Databázové systémy

Indexy, Performance

obrázky z knihy

Elmasri, Navathe: Fundamentals of Database Systems, fourth edition, Pearson Education, 2004

Markus Winand: SQL Performance Explained, 2012

Kde sa ukladajú dáta?

- Primary storage
  - rýchle
  - drahé a preto malá kapacita
  - volatilné (až na flash)
- Secondary storage
  - pomalé a p o m a l é
  - lacné a preto vysoká kapacita
  - naozaj perzistentné (non-volatilné)
- CPU pracuje len s primary storage

Kde sa ukladajú dáta?

- menšie databázy (gigabajty) celé v pamäti
- väčšie databázy na disk
  - ešte nemyslime na všemožné distribuované NoSQL
- na disku v súboroch
- záznamy by v súboroch mali byť uložené nejako rozumne
  - tak, aby sme ich vedeli efektívne lokalizovať, keď je to potrebné

Uloženie dát

- Primárna organizácia dát
  - determinuje fyzické uloženie dát
    - a teda aj spôsob prístupu k nim
- Sekundárna organizácia (prídavná)
  - umožňuje efektívny prístup k zaznámom cez iné atribúty ako tie, podľa ktorých sa vedie primárna organizácia dát
  - toto su tie indexy, ku ktorým to celé smeruje :)

Uloženie dát na disku

- Halda (Heap file, unordered file)
- Usporiadaný súbor (Sorted file)
  - utriedené podľa kľúča (príslušného atribútu)
- Hašovaný súbor (Hashed file)
  - hašovacia funkcia nad atribútom pre určenie umiestnenia
- Stromová štruktúra

Halda

- záznamy sú uložené v poradí v akom boli vkladané
  - super rýchle vkladanie nových záznamov
  - problém so všetkým ostatným
  - potreba pravidelnej defragmentácie
- vyhľadávanie
  - Ak má súbor b blokov, tak v priemere musíme načítať b/2 blokov aby sme získali blok s hľadaným záznamom

Usporiadaný súbor

- usporiadané podľa atribútu
  - vo všeobecnosti môže, ale nemusí byť UNIQUE
  - v skutočnosti to býva PRIMARY KEY (MySQL)
- čítanie
  - super ak v app. využívame usporiadanie z disku
  - next value väčšinou v rovnakom bloku
    - ktorý je už načítaný v pamäti
  - binárne vyhľadávanie podľa triediaceho atribútu
    - $\log_2(b)$

Binary search z wikipedie

```
int binary_search(int A[], int key, int min, int max)
{
   if (max < min): return KEY_NOT_FOUND; endif;
   int mid = (min + max) / 2;
   if (A[mid] > key):
        return binary_search(A, key, min, mid-1);
  else if (A[mid] < key):
        return binary_search(A, key, mid+1, max);
  else:
        return mid;
  endif;
}
```

Usporiadaný súbor

- Zapisovanie
  - veľmi drahé
  - v priemere musím poposúvať polovicu záznamov aby som spravil miesto pre nový záznam
- Mazanie
  - was_deleted marker
- overflow/transaction file
  - zapisujem do pomocného súboru (halda)
  - ten je periodicky sortovaný a spájaný s hlavným súborom
  - trošku komplikovanejšie vyhľadávanie

Hašovaný súbor

- hašovacia funkcia nám vráti adresu bloku, ktorý obsahuje požadovaný záznam
  - v skutočnosti adresu tzv. bucketu, ktorý obsahuje tento blok
- rýchle vyhľadávanie
  - podmienka vyhľadávania musí byť rovnosť (==)

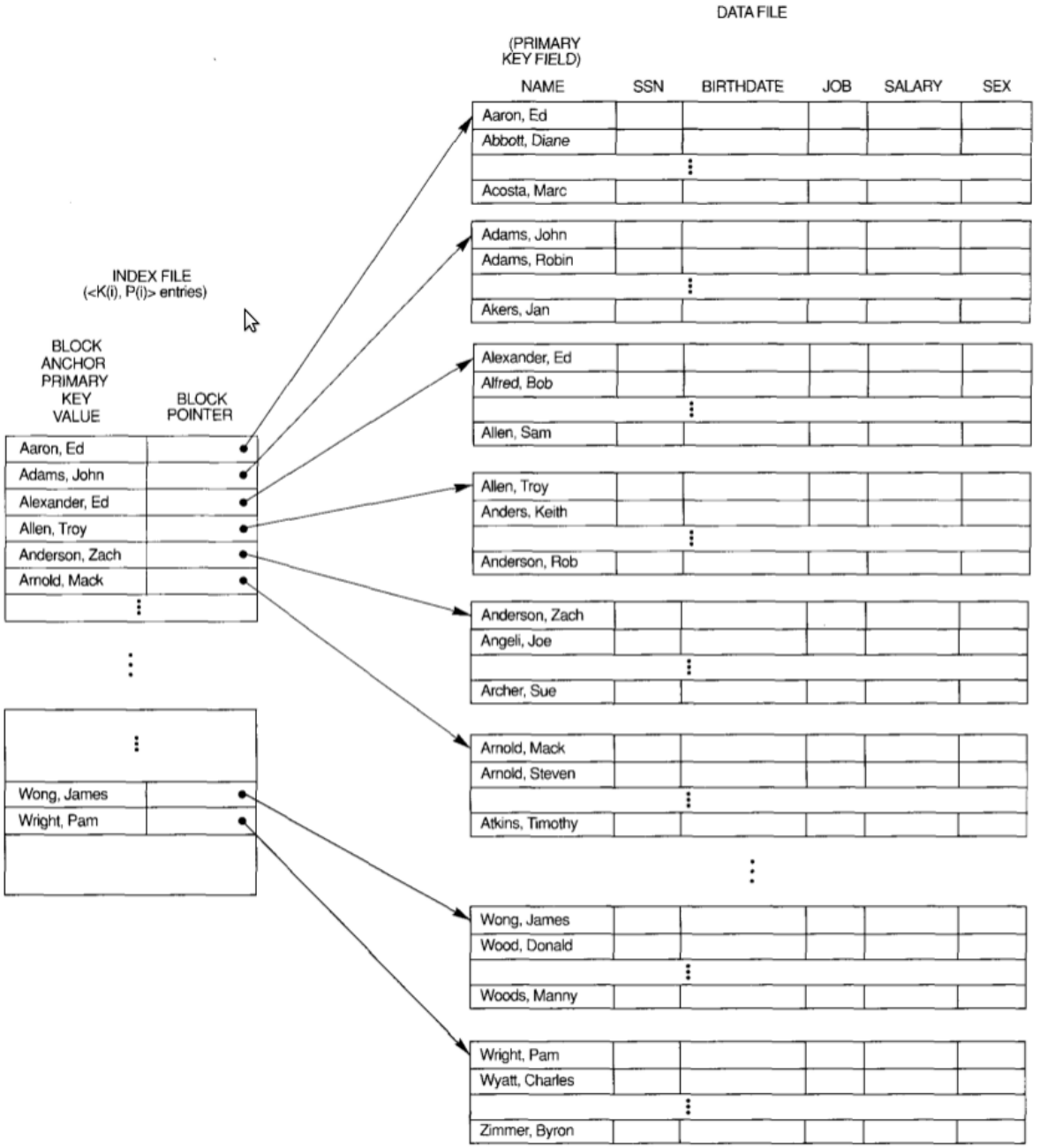
Indexy

- Motivácia rovnaká ako pre index na konci učebnice
- Zaberajú miesto
  - niekedy veľa miesta
- Je potrebné ich udržiavať
  - učebnica sa vytlačí a je pokoj, dáta sa však neustále menia

Primárny index

- jeden záznam v indexe pre každý blok
- každý záznam v indexe obsahuje hodnotu primárneho kľúča pre prvý záznam v bloku

Primárny index



Indexy všeobecne

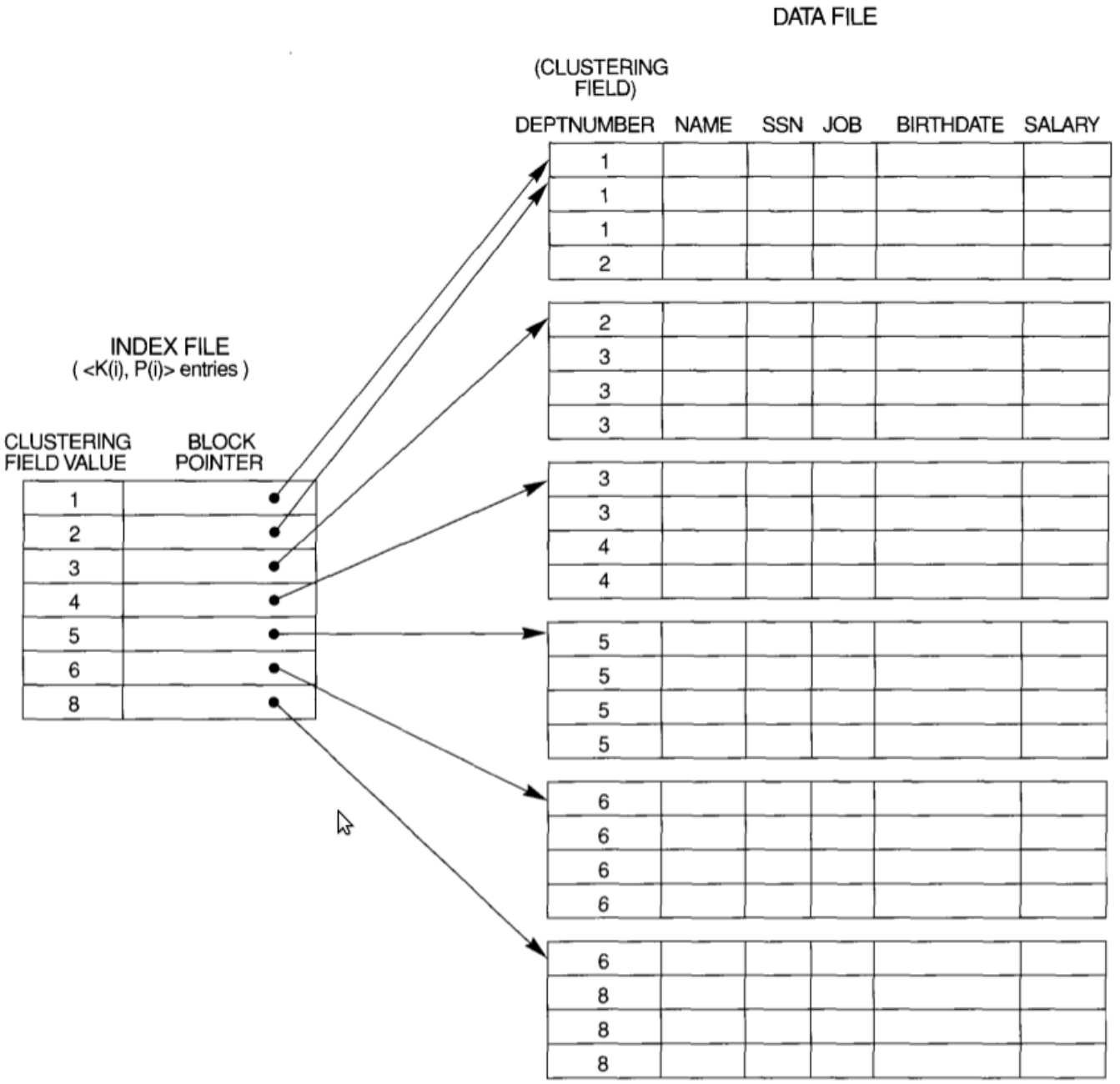
- menej záznamov v indexe ako je záznamov v tabuľke
  - nemusí platiť vždy
- menšie záznamy
  - iba vybraný atribút

- index teda zaberá menej blokov
  - aj binary search teda bude rýchlejší
- Vyhľadávanie
  - vyhľadanie bloku (binary search) + samotný prístup k bloku

Zhlukovací index

- v prípade, že sa mi môžu hodnoty opakovať
  - nemám primary key
  - mám atribút, podľa ktorého sú záznamy sortované
- v indexe mám záznam pre každú možnú hodnotu
  - hodnota – blok, v ktorom je prvý záznam s touto hodnotou
- úprava: každá hodnota má vyhradený svoj blok

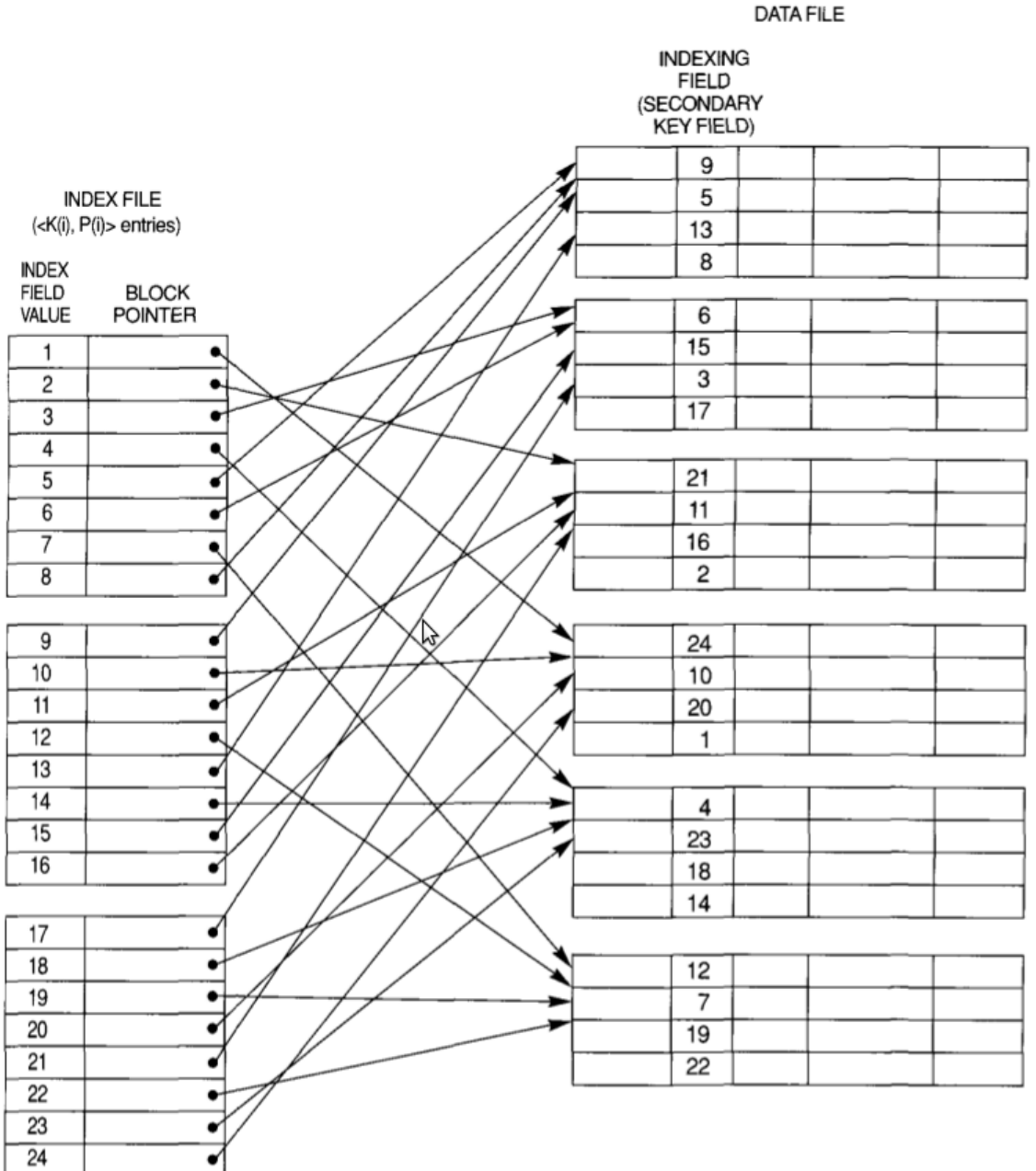
Zhlukovací index



Sekundárny index

- Dodatočná štruktúra
  - robím index podľa niektorého atribútu
  - zabezpečím si binárne namiesto lineárneho vyhľadávania
- záznam v indexe pre každý záznam v tabuľke
  - keďže záznamy v tabuľke nie sú zoradené podľa indexovaného atribútu
  - čo v prípade, že atribút nie je UNIQUE?
    - index neukazuje na záznam v tabuľke, ale na pole pointrov na záznamy v tabuľke

Sekundárny index

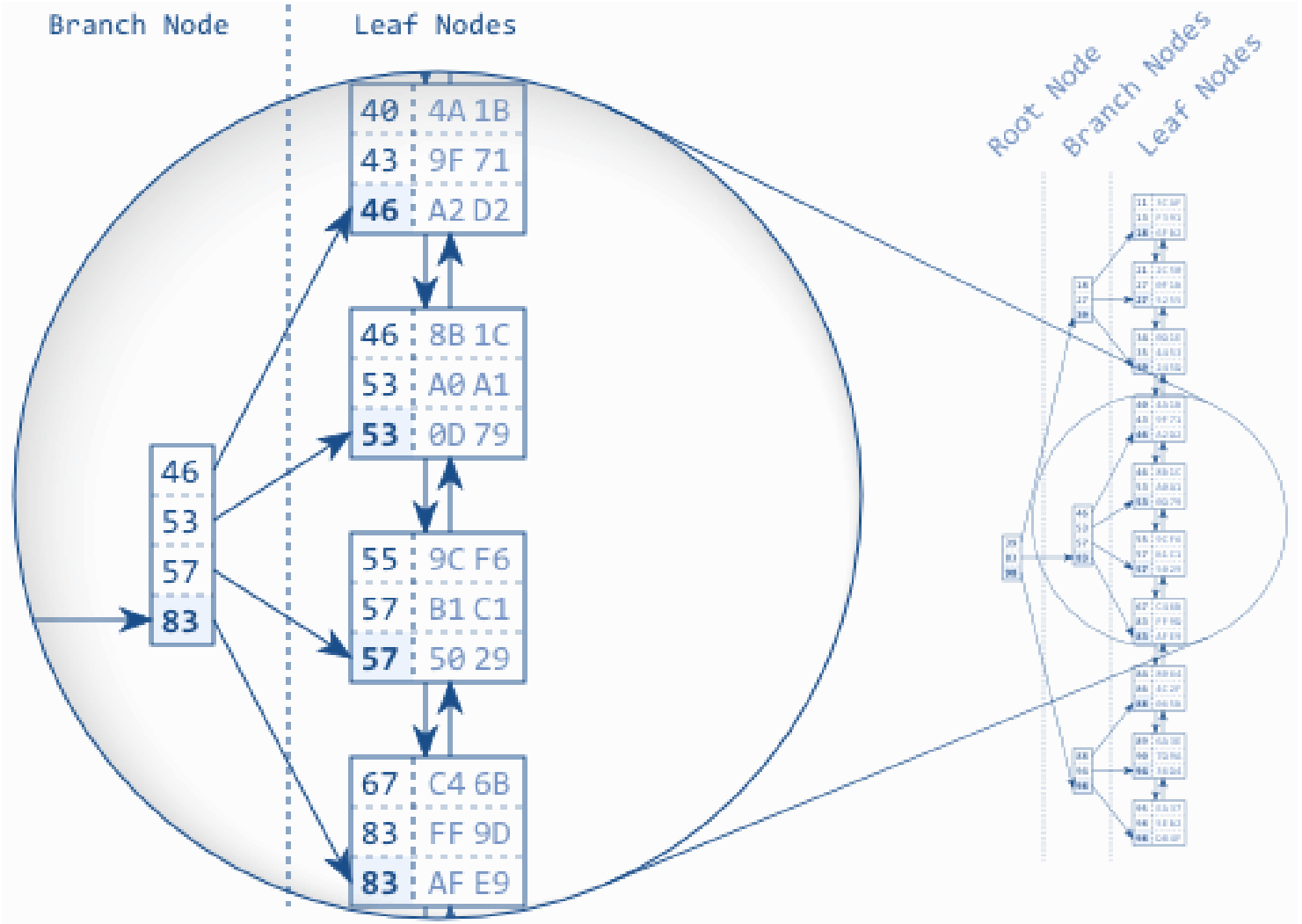


Viac-úrovňový index

- Ak sa nám index nezmestí do jedného bloku, vieme ho indexovať
  - a toto opakovať

- V DB sa používa B+ tree

B^+-Tree


Branch Node
Leaf Nodes
40 4A 1B
43 9F 71
46 A2 D2
46 8B 1C
53 A0 A1
53 0D 79
55 9C F6
57 B1 C1
57 50 29
67 C4 6B
83 FF 9D
83 AF E9
46
53
57
83
Root Node
Branch Nodes
Leaf Nodes

Ako potom vyzerá vyhľadávanie

- Index access – prechod stromom k listom
  - Toto nie je nikdy problém, strom je balanced
- Index range scan – prehľadávanie zoznamu listov
  - UNIQUE?
  - Ak toho musím prejsť veľa, tak je to problém
- Table access – vytiahnutie dát z tabuľky
  - Ak toho musím prejsť veľa, tak je to problém

Index nad viacerými atribútmi

- Záleží na poradí
- Sortovanie podľa prvého
  - v rámci jednej hodnoty prvého sortované podľa druhého
- Záleží na poradí!!
  - Najprv chcete prehľadávať strom, až potom listy

- covering index
  - id, name
    - a nemusím sa vôbec chodiť pozerať do dát

Kde sa využije index?

- pre rýchle nájdenie riadkov, ktoré vyhovujú `WHERE` podmienke
  - Ale nie ľubovoľnej WHERE podmienke
- pre elimináciu uvažovaných riadkov vyberá index, ktorý nájde najmenší počet riadkov
- pre získanie riadkov z inej tabuľky pri vykonávaní `JOIN`u
  - je vhodné aby boli stĺpce rovnakého typu, bez nutnosti konverzií
- pre získanie MAX(), MIN()
- pri sortovaní

Zložený index, covering index

majme index (col1,col2,col3)

```
SELECT * FROM tbl_name WHERE col1=val1;

SELECT * FROM tbl_name WHERE col1=val1 AND
col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;

SELECT * FROM tbl_name WHERE col2=val2 AND
col3=val3;
```

B-Tree index a LIKE

```
SELECT * FROM tbl_name WHERE key_col
LIKE 'Patrick%';

SELECT * FROM tbl_name WHERE key_col
LIKE 'Pat%_ck%';


SELECT * FROM tbl_name WHERE key_col
LIKE '%Patrick%';

SELECT * FROM tbl_name WHERE key_col
LIKE other_col;
```

- aby využil index musí byť konštanta

Indexovanie funkcie

```
SELECT first_name, last_name, phone_number
FROM employees
WHERE UPPER(last_name) = UPPER('winand')
#btw, I really recommend the book
```

```
CREATE INDEX emp_up_name
    ON employees (UPPER(last_name));
```

~~MySQL~~

Pozor na nedeterministické funkcie!

EXPLAIN

EXPLAIN [ANALYZE] statement

http://explain.depesz.com/

Čo ešte vplýva na výkon?

- Cachovanie
- Nastavenie buffrov

Cache v PostgreSQL

- shared buffers
  - default je dosť nepoužiteľný (málo)
  - príliš veľa však tiež nie je dobré
    - spoliehame sa na OS, že cachuje
  - pokročilé LRU
    - jednorázový sekvenčný scan celej tabuľky mi úplne zruší drahocennú cache
    - second chance (unsuitable for eviction bit pri prístupe)
    - clock sweep s usage_count

Zhrnutie

- Použitie indexov môže významne urýchliť vyhľadávanie záznamov
- Nie je to zadarmo
  - Réžia pri vkladaní, mazaní
  - Query planner
  - Zaberajú miesto

- http://use-the-index-luke.com/ !!!!