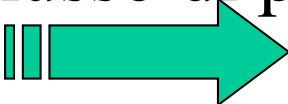


Organizzazione fisica dei dati

2 punti di vista

- Come costruire un DB ?

Flusso di progetto, schemi
logici  livello logico

- Come memorizzare i dati ?

Struttura dei dati
 livello fisico

DBMS : Architettura a livelli (Strati della conoscenza)

- Vista utente

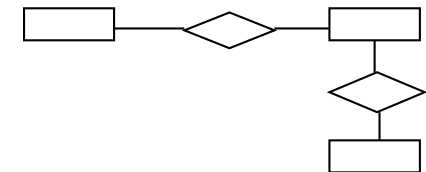


- Schema esterno : TABELLE



- Vista amministratore DB o azienda

- Schema concettuale : Schema E-R o RM/T



- Vista fisica

- Schema interno : memorie, buffer, tipi file, indici, alberi, 'motori'

Organizzazione fisica dei dati

- L'organizzazione fisica dei dati di un database deve essere **efficiente**, ciò significa che il sistema di gestione di un database (DBMS) deve avere la capacità di rispondere alle richieste dell'utente il più velocemente possibile.

Organizzazione fisica dei dati: perché studiarla?

- I DBMS offrono i loro servizi in modo "trasparente" e permettono di farci ignorare le proprie strutture fisiche
 - Il DBMS utilizza in maniera efficiente le strutture
 - abbiamo considerato il DBMS come una "scatola nera"
- Perché aprirla?
 - capire come funziona può essere utile per un migliore utilizzo
 - Non sempre il "default" è meglio!

Gestore degli accessi e delle interrogazioni



Vedi figura 1.1

del libro di riferimento del corso

La descrizione che segue
approfondirà lo schema precedente
dal basso verso l' alto.....

Caratteristiche memorie secondarie

- Si chiamano così perché i dati, per essere letti o scritti, devono essere trasferiti nella memoria principale.
- Organizzate in **‘blocchi’**.
- Lettura/scrittura solo sull’ intero blocco.
- Costo delle operazioni in mem. principale trascurabile rispetto a quello di accesso a un blocco di mem. secondaria

I 'Blocchi'

- Un disco è partizionato in blocchi
- I Blocchi hanno formato fisso (compreso tra 512 e 4096 Byte)
- Un blocco corrisponde in genere al settore di una traccia di disco
- Il tempo di risposta si valuta in numero di accessi al disco e dipende da :
 - tempo di **posizionamento della testina** (10-50ms)
 - tempo di **latenza** (5-10ms)
 - tempo di **trasferimento** (1-2ms)

Il Buffer

- Apposita zona di mem. centrale che, nei DBMS, agisce da interazione tra mem. secondaria e mem. principale.
- Organizzato in pagine con un numero intero di blocchi
- Con la riduzione dei costi si hanno buffer sempre più grandi.

Organizzazione delle tuple nelle pagine



Vedi figura 1.2

del libro di riferimento del corso

Gestore del buffer

- Si occupa del caricamento/scaricamento pagine, decidendo inoltre cosa farne.
- Segue la legge del ‘data location’ (20/80).
- Una directoy descrive il contenuto corrente del buffer
- Ogni pagina ha delle variabili di stato (contatore, bit di stato)
- Il gestore del buffer segue un algoritmo preciso

Organizzazione fisica dei file

- I dati sono organizzati in **file**
 - I file sono divisi in **record**
 - **Record logici**, quelli visibili
 - **Record fisici**, con informazioni del record e campi

INFORMAZIONI	CAMPI
---------------------	--------------

Organizzazione fisica dei file (2)

- **INFORMAZIONI**
 - **Puntatore** al record
 - Tipo record
 - Lunghezza record
 - Bit di cancellazione
 - Eventuali Offset dei campi
 - Eventuale puntatore al prossimo record

Un **puntatore** è una coppia (b,k)

Organizzazione fisica dei file (3)

- CAMPI
 - Eventuale offset del campo
 - Campo vero e proprio

Piccolo esempio per non perdere di vista l' obiettivo.....

- Limitare la dimensione dei file separando gli attributi con accesso frequente da quelli con minor accesso semplifica la gestione del buffer :
 - Si consideri R con $T=500.000$ tuple con attributi $K, A1, A2$ ognuno lungo $a=5$ byte
 - Si consideri un sistema con blocchi di dimensione $B=1$ Kb
 - Se le operazioni più frequenti chiedono la proiezione su uno solo dei due attributi R diventa $R1(K,A1)$ e $R2(K,A2)$
 - Ognuna di $T/(B/2a) = 5000$ blocchi ---> proiezione = 5000 accessi. Altrimenti sarebbero 7500 accessi

Strutture primarie per l'organizzazione dei file e metodi di accesso

Criteri secondo i quali sono disposte le tuple nell'ambito del file \implies Tipi di file.

3 Tipi di file :

- Sequenziali (**Heap**)
- Ad accesso calcolato (**Hash**)
- Ad albero (**Isam, Btree, B+tree**)

Tipi di file

Strutture sequenziali

- File **Heap** (mucchio)
 - Record inseriti nei blocchi senza ordine, seriale e disordinata
 - Inserimenti efficienti, cancellazioni e aggiornamenti no ---> proliferazione di indirizzamenti indiretti
 - Accesso ai record tramite directory di puntatori ai blocchi
 - Scansione sequenziale
 - > buono per letture/scritture sequenziali
 - > cattivo per ricerche singole

Tipi di file

Strutture sequenziali (2)

- Strutture sequenziali ad array (clustered)
 - Tuple di dimensione fissa e occupazione blocchi già nota in partenza
 - Non utilizzate per i DBMS
- Strutture sequenziali ordinate
 - Ordinate in base al valore di una chiave
 - Utilizzate solo in associazione con indici (vedi dopo su ISAM)

Tipi di file

Strutture ad accesso calcolato

- File **Hash**
 - La locazione fisica dei dati dipende dal valore della chiave
 - Record ripartiti in ‘bucket’ in base al valore della chiave
 - Ogni bucket ha 1 o più blocchi ed è organizzato come un Heap
 - I blocchi di un bucket sono collegati mediante puntatori a lista
 - Importanza di una funzione Hash “buona”

Tipi di file

Strutture ad accesso calcolato (2)

Funzione Hash

- Funzione hash buona se ripartisce uniformemente i record nei bucket (stessa probabilità di accesso) --> equidistribuzione
- Esempio : $F = \text{resto}(I/B)$ dove
 - I = chiave normalizzata in un intero
 - B = numero bucket
- Esempio : 40 studenti e numero di matricola di 6 cifre (un milione di possibili chiavi)

Un esempio



Vedi figura 1.3

del libro di riferimento del corso

Un file hash



Vedi figura 1.4

del libro di riferimento del corso

Tipi di file

Strutture ad accesso calcolato (3)

Ottimizzazione funzione **Hash**

- Tanti bucket ----> più basso il costo di ogni operazione
-ma la bucket directory deve entrare in memoria principale
- ...e devo minimizzare le catene di overflow

- Formula : $B = T / (f \times F)$ dove
 - **B** = numero bucket
 - **T** = numero tuple previste in un file
 - **f** = frazione di spazio fisico da utilizzare
 - **F** = fattore di blocco

Tipi di file

Strutture ad accesso calcolato (4)

Conclusioni

- È l'organizzazione più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza (accesso puntuale): costo medio di poco superiore all'unità (il caso peggiore è molto costoso ma talmente improbabile da poter essere ignorato)
- Non è efficiente per ricerche basate su intervalli (né per ricerche basate su altri attributi)
- I file hash "degenerano" se si riduce lo spazio sovrabbondante: funzionano solo con file la cui dimensione non varia molto nel tempo

Tipi di file

Strutture ad albero

File ad indice

- L'organizzazione ad albero realizza sia strutture primarie (con dati), sia secondarie (accedono ma non hanno dati)
- L'indice secondario di un file è un file contenente il valore della chiave e il relativo indirizzo fisico
- L'indice primario di un file è un file che contiene la chiave primaria e i dati
- Una sola chiave primaria, ma tanti indici secondari

Tipi di file

Strutture ad albero

File a indice ISAM

ISAM (indexed sequential access method), file con indice sparso

- Record ordinati in crescita (lasciando il 20% dei blocchi liberi)
- File indice formato da coppie (v,b)
- Directory del file indice + tanti file indici
- Metodi di scansione
 - Ricerca binaria o dicotomica
 - Ricerca per interpolazione

Tipi di file

Strutture ad albero

File ISAM - Ricerca binaria

- File indice formato da coppie (v, b)
 - dove in b chiavi $\geq v$ e in $b-1$ chiavi $< v$
- Se $B_1 \dots B_m$ blocchi file indice e w chiave da cercare

B_1	v_1	b_1
$B_{m/2}$	$v_{m/2}$	$b_{m/2}$
B_m	v_m	b_m

se $w < v_{m/2}$ ripeto ricerca con $B_1 - B_{m/2}$
 se $w = v_{m/2}$ trovato, scandisco tutto $B_{m/2}$
 se $w > v_{m/2}$ ripeto ricerca con $B_{m/2} - B_1$

Itero il procedimento fino a ridurmi ad unico blocco $(v_k < w < v_{k+1})$ che scandirò tutto (B_k)

Tipi di file

Strutture ad albero

- I file Isam sono tipici delle chiavi primarie o indici primari \implies ad ‘indice sparso’ perché non tutti i valori della chiave sono nell’ indice
- Un indice secondario invece deve contenere i riferimenti fisici per tutti i valori della chiave \implies ad ‘indice denso’ perché i valori possono non essere consecutivi
 - La ricerca puntuale su indice secondario è più efficiente

Indice primario “sparso”



Vedi figura 1.6

del libro di riferimento del corso

Indice secondario “denso”



Vedi figura 1.7

del libro di riferimento del corso

Tipi di file

Strutture ad albero

BTree

Btree (Binary? Balanced? Bayer? Boeing? Tree)

- Generalizzazione del file indice ISAM
- Il concetto dell' albero evita spreco di spazio
 - in (v,b) b può puntare ai blocchi di altro file indice
- Strutture dinamiche e quindi efficienti anche per gli aggiornamenti

Indice primario multilivello (non molto dinamico)



Vedi figura 1.8

del libro di riferimento del corso



Indice secondario multilivello

**Vedi figura 1.7
del libro di riferimento del corso**

Tipi di file

Strutture ad albero

Btree - Funzionamento

- Ogni nodo coincide con un blocco e ha più successori
- Gli alberi devono essere ‘bilanciati’ per funzionare bene
 - Profondità dell’ albero costante in ogni ramo
- Ogni nodo intermedio ha una sequenza ordinata dei valori della chiave e contiene n chiavi e $n+1$ puntatori
 - Dove n è il massimo possibile tale da ridurre il numero di livelli dell’ albero

Organizzazione dei nodi del B-tree



**Vedi figura 1.9
del libro di riferimento del corso**

Tipi di file

Strutture ad albero

Btree - Efficienza

- Altamente efficienti sulle ricerche (proporzionale alla profondità dell' albero)
- Efficienti negli aggiornamenti ed inserimenti
-pochissimo efficienti sugli accessi per intervalli di chiavi

Tipi di file

Strutture ad albero

Btree - Tecniche di accesso

- Per le ricerche --> stesso concetto di ricerca dicotomica, partendo dalla radice ed arrivando ai nodi foglia
- Per inserimenti ----> in mancanza di spazio operazione di 'split' che può propagarsi verso l'alto
- Per cancellazioni ----> operazione 'merge' che agisce al contrario

Se split e merge funzionano correttamente ==>
riempimento medio di ciascun nodo è del 70 %

Split e merge



**Vedi figura 1.10
del libro di riferimento del corso**

Tipi di file

Strutture ad albero

B+tree

- Come i Btree e in più vengono collegati tra di loro, sequenzialmente, tramite puntatori (lista), i nodi foglia
- questo rende efficiente la ricerca su intervalli di valori.

Un B+ tree



**Vedi figura 1.11
del libro di riferimento del corso**

Un B-tree



**Vedi figura 1.12
del libro di riferimento del corso**

Tipi di file

Strutture ad albero

Rtree - indici spaziali

- Utilizzati per interrogare i dati (oggetti) in base alle loro coordinate geografiche.
- Utilizzano il concetto di MBR (Minimum Bounding Rectangle)
- Esistono anche in MySQL

Tipi di file

Caratteristiche comuni nei DB

- In tutti i sistemi è possibile definire indici, sia primari che secondari
- Alcuni sistemi prevedono la possibilità di memorizzare in modo contiguo tuple di una tabella con gli stessi valori su un certo campo ==> tecnica del 'cluster'
- La sintassi generica per costruire indici :
 - Create [unique] index <nomeindice> on
<nometab> (lista attributi);

Come scegliere tipi file e indici da usare ?

- Si ricordi che :
 - Una struttura Hash è efficiente per accessi puntuali, ma peggiore di una a indice per ricerche su intervalli
 - Il costo del singolo accesso può variare per la contiguità dei blocchi
 - L' ottimizzatore delle query può fare di testa sua
- eventuale esempio.....
- Consigli :
 - Aggiungere indici è sempre un aggravio per gli aggiornamenti
 - Fare comunque attività di 'tuning'

MySQL

Schema optimization and indexing

- È importante saper scegliere :
 - Tipi di dati
 - Normalizzazione/Denormalizzazione
 - Indici/keys : efficienti per il recupero dati ma critici per una buona performance

MySQL

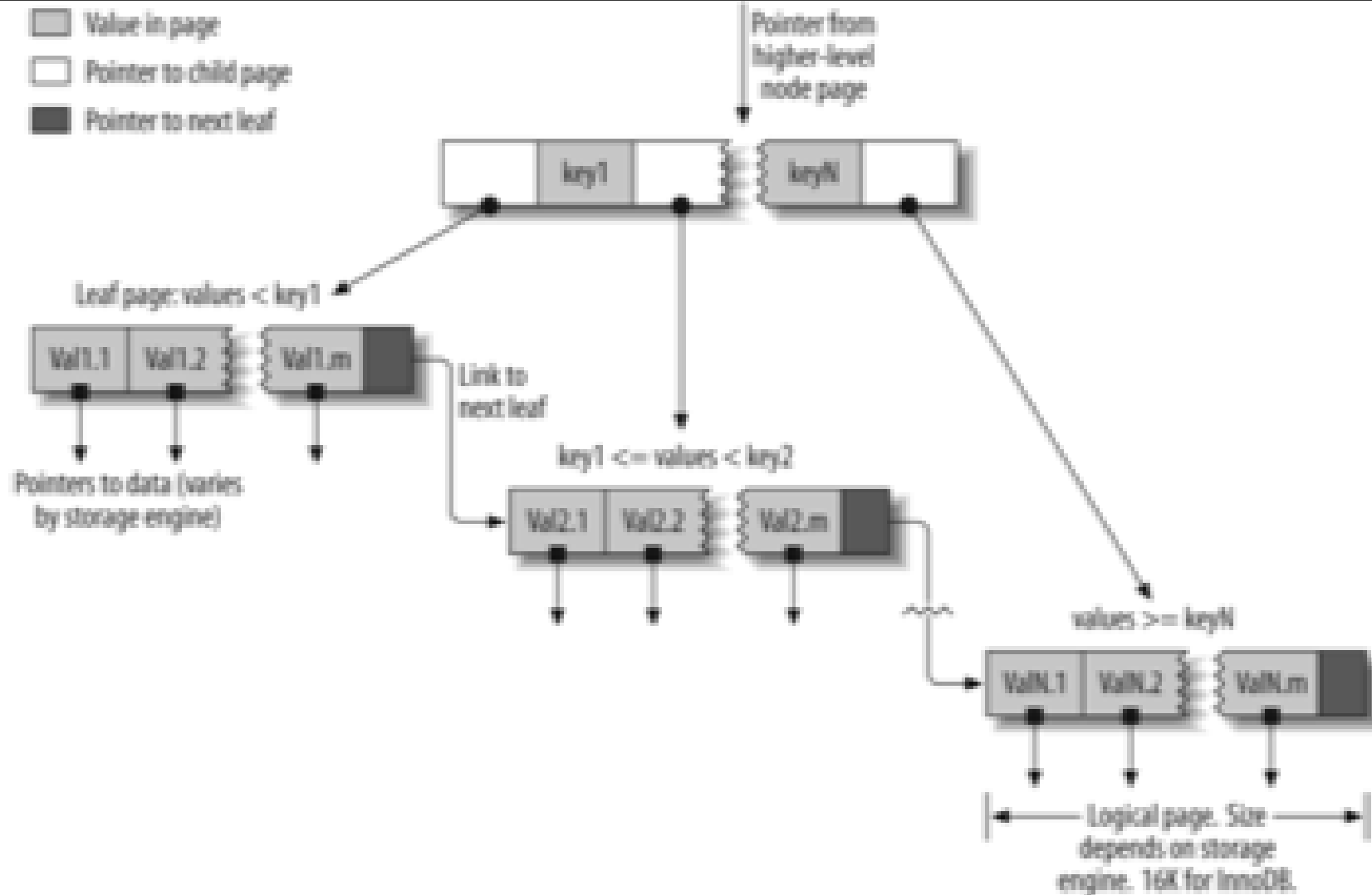
Indici

- Con campi multipli è importante l'ordine
 - Mysql cerca efficientemente sul prefisso più a sinistra
- Gli indici sono implementati a livello di storage engine , non di server

MySQL

Btree e B+tree

- MyIsam engine usa i Btree sulle righe indexate, mentre InnoDB fa a loro riferimento per i valori della chiave primaria
- Ciascuna foglia è alla stessa distanza dalla radice
- MyIsam usa delle tecniche di compressione per rimpicciolire gli indici
- Il tipo di puntatori varia a seconda dello Storage Engine usato



MySQL Hash

- Utile solo per ricerche esatte
- Per ogni riga lo Storage Engine calcola un “hash code”
- Supportato solo dal Memory Storage Engine
 - dove gli indici hash sono molto compatti
- Si possono costruire hash index ad hoc (tramite i trigger)

MySQL

indici spaziali

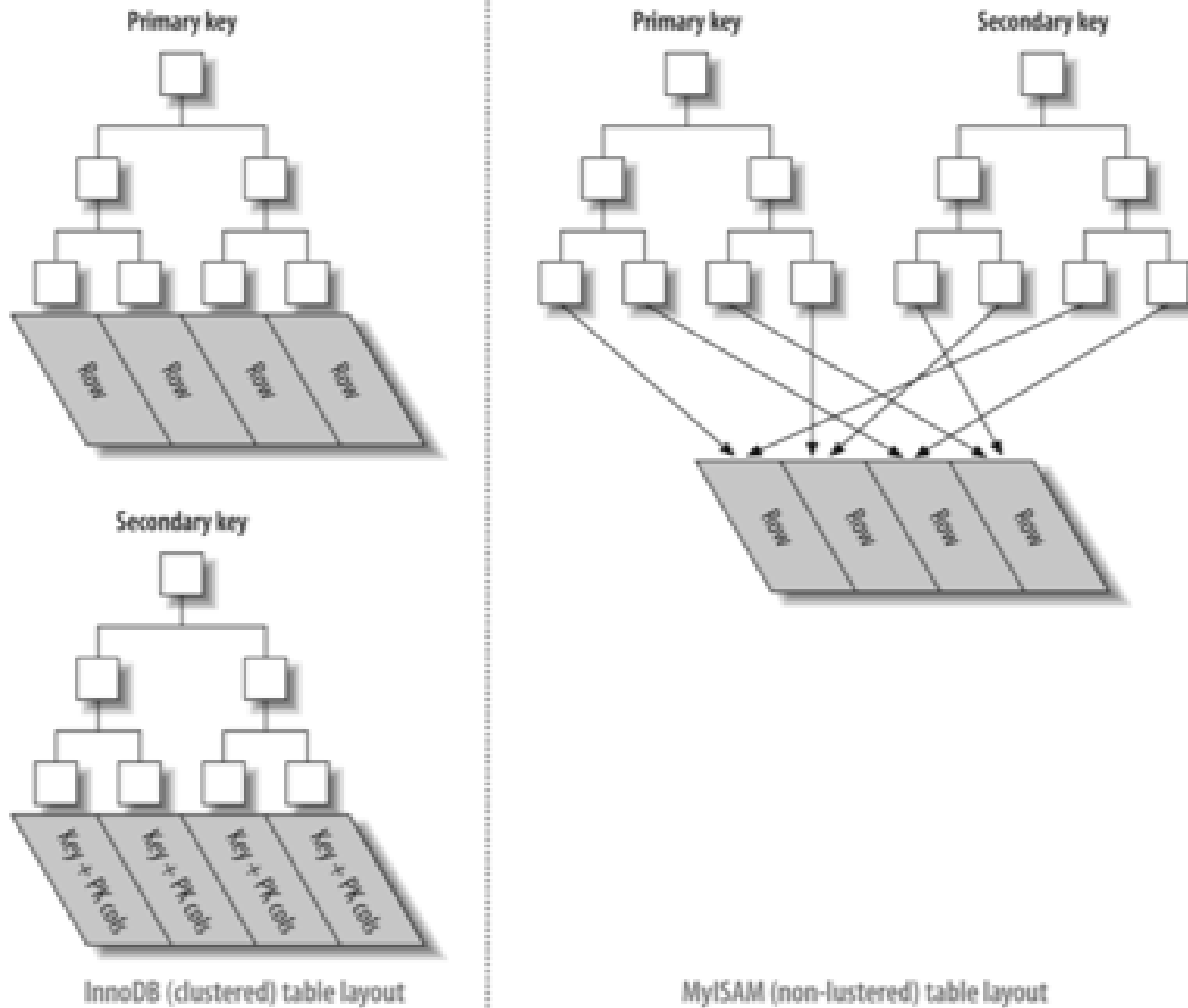
- Supportati solo da MyIsam engine tramite le MySQLGIS functions --> MBRCONTAINS()

MySQL

indici clustered

- Non sono un tipo di indici ma piuttosto un approccio alla memorizzazione
- Si basano sui Btree e sono supportati dai motori InnoDB e SolidDB
- In InnoDB le primary key sono memorizzate come clustered index, mentre gli indici secondari no.
- MyIsam invece non fa differenza tra chiavi primarie e secondarie, sono tutte non clusterizzate

Figure 3-9. Clustered and nonclustered tables side-by-side



Notice the autoincrementing integer primary key.

MySQL

Normalization/Denormalization

- Gli schemi normalizzati sono i modelli migliori
- Però rendere efficiente il sistema può comprendere denormalizzare lo schema
- Note :
 - È comunque difficile valutare l'efficienza solo dal tempo di esecuzione
 - L'Sql è comunque un linguaggio procedurale

MySQL

Normalization/Denormalization

- La normalizzazione :
 - evita la ridondanza dei dati
 - evita l' inconsistenza dei dati (anche temporanea)
 - usa tabelle più piccole e select più facili
- **MA.....**

MySQL

Normalization/Denormalization

-l' operatore di join è pesante e costoso
- ...e denormalizzare evita parecchi join

- La scelta giusta sta nella via di mezzo.