioni tra le stesse entità per separare le occorrenze:



  - è possibile accorpare due (o più) associazioni tra le medesime entità (che si riferiscono a due aspetti dello stesso concetto) in un'unica associazione.
- **scelta degli identificatori principali**:
  - essenziale nelle traduzioni verso il modello relazionale:
    - usate per stabilire legami tra dati in relazioni diverse;
    - i sistemi di gestione utilizzano la chiave primaria per la costruzione automatica di *indici*.
  - criteri di decisione:
    - gli attributi con valori nulli non possono essere identificatori principali;
    - un identificatore composto da uno o da pochi attributi è da preferire a quelli costituiti da molti attributi, in quanto:
      - garantisce che gli indici siano di dimensioni ridotte;
      - risparmio di memoria nella realizzazione dei legami logici tra le relazioni;
      - facilita le operazioni di join.
    - un identificatore interno con pochi attributi è preferibile rispetto ad uno esterno, infatti gli identificatori esterni vengono tradotti in chiaviche includono gli identificatori delle entità coinvolte nell'identificazione esterna;
    - un identificatore che viene utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri.
  - se nessuno degli identificatori candidati soddisfa i criteri viene introdotto un nuovo attributo *codice* generato appositamente per identificare le occorrenze delle entità.

Traduzione verso il modello relazionale

Questa seconda fase corrisponde a una traduzione tra modelli di dati diversi: a partire dallo schema E-R ristrutturato (senza generalizzazioni e attributi multivalore, con un solo identificatore) si costruisce uno schema logico *equivalente*, in grado di rappresentare le stesse informazioni.

- **entità e associazioni molti a molti**

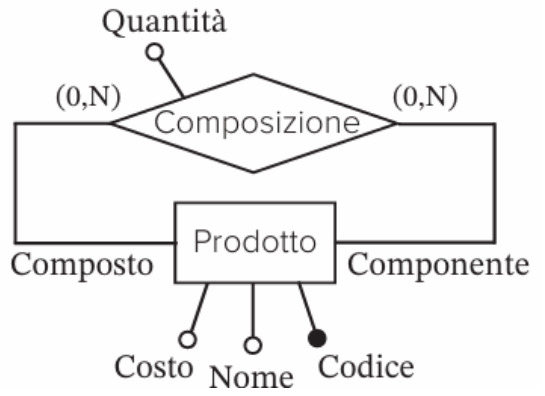
- *Relazioni binarie*:

la sua traduzione nel modello relazionale prevede:

  - per ogni entità:
    - una relazione con lo stesso nome;
    - avente come attributi gli stessi dell'entità;
    - per chiave il suo identificatore.
  - per ogni associazione:
    - una relazione con lo stesso nome;
    - avente per attributi gli stessi dell'associazione;
    - gli identificatori delle entità coinvolte che formano la chiave della relazione.

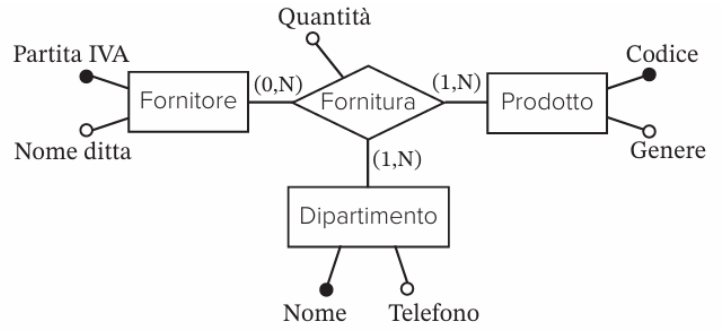
    schema relazionale corrispondente:
    Impiegato(Matricola, *Cognome*, *Stipendio*)
    Progetto(Codice, *Nome*, *Budget*)
    Partecipazione(Matricola, Codice, *DataInizio*)
    Per rendere più comprensibile il significato dello schema è conveniente effettuare alcune ridenominazioni: **Partecipazione**(Impiegato, Progetto, *DataInizio*)
    C'è un vincolo di integrità referenziale tra:
  - Matricola in **Partecipazione** e la chiave di **Impiegato**;
  - Codice in **Partecipazione** e la chiave di **Progetto**;

- *Relazioni ricorsive*:



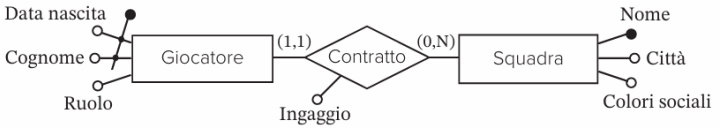
schema relazionale corrispondente:
Prodotto(Codice, *Nome*, *Costo*)
Composizione(Composto, Componente, *Quantità*)
gli attributi Composto e Componente contengono codici di prodotti: il primo ha il secondo come componente.

- *Associazioni con più di due entità*:

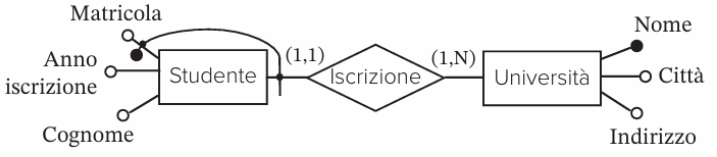


si traducono in maniera analoga alle associazioni binarie:
Fornitore(PartitaIVA, *NomeDitta*)
Prodotto(Codice, *Genere*)
Dipartimento(Nome, *Telefono*)
Fornitura(Fornitore, Prodotto, Dipartimento, *Quantità*)

- **associazioni uno a molti**
  - *associazione binaria*



schema relazionale corrispondente:
Giocatore(Cognome, DataNascita, *Ruolo*)
Squadra(Nome, *Città*,*ColoriSociali*)
Contratto(Giocatore, DataNascitaGiocatore, *NomeSquadra*, *Ingaggio*)
In **Contratto** la chiave è costituita solo dall'identificatore di **Giocatore** perché la sua cardinalità implica che ogni giocatore ha un contratto con una sola squadra. Dal momento che **Giocatore** e **Contratto** hanno la stessa chiave è possibile fonderle in un'unica relazione:
Giocatore(Cognome, DataNascita, *Ruolo*, *NomeSquadra*, *Ingaggio*)
Squadra(Nome, *Città*, *ColoriSociali*)
Con questa soluzione:
    - abbiamo meno relazioni;
    - è possibile avere valori nulli sugli attributi NomeSquadra e Ingaggio.
      Vincolo di integrità referenziale tra:
    - *NomeSquadra* di **Giocatore** e Nome di **Squadra**.
  - *associazioni ternarie*
    L'*entità* che partecipa all'associazione ternaria con cardinalità massima uguale a 1, viene tradotta in una relazione che contiene anche gli identificatori delle altre entità coinvolte nell'associazione. Se l'entità **Prodotto** nell'esempio di associazione ternaria precedente avesse cardinalità (1,1), quindi per ogni prodotto esiste un solo fornitore e un solo dipartimento al quale viene fornito, allora:
    Fornitore(PartitaIVA, *NomeDitta*)
    Dipartimento(Nome, *Telefono*)
    Prodotto(Codice, *Genere*, *Fornitore*, *Dipartimento*, *Quantità*)
    Vincoli di integrità referenziale:
    - tra l'attributo Fornitore della relazione **Prodotto** e l'attributo PartitaIVA di **Fornitore**;
    - tra l'attributo *Dipartimento* della relazione **Prodotto** e l'attributo *Nome* della relazione **Dipartimento**.
- **entità con identificatore esterno**:
  danno luogo a relazioni con chiavi che includono gli identificatori delle entità "identificanti".



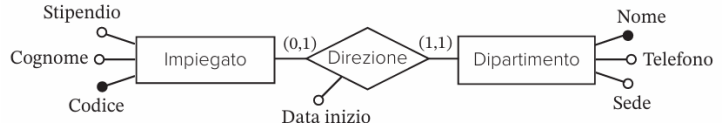
schema relazionale:
Studente(Matricola, NomeUniversità, *Cognome*, *AnnoIscrizione*)
Università(Nome, *Città*, *Indirizzo*)
vincolo di integrità referenziale:

- tra l'attributo NomeUniversità della relazione **Studente** e l'attributo Nome di **Università**
  Rappresentando l'identificatore esterno si rappresenta direttamente anche l'associazione, infatti le entità identificate esternamente partecipano all'associazione sempre con una cardinalità minima e massima pari a 1. Inoltre, questo tipo di traduzione è valido indipendentemente dalla cardinalità con cui l'altra entità partecipa all'associazione.

- **associazioni uno a uno**
  diverse possibilità di traduzione:
  - *partecipazioni obbligatorie per entrambe le entità*

    due possibilità:
    1. **Direttore**(Codice, *Cognome*, *Stipendio*, *DipartimentoDiretto*, *InizioDirezione*)
       Dipartimento(Nome, *Telefono*, *Sede*)
    2. **Direttore**(Codice, *Cognome*, *Stipendio*)
       Dipartimento(Nome, *Telefono*, *Sede*, *Direttore*, *InizioDirezione*)
       E' possibile rappresentare l'associazione in una qualunque delle relazioni che rappresentano le due entità.
  - *partecipazione opzionale per una sola entità*

    una sola soluzione è preferibile rispetto alle altre:
    Impiegato(Codice, *Cognome*, *Stipendio*)
    Dipartimento(Nome, *Telefono*, *Sede*, *Direttore*, *InizioDirezione*)
    in quanto in questo modo non è possibile avere valori nulli.
  - *partecipazione opzionale*
    Impiegato(Codice, *Cognome*, *Stipendio*)
    Dipartimento(Nome, *Telefono*, *Sede*)
    Direzione(Direttore, *Dipartimento*, *DataInizioDirezione*)
    - non presenta mai valori nulli sugli attributi dell'associazione;
    - abbiamo una relazione in più ⇒ rende più complessa la base di dati.
      Questa soluzione è da prendere in considerazione solo se il numero di occorrenze dell'associazione è molto basso rispetto alle occorrenze delle entità che partecipano all'associazione.

Normalization

La "forma normale" è una proprietà di un database relazionale che ne garantisce la qualità. Quando una relazione non è in forma normale:

- presenta ridondanze;
- può avere comportamenti indesiderati durante gli aggiornamenti;
  La **normalizzazione** deve essere usata come una tecnica di verifica per testare il risultato del design della base di dati, non è una metodo per il design del database.
  Esempio:

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20 000	Marte	2000	Tecnico
Verdi	35 000	Giove	15 000	Progettista
Verdi	35 000	Venere	15 000	Progettista
Neri	55 000	Venere	15 000	Direttore
Neri	55 000	Giove	15 000	Consulente
Neri	55 000	Marte	2000	Consulente
Mori	48 000	Marte	2000	Direttore
Mori	48 000	Venere	15 000	Progettista
Bianchi	48 000	Venere	15 000	Progettista
Bianchi	48 000	Giove	15 000	Direttore

  le tuple di questa relazione rispettano queste proprietà:
- lo stipendio di ogni impiegato è unico ed è funzione del solo impiegato, indipendentemente dai progetti cui partecipa;
- il bilancio di ciascun progetto è unico e dipende dal solo progetto, indipendentemente dagli impiegati che partecipano.
  Anomalie:
- **Ridondanza**: il valore dello stipendio di ciascun impiegato è ripetuto in tutte le tuple;
- **Anomalia di aggiornamento**: se lo stipendio di un impiegato varia, è necessario andarne a modificare il valore in tutte le tuple corrispondenti, comporta la necessità di più modifiche contemporaneamente;
- **Anomalia di cancellazione**: se un impiegato interrompe la partecipazione a tutti i progetti senza lasciare l'azienda, tutte le tuple corrispondenti vengono eliminate e non è possibile conservare traccia del suo nome e del suo stipendio;
- **Anomalia di inserimento**: se si hanno informazioni su un nuovo impiegato, non è possibile inserire finché non viene assegnato ad un progetto.
  Queste anomalie sono causate dal fatto che è stata utilizzata una sola relazione per rappresentare relazioni che riuniscono concetti fra loro disomogenei.

Dipendenze Funzionali

Sono un particolare vincolo di integrità che descrive legami di tipo funzionale tra gli attributi di una relazione. Formalizzato:

> **Def**
>
> Data una relazione r su uno schema $R(X)$ e due sottoinsiemi non vuoti Y e Z di X, esiste su r una dipendenza funzionale tra Y e Z se, per ogni coppia di tuple t_1 e t_2 di r aventi gli stessi valori sugli

attributi Y, risulta che t_1 e t_2 hanno gli stessi valori anche sugli attributi di Z.
Una dipendenza funzionale tra Y e Z viene generalmente indicata con $Y \rightarrow Z$.
Viene associata ad uno schema: una relazione su quello schema verrà considerata corretta se soddisfa tale dipendenza funzionale.

Nel nostro esempio: *Impiegato* → *Stipendio*
Progetto → *Bilancio*
Impiegato Progetto → *Funzione*

Osservazioni:

se l'insieme Z è composto dagli attributi $A_1, A_2, ..., A_k$, allora una relazione soddisfa $Y \rightarrow Z \Leftrightarrow$ soddisfa tutte le k dipendenze $Y \rightarrow A_1$, $Y \rightarrow A_2$, ..., $Y \rightarrow A_k$.

Diremo che una dipendenza funzionale $Y \rightarrow A$ è ***non banale*** se A non compare tra gli attributi di Y.
Impiegato Progetto → *Funzione* è una dipendenza *banale*.

Osservazione sulle dipendenze funzionali e il legame con la chiave:

Se prendiamo una chiave K di una relazione r, si verifica facilmente che esiste una dipendenza funzionale tra K e ogni altro attributo dello schema di r; infatti, per definizione stessa di vincolo di chiave, non possono esistere due tuple con gli stessi valori su K.

Nel nostro esempio:

- le prime due dipendenze funzionali non sono chiavi e causano anomalie;
- la terza dipendenza funzionale (*Impiegato Progetto*) è una chiave e non causa anomalie;

Forma normale di Boyce-Codd

Def

Una relazione r è in *forma normale di Boyce-Codd* se per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, X contiene una chiave K di r, cioè X è superchiave per r.
Anomalie e ridondanze non si presentano per relazioni in forma normale di Boyce-Codd, perchè i concetti indipendenti sono separati, uno per relazione.

Decomposizione in BCNF

Data una relazione che non soddisfa la BCNF è possibile, in molti casi, sostituirla con due o più relazioni normalizzate attraverso un processo detto di ***normalizzazione***: se una relazione rappresenta più concetti indipendenti, allora va decomposta in relazioni più, piccole, una per ogni concetto.

Impiegato	Stipendio
Rossi	20 000
Verdi	35 000
Neri	55 000
Mori	48 000
Bianchi	48 000

Progetto	Bilancio
Marte	2000
Giove	15 000
Venere	15 000

Impiegato	Progetto	Funzione
Rossi	Marte	Tecnico
Verdi	Giove	Progettista
Verdi	Venere	Progettista
Neri	Venere	Direttore
Neri	Giove	Consulente
Neri	Marte	Consulente
Mori	Marte	Direttore
Mori	Venere	Progettista
Bianchi	Venere	Progettista
Bianchi	Giove	Direttore

Nell'esempio vengono costruite delle nuove relazioni in modo che a ciascuna dipendenza corrisponda una diversa relazione la cui chiave è proprio il primo membro della dipendenza stessa.
In molti casi, la decomposizione può essere effettuata producendo tante relazioni quante sono le dipendenze funzionali definite; in generale, le dipendenze possono avere una struttura complessa: può non essere necessario, o possibile, basare la decomposizione su tutte le dipendenze e può essere difficile individuare quelle su cui si deve basare la decomposizione.

Employee	Project	Office
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan

Employee → Office

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

Project → Office

Project	Office
Mars	Rome
Jupiter	Milan
Venus	Milan
Saturn	Milan

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

⋈

Office	Project
Rome	Mars
Milan	Jupiter
Milan	Venus
Milan	Saturn

=

Employee	Office	Project
Jones	Rome	Mars
Smith	Milan	Jupiter
Smith	Milan	Venus
Smith	**Milan**	**Saturn**
White	**Milan**	**Jupiter**
White	Milan	Saturn
White	Milan	Venus

≠

Employee	Office	Project
Jones	Rome	Mars
Smith	Milan	Jupiter
Smith	Milan	Venus
White	Milan	Saturn
White	Milan	Venus

DIFFERENT FROM THE ORIGINAL RELATION!

Decomposizione senza perdita

Def

Caso Generale: data una relazione r su un insieme di attributi X, se X_1 e X_2 sono due sottoinsiemi di X la cui unione sia pari a X stesso, allora il join delle due relazioni ottenute per proiezione da r su X_1 e X_2, rispettivamente, è una relazione che contiene tutte le tuple di r, più eventualmente altre che possiamo chiamare "spurie". Diciamo che r si *decompone senza perdita* su X_1 e X_2 se il join delle due proiezioni è uguale a r stessa (cioè non contiene tuple spurie).

Condizione per la decomposizione senza perdita:

Sia r una relazione su X e siano X_1 e X_2 sottoinsiemi di X tali che $X_1 \cup X_2 = X$;
inoltre, sia $X_0 = X_1 \cap X_2$;
allora: r si decompone senza perdita su X_1 e X_2 se soddisfa la dipendenza funzionale $X_0 \rightarrow X_1$, oppure la dipendenza funzionale $X_0 \rightarrow X_2$.
In altre parole, r si decompone senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazione è chiave per almeno una delle relazioni composte.

Conservazione delle dipendenze

Per garantire che tutte le dipendenze funzionali dello schema originale siano ancora rispettate nella decomposizione è necessario che: ogni dipendenza funzionale dello schema originale coinvolga attributi che compaiono tutti in almeno uno degli schemi della decomposizione; questo assicura che i vincoli e le relazioni tra i dati originali rimangano validi anche dopo la decomposizione.
Esempio:

- schema originale: **R**(A, B, C)
- dipendenza funzionale: A → B, ovvero il valore dell'attributo A determina univocamente il valore dell'attributo B
- decomposizione:
  1. $\mathbf{R}_1$(A, B)
  2. $\mathbf{R}_2$(A, C)
- verifica:
  - dopo la decomposizione lo schema $\mathbf{R}_1$(A, B) conserva la dipendenza A → B perché entrambi gli attributi A e B sono inclusi in $\mathbf{R}_1$.
  ⇒ la decomposizione conserva le dipendenze funzionali dello schema originale.

Qualità delle decomposizioni

- La **decomposizione senza perdita** garantisce che le informazioni nella relazione originaria siano ricostruibili con precisione;
- la **conservazione delle dipendenze** garantisce che le relazioni decomposte hanno la stessa capacità della relazione originaria di rappresentare i vincoli di integrità e quindi di rilevare aggiornamenti illeciti: a ogni aggiornamento lecito sulla relazione originaria corrisponde un aggiornamento lecito sulle relazioni decomposte.

Terza forma normale

- **limitazioni BCNF**:

Chief	Project	Office
Smith	Mars	Rome
Johnson	Jupiter	Milan
Johnson	Mars	Milan
White	Saturn	Milan
White	Venus	Milan

Project Office → Chief

Chief → Office

la relazione non è in BCNF perché il primo membro della dipendenza Chief → Office non è superchiave. Inoltre non è possibile decomporre bene questa relazione in quanto la dipendenza **Project Office → Chief** coinvolge tutti gli attributi.
⇒ Talvolta la forma normale di Boyce-Codd non è raggiungibile.

Terza Forma Normale:

Diciamo che una relazione *r* è in *terza forma normale* se, per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r;
- A appartiene ad almeno una chiave di r.

- la BCNF è più forte della 3NF (la 3NF ammette relazioni con anomalie);
- 3NF può essere sempre raggiunta;
- Se una relazione ha una sola chiave, è in BCNF se e solo se è in 3NF.
  Infatti, la dipendenza **Project Office → Chief** ha come primo membro una chiave della relazione, mentre **Chief → Office**, pur non contenendo una chiave al primo membro, ha un unico attributo a secondo membro che fa parte della chiave **Project → Office**.

Decomposizione in terza forma normale

Una relazione che non soddisfa la terza forma normale si decompone in relazioni ottenute per proiezione sugli attributi corrispondenti alle dipendenze funzionali, con l'accortezza di mantenere sempre una relazione che contiene una chiave della relazione originaria.

1. identificare le dipendenze funzionali nello schema
2. decomporre lo schema originale in più schemi, in modo che ogni schema soddisfi i requisiti della 3NF;
3. assicurarsi che la decomposizione conservi le dipendenze e che sia possibile ricostruire lo schema originale usando i sottoschemi (proprietà di lossless join).

Esempio:
R(A, B, C)
dipendenze funzionali sono:

4. A → B
5. B → C

dove A è la chiave primaria;

- la dipendenza B → C crea una dipendenza transitiva:
  - A → B e B → C ⇒ A → C, ma C dipende indirettamente da A tramite B
  difatti questo schema non è 3NF:
- creiamo due schemi:
  - R_1 (A, B): per rappresentare A → B
  - R_2 (B, C): per rappresentare B → C
- verifica:
  - schema R_1 (A, B): È in 3NF, perché B dipende direttamente dalla chiave primaria A;
  - Schema R_2 (B, C): È in 3NF, perché C dipende direttamente dalla chiave primaria B.
    Unendo R_1 e R_2, possiamo ricostruire lo schema originale senza perdita di dati.

Teoria delle dipendenze

Data una relazione e un insieme di dipendenze funzionali su di essa, generare una decomposizione di tale relazione che contenga solo le relazioni in forma normale che soddisfi le proprietà della decomposizione di cui abbiamo già parlato:

- decomposizione senza perdita;
- preservazione delle dipendenze.

> **Def**
> Diciamo che un insieme di dipendenze funzionali *F implica* un'altra dipendenza *f* se ogni relazione che soddisfa tutte le dipendenze in *F* soddisfa anche *f*.

Esempio: A → B, B → C ⇒ A → C

> **Chiusura di un insieme di attributi**
>
> Siano dati uno schema di relazione R(U) e un insieme di dipendenze funzionali F definite sugli attributi in U. Sia X un insieme di attributi contenuti in U (cioè $X \subseteq U$); la *chiusura* di X rispetto a F, indicata con X_F^+, è l'insieme degli attributi che dipendono funzionalmente da X (esplicitamente o implicitamente):
> $X_F^+ = \{A \mid A \in U \text{ e } F \text{ implica } X \to A\}$
> Se vogliamo vedere se X → A è implicata da F, basta vedere se A appartiene a X_F^+, a patto di saper calcolare X_F^+.

Esiste un algoritmo per il calcolo di X_F^+:
Input: un insieme X di attributi e un insieme F di dipendenze.
Output: un insieme X_P di attributi.

1. Inizializziamo X_P con l'insieme di input X.
2. Esaminiamo le dipendenze in F; se esiste una dipendenza $Y \rightarrow A$ con $Y \subseteq X_P$ e $A \notin X_P$ allora aggiungiamo A a X_P.
3. Ripetiamo il passo 2 fino al momento in cui non vi sono ulteriori attributi che possono essere aggiunti a X_P.

   Il concetto di chiusura X_F^+ è utile anche per formalizzare il legame fra il concetto di dipendenza funzionale e quello di chiave:

   > **Def**
   > un insieme di attributi K è chiave per uno schema di relazione R(U) su cui è definito un insieme di dipendenze funzionali F se F implica $K \rightarrow U$. Di conseguenza, l'algoritmo mostrato può essere utilizzato per verificare se un insieme è chiave.

Un insieme F è:

- *non ridondante* se non esiste dipendenza $f \in F$ tale che $F - \{f\}$ implica f;
- *ridotto* se è non ridondante e non esiste un insieme F' equivalente a F ottenuto eliminando attributi dai primi membri di una o più dipendenze di F.
  Esempi:
- $F_1 = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C\}$
- $F_2 = \{A \rightarrow B, AB \rightarrow C\}$
- $F_3 = \{A \rightarrow B, A \rightarrow C\}$
  $\Rightarrow F_1$ è ridondante, perché $\{A \rightarrow B, AB \rightarrow C\}$ implica $A \rightarrow C$; F_1 è equivalente a F_2;
  $\Rightarrow F_2$ è non ridondante ma non è ridotto, perché B può essere eliminato dal primo membro della seconda dipendenza: F_2 è equivalente a F_3;
  $\Rightarrow F_3$ è ridotto.
  Due insiemi di dipendenze funzionali F_1 e F_2 sono equivalenti se F_1 implica ogni dipendenza in F_2 e viceversa.
  Se due insiemi sono equivalenti diciamo che sono uno copertura del altro.

Calcolare la copertura minima

Per trovare una copertura minima non ridondante esaminiamo ripetutamente le dipendenze dell'insieme dato, eliminando quelle implicate da altre, si procede in tre passi:

1. sostituiamo l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi;
2. eliminiamo le dipendenze ridondanti;
3. per ogni dipendenza verifichiamo se esistono attributi eliminabili dal primo membro: se F è l'insieme corrente, per ogni dipendenza $Y \rightarrow A \in F$, verifichiamo se esiste $Y \subseteq X$ tale che F è equivalente a $F - \{X \rightarrow A\} \cup \{Y \rightarrow A\}$.

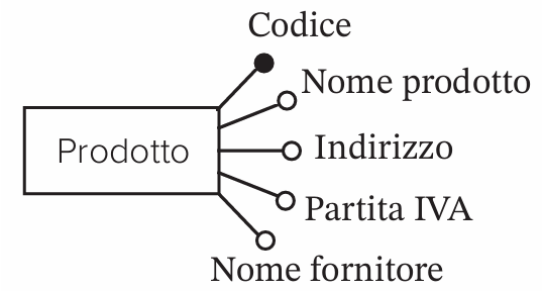
Algoritmo di sintesi per la terza forma normale

Def

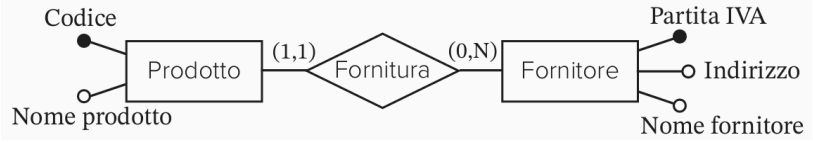
Uno schema di relazione R(U) con l'insieme di dipendenze F è in *terza forma normale* se, per ogni dipendenza funzionale (non banale) $X \rightarrow A \in F$, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di *r*: cioè $X_F^+ = U$;
- A è contenuto in almeno una chiave di *r*: esiste un insieme di attributi $K \subseteq U$ tale che $K_F^+ = U$ e $(K - A)_F^+ \subset U$.

Verifiche di normalizzazione sulle entità



- Partita IVA → NomeFornitore, Indirizzo
  Tutti gli attributi dipendono funzionalmente da Codice, ovvero l'identificatore di Prodotto.
  ⇒ l'entità viola la terza forma normale perché la dipendenza *Partita IVA → NomeFornitore, Indirizzo* ha un primo membro che non contiene l'identificatore e un secondo membro composto da attributi che non fanno parte della chiave.
  Decomposizione:

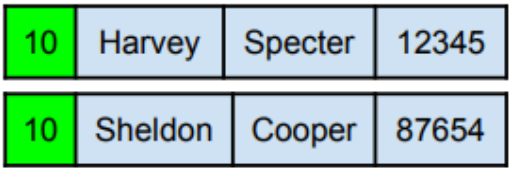


Indexes & B+trees

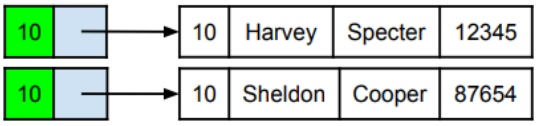
Indici

L'indicizzazione è una tecnica di ottimizzazione utilizzata per velocizzare le interrogazioni, gli indici sono una struttura dati che contiene informazioni complementari che supportano un accesso efficiente ai dati.
La chiave di ricerca è definita utilizzando alcuni attributi; le chiavi di ricerca non sono chiavi primarie, infatti la stessa chiave può contenere più valori.
Un indice è una coppia *<key, label>* e supporta il recupero di tutte le etichette con un dato valore K in modo efficiente.
Le *etichette* (labels) possono essere:

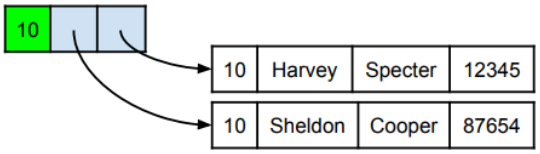
1. il dato stesso;



2. l'identificatore del dato (RID) con il valore K della chiave;



3. una lista di identificatori dello stesso valore con chiave K;



   La rappresentazione delle etichette è indipendente dal metodo di ricerca.

Osservazioni:

- in una base di dati, è possibile avere al massimo un solo indice sui dati utilizzando la prima rappresentazione;
- utilizzando la prima rappresentazione, la dimensione dell'indice è la stessa dei dati;
- la stessa chiave di ricerca può contenere più valori;
- la terza rappresentazione è la soluzione più compatta, ma le etichette hanno dimensioni variabili.

In SQL:

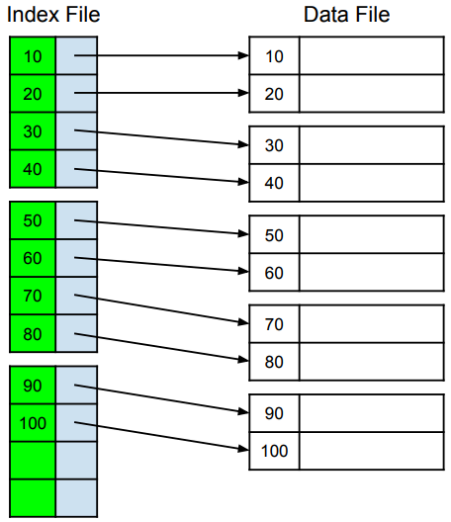
```
//per creare indici
CREATE [UNIQUE] INDEX IndexName ON Table(AttributeList)

//per eliminare un indice
DROP INDEX IndexName
```

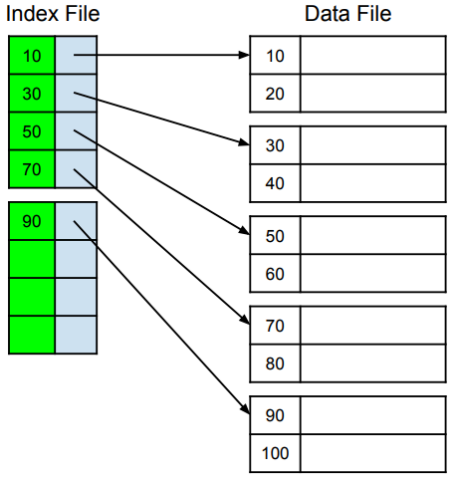
Classificazione

- **indice primario**:
  - è un indice basato su un insieme di attributi che include la chiave primaria;
  - i dati sono ordinati in base a questi attributi;
  - se l'indice non è basato sulla chiave primaria, allora è un **indice secondario**.
- **indice denso**:
  - ogni valore chiave di ricerca nel file dei dati ha almeno una voce corrispondente nell'indice;
  - se non tutti i valori chiave di ricerca sono rappresentati, allora l'indice è **sparso**.
- **indice clusterizzato**:
  - l'ordine dei record nel file dati corrisponde (o è simile) all'ordine delle etichette (chiavi) nell'indice;
  - se l'ordine dei record non corrisponde, allora l'indice è **non clusterizzato**.

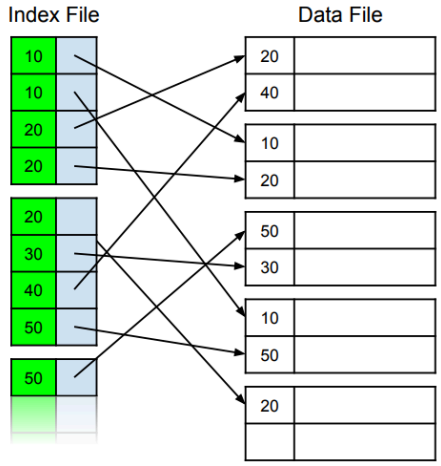
  Esempi:
- indice clusterizzato e denso:



- indice sparso e clusterizzato:



- indice secondario, denso e non clusterizzato



  gli indici non clusterizzati danno meno efficienza nell'accesso ai dati (ad esempio tre record con lo stesso valore sono memorizzati in tre blocchi diversi).

B+trees

Le strutture ad albero dinamiche di tipo B+trees (un tipo speciale di B-alberi), sono le più frequentemente usate nei DBMS relazionali per la realizzazione degli indici.

- Ogni albero è caratterizzato da un nodo radice, vari nodi intermedi e vari nodi foglia;
- ogni nodo ha un numero di discendenti che dipende dall'ampiezza della pagina;
- gli alberi sono **bilanciati**, ovvero la lunghezza di un cammino che collega il nodo radice a un qualunque nodo foglia è costante; in questo modo il tempo di accesso alle informazioni contenute nell'albero è lo stesso per tutte le foglie ed è pari alla profondità dell'albero.

Regole:

- le chiavi nei nodi foglia sono copie delle chiavi del data file. Queste chiavi sono distribuite tra le foglie in modo ordinato, da sinistra a destra.
- alla radice, ci sono almeno due puntatori utilizzati (con almeno due record di dati nel file). Tutti i puntatori puntano ai blocchi del livello sottostante;
- in presenza di n chiavi, bisogna avere n+1 puntatori;
- In un nodo interno, tutti i puntatori utilizzati puntano a blocchi al livello immediatamente inferiore e almeno $\lceil (n+1)/2 \rceil$ devono essere utilizzati;
- in una foglia, l'ultimo puntatore punta al blocco foglia successivo a destra. Tra gli altri puntatori in un blocco foglia, almeno $\lfloor (n+1)/2 \rfloor$ di essi sono utilizzati e puntano a un record di dati.