Databázy 1

Indexy, Performance

Povinná literatúra

Markus Winand: SQL Performance Explained, 2012

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

Kde sa ukladajú dáta?

- Primary storage
  - rýchle
  - drahé a preto malá kapacita
  - volatilné (až na flash)
- Secondary storage
  - pomalé a pomalé
  - 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 :)

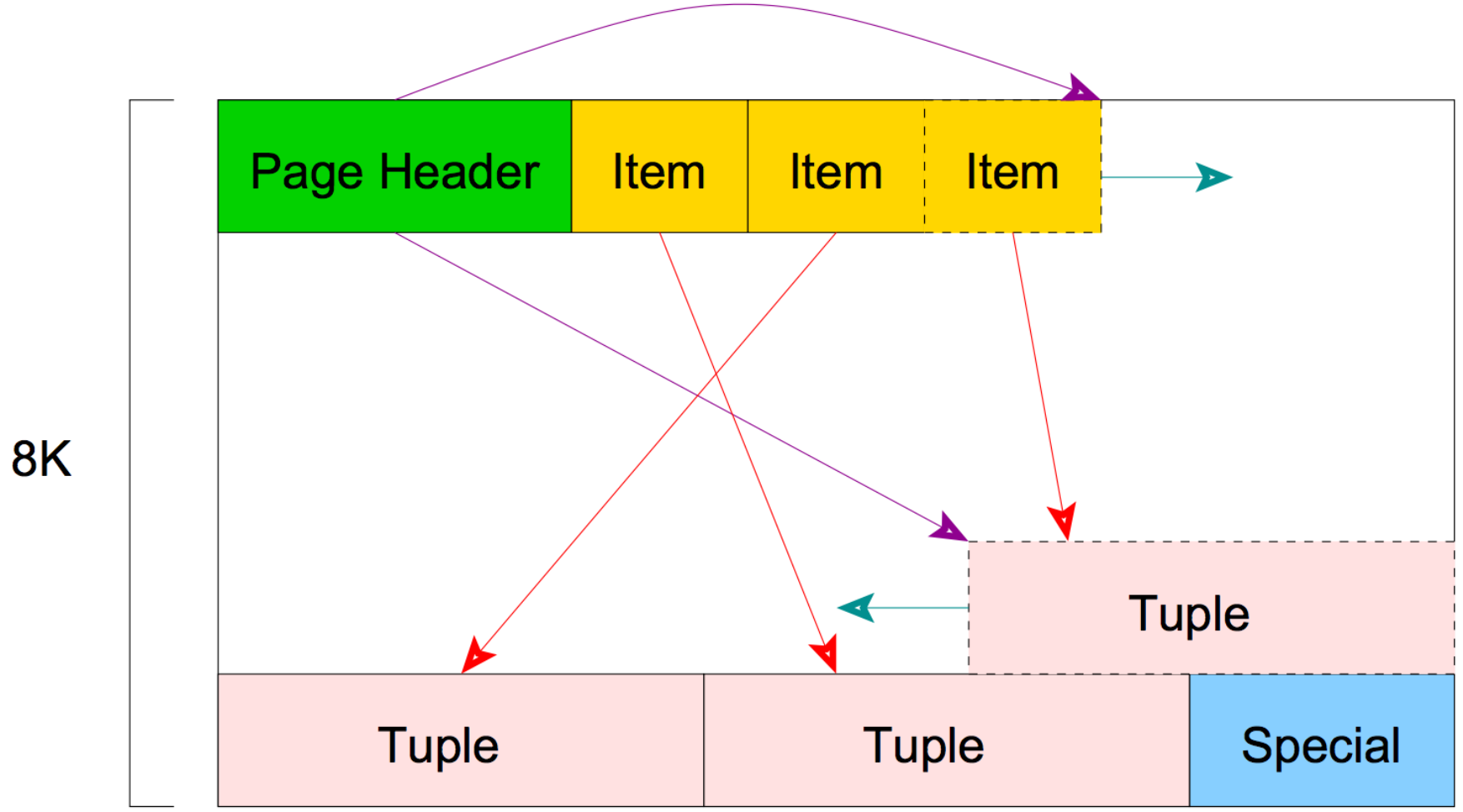
Uloženie dát na disku

- 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

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

Postgres page structure



http://rachbelaid.com/introduction-to-postgres-physical-storage/

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

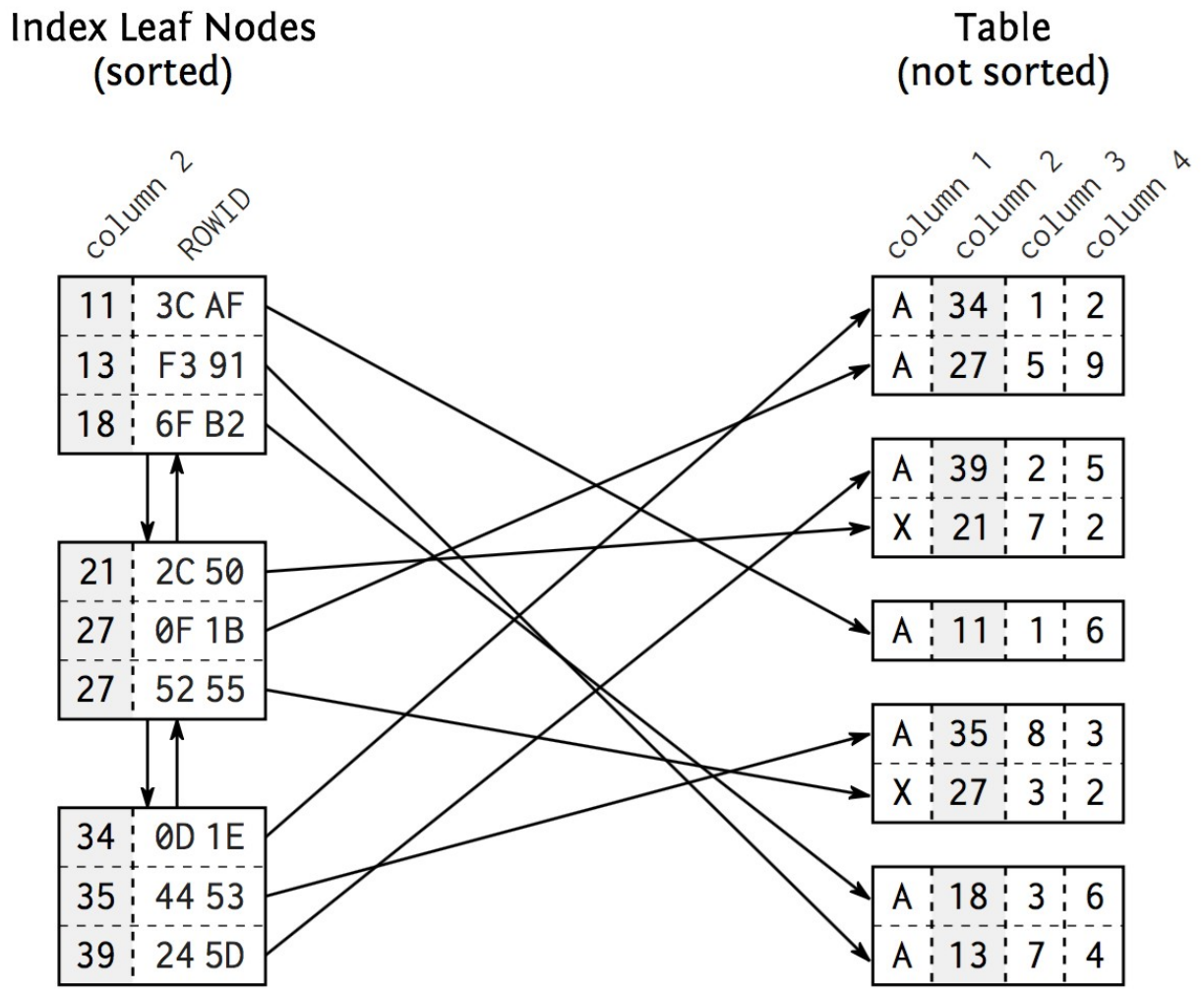
- Zapisovanie
  - veľmi drahé
  - v priemere musím poposúvať polovicu záznamov aby som spravil miesto pre nový záznam
- Mazanie
  - was_deleted marker
- overflow/transaction file
  - zapisujem do pomocného súboru (halda)
  - ten je periodicky sortovaný a spájaný s hlavným súborom
  - trošku komplikovanejšie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

Hašovaný súbor

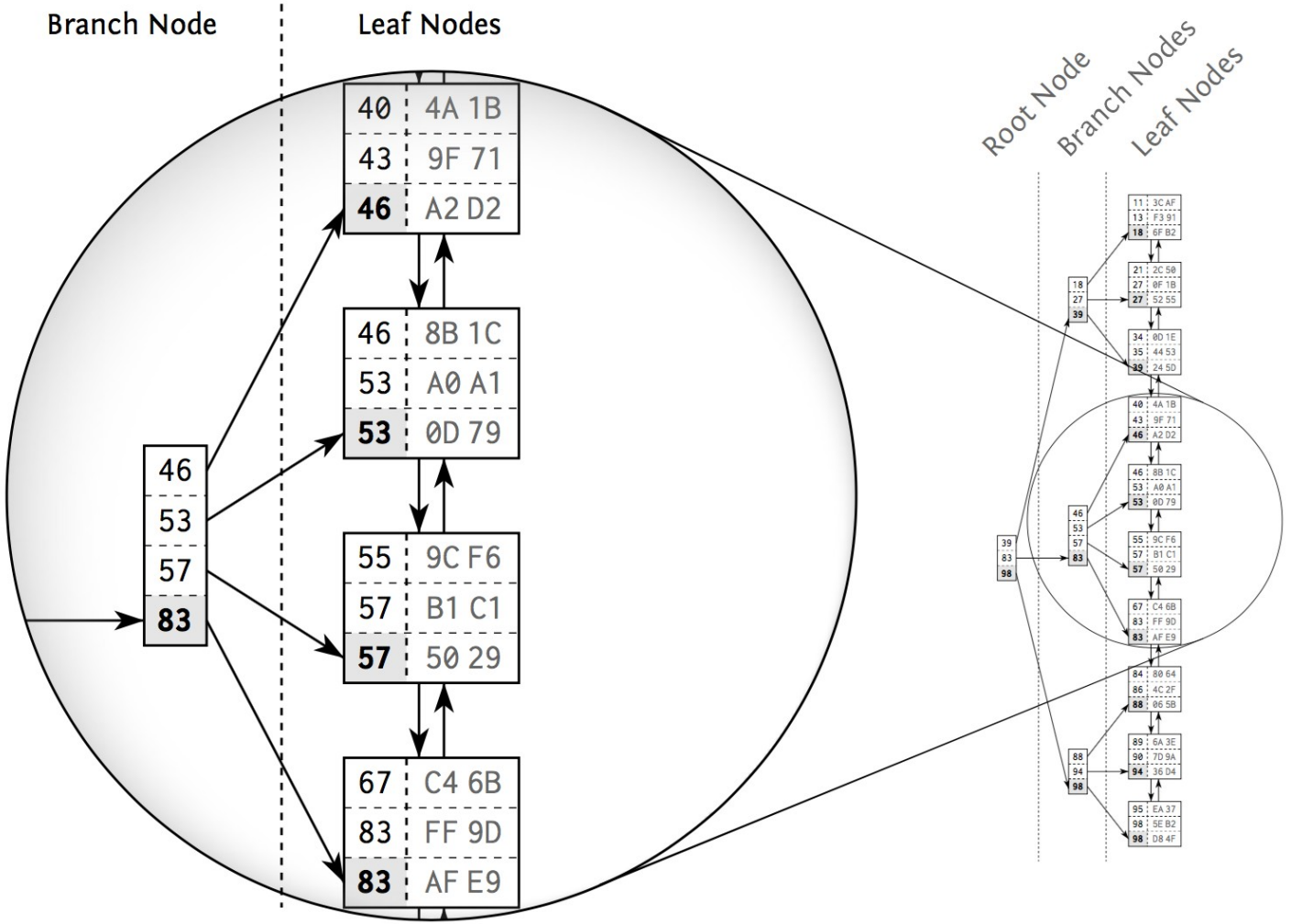
- hašovacia funkcia nám vráti adresu bloku, ktorý obsahuje požadovaný záznam
  - v skutočnosti adresu tzv. bucketu, ktorý obsahuje tento blok
- rýchle vyhľadávanie
  - podmienka vyhľadávania musí byť rovnosť (==)

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


Index Leaf Nodes
(sorted)
Table
(not sorted)
column 2
ROWID
11 3C AF
13 F3 91
18 6F B2
21 2C 50
27 0F 1B
27 52 55
34 0D 1E
35 44 53
39 24 5D
column 1
column 2
column 3
column 4
A 34 1 2
A 27 5 9
A 39 2 5
X 21 7 2
A 11 1 6
A 35 8 3
X 27 3 2
A 18 3 6
A 13 7 4

B+ strom


Branch Node
Leaf Nodes
46
53
57
83
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
Root Node
Branch Nodes
Leaf Nodes

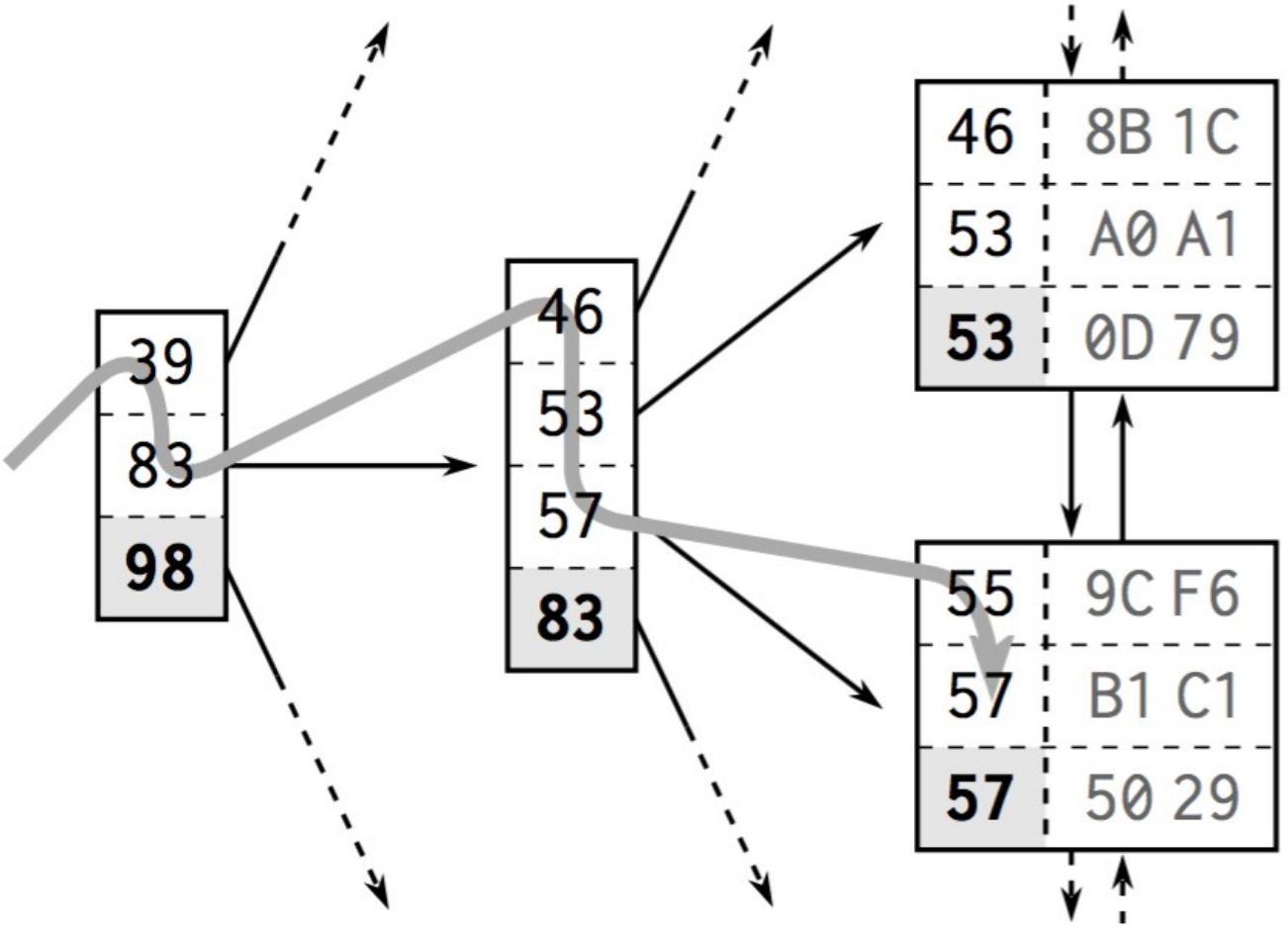
B+ strom

- B for balanced – vyvážený
  - Vzdialenosť medzi koreňom a akýmkoľvek listom je rovnaká
    - Rovnaký počet krokov pre dosiahnutie akejkoľvek hodnoty
  - Treba ho udržiavať

Ako potom vyzerá vyhľadávanie

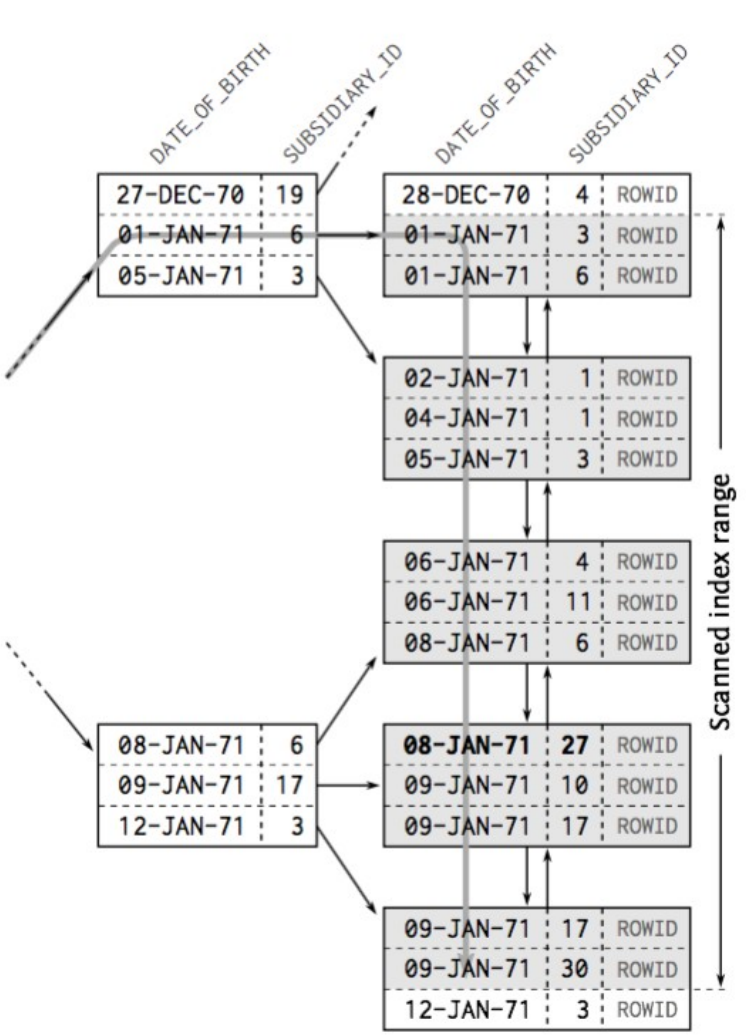
- Index access – prechod stromom k listom
  - Toto nie je nikdy problém, strom je balanced
- Index range scan – prehľadávanie zoznamu listov
  - UNIQUE?
  - Ak toho musím prejsť veľa, tak je to problém
- Table access – vytiahnutie dát z tabuľky
  - Ak toho musím prejsť veľa, tak je to problém

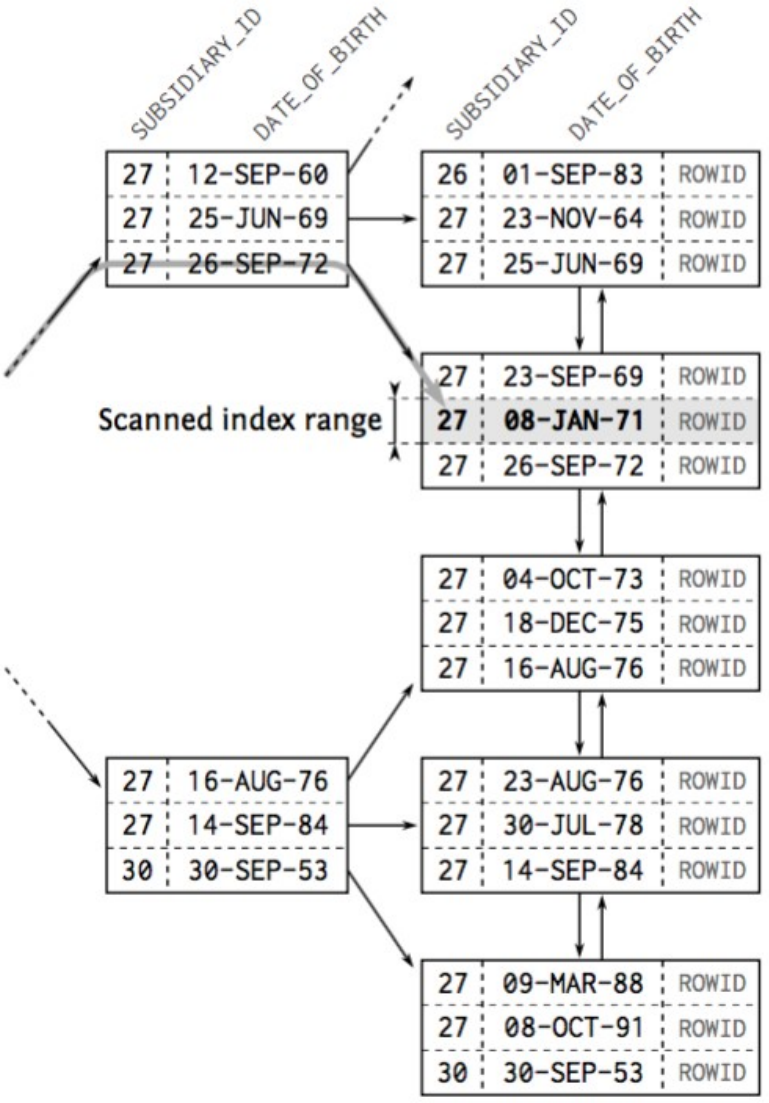
Prechod B+ stromom



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
  - Najprv rovnosť, potom range


DATE_OF_BIRTH
SUBSIDIARY_ID
DATE_OF_BIRTH
SUBSIDIARY_ID
27-DEC-70 19
01-JAN-71 6
05-JAN-71 3
28-DEC-70 4 ROWID
01-JAN-71 3 ROWID
01-JAN-71 6 ROWID
02-JAN-71 1 ROWID
04-JAN-71 1 ROWID
05-JAN-71 3 ROWID
06-JAN-71 4 ROWID
06-JAN-71 11 ROWID
08-JAN-71 6 ROWID
08-JAN-71 6
09-JAN-71 17
12-JAN-71 3
08-JAN-71 27 ROWID
09-JAN-71 10 ROWID
09-JAN-71 17 ROWID
09-JAN-71 17 ROWID
09-JAN-71 30 ROWID
12-JAN-71 3 ROWID
Scanned index range


SUBSIDIARY_ID
DATE_OF_BIRTH
SUBSIDIARY_ID
DATE_OF_BIRTH
27 12-SEP-60
27 25-JUN-69
27 26-SEP-72
26 01-SEP-83 ROWID
27 23-NOV-64 ROWID
27 25-JUN-69 ROWID
Scanned index range
27 23-SEP-69 ROWID
27 08-JAN-71 ROWID
27 26-SEP-72 ROWID
27 04-OCT-73 ROWID
27 18-DEC-75 ROWID
27 16-AUG-76 ROWID
27 16-AUG-76
27 14-SEP-84
30 30-SEP-53
27 23-AUG-76 ROWID
27 30-JUL-78 ROWID
27 14-SEP-84 ROWID
27 09-MAR-88 ROWID
27 08-OCT-91 ROWID
30 30-SEP-53 ROWID

Covering index

- Ak často nepotrebujem celý riadok
- Pridám si do indexu atribút(y), ktoré naozaj potrebujem
- Hľadám podľa id, zobrazujem name
- 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

Kedy sa nevyužije index?

- Lacnejši full table scan
  - Index filter vracia príliš veľa riadkov

http://use-the-index-luke.com/sql/where-clause/obfuscation

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!

Prepared statements

- Bezpečnosť
- Caching query plánu

```
int subsidiary_id;
PreparedStatement command =
connection.prepareStatement(
                    "select last_name"
                  + " from employees"
                  + " where subsidiary_id = ?"
                  );
command.setInt(1, subsidiary_id);
```

Apropos bezpečnosť

SQL injection

```
statement =
"SELECT * FROM users WHERE name =
'" + userName + "';"


userName = ' OR '1'='1
```

Ako je to s JOIN?

- JOIN operuje vždy len nad dvoma tabuľkami
- Nested Loop
  - N + 1 problém
  - Bola dlho jediná možnosť v MySQL
- Hash JOIN
  - Menšia tabuľka ide do hashe
  - Indexy pre WHERE condition
  - Výber menšieho počtu stĺpcov
- Sort-Merge JOIN
  - Kombinovanie dvoch zoradených zoznamov

ORDER BY

- Index je už zoradený…
  - Ak sedí index s ORDER BY, db nemusí robiť explicitnú fázu zoraďovania výsledku
- Index je doubly linked list
  - Vieme ho čítať v opačnom poradí
  - Pozor na miešanie ASC a DESC

```
CREATE INDEX sales_dt_pr
    ON sales (sale_date ASC, product_id DESC);
```

    - MySQL to ignoruje

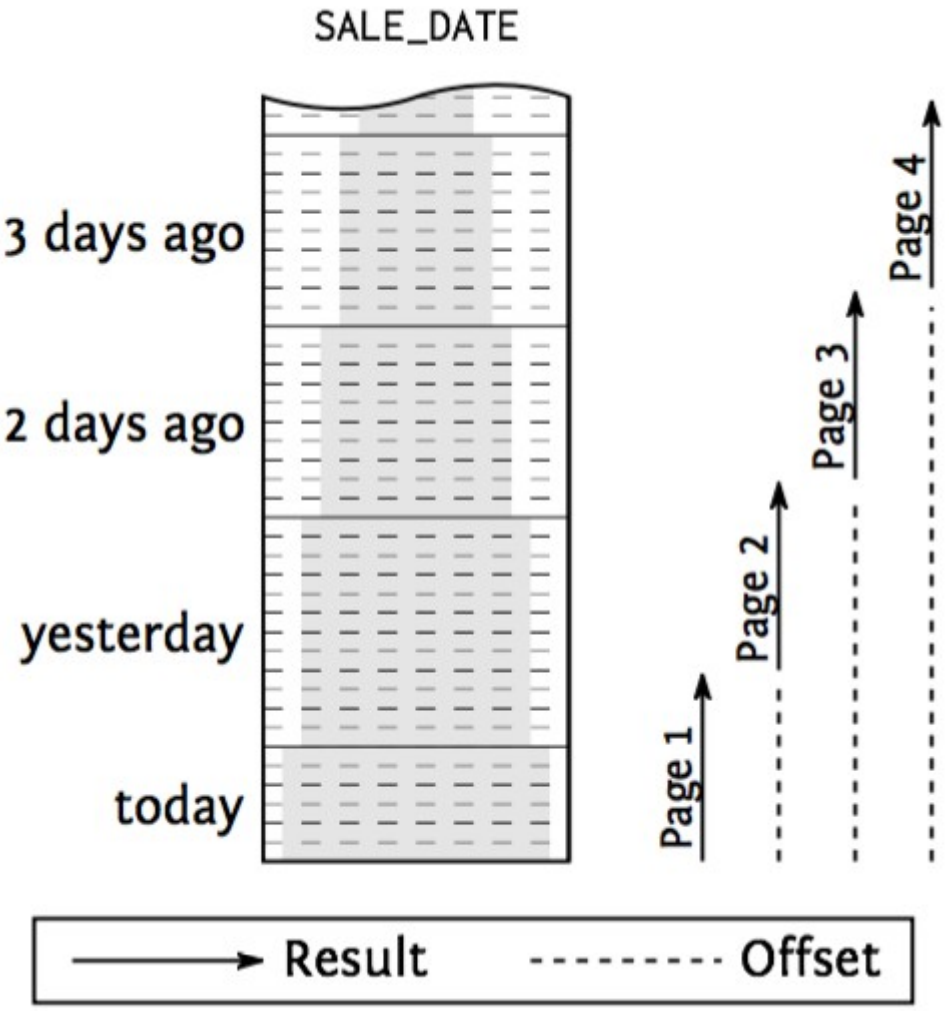
GROUP BY

- Aké možnosti má databáza?
  - Agreguje cez hash tabuľku
  - Zoradí riadky & postupne agreguje
    - Tu sa môže použiť index, keďže je zoradený

Pagination

- LIMIT + OFFSET
  - :(

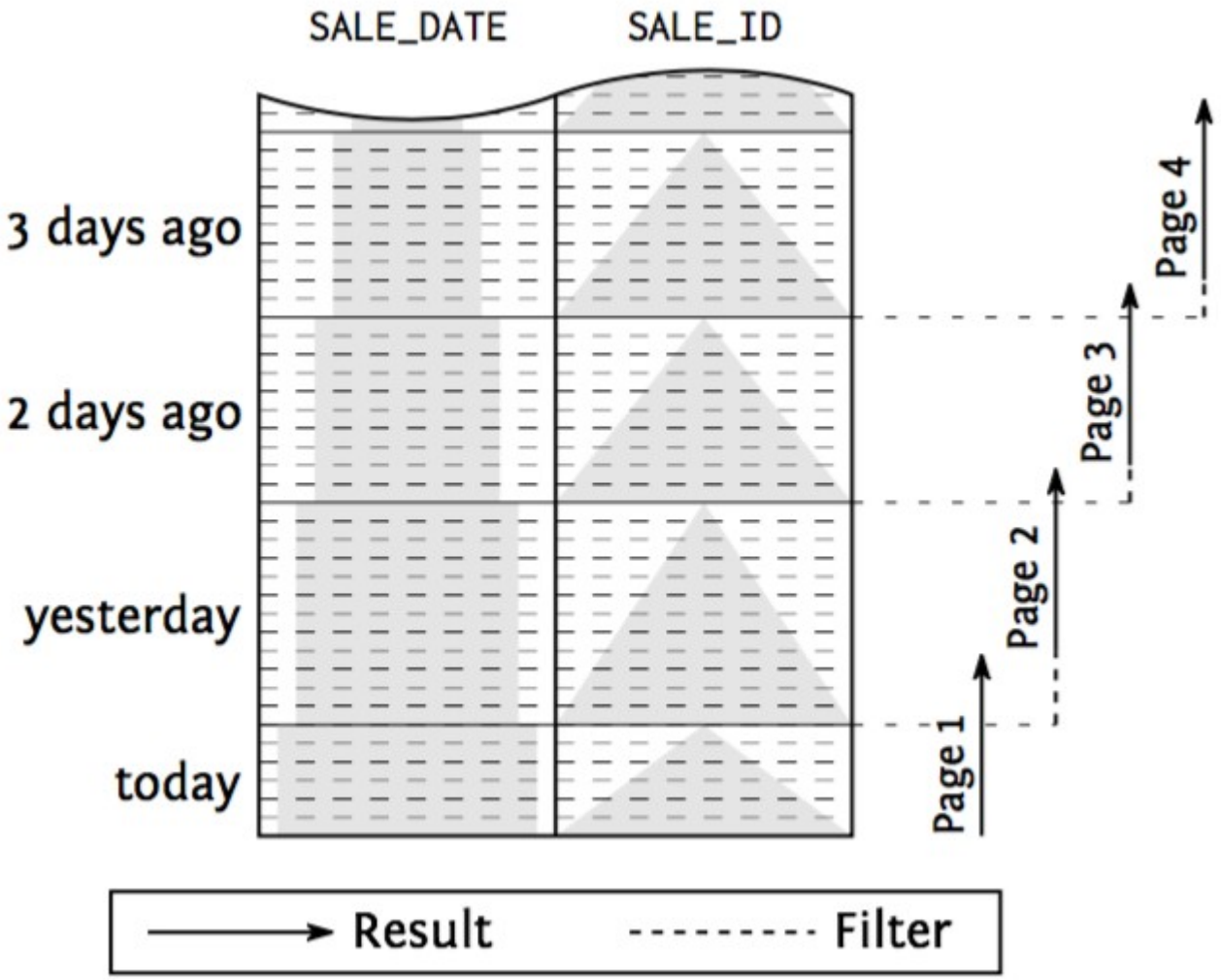
```
SELECT *
  FROM sales
 ORDER BY sale_date DESC
 LIMIT 10 OFFSET 10
```



Pagination

- Seek method

```
SELECT *
  FROM sales
 WHERE sale_date < ?
 ORDER BY sale_date DESC
 LIMIT 10
```



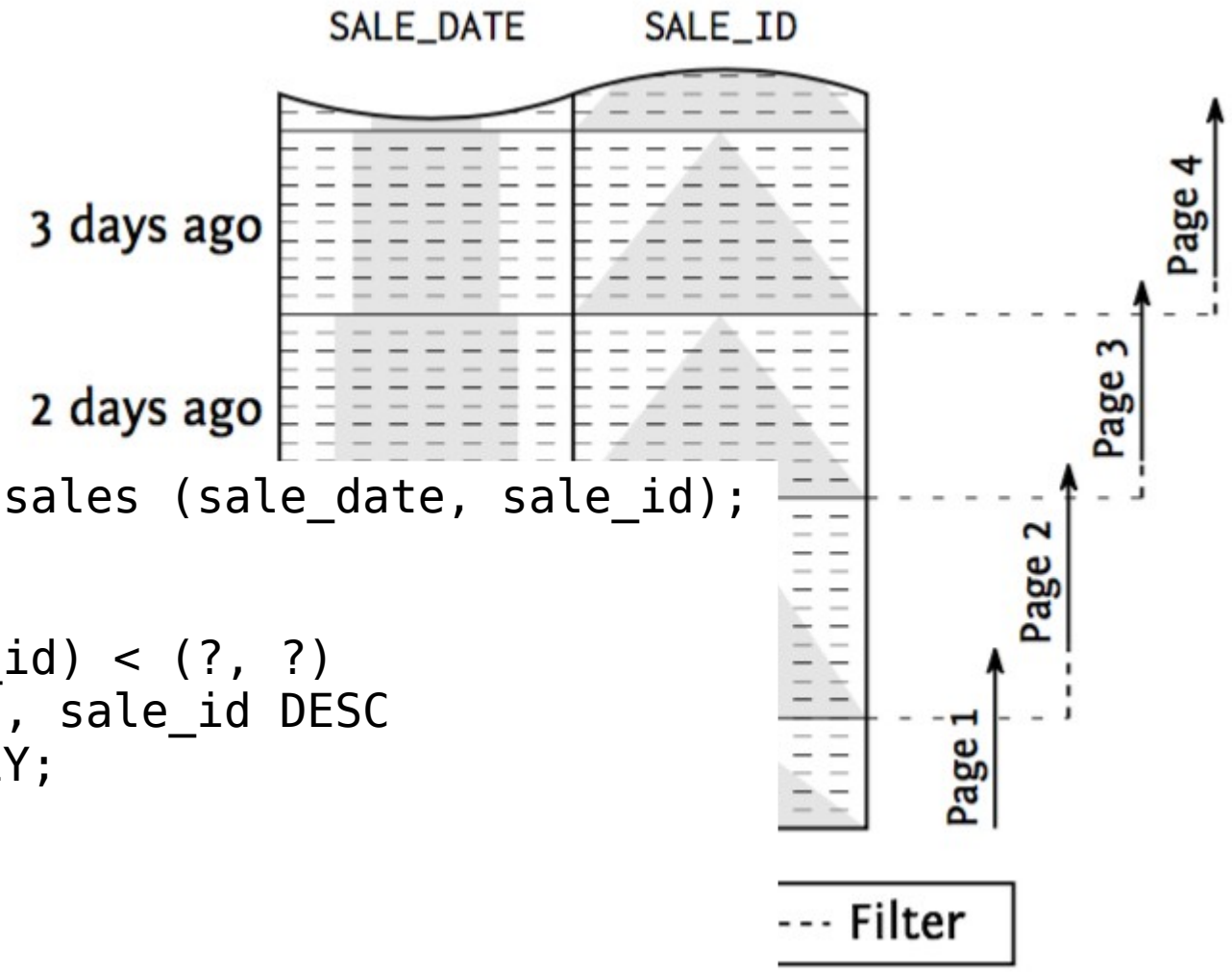
Pagination

- Seek method

```
SELECT *
  FROM sales
 WHERE sale_date < ?
 ORDER BY sale_date DESC
 LIMIT 10

CREATE INDEX sl_dtid ON sales (sale_date, sale_id);
SELECT *
  FROM sales
 WHERE (sale_date, sale_id) < (?, ?)
 ORDER BY sale_date DESC, sale_id DESC
 FETCH FIRST 10 ROWS ONLY;
```

PostgreSQL only



EXPLAIN

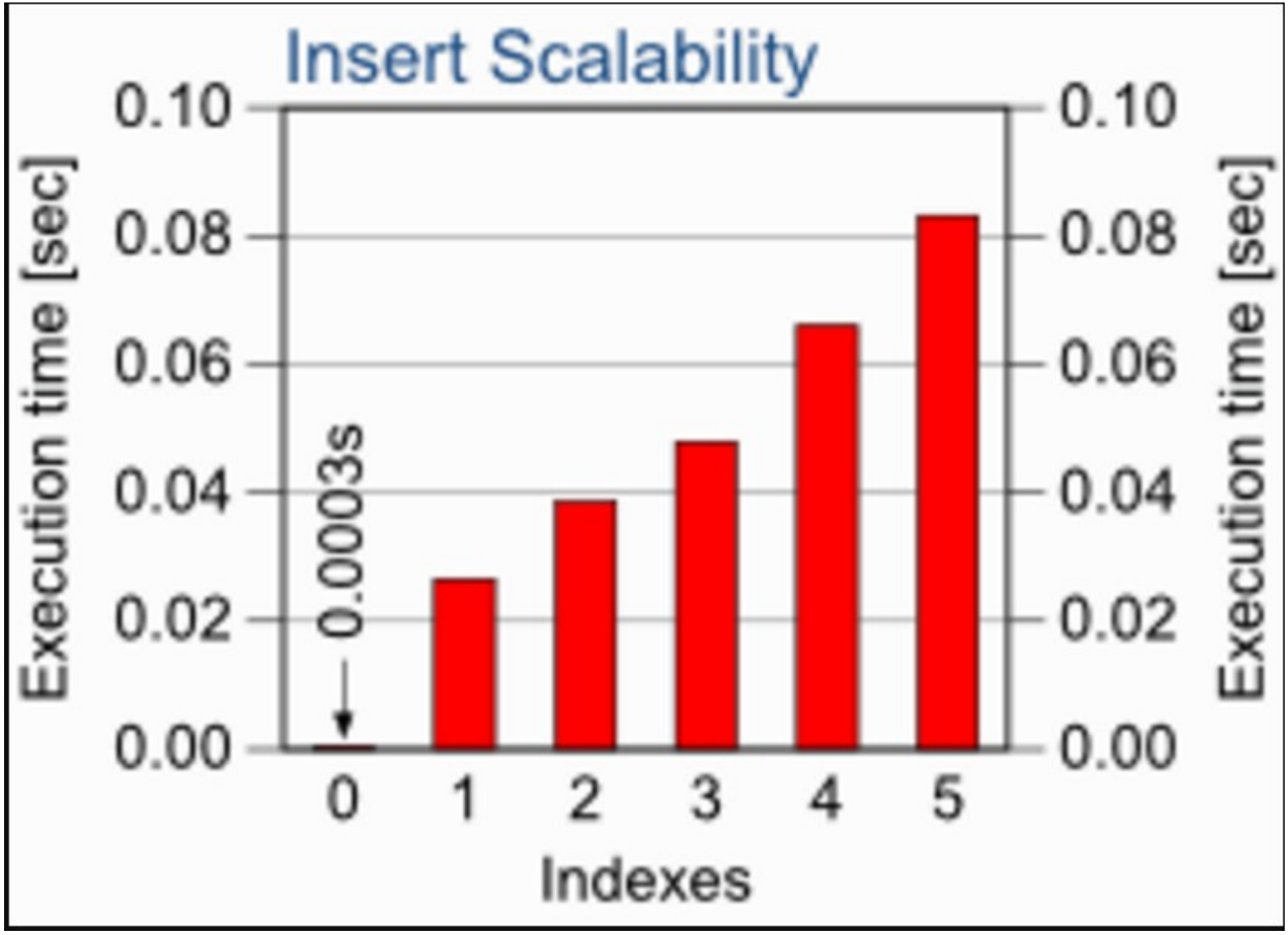
EXPLAIN [ANALYZE] statement

Pozor na to ANALYZE

http://explain.depesz.com/

VACUUM ANALYZE

INSERT/UPDATE/DELETE



BULK ops

- Loadovanie väčšieho množstva dát sa môže dramaticky zrýchliť, ak dočasne vypnete indexy
  - A potom ich zapnete

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

    https://www.2ndquadrant.com/wp-content/uploads/2019/05/Inside-the-PostgreSQL-Shared-Buffer-Cache.pdf

    https://www.interdb.jp/pg/pgsql08.html#_8.4.4.

PG Tune

- Odporúča nastavenie premenných súvisiacich s výkonom v postgresql.conf
- http://pgtune.leopard.in.ua

Zhrnutie

- Použitie indexov môže významne urýchliť vyhľadávanie záznamov
- Nie je to zadarmo
  - Réžia pri vkladaní, mazaní
  - Query planner
  - Zaberajú miesto
  - Slow query log, https://wiki.postgresql.org/wiki/Logging_Difficult_Queries
- http://use-the-index-luke.com/ !!!!