Databázové technologie

# Reprezentace dat a indexy

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

# Motivace

- množství dat je často řádově vyšší něž je schopna pojmout operační paměť
- přístup na disk je řádově pomalejší než do RAM
- snaha eliminovat přesuny mezi primární a sekundární pamětí (diskem)
- optimalizované algoritmy (quicksort vs. mergesort vs. two phase mergesort) struktury
- optimalizované datové struktury (B-stromy vs. AVL nebo RB stromy)
- někdy dublují funkce OS
- v minulosti DB řešily i přímo práci s diskem (vystavení hlavičky disku)
- dnes trend in-memory databáze
- budeme uvažovat primárně OLTP databáze

# Odbočka: přístupové doby

| | | |
|---|---|---|
| L1 cache reference | 0.5 ns | |
| Branch mispredict | 5 ns | |
| L2 cache reference | 7 ns | |
| Mutex lock/unlock | 25 ns | |
| **Main memory reference** | **100 ns** | |
| Compress 1K bytes with Zippy | 3,000 ns | |
| Send 1K bytes over 1 Gbps network | 10,000 ns | (0.01 ms) |
| **Read 4K randomly from SSD*** | **150,000 ns** | (0.15 ms) |
| **Read 1 MB sequentially from memory** | **250,000 ns** | (0.25 ms) |
| Round trip within same datacenter | 500,000 ns | (0.5 ms) |
| **Read 1 MB sequentially from SSD*** | **1,000,000 ns** | (1 ms) |
| **Disk seek** | **10,000,000 ns** | (10 ms) |
| **Read 1 MB sequentially from disk** | **20,000,000 ns** | (20 ms) |
| Send packet CA-Netherlands-CA | 150,000,000 ns | (150 ms) |

# Datové typy

**pevné velikosti**

- standardní čísla (integer, double), datum/čas
- `char(n)` (doplněny mezery)

**proměnlivé velikosti**

- varchar(n)
- array
- dvě možná řešení
  - velikost + data
  - data + zarážka

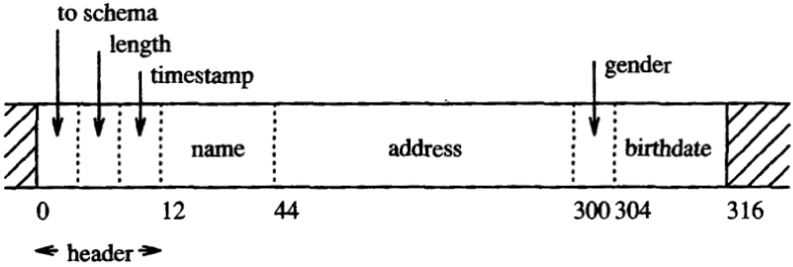
**BLOB**

- binary large object
- nestrukturovaná data o velikosti až v GB
- typicky uloženy v oddělených blocích

# Záznamy (records, ntice, řádky)

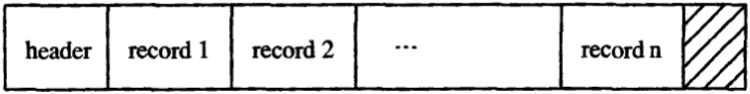
**záznamy s datovými typy pevné délky**

- jednoduché na procházení manipulaci
- uspořádání podobné jako např. struktury v C
- může být výhodné zarovnávat hodnoty podle architektury (např. int na 4 B)
- často bývá doplněna hlavička (schéma, délka záznamu, čas přístupu, příznak odstranění)



# Bloky I

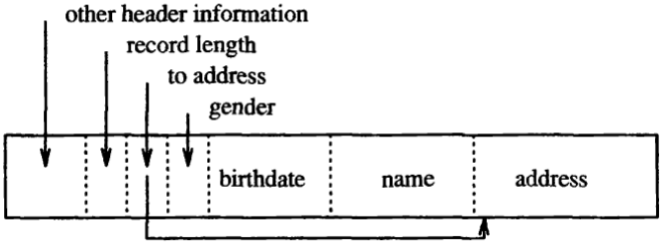
- záznamy seskupeny do bloků
- základní jednotka pro přesun mezi primární a sekundární paměti
- volitelně obsahují hlavičku
  - identifikátor bloku případně typu záznamů
  - odkazy na související datové bloky, případně vztah k nim
  - odkazy na jednotlivé záznamy
  - informace o časech přístupu



# Záznamy (records, ntice, řádky)

**záznamy s datovými typy proměnlivé délky**

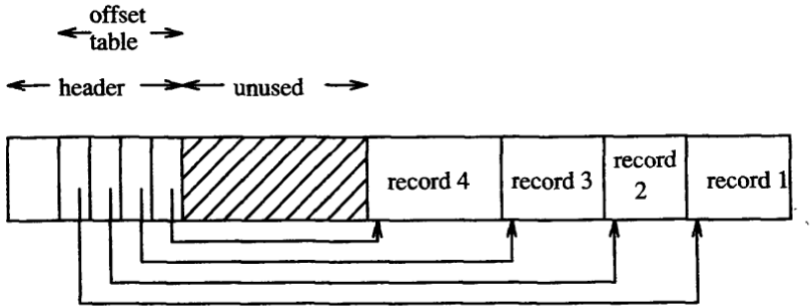
- součástí záznamu je jeho celková délka
- hodnoty datových pevné délky seskupeny na začátku záznamu
- **odkazy** na hodnoty typů proměnlivé délky (offset v rámci záznamu)
- **honoty** typů proměnlivé délky




- alternativně mohou odkazy vést do samostatné, k tomu určené, oblasti paměti (menší záznamy, vyšší I/O)
- oba přístupy je možné kombinovat

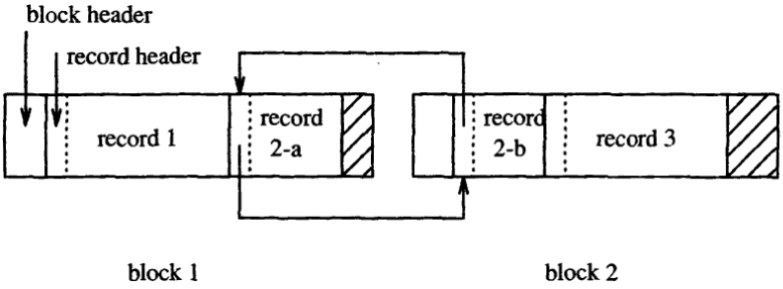
# Bloky II

- rozdělení bloku na několik částí – hlavička, odkazy, volné místo, záznamy
- vhodné pro záznamy různé velikosti
- jednoduché přeuspořádání záznamu v bloku (jen se mění odkazy)
- možnost přesunout záznam do jiného bloku (pokud to offset umožňuje)
- jednoduché odstranění záznamu – stačí offset označit jako neplatný



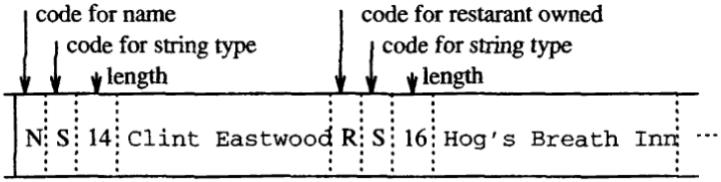
# Bloky III

- velké záznamy, riziko vnější fragmentace
- rozdělení záznamu na několik fragmentů
- nutné signalizovat (v hlavičce)
  - jestli se jedná o kompletní záznam nebo fragment
  - první a koncový fragment
  - odkazy na předchozí a další fragmenty



# Záznamy variabilního formátu

- není k dispozici schéma
- záznam je tvořen sekvencí označkovaných (tagged) hodnot
- značka se skládá z:
  - identifikátoru atributu
  - typu atributu (pokud to není zjevné z identifikátoru atributu)
  - délky hodnoty (pokud to není zjevné z typu)
- vhodné pro dokumentové databáze
- nebo databáze, kde je velké množství atributu s NULL „hodnotou"

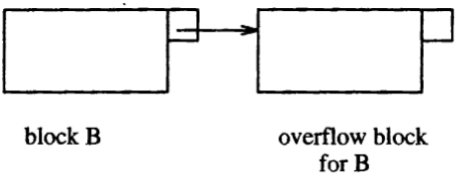


# Manipulace se záznamy

- plyne z organizace dat v blocích a požadavku, zda mají být data uspořádána

**vkládání**

- pokud nevyžadujeme uspořádání, stačí najít vhodný blok s volným místem
- pokud vyžadujeme uspořádání a není v odpovídajícím bloku místo
  1. lze největší záznam (z pohledu uspořádání přesunout do vedlejšího bloku)
  2. připojit *overflow block*



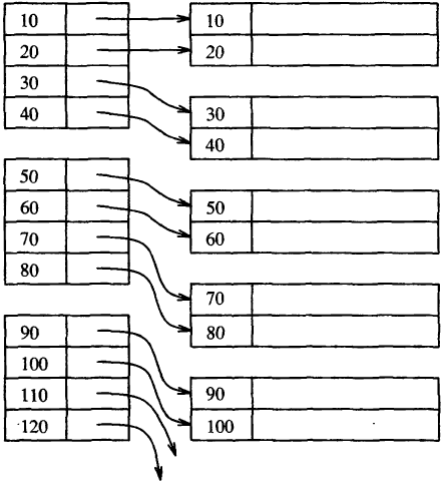

**odstranění**

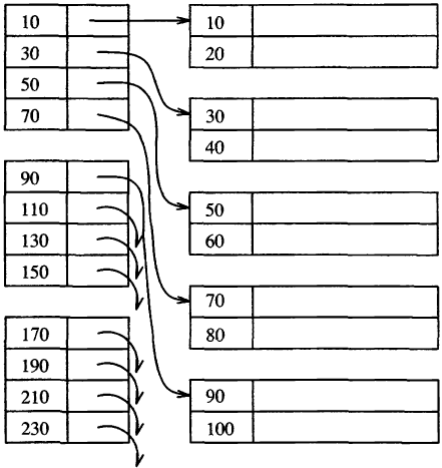
- příznak odstranění záznamu (tombstone)
- snadný přesun záznamu v rámci jednoho bloku (příp. overflow bloku)

# Indexy

- dva soubory (každý soubor je kolekce bloků)
  - **datový soubor** (obsahuje kompletní záznamy)
  - **indexový soubor** obsahuje dvojice klíč-ukazatel do datového souboru
- indexový soubor většinou výrazně menší
- vyhledání podle klíče vyžaduje $\log_2 n$ načtení bloků indexu (kde $n$ je počet bloků indexu)
- **hustý (dense) index** pro každý záznam v datovém souboru existuje záznam v indexovém souboru
- **řídký (sparse) index** pro každý blok v datovém souboru existuje v indexovém souboru záznam ukazující na jeho začátek
- řídké indexy jsou menší, ale pro některé typy dotazů vyžadují načtení datového bloku
- co v případě, že jsou povoleny duplicitní klíče?
- manipulace s indexem v podobném duchu jako s datovým souborem

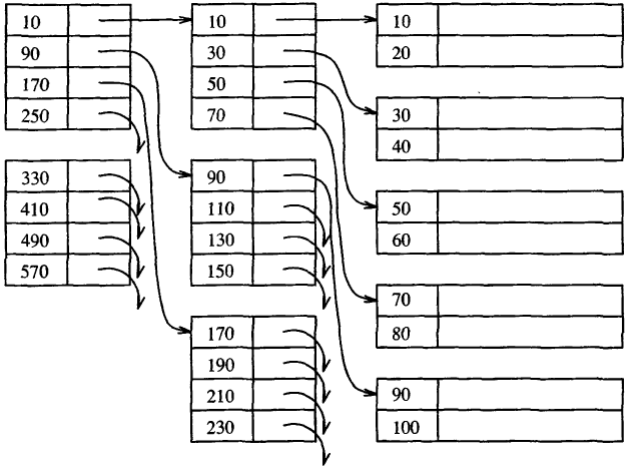
# Hustý a řídký index (ilustrace)



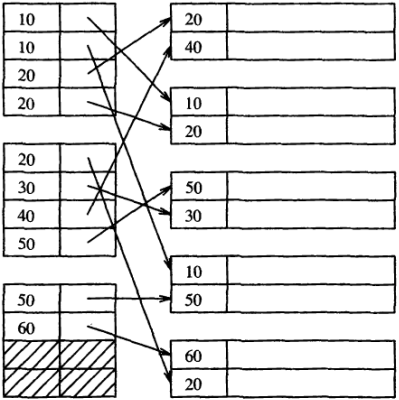
# Víceurovňový index

- možné kombinovat více úrovní a typů indexů
- ne všechny kombinace dávají smysl, např. hustý a hustý nepřináší žádnou úsporu



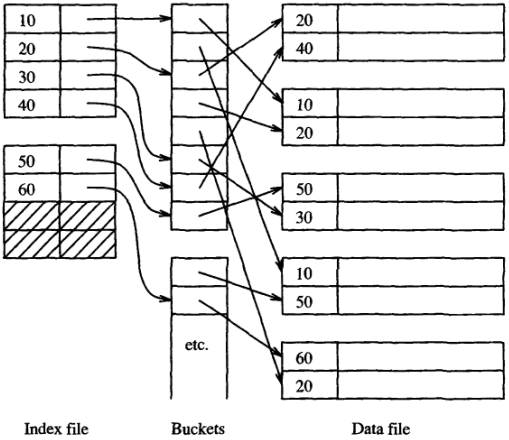
# Sekundární indexy

- primární index určuje pozici záznamu v datovém souboru
- sekundární index určuje uspořádání vzhledem k primárnímu indexu (nebo jiným atributům)
- musí být vždy hustý (nemůže odkazovat na jednotlivé bloky dat)
- typicky povoluje duplicity



# Nepřímé odkazy v sekundárním indexu

- umožňuje zbavit index duplicitních záznamů
- problém přesunut do mezilehlé úrovně obsahující odkazy na řádky s duplicitními klíči
- umožňuje optimalizovat některé operace (vyhledávání podle dvou indexů, lze realizovat jako průnik odkazů na mezilehlém indexu)



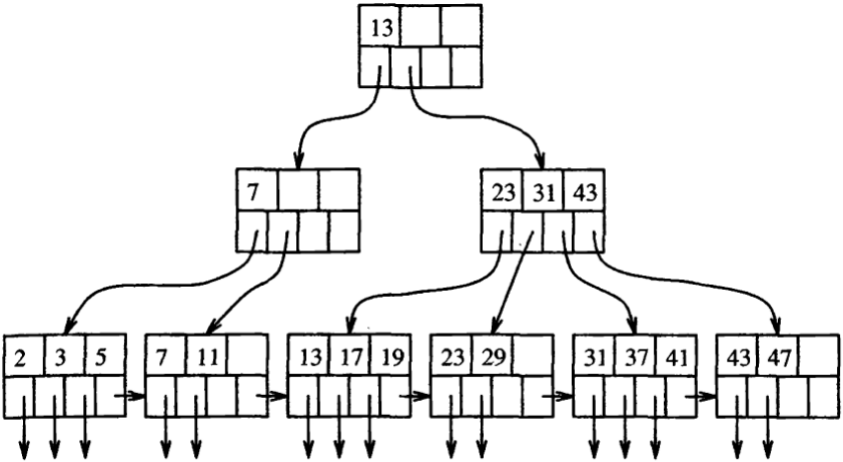
# B-stromy

- zobecněné řešení, vyvážené stromy (zobecnění 2,3-stromů, 2,3,4-stromů)
  - všechny listy jsou ve stejné hloubce
  - každý uzel (mimo kořenového) obsahuje nejméně $t-1$ klíčů, tj. má $t$ potomků; v neprázdném stromě kořen obsahuje alespoň jeden klíč
  - každý uzel má nanejvýš $2t-1$ klíčů $\Rightarrow$ $2t$ potomků $\Rightarrow$ plný uzel
  - **Věta:** Pokud je počet klíčů $n \geq 1$, pak pro B-strom stupně $t \geq 2$ a výšky $h$ platí

$$h \leq \log_t \frac{n+1}{2}.$$

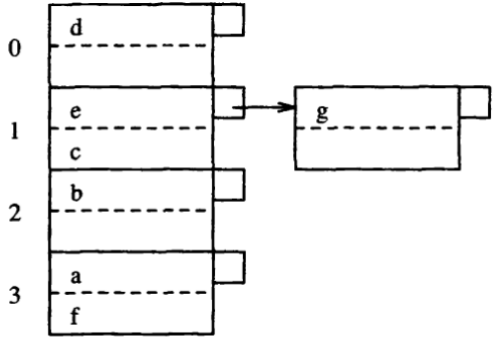
- ne všechna data v paměti (vs. běžné binarní stromy)
- preferované sekvenční čtení $\Rightarrow$ načtení celých bloků
- složitost vyhledávání, vložení: $O(th) = O(t \log_t n)$
- počet přístupů na disk: $O(h) = O(\log_t n)$
- v databázových systémech se používají se B+stromy (zaplnění min. ze $\frac{2}{3}$, **zřetězení listů**)

# B-stromy



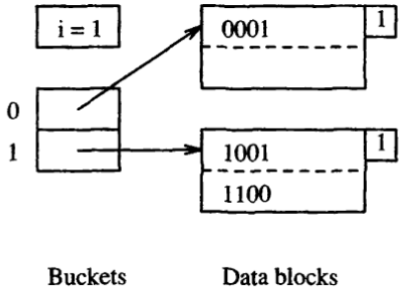
# Hašovací tabulky

- hašovací tabulka pracující se sekundarním uložištěm se podstatně liší od tradičního (in-memory řešení)
- každé pole (bucket) hašovací tabulky obsahuje záznamy se stejným hashem
- pokud dojde přetečení, je přidán overflow block

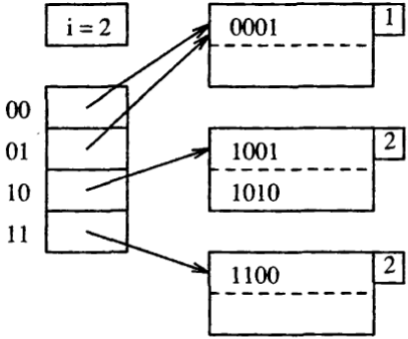


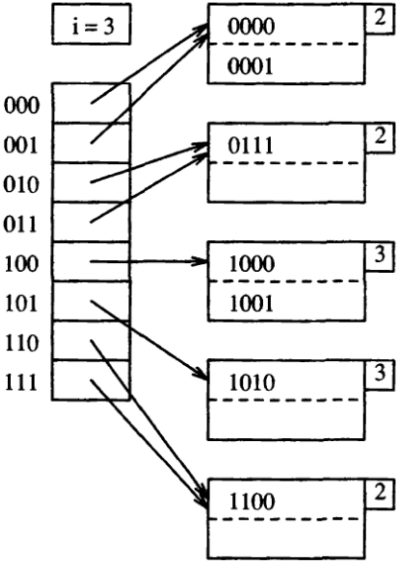
# Dynamické hašovací tabulky

- přidána jedna úroveň odkazů
- pole ukazatelů o velikosti $2^n$ na jednotlivé datové bloky
- prefix hashe daného záznamu představuje odkaz do tabulky ukazatelů
- datové bloky mohou být sdíleny, pokud nejsou dostatečně zaplněny



# Dynamické hašovací tabulky



# Poznámka

Všechny obrázky:
Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database System Implementation.