

SELECT Distinct città  
 FROM Clienti  
 ↓  

CS
RC

→ La Distinct si applica  
 di default a tutti gli  
 attributi passati

## RELAZIONE

La relazione ha due accezioni :

- Schema di relazione → nome di relazione seguito da uno schema di attributi ciascuno dei quali ha un proprio dominio.  
es. NomeRelazione ( $A_1; D_1, A_2; D_2 \dots$ )
- Istanza dello schema → dato uno schema di relazione, un'istanza dello schema è un sottoinsieme del prodotto cartesiano dei domini ovvero  $C \subseteq D_1 \times D_2 \dots$   
SQL utilizza una concezione di istanza che include duplicati

Più schemi di relazioni sono detti schemi di database mentre istanze di database sono un insieme di istanze di schemi relativi ma per ogni schema di relazione presente nello schema di database

GAIA BERTOLINO

Viavolo di chiave → esso è una espressione del tipo  $k : B_1, \dots, B_n$  dove  $B_i$  è un sottoinsieme delle colonne  $A_1, \dots, A_n$ . Questo viavolo impone una verifica che verrà operata sulle istanze.  
Una istanza soddisfa il viavolo dato se non esistono più tuple dell' istanza che coincidono in tutti i campi della chiave.

## DOMANDE D'ESAME

- 1) Definizione di chiave nel modello relazionale
- 2) Definizione di chiave esterna
- 3) Definizione di dipendenza funzionale

Natural join → Una join senza argomenti è detta natural join e le condizioni sono implicite

Le condizioni sono implicite

Equi join → Le condizioni sono di ugualanza

Outer join → Le condizioni sono di non ugualanza

Left outer join → è una join che ricopia anche le tuple della tabella sinistra che non hanno corrispondenze a destra e popola i suoi campi a destra con valori null.

Si indica con  $\Delta L$  ed esiste anche la right outer join  $\Delta R$  e la full outer join  $\Delta F$

Semi join → è una join che proietta solo attributi di A

VINCOLO DI CHECK →

```
CREATE TABLE t AS
(
    eta NUMBER(3) CHECK >= 0,
    ...
)
```

## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## Dipendenze funzionali

venerdì 28 gennaio 2022 12:31

La dipendenza funzionale è un vincolo sul database relazionale. È specificata su uno schema di relazione e afferma quali istanze dello schema sono valide.

Se ad esempio ho una Tabella di CAP e città, per evitare che ci siano più città con uno stesso CAP si può impostare un vincolo di dipendenza funzionale.

Ad esempio posso definire che se due tuple hanno lo stesso valore di CAP allora hanno lo stesso valore di comune.

Esa si esprime come CAP → CITTÀ

Lo schema generale di una dipendenza funzionale sarà:

$X \rightarrow Y$  dove  $X$  e  $Y$  sono due insiemi di attributi  
dello schema di relazioni su cui è definita  
la dipendenza funzionale

Le dipendenze possono essere costituite da più elementi.

Ad esempio, l'IVA di un farmaco dipende dallo stato e dalla categoria:

STATO, CATEGORIA → IVA

$\neq$   
 $\left\{ \begin{array}{l} \text{STATO} \rightarrow \text{IVA} \\ \text{CATEGORIA} \rightarrow \text{IVA} \end{array} \right.$  } ciò permette che in uno stato a più  
categorie corrispondano IVA diverse

Dunque la parte sinistra di una dipendenza non può essere spezzata  
mentre la parte destra sì.

es. CF → nome, città

$\equiv$

$\left\{ \begin{array}{l} \text{CF} \rightarrow \text{nome} \\ \text{CF} \rightarrow \text{città} \end{array} \right.$

Una dipendenza con a destra un solo attributo è detta in forma canonica

## Definizione formale:

Sia dato uno schema di relazione  $R(A_1, \dots, A_n)$ . Una dipendenza funzionale è una espressione del tipo  $x \rightarrow y$  con  $x, y \subseteq A_1, \dots, A_n$

Una istanza  $t$  di  $R$  è consistente rispetto a  $x \rightarrow y$  se, essendo

$$x = \{x_1, \dots, x_n\}, y = \{y_1, \dots, y_n\} : \forall t_1, t_2 \in t \quad t_1[x_1] = t_2[x_1] \dots \\ \Rightarrow t_1[y_1] = t_2[y_1] \dots$$

Dunque se due tuple coincidono in  $X$  allora devono coincidere in  $Y$ .

Dato uno schema di relazione è possibile esprimere il vincolo di chiave come dipendenza funzionale

es. Esame (matricola, nome, data, materia, docente, voto)

↓ DIPENDENZA

Matricola, matricola  $\rightarrow$  nome, data, docente, voto  
↳ chiave

Matricola  $\rightarrow$  nome

↳ risolve il problema di avere una matricola e più nomi

Materia  $\rightarrow$  docente

↳ una materia è insegnata sempre dallo stesso docente

→ INSIEME DI DIPENDENZE

Una istanza è **CONSISTENTE** verso un insieme di dipendenze se lo è per ogni dipendenza

In SQL le dipendenze funzionali vengono definite tramite del codice.

In uno schema vi è la presenza di ridondanza nel caso in cui siano presenti dipendenze con parte sinistra che non è una chiave. La presenza di ridondanza può causare incostituzionalità per cui, tramite la dipendenza, è possibile strutturare gli schemi per eliminarla.

## RELAZIONE BOYCE-CODD

Dato uno schema del tipo

$$\left\{ \begin{array}{l} R(A_1, \dots, A_n) \\ F. \text{ insieme di dep. funzionali: } R \end{array} \right.$$

Dato uno schema del tipo

$$\left\{ \begin{array}{l} R(A_1, \dots, A_n) \\ F. insieme di dep. funzionali R \end{array} \right.$$

Si dice che  $R$  è una **forma normale** di Boyce Codd (BCNF) se  $\forall X \rightarrow Y$   
non triviale l'insieme  $X$  è una superchiave (un sottinsieme di una  
chiave).

Ciò significa che non ci sono ridondanze.



Dunque una relazione è in BCNF quando per ogni relazione di dipendenza  
funzionale non triviale la parte sinistra contiene una chiave.

### Esempio

chiave ipotizzando che si registrino  
solo le promozioni

Esame (matricola, nome, materia, docente, data, voto)

$\left\{ \begin{array}{l} \text{matricola} \rightarrow \text{voto} \\ \text{materia} \rightarrow \text{docente} \\ \text{matricola}, \text{materia} \rightarrow \text{nome, docente, data, voto} \end{array} \right. \quad ? \quad \begin{array}{l} \text{Da queste due riesco ad} \\ \text{ottenere la definizione della chiave} \end{array}$

Ragionando sulle dipendenze funzionali posso ricavare la chiave  
definendo quali sono gli attributi che dipendono dagli altri e  
quali sono "indipendenti" ovvero quelli da cui dipendono tutti gli  
altri.

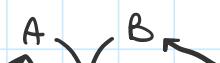
es.  $R(A B C D E)$

$$\left\{ \begin{array}{l} AB \rightarrow C \\ CD \rightarrow E \\ E \rightarrow AB \end{array} \right.$$

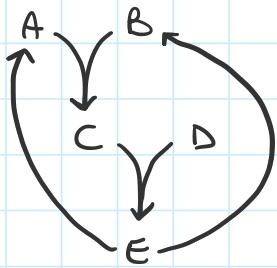
Keys  $\{CD, DE, ABD\}$

$CD \rightarrow EAB$
$DE \rightarrow ABC$
$ABD \rightarrow CE$

Per ottenere le chiavi è possibile usare un grafo dove  
indico con delle frecce le dipendenze funzionali



Osservo e combino gli attributi.



Osservo e combino gli attributi delle dipendenze e vedo se, partendo da essi, riesco a coprire tutti gli altri attributi. In caso affermativo allora ottengo una **SUPERCHIAVE**

L'insieme di attributi in cui due tuple devono coincidere avendo gli attributi a destra di una dipendenza è detta **CHIUSURA**

$$\text{es. } (AE)^+ = AEBC$$

Le chiavi candidate si calcolano dunque analizzando tutte le combinazioni di attributi e verifico di coprire tutto il grafo.

Inoltre, è possibile semplificare la ridondanza nei vincoli!

Esempio (matricola, nome, materia, docente, data, voto)

$$\left\{ \begin{array}{l} \text{matricola} \rightarrow \text{nome} \\ \text{materia} \rightarrow \text{docente} \\ \text{matricola}, \text{materia} \rightarrow \text{voto} \end{array} \right.$$

Verificati preliminariamente

Esempio di ridondanza:

$$\left\{ \begin{array}{l} A \rightarrow B \\ B \rightarrow C \\ A \cancel{\rightarrow} D \end{array} \right.$$

$\hookrightarrow A \rightarrow C$  dunque è ridondante

Esempio

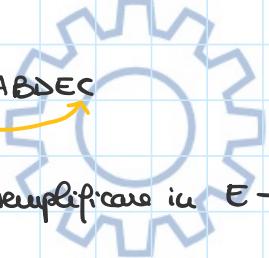
$$R(A B C D E)$$

$$\left\{ \begin{array}{l} AB \rightarrow C \\ CD \rightarrow E \\ E \rightarrow ABD \\ D \rightarrow B \\ EC \rightarrow D \end{array} \right.$$

Per semplificare vado ad analizzare le chiusure degli attributi:

1)  $AB \rightarrow C$      $A^+ = A$     dunque la parte sinistra non può essere semplificata  
 $B^+ = B$

2)  $CD \rightarrow E$      $C^+ = C$     dunque non posso ridurla  
 $D^+ = BD$

3)  $EC \rightarrow D$      $E^+ = ABDE = ABDEC$   
  
dunque  $E \rightarrow C$  e posso semplificare in  $E \rightarrow D$

Il sistema diventa:

$$\left\{ \begin{array}{l} AB \rightarrow C \\ CD \rightarrow E \\ E \rightarrow ABD \\ D \rightarrow B \\ E \rightarrow D \end{array} \right.$$

## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

Provo a semplificare le parti destre:

1)  $AB \rightarrow C$      $(AB)^+ = AB$     dunque non posso eliminarla

2)  $CD \rightarrow E$      $(CD)^+ = CDB$     dunque non posso eliminarla

3)  $E \rightarrow ABD$     (A)  $E \rightarrow BD$  non è eliminabile  
(B)  $E \rightarrow AD$      $E^+ = EADB$     dunque B è ridondante

↓ opera una modifica

lavoro sul nuovo sistema

Nuovo sistema

Nuovo sistema

↓ operata una modifica  
lavoro sul nuovo sistema

$$\left\{ \begin{array}{l} AB \rightarrow C \\ CD \rightarrow E \\ E \rightarrow AD \\ D \rightarrow B \\ E \rightarrow D \end{array} \right.$$

(D)  $E \rightarrow A$  non superflua

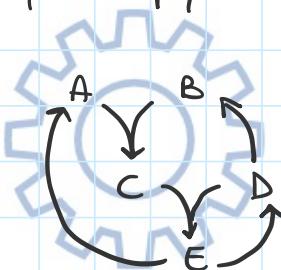
4)  $D \rightarrow B \quad D^+ = D$

5)  $E \rightarrow D \quad E^+ = ADE \text{ dunque è superflua}$

Nuovo sistema :

$$\left\{ \begin{array}{l} AB \rightarrow C \\ CD \rightarrow E \\ E \rightarrow AD \\ D \rightarrow B \end{array} \right.$$

Grafo



## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

$$E^+ = A D B C \rightarrow \text{superflue}$$

$$(CD)^+ = E A D B \rightarrow \text{chiave}$$

$$(AD)^+ = A D B C E \rightarrow \text{chiave}$$

} Keys : { E, CD, AD }

Una chiave minima ha un numero minimo di componenti vera è tale se ha sottoinsiemi che sono chiavi.

## DECOPOLISZIONE IN BCNF

Scoposizione in forma BCNF

## Esempio 1

Oraio (Aula, Ora, Materia, Docente)

Aula	Ora	Materia	Docente
A1	L9	BD	F
A2	L9	ED	P
A3	M9	BD	F

I vincoli saranno:

$$\left\{ \begin{array}{l} \text{Materia} \rightarrow \text{Aula, Docente} \\ \text{Aula, Ora} \rightarrow \text{Materia} \end{array} \right.$$

Grafo:



keys  $\{ \text{Aula Ora, Materia Ora} \}$

## APPUNTI DI INGEGNERIA INFORMATICA

Verifico ora che le parti sinistre delle relazioni sono chiavi:

$$\left\{ \begin{array}{l} \text{Ricordanza} \\ \underline{\text{Materie} \rightarrow \text{Aula, Docente}} \\ \underline{\text{Aula, Ora} \rightarrow \text{Materia}} \end{array} \right.$$

Per eliminare la ridondanza scopro quelle relazioni dallo schema principale.

In questo potremmo avere la seguente scomposizione:

① Rx (Materia, aula, docente)

$$\left\{ \begin{array}{l} \text{Materia} \rightarrow \text{aula, docente?} \end{array} \right.$$

②  $R_2$  (Aula, Ora, Materia)

$$\left\{ \begin{array}{l} \text{Aula Ora} \rightarrow \text{Materia} \\ \text{Materia} \rightarrow \text{Aula, docente} \end{array} \right\}$$

cioè non è in BCNF

Risolviamo allora le dipendenze

①  $R_2$  (Materia, Aula, docente)

$$\left\{ \text{Materia} \rightarrow \text{aula, docente} \right\}$$

②  $R_2$  (Aula, Ora)

$$\left\{ \emptyset \right\}$$

Questa è una decomposizione in BCNF ed è lossless join ma ha il problema di non permettere di ricomporre l'integrità in quanto si perdono delle dipendenze.

Dunque non è sempre possibile mantenere ottenere un BCNF anche integro ma è possibile eliminare comunque ridondanza.

Esempio 2

Esame (matricola, nome, materia, docente, data, voto)

$$\left\{ \begin{array}{l} \text{matricola} \rightarrow \text{nome} \\ \text{materia} \rightarrow \text{docente} \\ \text{matricola, materia} \rightarrow \text{data, voto} \end{array} \right.$$

① Studente (matricola, nome)

$$\left\{ \text{matricola} \rightarrow \text{nome} \right\}$$

② Insegnamento (materia, docente)

(2) l'eseguimento (materia, docente)

$$\{ \text{materia} \rightarrow \text{docente} \}$$

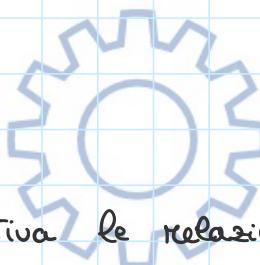
(3) Eseme  $\{ \underline{\text{matricola}}, \underline{\text{materia}}, \text{data}, \text{voto} \}$

$$\{ \text{matricola}, \text{materia} \rightarrow \text{data}, \text{voto} \}$$

Si dice che la decomposizione gode della proprietà di **LOSELESS JOIN** ovvero, non si perde l'informazione originaria che è ottenibile attraverso un join. Ciò non implica l'integrità.

Inoltre le proprietà iniziali vengono mantenute ovvero non si perdono le dipendenze funzionali.

Definizione formale



Scomponendo in maniera iterativa le relazioni si ottiene ad un certo punto la forma BCNF

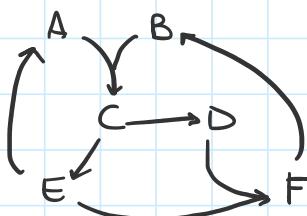
Esempio

## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

R(ABCDEF)

$$\begin{cases} AB \rightarrow C \\ C \rightarrow ED \\ ED \rightarrow F \\ \underline{E \rightarrow A} \\ \underline{F \rightarrow B} \end{cases}$$



non è in BCNF keys  $\{ AB, C, ED, EF, BE, AF \}$   
poiché E non  
è chiave

Scompongo chiave R:

Come x scelgo una dipendenza che soddisfa, come Y ea

Come X scelgo una dipendenza che soddisfa, come Y la chiusura di X e in Z metto il resto:

$$X = E, Y = A, Z = BCDF$$

Partiziono dunque in due relazioni dove metto in una XY e nell'altra il resto:

$$R_1 (AE)$$

$$\{ E \rightarrow A \}$$

BCNF

$$R_2 (BCDEF)$$

$$\{ \begin{array}{l} C \rightarrow ED \\ ED \rightarrow F \\ F \rightarrow B \\ EB \rightarrow C \end{array} \}$$

era implicata una ora la devo indicare

una BCNF  $\rightarrow$  ifero

$$\downarrow \\ X = F \\ Y = B$$

APPUNTI DI INGEGNERIA

INFORMATICA

GAIA BERTOLINO

$$R_3 (BF)$$

$$\{ F \rightarrow B \}$$

BCNF

$$R_4 (CDEF)$$

$$\{ \begin{array}{l} C \rightarrow ED \\ ED \rightarrow F \\ ED \rightarrow C \end{array} \}$$

BCNF

Dunque  $R_1, R_3$  e  $R_4$  sono una decomposizione BCNF senza perdita di informazioni. Tuttavia si sono perse dipendenze funzionali ovvero  $AB \rightarrow C$

### Definizione formale

Una decomposizione di R in  $R_1, \dots, R_u$  è in BCNF se  $\forall i \in [1 \dots u] R_i$  è

Una decomposizione di  $R$  in  $R_1, \dots, R_u$  è in BCNF se  $\forall i \in [1 \dots u]$   $R_i$  è in BCNF (per determinare se una  $R_i$  è in BCNF occorre riferirsi all'insieme di dipendenze funzionali  $f_i$  ottenute proiettando l'insieme di dipendenze funzionali  $f$  (definiti su  $R$ ) su  $R_i$ .

Si cercano due caratteristiche

- 1) Non voglio join: non voglio perdere informazioni
- 2) Non voglio perdere dipendenze funzionali

Tuttavia non è sempre possibile garantire 1 e 2.

L'algoritmo iterativo garantisce la proprietà 2.

### Esempio

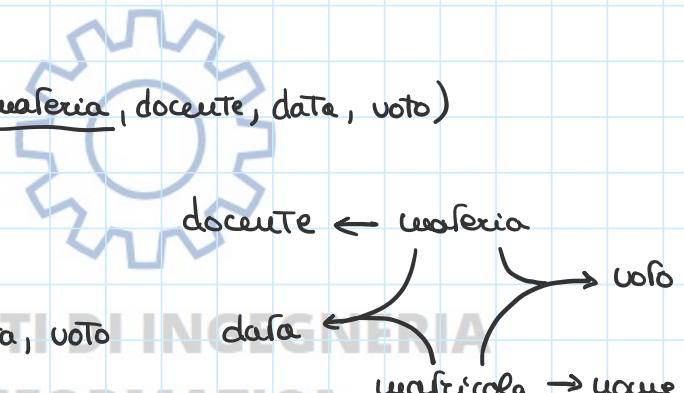
Esame (matricola, nome, materia, docente, data, voto)

$$\left\{ \begin{array}{l} \text{matricola} \rightarrow \text{nome} \\ \text{materia} \rightarrow \text{docente} \\ \text{matricola}, \text{materia} \rightarrow \text{data}, \text{voto} \end{array} \right.$$

$$x = \text{matricola}$$

$$y = \text{nome}$$

$$z = \text{materia, docente, data, voto}$$



Studente (matricola, nome)

$$\left\{ \text{matricola} \rightarrow \text{nome} \right\}$$

BCNF

$$\left\{ \begin{array}{l} \text{materia} \rightarrow \text{docente} \\ \text{matricola}, \text{materia} \rightarrow \text{data}, \text{voto} \end{array} \right\}$$

$$x = \text{matricola}$$

$$y = \text{docente}$$

$$z = \text{data, voto}$$

Insegnamento (materia, docente)  
 $\{ \text{materia} \rightarrow \text{docente} \}$

BCNF

Esame " (materie, materia, data, voto )  
 $\{ \text{materie, materia} \rightarrow \text{data, voto} \}$

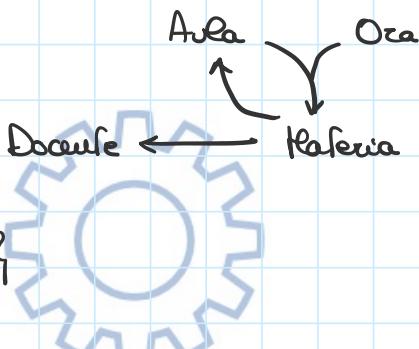
BCNF

Coda di Posless giù e non ho perso dipendenze funzionali

### Esempio

Oraio (Aula, Ora, Materia, Docente)

$\{ \text{Materia} \rightarrow \text{Aula, Docente} \}$   
 $\{ \text{Aula, Ora} \rightarrow \text{Materia} \}$



Keys  $\{ \text{Aula}, \text{Ora}, \text{Materia}, \text{Ora} \}$

$X = \text{Materia}$   
 $Y = \text{Docente, Aula}$   
 $Z = \text{Ora}$

## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

R<sub>1</sub> (Materia, Docente, Aula)

$\{ \text{Materia} \rightarrow \text{Docente, Aula} \}$

BCNF

R<sub>2</sub> (Materia, Ora)

$\{ \emptyset \}$

BCNF

Coda di Posless giù ma vi è perdita di dipendenze funzionali.

Cioè vuol dire che se distribuisco gli attributi: non ottengo una soluzione esattamente coerente con la soluzione iniziale.

In particolare quando ho cicli nei grafi non si riesce ad arrivare a una decomposizione che rispetti più delle 3 norme.

In particolare quando ho cicli nei grafi non si riesce ad ottenere una decomposizione che rispetti entrambe le proprietà.

Esiste un'altra forma, detta 3NF (Third Normal Form) tale che una relazione  $R$  è in 3NF se  $\forall x \rightarrow Y \in f$  in forma canonica vale almeno una delle due proprietà seguenti! copertura minima

- 1)  $x$  è una chiave
- 2)  $Y$  appartiene ad una chiave

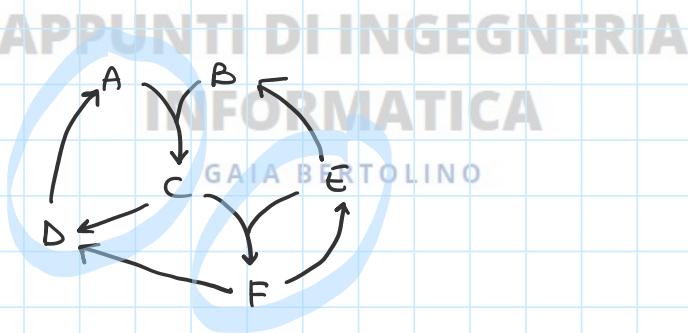
es. data  $AB \rightarrow CD$ , essa soddisfa P2) se  $C$  appartiene a una chiave e se  $D$  appartiene a una chiave (non è necessario che  $C$  e  $D$  appartengano alla stessa chiave).

Vale che se  $R$  è in BCNF allora è in 3NF ma non vale il viceversa.

Tuttavia, sempre una decomposizione in 3NF senza perdita di informazioni e senza perdite di dipendenze funzionali.

es.  $R(A B C D E F)$

$$\left\{ \begin{array}{l} AB \rightarrow C \\ C \rightarrow D \\ D \rightarrow A \\ CE \rightarrow F \\ E \rightarrow B \\ F \rightarrow E \\ F \rightarrow D \end{array} \right.$$



1) Verificare che la copertura sia minima. (da fare come esercizio)

2) Trovare le chiavi:

keys  $\{F, ED, EC, AE\}$

3) Data la presenza di cicli, una decomposizione in BCNF avrà sicuramente una perdita di informazione

4) Verifica se è in 3NF:

$$\left\{ \begin{array}{l} AB \rightarrow C \\ C \rightarrow D \\ D \rightarrow A \\ CE \rightarrow F \\ E \rightarrow B \\ F \rightarrow E \\ F \rightarrow D \end{array} \right.$$

Tutte le relazioni sono in 3NF  
Tranne 1 quindi si può eliminare  
ridondanza

Per scomporre in 3NF si prendono tutte le dipendenze funzionali accorpandole  
una in forma canonica, si crea una relazione per ogni dipendenza:

$$\begin{aligned} R_1(ABC) & \quad \left\{ AB \rightarrow C, \underline{C \rightarrow A} \right\} \\ R_2(CEF) & \quad \left\{ CE \rightarrow F, \underline{F \rightarrow EC} \right\} \\ R_3(DEF) & \quad \left\{ F \rightarrow DE, \underline{DE \rightarrow F} \right\} \\ R_4(CD) & \quad \left\{ C \rightarrow D \right\} \\ R_5(DA) & \quad \left\{ D \rightarrow A \right\} \\ R_6(EB) & \quad \left\{ E \rightarrow B \right\} \end{aligned}$$

Inserisco le dipendenze e le  
proiezioni anche se sono presenti  
nel sistema di paranza

## APPUNTI DI INGEGNERIA

### INFORMATICA

GAIA BERTOLINO

Per garantire che non si perdano informazioni, almeno una relazione deve  
contenere una chiave di paranza, altrimenti bisogna inserire una relazione  
relativa ad una chiave del tipo  $R_7(AE) \{ \phi \}$

5) Per ottenere la decomposizione minima verifico se vi sono relazioni che possono  
essere accoppiate:

$$\begin{aligned} R_1(ABC) & \quad \left\{ AB \rightarrow C, C \rightarrow A \right\} \\ R_2(CEF) & \quad \left\{ CE \rightarrow F, F \rightarrow EC \right\} \\ R_3(DEF) & \quad \left\{ F \rightarrow DE, DE \rightarrow F \right\} \\ R_4(CD) & \quad \left\{ C \rightarrow D \right\} \\ R_5(DA) & \quad \left\{ D \rightarrow A \right\} \\ R_6(EB) & \quad \left\{ E \rightarrow B \right\} \end{aligned}$$

Posso implicare solo con gli argomenti della relazione

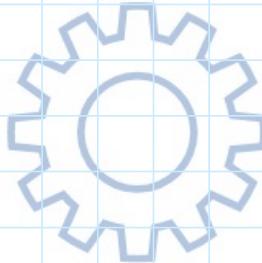
Tuttavia una è in 3NF

Dunque procedere per tentativi è inutile. Dunque ragiono accoppiando cicli

$R_1 (ABC)$	$\{ AB \rightarrow C, C \rightarrow A \}$	$+$	$R_{45} (ABCD) \quad \{ AB \rightarrow C, C \rightarrow D, D \rightarrow A \}$ keys $\{ AB, CB, DB \}$ che è un accoppiamento corretto
$R_2 (CEF)$	$\{ CE \rightarrow F, F \rightarrow EC \}$		
$R_3 (DEF)$	$\{ F \rightarrow DE, DE \rightarrow F \}$		
$R_4 (CD)$	$\{ C \rightarrow D \}$	$+$	
$R_5 (DA)$	$\{ D \rightarrow A \}$	$+$	
$R_6 (EB)$	$\{ E \rightarrow B \}$		

Otengo dunque :

$R_{45} (ABCD)$	$\{ AB \rightarrow C, C \rightarrow D, D \rightarrow A \}$
$R_2 (CEF)$	$\{ CE \rightarrow F, F \rightarrow EC \}$
$R_3 (DEF)$	$\{ F \rightarrow DE, DE \rightarrow F \}$
$R_6 (EB)$	$\{ E \rightarrow B \}$



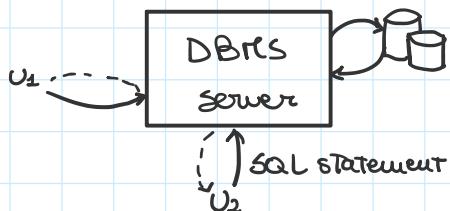
## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## Rappresentazione in memoria

sabato 29 gennaio 2022 12:06

Soltanente i DBMS hanno una architettura del tipo client-server ovvero vi è un DBMS server a cui arrivano delle richieste formulate in codice SQL. Tale server si interfaccia con un storage di dati.



Ogni DBMS può decidere come rappresentare / codificare i dati o come si appoggiano allo spazio fisico.

Essi, infine, gestiscono la granularità di accesso diversamente da un file system.

La gestione delle tabelle avviene tenendo in conto il concetto di durata dei dati: un dato deve permanere fino a quando un utente non decide di cancellarlo. Dunque, una RAM non è utilizzabile se si usano gli hard disk.

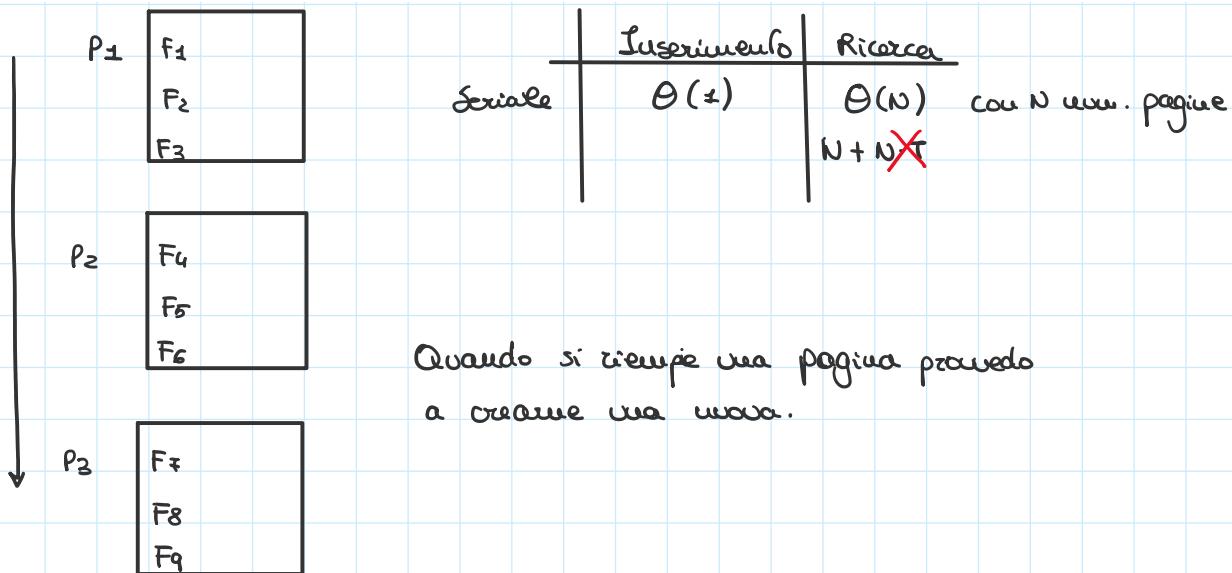
Le funzioni hash potrebbero sembrare una soluzione ottima ma esse risultano inefficienti quando si tratta di elasticità nell'accesso dei dati.

In un SO si parla di word, nella memoria secondaria, più lenta di una RAM, si trovano pezzi più grossi di un word che riesce a contenere ben più di una tupla. Dunque, bisogna pensare al fatto che accedendo ad un dato è facile accedere anche a quelli vicini.

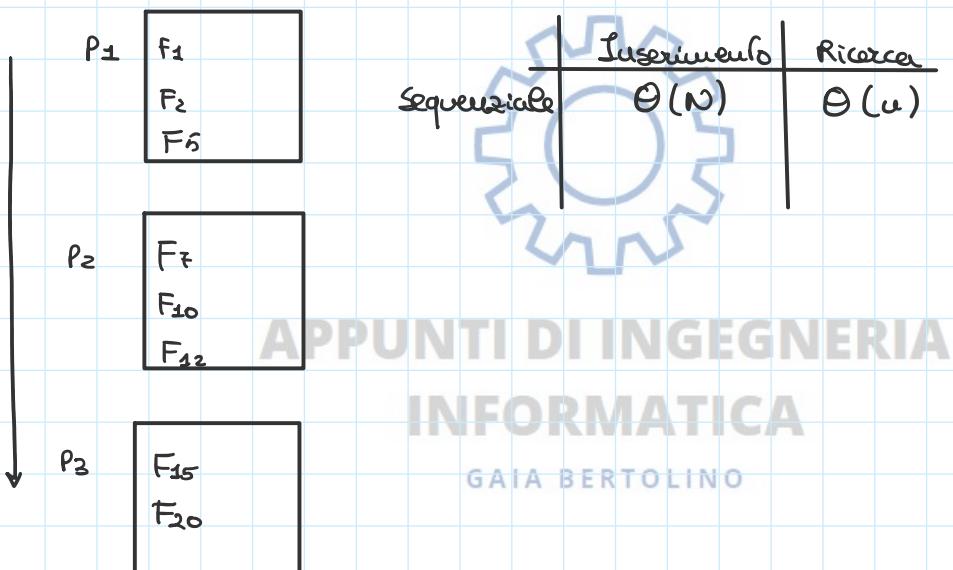
La memoria è organizzata in pagine per cui il costo di ricerca non è dato al numero di tuple da visitare ma dal numero di pagine da accedere in quanto l'accesso ad una pagina ha un costo decisamente maggiore della scissione degli elementi al suo interno che dunque diventa trascurabile.

Inoltre, un HDD ha una latenza mille volte peggiore.

Per memorizzare le tuple utilizzo un meccanismo **SERIALE** ovvero provvedo a riempire le pagine con i dati a cascata



Un altro tipo di memorizzazione è **SEQUENZIALE** ovvero ordino le tuple di una pagina secondo la chiave



## APPUNTI DI INGEGNERIA

### INFORMATICA

GAIA BERTOLINO

L'inserimento avviene con costo pari all'insertion sort poiché per inserire una tupla in una pagina devo far scorrere gli indirizzi, anche di pagina.

Ovviamente in queste situazioni i costi bisogna aggiungere i costi di **DEFRAGMENTAZIONE** delle pagine.

I tempi di attesa sequenziali e seriali sono troppo lunghi e dunque si utilizzano altre strutture.

### HASHING

In una tabella hash si trasforma uno o più campi della tupla da

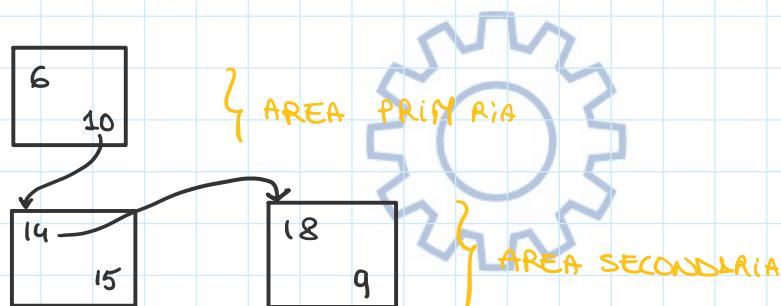
## HASHING

In una tabella hash si trasforma uno o più campi della tupla da inserire in una chiave tramite una funzione hash.

Una tupla di chiavi si trova nella pagina  $H(k) = k \bmod m$  dove  $m$  è il numero delle pagine.

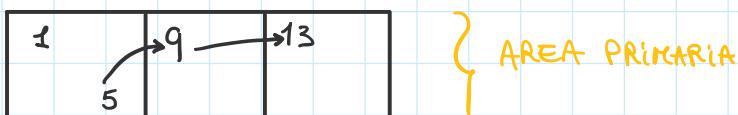
Tale disposizione permette anche di gestire eventuali collisioni. Tuttavia, il trabocco non è direttamente gestito e richiede che a ogni pagina, quando essa si riempie, venga legato un puntatore che punta ad un'altra lista che fa riferimento alla stessa pagina. Ne consegue che aumenta leggermente il costo di ricerca.

Per migliorare la tabella hash è possibile creare una sola tabella di trabocco tale che essa continuerà valori di trabocco di altre pagine e ogni pagina sarà puntata dall'ultimo valore precedente relativo alla chiave. Dunque si creano puntatori fra un valore e una pagina.



## APPUNTI DI INGEGNERIA INFORMATICA

Un'altra tecnica consiste nell'autore a posizionare il valore che andrebbe nella pagina piena nella pagina più occupata o in quella a destra se non è piena e uso puntatori fra valori



In questo caso il costo dunque risce perché queste pagine sono allocate inizialmente e dunque anche posizionate nell'hard disk in maniera ottimale. Invece, la creazione di una pagina di trabocco avviene dove c'è spazio e dunque è meno efficiente.

Un sistema simile che usa i puntatori si chiama **hashing lineare ad indirizzamento aperto** il quale, anziché avere il valore  $k$  in  $H(k) = k \bmod m$  per un trabocco ma il valore  $k \bmod m$  da trasferire la sua posizione. C'è causa però il fenomeno di **agglomerazione primaria** ovvero tutte le celle hanno la stessa probabilità di essere destinazione della tupla vera, quando avviene

Cioè causa però il fenomeno di **agglomerazione primaria** ovvero tutte le celle hanno la stessa probabilità di essere destinazione della tupla vera, quando avviene un trabocco, la probabilità della pagina a disfatta aumenta ed essa viene oberata di lavoro. Cioè causa anche fenomeni di trabocco a catena.

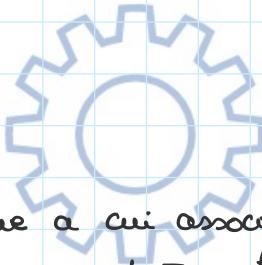
Per evitare questo problema il valore si può essere scelto casualmente per evitare di obbligare eccessivamente delle pagine.

Una tecnica di hashing è detta **STATICA** se l'area di memoria usata varia ovvero equivale sempre a quella primaria.

Avere le pagine misero a parte causa un rallentamento nella ricerca di elementi secondo un campo diverso dalla chiave in quanto devono essere analizzate tutte le pagine con un costo maggiore rispetto al caso in cui fossero state ammesso-

## HASHING DINAMICO

### Hashing virtuale



Si parte da un certo numero di pagine a cui associo un vettore di bit dove ogni cella contiene un 1 se la pagina corrispondente ha almeno una tupla, 0 altrimenti.

es.  $M=3$

0	1	2
3	10	
6		

$$H(k) = k \bmod M$$

B	01	10	0

Se capienza=2 e arriva ora un 9, ho un trabocco. Dunque raddoppio le pagine e il vettore di bit e ridistribuisco i valori della pagina traboccati usando una nuova funzione di hashing  $H_1(k) = k \bmod 2M$

0	1	2	3	4	5
3	10		3		
6			9		

B	01	10	01	0

La tecnica prevede dunque di raddoppiare l'area dati ad ogni iterazione. Ne consegue che la funzione di hashing diventa  $H_i(k) = k \bmod 2^i \cdot M$

Il vettore B serve per operare una ricerca.

Se ad esempio ricercavo un valore vado a visitare la pagina relativa. Se la pagina ha  $B=0$  ne consegue che o non è stato inserito o si trova nella pagina corrispondente usando la funzione di hashing con  $i-1$ .

Trovare una tupla dunque il costo è pari a 1 accesso al disco perché trovare l'indice della pagina ha costo 0. Nel caso dell'inserimento, il costo è di due accessi al disco. In caso di trabocco, la redistribuzione ha costo dipendente dall'operazione aritmetica del modulo.

Anche nell'inserimento si usa il meccanismo di ricerca poiché bisogna verificare l'esistenza.

In questa tecnica si raddoppia e non si aggiunge solo una pagina perché aggiungendo solo una pagina la nuova pagina sarebbe  $k \bmod M+1$  e ciò implica la redistribuzione di tutti i valori e dunque non potrebbe più essere applicata la tecnica precedente di ricerca tramite il vettore di bit.

Tuttavia, il raddoppio non è sostanziale e questo è il motivo per cui è detto virtuale; esso infatti ha crescita esponenziale.

### Hashing lineare

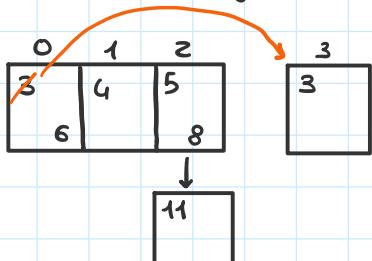
È diverso da quello statico ad indirizzamento statico.

Si memorizza in una variabile l'indice della pagina più a sinistra che non è stata redistribuita.

0	1	2
3	4	5
6		8

$$H(k) = k \bmod M \quad p=0$$

Se ora arriva l'11, nella pagina che dovrebbe traboccare (se non è pari a  $p$ ) inserisco una pagina di trabocco, aumento di una pagina e creo una nuova funzione di hashing  $H_1(k) = k \bmod 2M$ . Prendo poi la pagina  $p$  e la redistribuisco:



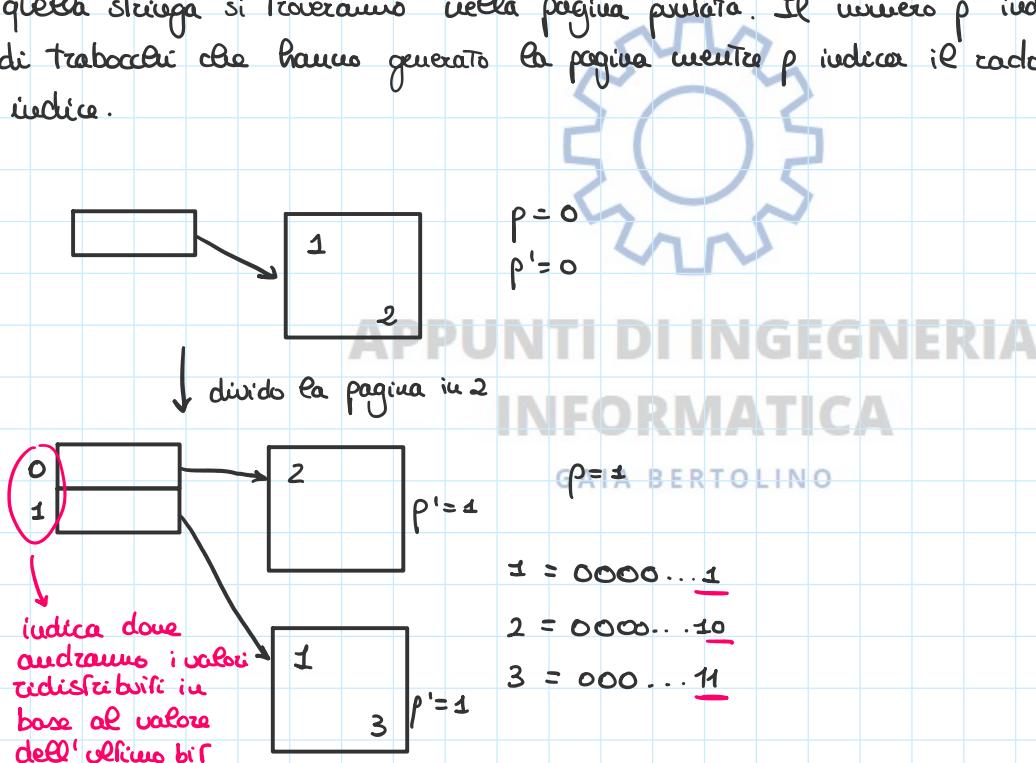
$$p=1$$
$$H_1(k) = k \bmod 2M$$

Dopo il primo Trabocco, per ogni inserimento verifico di poter applicare la funzione  $H$  altrimenti una se ottengo una pagina che ancora non esiste vado ad utilizzare la funzione  $H$  ed eventualmente ridistribuisco secondo il procedimento visto prima. Bisogna ricordare che creo una lista di trabocchi solo se la pagina da ridistribuire è diversa da  $p$ , altrimenti ridistribuisco aggiungendo una pagina.

Quando si raddoppiano tutte le pagine si azzera  $p$  e  $M$  cresce fino ad essere pari al doppio di quella di partenza. Ne consegue che si utilizzano solo due funzioni di hash.

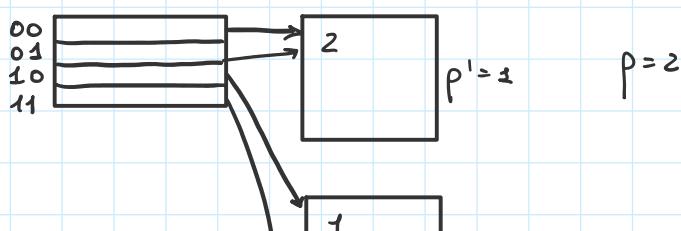
### Hashing infiduciale

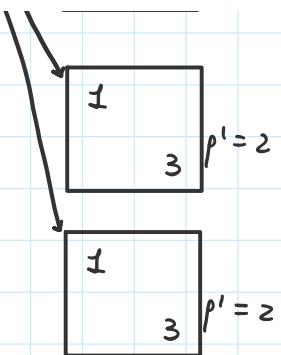
In questo tipo di hashing si ha un vettore dove ogni cella associata ad una stringa di bit che dice che tutte le tuple la cui chiave ha come ultimo bit proprio quella stringa si troveranno nella pagina puntata. Il numero  $p$  indica il numero di trabocchi che hanno generato la pagina mentre  $p'$  indica il raddoppio dell'area indice.



Ogni volta che una pagina trabocca vado a dividere la pagina che trabocca.

L'indice  $p$  di istruzione indica il numero di bit da guardare nella chiave per ridistribuire il valore





Dunque, in risposta al tabacco, spezza una pagina e dunque il numero complessivo aumenta di 1.

Il problema dell'hashing è che devo conoscere tutti i valori da cercare / inserire in quanto dispone i valori in maniera casuale e dunque non è possibile operare una ricerca per range.

Inoltre, non è possibile operare alcuna ottimizzazione a causa della disposizione specifica dell'hashing.

L'hashing è una tecnica procedurale poiché fa uso di una procedura.

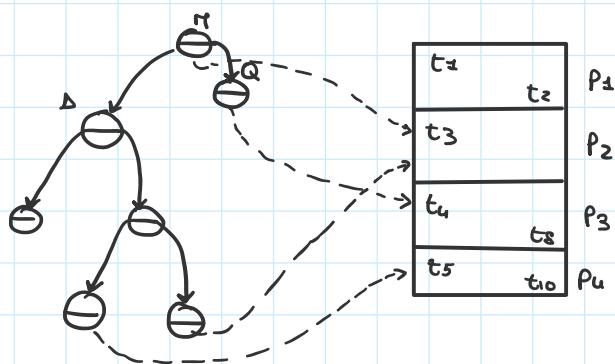
## B-TREE

# APPUNTI DI INGEGNERIA INFORMATICA

Tale processo viene detto **tabellare**.



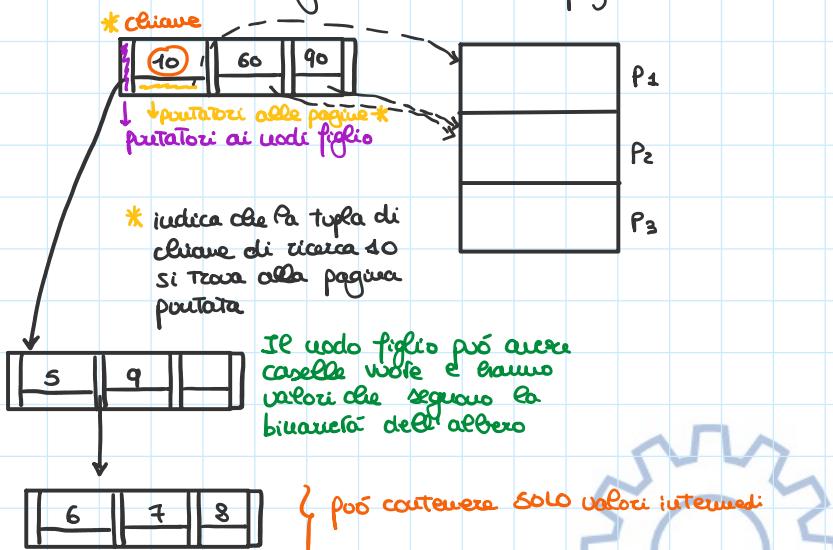
Per indicizzare le pagine e il contenuto scrivendo un albero binario.



Tale albero va bilanciato e non è facile rappresentarlo in memoria secondaria e bilanciarlo.

I B-tree riescono a gestire correttamente e abilmente il bilanciamento in memoria secondaria.

In un B-tree ogni nodo è una pagina e ha una forma tabellare del tipo



Ogni nodo con n chiavi ha n+1 figli e il numero di chiavi più di uno di ogni nodo indica il **GRADO** ovvero il numero di figli massimo che ogni nodo può avere.

Il grado dell'albero è calcolato a priori dal sistema in base alle sue caratteristiche.

La ricerca avviene per scansione del B-tree. Quando arrivo ad una foglia senza aver trovato il valore, allora posso concludere che l'elemento non esiste.

In un albero il costo della ricerca è al più lineare.

Un albero si dice **BILANCIATO** se per ogni nodo tutti i sottoalberi destro e sinistro differiscono per al massimo un nodo.

In un B-tree invece tutte le foglie stanno alla stessa altezza.

In un B-tree la profondità è molto bassa e la base del raggiungimento dell'altezza dipende dal grado.

Difatto, i nodi all'ultimo livello sono quelli più numerosi e ne consegue che esso sta in memoria secondaria mentre il resto dei nodi in memoria principale.

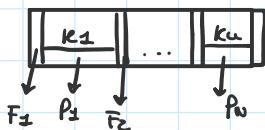
## Proprietà

In un B-tree conviene addensare i valori ed evitare la troppa dispersione. Dunque non molti nodi vuoti non ci sono!

In un B-tree conviene addensare i valori ed evitare la troppa dispersione. Dunque esso gode delle seguenti proprietà:

B-TREE di grado  $u$

1) Ogni nodo ha questa struttura



dove le  $k_i$  sono ordinate e i sottobalzoni figli mantengono l'ordinamento

2) Se un nodo ha  $x$  chiavi e una è foglia allora ha  $x+1$  figli e viceversa

3) Tutte le foglie sono alla stessa altezza (**proprietà di bilanciamento**)

4) Ogni nodo non radice prende almeno  $\lceil \frac{u}{2} \rceil - 1$  chiavi (proprietà che prevede che la struttura sia sparsa)

**proprietà del 50%**

Le operazioni di inserimento vengono fatte in memoria principale e dunque hanno costo trascurabile.

## APPUNTI DI INGEGNERIA

Nel caso di presenza di celle disponibili nel nodo foglia opera traslazioni e inserisco il valore in posizione corretta. Nel caso in cui sia pieno invece proviamo un valore nella radice e trasformo gli altri in figli!

es. 1)



2)

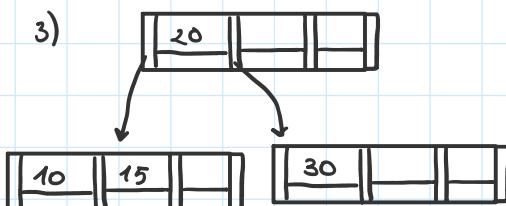


provo una codifica

sinistra → 10 15 20 30 destra

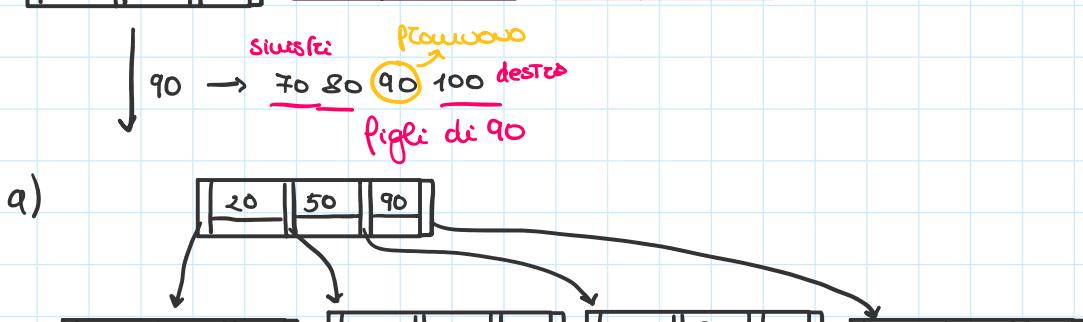
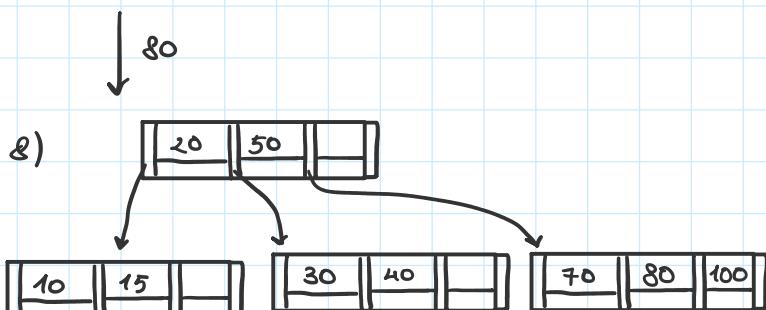
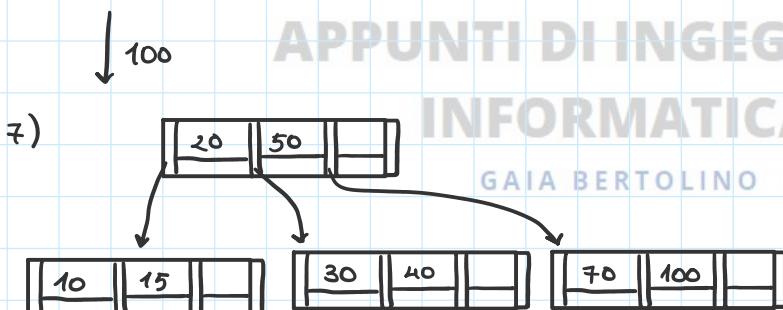
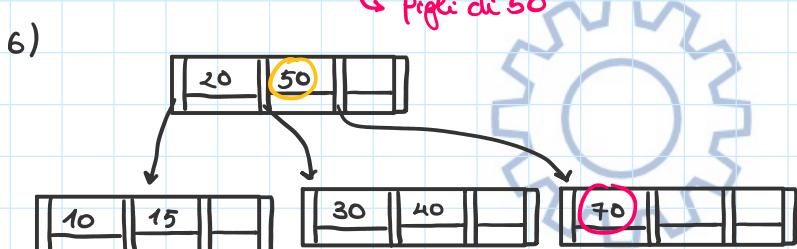
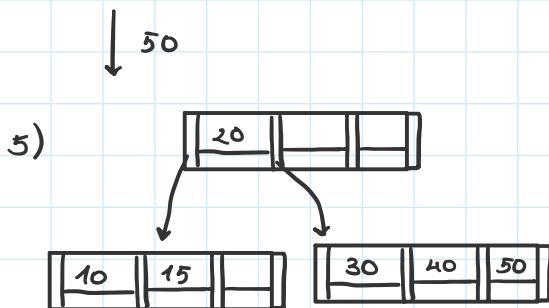
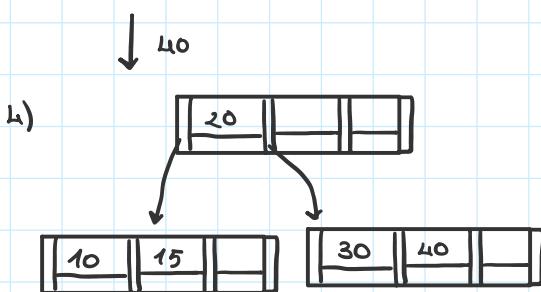
diventano figli

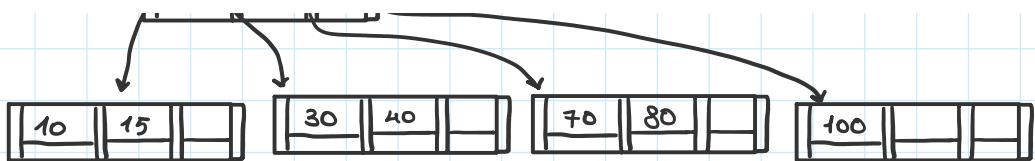
3)



Ogni volta che inserisco un valore lo inserisco nell'ultimo buco

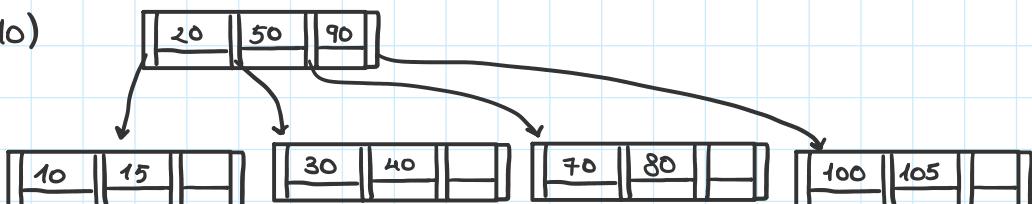
Ogni volta che inserisco un valore lo inserisco nell'ultimo blocco





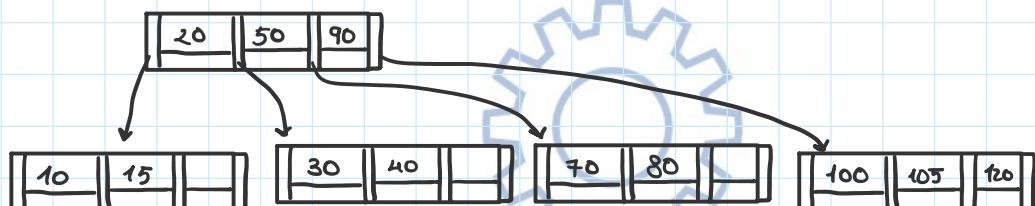
↓ 105

10)



↓ 120

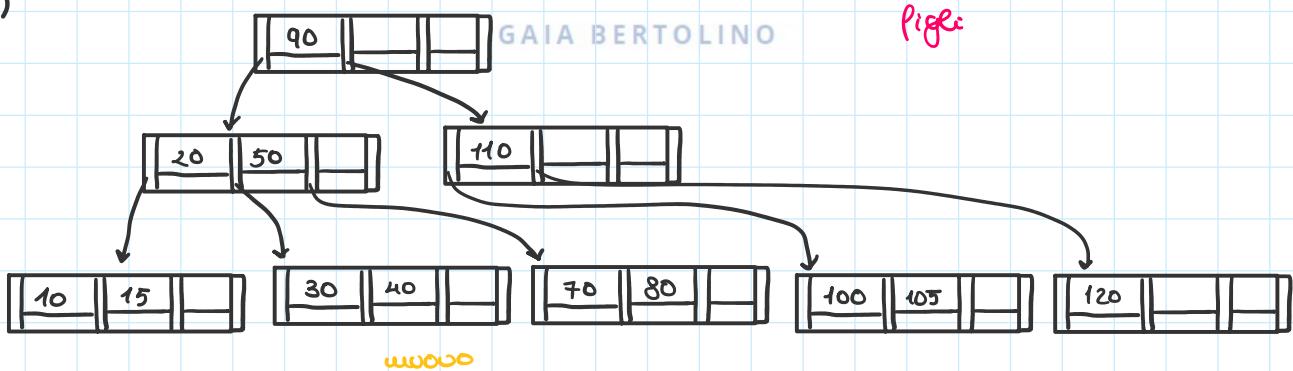
11)



↓ 110

*proponendo  
sinistra 100 105 110 120  
destra però il nodo sopra è pieno  
quiudi creo una nuova radice  
figli*

12)

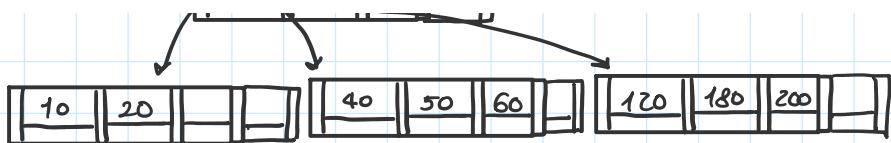


*nuovo*

I nodi all'ultimo livello saranno pari a  $\frac{m^{k-1} - 1}{m - 1}$

Rimozione :



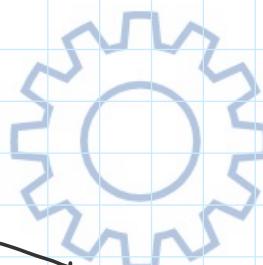


Se volevo eliminare il 20 visto la soglia minima  $\lceil \frac{m}{2} \rceil - 1 = 2$



La rotazione si limita al livello e può avvenire solo se la foglia a destra ha elementi da prestare

↓ Nodo 30



Se tutti i nodi a destra sono a riempimento minimo allora posso eliminare un nodo

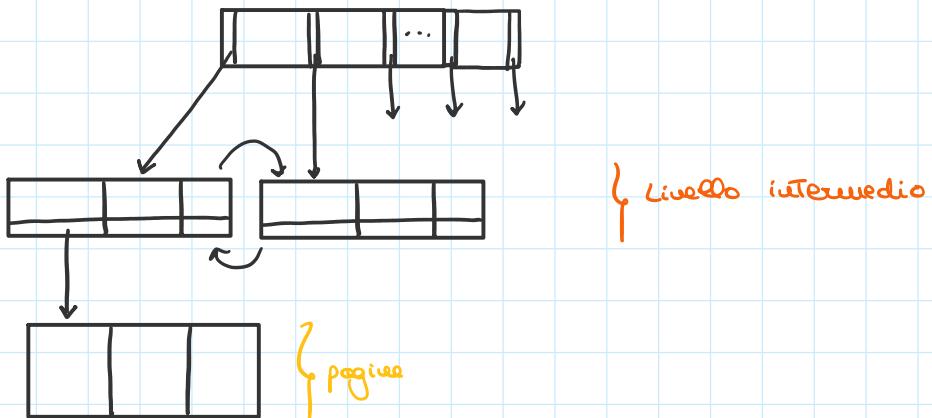


La cancellazione dei nodi si propaga fino alla radice esclusa.  
Tuttavia, è possibile che diminuisca l'altezza dell'albero.

Apartiti sui costi: lez. 3 dicembre

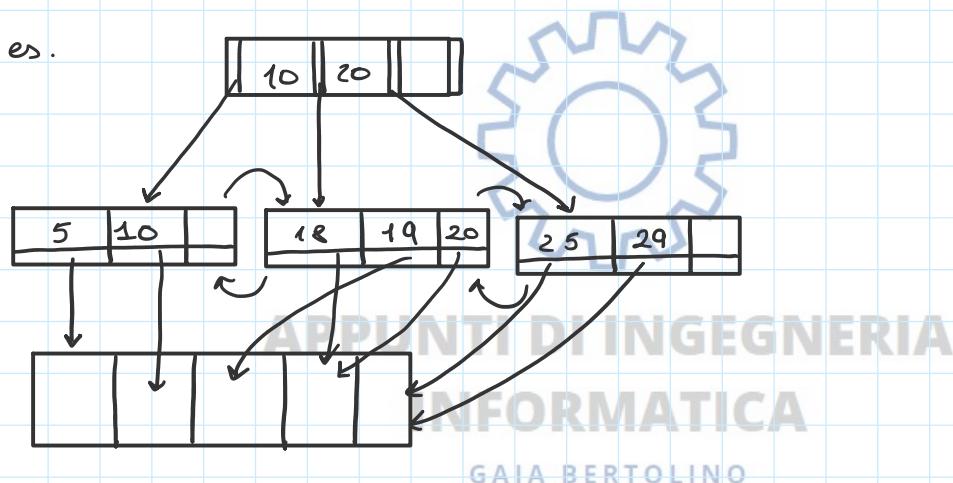
$B^+$  Tree





Tutte le chiavi intermedie sono riportate anche nel livello foglia ovvero sono ripetute. Inoltre i nodi a livello foglia sono concatenati in una lista. Tramite puntatori.

Dunque nell'ultimo livello vi è un ordinamento lessicografico in base alla chiave; dunque è ideale ad operare la ricerca per range.



Negli alberi di scansione ordinata richiede degli indici. In B-tree del B<sup>+</sup>tree siamo ad indicizzare più chiavi perché necessario meno prefissi.

Dunque ogni nodo avrà un grado maggiore e l'altezza del B<sup>+</sup>tree è più bassa di quella di un B-tree.

La particolarità di un B<sup>+</sup>tree è quella di poter usare una pagina con due o più chiavi candidate ed è possibile utilizzarne una per ordinare i valori. Così facendo è possibile accedere ad un dato e scandire i successivi senza dover rivedere gli indici se devo operare una ricerca per range.

### Indice sparso

Inoltre, in una struttura ordinata succede che è possibile utilizzare pochi indici per mappare più valori in una pagina. Infatti, usando un valore chiave più piccolo o grande è possibile accedere alla pagina che contiene il range a cui si vuole fare riferimento.