

# Cheat Sheet Finale di Basi di Dati

di Alessandro Ricci (alessandro.ricci47@studio.unibo.it)

Motto:

*“Ogni volta che la matematica si impone con la sua dittatura grammaticale, costringendo al rispetto della corretta terminologia, anche a costo di complicarne inutilmente la comprensione da parte degli studenti, un Cheat Sheet viene creato appositamente per trasgredire alle regole, nella speranza di rendere l'apprendimento libero e approcciabile da chiunque”*

Disclaimer:

Tutti i capitoli di questo documento tentano di spiegare in modo più informale possibile gli argomenti e le tecniche imparate al corso di Basi di Dati 2023/24 della prof. Annalisa Franco. Tutto il contenuto è stato realizzato con cura, ma non è assicurato che sia esente da errori o inesattezze di ogni tipo.

## Sommario

Terminologia Fondamentale .....	2
Normalizzazione .....	3
Progettazione concettuale (E/R) .....	4
Terminologia Grafica.....	4
Vincoli importanti da rispettare .....	4
Vincoli meno importanti sulla nomenclatura .....	4
Progettazione logica.....	4
Vincoli da rispettare .....	4
Vincoli meno importanti sulla nomenclatura .....	5
Struttura di una Query SQL .....	5
Algebra Relazionale .....	5
Transazioni .....	6
Lost Update .....	7
Dirty Read .....	8
Unrepeatable Read.....	8
B-Tree .....	9
Terminologia .....	9
Vincoli da rispettare .....	9
Esercizio di eliminazione chiave .....	10

Esercizio di inserimento chiave (semplice) .....	11
Esercizio di inserimento chiave (complesso) .....	12

## Terminologia Fondamentale

- **Attributo** → sinonimo di “singola colonna”, “campo”, “proprietà” di una tabella
- **Record** → sinonimo di “riga” di una tabella, composta da uno o più attributi (o colonne)
- **Tupla** → insieme di attributi, ad esempio {Nome, Cognome, ecc...}.  
Ad esempio, una tupla composta da tutti gli attributi di una tabella è equivalente a dire “riga” della tabella
- **Dominio** → letteralmente il “tipo di dato” di un attributo (come se fosse il tipo di dato di una variabile in un qualsiasi linguaggio di programmazione), però usando un termine più “matematico”.  
Il dominio è, più precisamente, l’insieme di tutti i possibili valori che può assumere un determinato attributo. Ad esempio, il tipo di dato “Int” consente di avere qualsiasi numero intero compreso tra un intervallo “teoricamente infinito” (nella pratica delimitato da “MinInt” e “MaxInt” della macchina), ma un attributo chiamato “Mese” è giusto che, pur essendo di tipo “Int”, possa accettare solo valori interi compresi tra 1 e 12. Quindi, per meglio dire, il dominio è una “restrizione sul tipo di dato” di un attributo
- **Superchiave** → attributo singolo, o combinazione di attributi, i cui valori sono univoci nella tabella, ovvero, i cui valori (o combinazione di valori) sono diversi per ogni riga della tabella.  
Ad esempio (banale ma utile per capire), la combinazione di TUTTI gli attributi di una tabella è per forza superchiave, poiché tutte le righe di una tabella devono essere distinte, ovvero ogni riga deve presentare almeno una differenza (anche se minima) rispetto alle altre
- **Chiave** → attributo singolo, o combinazione di attributi, che identificano univocamente ogni riga della tabella, ovvero, una “superchiave minimale” (cioè, il numero minimo di attributi le cui combinazioni di valori sono univoci, cioè diversi per ogni riga).  
Questo vuol dire che una superchiave può contenere altri attributi “inutili”, cioè che non sono essenziali per determinare univocamente i record. Ad esempio, la combinazione (CodiceFiscale, Nome). In questo caso, la coppia (CodiceFiscale, Nome) è superchiave, ma, mentre lo stesso Nome può ripetersi più volte, è il CodiceFiscale a garantire l’unicità effettiva, cioè è proprio CodiceFiscale il “motivo” per cui le righe saranno tutte distinte, quindi la chiave è determinata soltanto da CodiceFiscale
- **Chiave Esterna** → chiave (composta da uno o più attributi) i cui valori sono vincolati da uno stesso attributo presente in un’altra tabella, dove in quest’ultima quell’attributo è chiave primaria.  
Ad esempio, una tabella “Persone” contiene gli attributi CodiceFiscale, Nome, Cognome e CodiceNazionalità. Nel campo “CodiceNazionalità” non può chiaramente essere scritta qualsiasi cosa, ma solo le effettive nazionalità esistenti, ad esempio IT, EN, eccetera. Per determinare tutti questi valori ammissibili, nel database sarà presente un’altra tabella “Nazionalità”, che contiene l’attributo “CodiceNazionalità” come chiave primaria. A questo punto, la tabella “Persone” si può “collegare” alla tabella “Nazionalità” imponendo che il suo attributo “Nazionalità” sia una chiave esterna, che punta proprio all’omonimo attributo della tabella “Nazionalità”. In questo modo, nel campo della tabella “Persone” possono essere inseriti i soli valori presenti nel campo della tabella “Nazionalità”

- **Attributo Primo** → attributo singolo che fa parte di una chiave (e, ovviamente, attributo NON-primo significa che NON fa parte di una chiave, sintassi che verrà usata spesso nella Normalizzazione)
- **Vincolo** → restrizione, di qualsiasi tipo, sui valori che può assumere un attributo di una tabella.  
Ad esempio, un vincolo di una chiave esterna, un vincolo che impone che l'importo inserito sia maggiore di zero, o ancora un vincolo che controlla il rispetto di una o più condizioni qualsiasi (detto anche “CHECK”, ad esempio CHECK StipendioNetto = StipendioLordo + Tasse)
- **FD (Functional Dependency)** → un attributo, o una combinazione di attributi, che DETERMINANO i valori di un altro attributo, o combinazione di attributi. In altre parole, per ogni combinazione di valori degli attributi “di sinistra”, i valori degli attributi di “destra” sono sempre gli stessi.  
Ad esempio, consideriamo la FD [CodiceFiscale → Nome, Cognome]. In questo caso, è evidente che (Nome, Cognome) sono DIPENDENTI da CodiceFiscale, quindi, vista dalla prospettiva opposta, CodiceFiscale DETERMINA i valori di (Nome, Cognome). Detto in altri termini, al cambiare del valore di CodiceFiscale cambiano anche i valori di (Nome, Cognome), e se due valori di CodiceFiscale sono uguali, anche i due valori di (Nome, Cognome) saranno uguali (se supponiamo che non si possa cambiare nome)
- **Query** → letteralmente “interrogazione” al database, ovvero, una qualsiasi sequenza di istruzioni che ha come scopo quello di “domandare” al database qualcosa, ad esempio la presenza o il valore di determinati record di una tabella, o ancora la richiesta di inserire un nuovo record o di cancellarlo
- **Atomico** → nel contesto di valori, un valore atomico è un valore singolo e indivisibile rispetto al suo contesto, ad esempio (13) o (“Gennaio”), mentre (“Gennaio, Febbraio, Marzo”) NON è atomico. Nel contesto di funzioni, una funzione atomica è una funzione composta da una o più istruzioni singole e indivisibili, che vengono o eseguite del tutto oppure non vengono eseguite affatto (concetto umano del “tutto o niente”)

## Normalizzazione

- **1NF** → Il dominio di ogni attributo deve essere composto da valori atomici (singoli e indivisibili), e il valore che viene dato ad ogni riga di ogni attributo deve essere atomico.  
Tradotto: non si può avere un campo il cui tipo di dato è una lista, o assegnare una lista di valori ad un singolo valore di un campo
- **2NF** → Ogni attributo non-primo (cioè NON facente parte della chiave) deve dipendere completamente (NON parzialmente) dalla chiave (cioè dalla combinazione degli attributi che insieme formano la chiave). Per analizzare queste dipendenze, si fa uso dello studio delle FD (Functional Dependency)
- **3NF** → Ogni attributo non-primo NON deve dipendere transitivamente dalla chiave (cioè nel caso in cui A (chiave) determina B (non-chiave), e B determina C (non-chiave), è vero che “transitivamente” A determina C; questo NON deve avvenire)

Ogni problema di mancata normalizzazione si risolve creando altre tabelle (o entità) che esprimono in una maniera più “intuitiva” le relazioni tra i vari attributi. Spesso ci si può far guidare dal “buonsenso”, ma può non essere facile per chi è alle prime armi.

# Progettazione concettuale (E/R)

## Terminologia Grafica

- **Entità (entity)** → rettangolo
- **Associazione (relation)** → rombo
- **Cardinalità** → vincoli sui tipi di relazione (uno-a-uno (1-1), uno-a-molti (1-N), zero-a-uno (0-1), zero-a-molti (0-N))
- **Chiave** → uno o più campi sottolineati

## Vincoli importanti da rispettare

1. Ogni entità DEVE avere una chiave, ovvero uno o più attributi (anche esterni) che formano la chiave primaria. Se, per l'entità presa in esame, nel testo non è specificata alcuna chiave, si deve provare a vedere se uno o più campi possono rappresentare la chiave. Se NON è così, bisogna CREARE un nuovo campo da usare come chiave. Solitamente, conviene sempre creare un codice (ad esempio, codiceCorso)
2. Le associazioni possono avere dei campi, ma NON possono avere una chiave
3. Quando un'entità si collega in chiave esterna ad un'altra entità (ad esempio, l'entità DETTAGLIO FATTURA che si collega a FATTURA), l'entità da cui è partito il collegamento si lega QUASI SEMPRE in rapporto uno a uno (1-1). “Quasi sempre” perché, a volte, si può collegare anche in modalità uno-a-molti (1-N) o zero-a-molti (0-N). Per capirlo, bisogna leggere bene il testo dell'esame e/o usare un po' di buonsenso

## Vincoli meno importanti sulla nomenclatura

1. I nomi delle entità DOVREBBERO essere tutti al singolare (non è un grosso problema, ma è uno standard più preferibile)
2. I nomi delle associazioni DEVONO essere concetti, verbi NON coniugati (ad esempio, usare APPARTENENZA invece di APPARTIENE), e i nomi delle associazioni NON DOVREBBERO ripetersi più volte nello stesso schema (non dovrebbero esserci due o più APPARTENENZA, ma occorre trovare dei sinonimi)

# Progettazione logica

## Vincoli da rispettare

1. Per ogni tabella, i campi che fanno parte di una chiave DEVONO essere sottolineati (anche se provengono da chiavi esterne)
2. I campi opzionali (ovvero quelli la cui cardinalità è zero-a-uno (0-1) o zero-a-molti (0-N)) DEVONO avere un asterisco alla fine del nome

## Vincoli meno importanti sulla nomenclatura

1. I nomi delle tabelle DOVREBBERO essere tutti al plurale (non è un grosso problema, ma è uno standard più preferibile)

## Struttura di una Query SQL

Legenda:

- **In blu** → Struttura base di qualsiasi query
- **In verde** → Struttura degli eventuali raggruppamenti per query di aggregazione
- **In rosso** → Struttura opzionale per imporre un particolare ordinamento dei dati

SELEZIONA QUALI CAMPI? **SELECT [...]**

DA DOVE? **FROM [...]**

CON QUALI CONDIZIONI? **WHERE [...]**

RAGGRUPPATI PER QUALI CAMPI? **GROUP BY [...]**

CON QUALI CONDIZIONI SUI RAGGRUPPAMENTI? **HAVING [...]**

ORDINATI COME? **ORDER BY [...]**

## Algebra Relazionale

Una “piccola oscenità matematica” (concedetemi il termine) per esprimere una query attraverso un linguaggio universale.

Solitamente viene sempre chiesto all'esame, in uno dei punti dell'esercizio 4 sulle query.

Per spiegarla brevemente, verrà fornito un esempio di espressione in algebra relazionale con affiancata l'equivalente query SQL, evidenziando le istruzioni equivalenti con lo stesso colore.

Prima diamo un'occhiata ai simboli principali e a quali istruzioni equivalgono in SQL:

- $\pi$  → **SELECT**
- $( )$  → **FROM**
- $\bowtie$  → **JOIN** (ovvero INNER JOIN)
- $\sigma$  → **WHERE**

Adesso proponiamo la seguente espressione in algebra relazionale e la query equivalente:

*Esame di Luglio 2024*

“Scrivere un’espressione in algebra relazionale che visualizzi le proiezioni che hanno incassato almeno 500€ in sale con una capienza massima di 50 posti (codFilm, codSala, incasso, dataProiezione)».

Soluzione:

$$\pi_{codFilm, codSala, incasso, dataProiezione} \left( \sigma_{incasso > 500}(PROIEZIONI) \bowtie \sigma_{posti \leq 50}(SALE) \right)$$

Query equivalente:

```
SELECT PR.codFilm, PR.codSala, PR.incasso, PR.dataProiezione  
FROM PROIEZIONI AS PR  
INNER JOIN SALE AS SA  
ON PR.codSala = SA.codSala  
WHERE PR.incasso > 500  
AND SA.posti <= 50
```

## Transazioni

In maniera semplicissima e riassuntiva:

Le transazioni sono sequenze di operazioni le cui modifiche apportate al database, a fronte dell’esecuzione, possono essere confermate (commit) oppure annullate del tutto (rollback).

Sono utili perché, al verificarsi di un qualsiasi tipo di errore, tutti noi vorremmo che l’intera sequenza di istruzioni effettuate vengano annullate (come se non fossero mai state eseguite), poiché, in caso contrario, ci esporremmo ad una serie di problemi gravi e impredicibili (immaginiamo di dover anche solo “capire” come sistemare un problema derivato da un programma che è crashato a metà, che ha eseguito tutte le istruzioni prima della metà e nessuna delle successive).

Nella pratica, ogni istruzione o sequenza di istruzioni SQL si può racchiudere in un cosiddetto “blocco” di transazione (delimitato da BEGIN TRAN ed END TRAN), seguito, alla fine, da una o più istruzioni COMMIT e ROLLBACK (ad esempio, if any error → ROLLBACK, else → COMMIT).

Detto questo, il corso fa molta leva sul concetto di concorrenza tra transazioni, ovvero, come il programma del database system gestisce i casi in cui più transazioni vengano eseguite contemporaneamente (immaginate il database di un qualsiasi social che deve gestire migliaia (se non milioni) di richieste di pubblicazione di post al secondo).

Quali sono alcuni metodi utilizzati per evitare i classici problemi di “errata gestione della sincronizzazione” nell’ambito delle transazioni in parallelo?

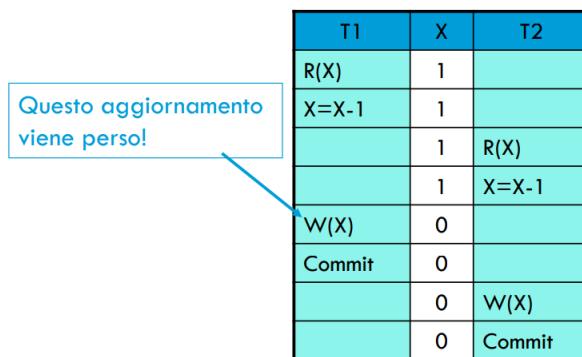
Risposta: in generale tutte le tecniche già viste e riviste per la programmazione concorrente (ad esempio, la mutua esclusione), ma, in questo caso, è importante citare principalmente il **Meccanismo di Lock/Unlock**, di cui si parla molto nelle slide.

Di seguito verranno mostrate le 3 casistiche di “bug di concorrenza” che **possono essere richieste all'esame**, cioè problemi che si verificherebbero con le transazioni se non ci fossero algoritmi appositi ad impedirlo.

Legenda dei simboli nelle tabelle successive:

- **R** → Read
- **W** → Write
- **X** → Valore di un certo elemento che si sta leggendo “R(X)” o scrivendo “W(X)”
- **Colonna T1** → Transazione 1 (simultanea a T2)
- **Colonna T2** → Transazione 2 (simultanea a T1)
- **Colonna X** → Valore di X in ogni istante (il tempo aumenta andando in basso)

## Lost Update



T1	X	T2
R(X)	1	
X=X-1	1	
	1	R(X)
	1	X=X-1
W(X)	0	
Commit	0	
	0	W(X)
	0	Commit

T1 legge X e ne modifica il valore localmente, ma, prima ancora di scrivere la modifica su disco, T2 (che vede ancora il valore originale di X) modifica X a sua volta.

Dopodiché, T1 riesce finalmente a confermare “globalmente” la propria modifica (cioè a scriverlo su disco + commit), ma subito dopo T2 fa lo stesso con il proprio valore locale di X.

In questo modo, **l'aggiornamento di T1 è andato perduto per sempre**.

## Dirty Read

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

Questa lettura  
è "sporca"!

T1 legge X, ne modifica il valore e lo scrive su disco, ma, subito dopo, T2 legge il valore attuale di X (appena modificato da T1).

Dopodiché, per qualche ragione, T1 esegue il rollback di quello che ha appena fatto, annullando quindi la modifica al valore di X.

T2, però, ha letto il valore di X **prima** del rollback di T1, e lo ha salvato in una variabile locale, senza poter venire a conoscenza che in realtà quel valore è stato poi annullato dal rollback di T1.

In questo modo, **T2 procederà ad eseguire le sue prossime istruzioni su un valore "falso" di X**, il cui esito si può riassumere in: "Dio solo sa cosa può succedere".

## Unrepeatable Read

T1	X	T2
R(X)	0	
	0	R(X)
	1	X=X+1
	1	W(X)
	1	Commit
R(X)	1	
Commit	1	

Le 2 letture sono  
tra loro inconsistenti!

Da ricordare come "errore delle 2 letture" o più formalmente "errore di analisi inconsistente".

T1 legge il valore di X. Successivamente, T2 legge lo stesso valore di X e, attraverso alcune operazioni, lo modifica e scrive il valore aggiornato su disco (facendo il commit).

Dopodiché, T1 ha bisogno di leggere una seconda volta il valore di X, aspettandosi naturalmente che sia uguale a quello di prima, ma in realtà il valore è stato cambiato da T2 proprio nel mezzo delle 2 letture di T1.

In questo modo, **T1 ha letto 2 valori diversi per una X che non sarebbe dovuta cambiare**, e andrà a riportare il "falso" dato all'utente attraverso il commit finale (per esempio perché l'utente ne ha richiesto la visione).

# B-Tree

## Terminologia

- **Radice** → nodo più alto dell'albero
- **Nodo** → nodo NON-radice che ha figli
- **Foglia** → nodo finale dell'albero che NON ha figli
- **Chiave** → letteralmente un “numero”, cioè un singolo elemento contenuto in un nodo
- **h** → altezza dell'albero, ovvero il numero di livelli (da vedere come una piramide, se aiuta)
- **g** → ordine dell'albero, ovvero il numero minimo di chiavi (numeri) che possono esserci in un nodo (esclusa la radice che NON vale). Il numero massimo di chiavi, invece, nei B-Tree è sempre  $2g$ :
  - Se il minimo è 1, il massimo è 2
  - Se il minimo è 2, il massimo è 4
  - Se il minimo è 3, il massimo è 6
  - E via dicendo...

## Vincoli da rispettare

1. **Vincolo Maggiore/Minore Verticale** → Tutto quello che sta a destra deve essere maggiore del padre, tutto quello che sta a sinistra deve essere minore del padre
2. **Vincolo Maggiore/Minore Orizzontale** → In uno stesso livello (nodi o foglie adiacenti), ogni chiave deve essere maggiore di ogni altra chiave a sinistra. In altre parole, leggendo da sinistra verso destra, l'ordine deve sempre essere crescente
3. **Vincolo dei figli (o puntatori)** → Ogni nodo deve avere almeno 2 puntatori a figli diversi, uno a sinistra e uno a destra. Se un nodo contiene più di una chiave, devono anche essere presenti dei puntatori in tutte le posizioni “di mezzo” tra due chiavi (ad esempio, un nodo con 2 chiavi deve avere 3 puntatori (sinistra, destra, in mezzo)
4. **Vincolo di g** → Ogni nodo (o foglia) deve avere minimo  $g$  chiavi, e massimo  $2g$  chiavi. Questo discorso NON vale per il nodo radice, che può avere un numero di chiavi al di sotto di  $g$ .
5. **Regola del più a destra** → Una regola inventata da me 😊 che si può riassumere in: “se sei indeciso su quale chiave spostare negli esercizi dei B-Tree, prediligi sempre quella più a destra”. Vedremo come questa regola ci aiuterà a svolgere correttamente gli esercizi elencati di seguito

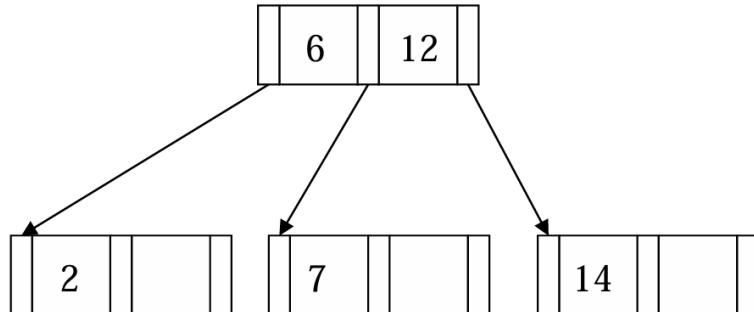
**NOTA:** per questioni di semplicità, negli esercizi successivi, alcuni B-Tree sono disegnati con ogni “freccia” direzionata verso un punto casuale del nodo figlio a cui puntano. Questo NON sarebbe graficamente corretto, poiché la freccia dovrebbe sempre puntare all'estremo sinistro del nodo figlio, predisponendo nel disegno degli appositi “rettangolini vuoti” da puntare, nelle posizioni di sinistra, destra e centro delle chiavi.

Si noti che, in ogni caso, è poco probabile che questo “dettaglio grafico” venga considerato in fase di valutazione all'esame (ci sono soluzioni degli esami passati dove la freccia punta il nodo in direzione casuale, suggerendo che neanche la prof ne tenga conto). Quindi, il consiglio è di prendere questa informazione alla leggera.

## Esercizio di eliminazione chiave

Esame di Giugno 2022

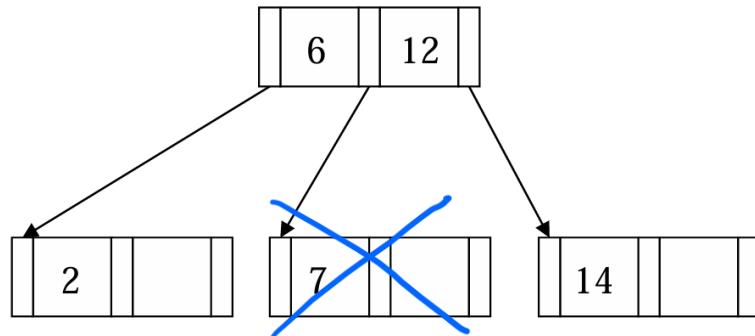
Viene fornito il seguente B-Tree di **ordine (g) = 1**:



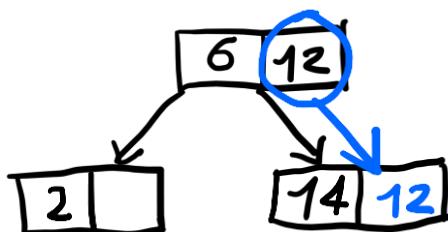
Viene richiesto di riportare la struttura a fronte dell'eliminazione della chiave **7**.

Come fare?

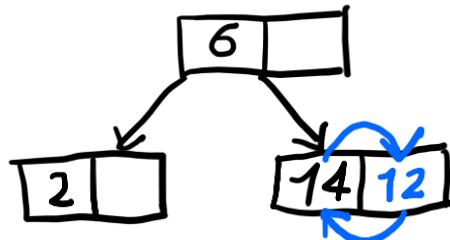
- Notiamo che la chiave da rimuovere farebbe andare il numero di chiavi nella foglia centrale al di sotto di **g** (in questo caso, tra l'altro, diventerebbe zero), quindi un vincolo NON è più rispettato, e perciò bisogna cambiare la struttura dell'albero.  
Immaginiamo quindi di cancellare il nodo centrale



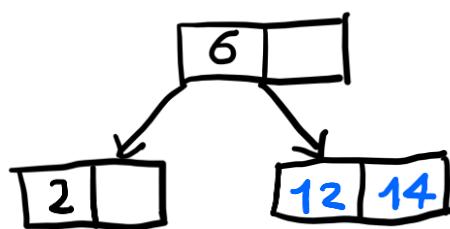
- Per sistemare l'albero, dobbiamo fare in modo che sparisca la "freccia" centrale (tra **6** e **12**), dato che il nodo a cui puntava è stato rimosso (è obbligatorio che ogni freccia punti a un figlio). L'unica possibile soluzione è "**far fluire**" una delle due chiavi del padre in una foglia. Usiamo la "**regola del più a destra**" per scegliere la chiave **12** (anziché **6**) e facciamola andare nella foglia di destra (poiché maggiore di **6**).  
Questo processo di "far fluire" una chiave dal nodo padre al nodo figlio si chiama **catenation** (va esplicitato nel caso venga richiesto di giustificare la risposta)



3. Ovviamente, in questo caso dobbiamo scambiare il **14** con il **12** per rispettare il **Vincolo Maggiore/Minore Orizzontale**



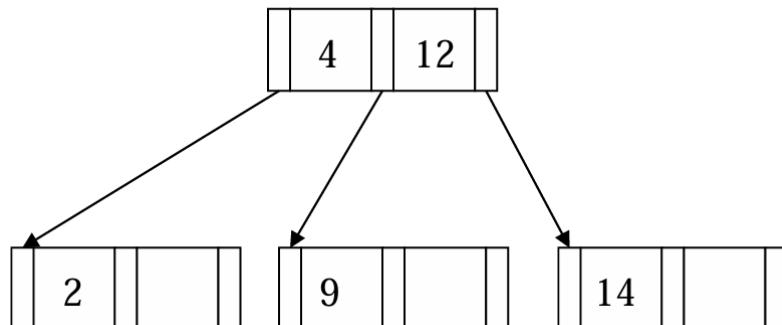
Risultato:



## Esercizio di inserimento chiave (semplice)

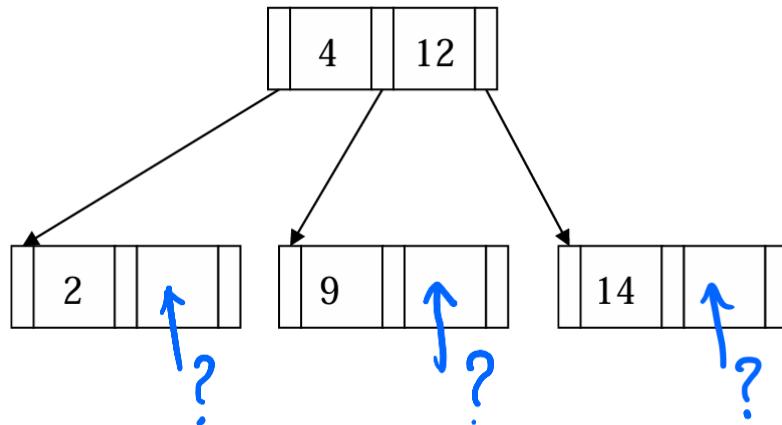
*Esame di Settembre 2022*

Viene fornito il seguente B-Tree di **ordine (g) = 1**:

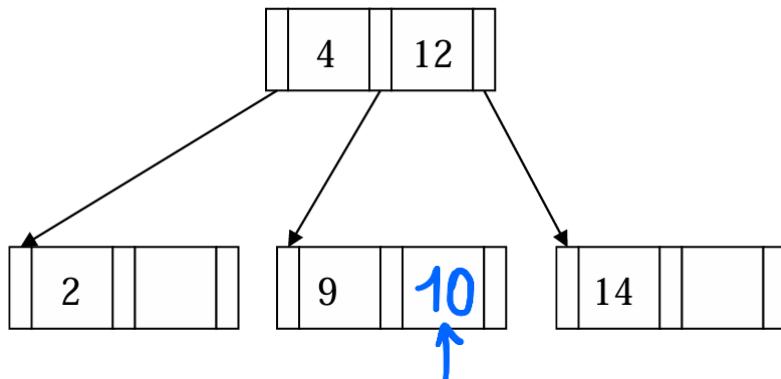


Viene richiesto di riportare la struttura a fronte dell'inserimento della chiave **10**.

1. Notiamo che la chiave da inserire può semplicemente essere messa in uno degli spazi liberi, in una delle foglie, senza dover alterare la struttura dell'albero



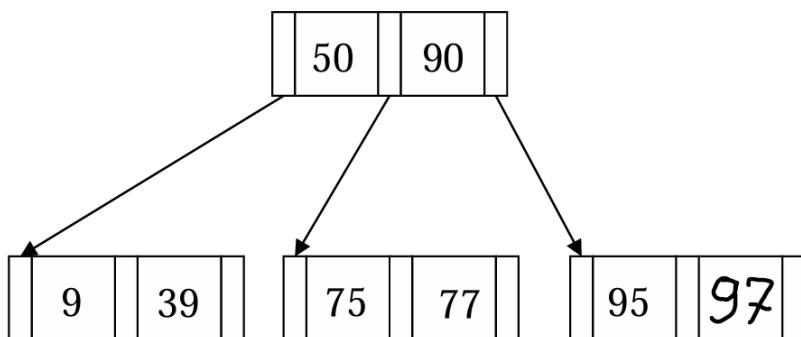
2. Scegliamo lo spazio libero stando solo attenti a rispettare il **Vincolo del Maggiore/Minore**. È evidente che l'unica posizione ammissibile è quella indicata di seguito:



## Esercizio di inserimento chiave (complesso)

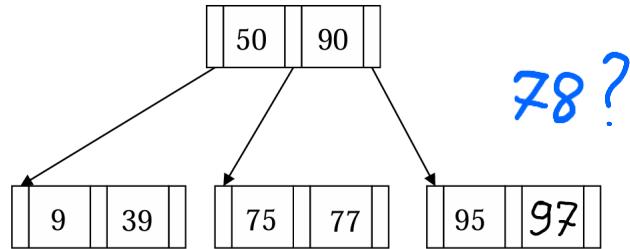
*Esame di Giugno 2024*

Viene fornito il seguente B-Tree di **ordine (g) = 1**:

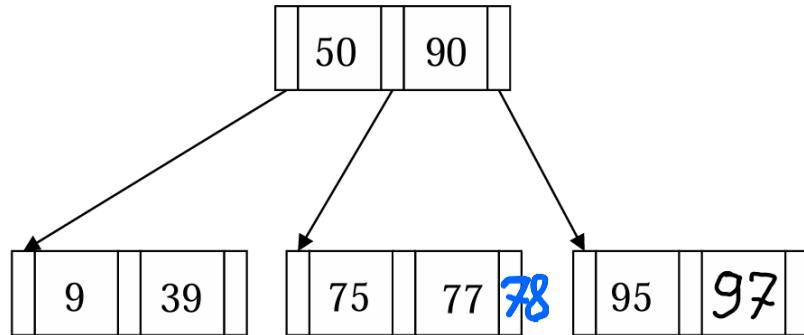


Viene richiesto di riportare la struttura a fronte dell'inserimento della chiave **78**.

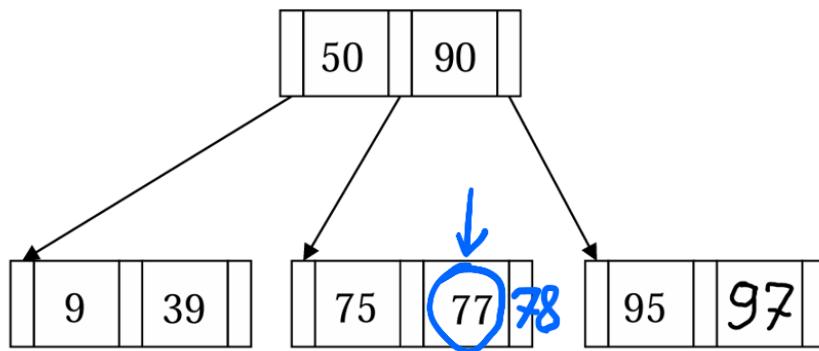
1. Notiamo che la chiave da inserire non può essere aggiunta facilmente, poiché abbiamo già esaurito tutti gli spazi disponibili nelle foglie



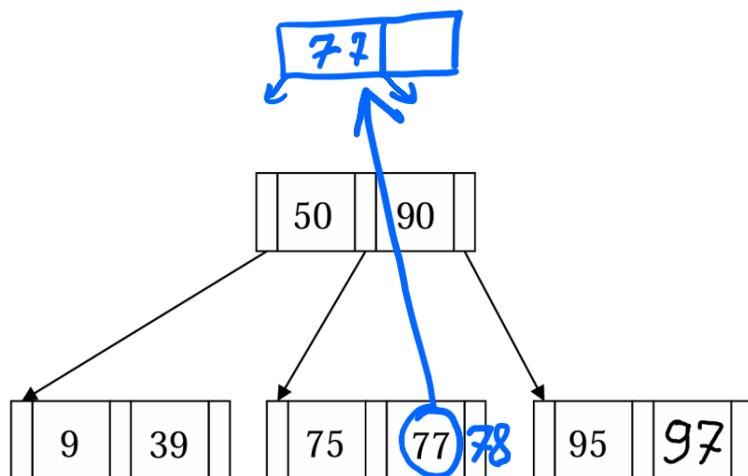
2. Immaginiamo ora di inserire la nostra chiave come terzo elemento nella foglia centrale



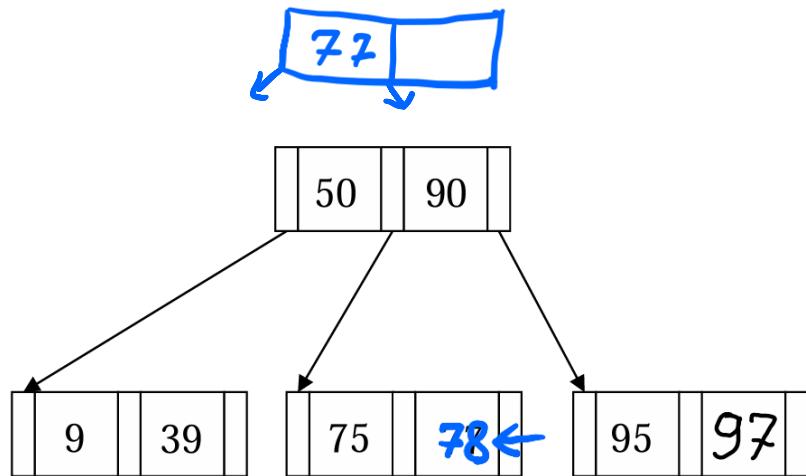
3. Usiamo la “**regola del più a destra**” per scegliere e lavorare sulla chiave **77** della foglia centrale



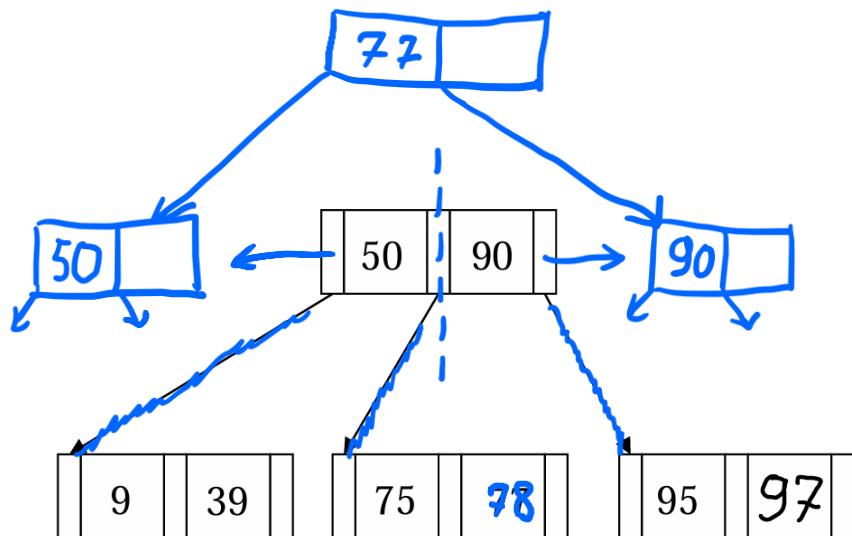
4. “**Pushamo in alto**” il 77, facendolo diventare il nodo radice, **aumentando così l'altezza dell'albero**



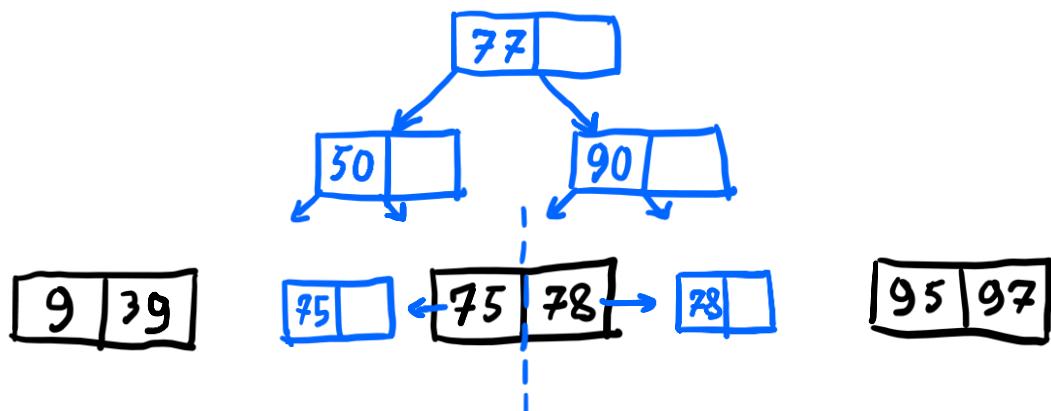
5. Adesso che la postazione del **77** si è liberata, mettiamoci il nostro numero da inserire (**78**)



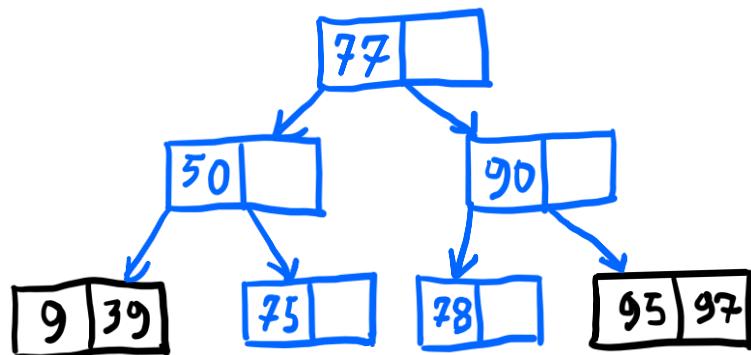
6. Adesso il nuovo nodo radice deve avere per forza due figli, uno a sinistra e uno a destra. Ma attualmente abbiamo solo un figlio disponibile, perciò, non ci resta che **splittare** il nodo che ora si trova al livello (altezza) 2



7. Adesso, per sistemare l'ultimo livello (quello delle foglie), occorre che i nuovi nodi derivati dallo split di prima abbiano in totale 4 figli, 2 ciascuno. Ma, in questo momento, noi abbiamo solo 3 foglie. La soluzione è intuitiva: **splittiamo la foglia centrale**, che contiene 2 chiavi, in modo da ritrovarci con 4 nodi foglia



8. Aggiustiamo coerentemente i puntatori a ciascun figlio, e poi abbiamo finito



9. Infine, possiamo verificare ad occhio il rispetto dei vincoli di maggiore/minore, ma è evidente che in questo caso è già così

