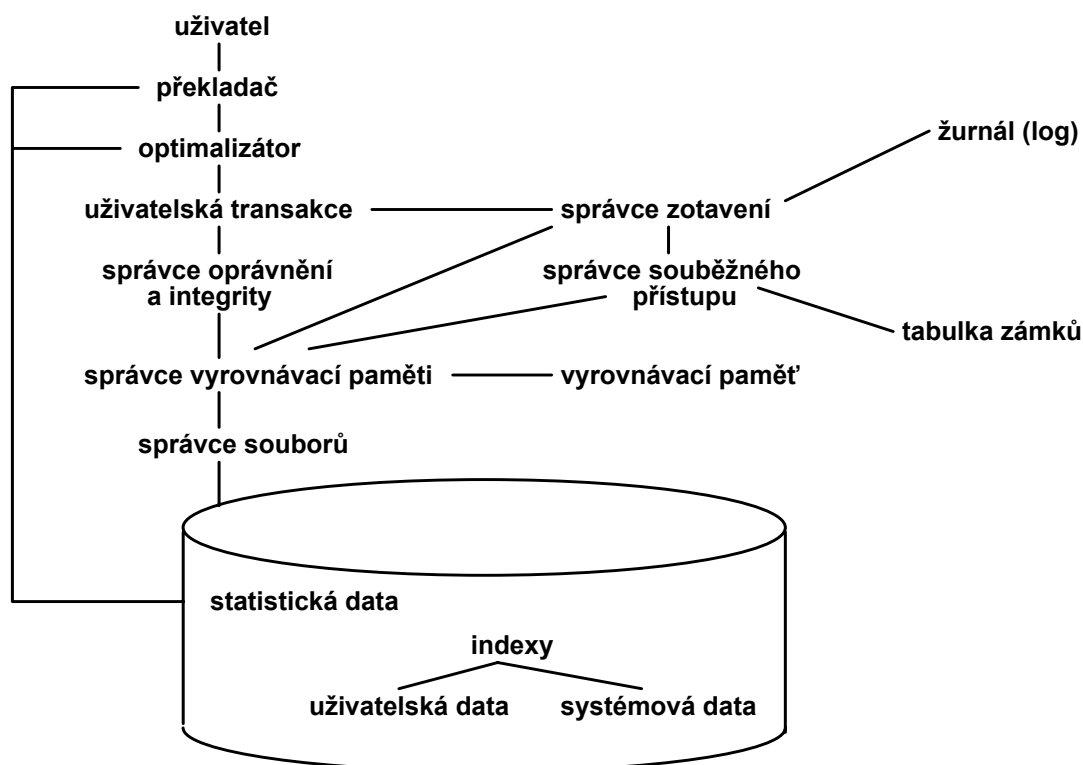


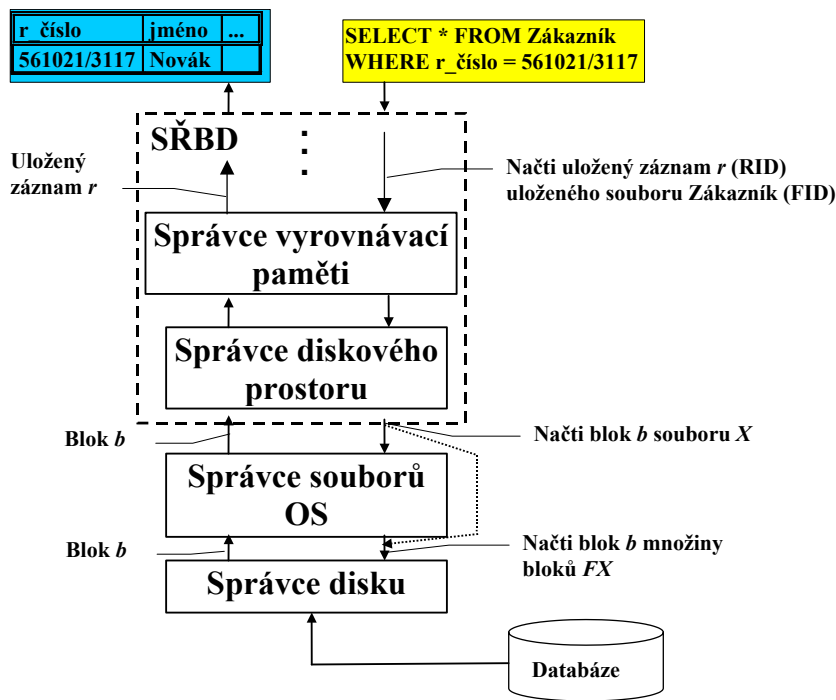
6. Fyzická (interní) úroveň databázového systému

6.1. Struktura databázového systému	2
6.2. Přístup k datům v databázi	3
6.3. Struktura souborů	4
6.4. Správa vyrovnávací paměti	8
6.5. Podstata indexování a hašování	10
6.6. Uspořádané indexy	12
6.7. B ⁺ strom	17
6.8. Hašování.....	25
6.9. Shlukování (clustering)	29
6.10. Bitmapové indexy	30
6.11. Fyzický návrh databáze	33
Literatura.....	37

6.1. Struktura databázového systému



6.2. Přístup k datům v databázi



- hierarchie pamětí: vnitřní, sekundární (disk), terciální (mag.páska)
- Cíl: minimalizace přístupů k disku.**

6.3. Struktura souborů

Soubor posloupnost záznamů seskupovaných do bloků tvořících logický celek.

- Organizace záznamů v souborech

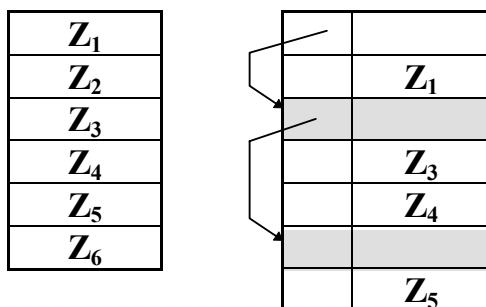
Neuspořádaný soubor (heap organization) – záznam je umístěn tam, kde je místo, neexistuje žádné uspořádání záznamů.

Sekvenční soubor – záznamy jsou uspořádány podle hodnoty tzv. **vyhledávacího klíče** (search key). V případě takového uspořádání, kdy jsou logicky svázané záznamy umístěny fyzicky blízko u sebe, hovoříme o **shlukování** (clustering). Klíč pro shlukování se pak označuje jako **shlukovací klíč** (cluster key).

Hašovaný soubor – záznamy jsou umístěny do bloků na základě hodnoty **hašovací funkce**.

- Soubory se záznamy pevné délky

- Musí být zajištěno, aby identifikace záznamu (RID, ROWID) byla konstantní po celou dobu jeho života.



- Použití *přetokových oblastí* (bloků) u sekvenčních souborů
- Soubory se záznamy proměnné délky
 - Potřeba vzniká v těchto případech:
 - uložení záznamů různých typů v jednom souboru,
 - typ záznamu, dovolující proměnnou délku některé položky,
 - typ záznamu umožňující opakující se položky (seznam) – není v klasické relační databázi.

Možnosti realizace:

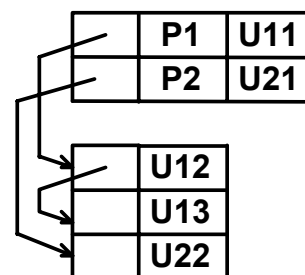
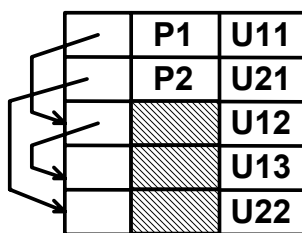
- řetězec bytů proměnné délky,
- několik záznamů pevné délky

S rezervovaným prostorem

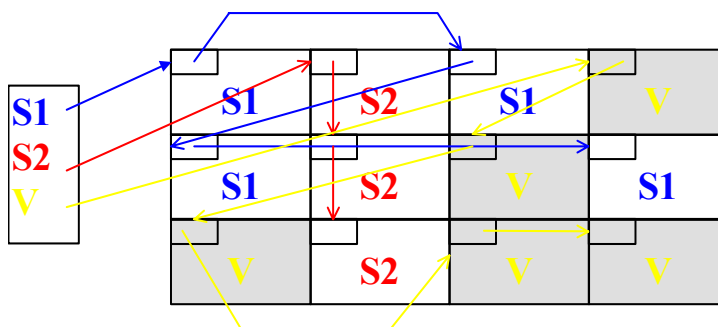
P1	U11	U12	U13	⊥
P2	U21	U22	⊥	

P1	U11	U12	U13	⊥
P2	U21	U22	⊥	⊥

Použitím ukazatelů



• Organizace bloků (stránek)



- **Vztah databáze a souboru operačního systému:**
 - Každá tabulka (příp. indexy) v samostatném souboru OS. Typické pro databáze systémů s architekturou PC file-server (např. dBase).
 - Celá databáze (typicky obsahující řadu schémat) v jednom nebo několika souborech OS. Typické pro databáze systémů s architekturou klient-server (např. Oracle, SQLBase).

6.4. Správa vyrovnávací paměti

- **Požadavky**
 - co nejlepší strategie výměny bloků,
 - omezení času, kdy lze blok zapsat na disk,
 - možnost vynuceného zápisu modifikovaných bloků na disk,
 - často používané bloky (katalogu, indexů).
- **Možnosti řešení**
 - vyrovnávací paměť spravovaná operačním systémem (OS),
 - vyrovnávací paměť spravovaná SŘBD.
- **Problémy vyrovnávací paměti ve správě OS**

- strategie výměny bloků LRU, SŘBD může lépe "předvídat potřebu" ,

Př) Zákazník JOIN Účet metodou zanořených cyklů

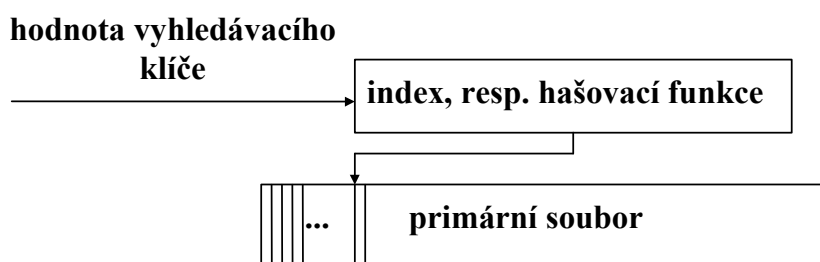
```
for each blok tabulky Zákazník do
  for each blok tabulky Účet do
    .....
```

- hodila by se spíše strategie MRU

- nelze omezit čas, kdy lze zapsat blok na disk

⇒ vyrovnávací paměť spravovaná SŘBD

6.5. Podstata indexování a hašování



Primární soubor – soubor se záznamy tabulky.

Vyhledávací klíč – sloupec (případně složený) tabulky, prostřednictvím jehož hodnoty budeme přistupovat k požadovaným záznamům.

Primární vyhledávací klíč – vyhledávací klíč, podle jehož hodnot je seříděn primární soubor.

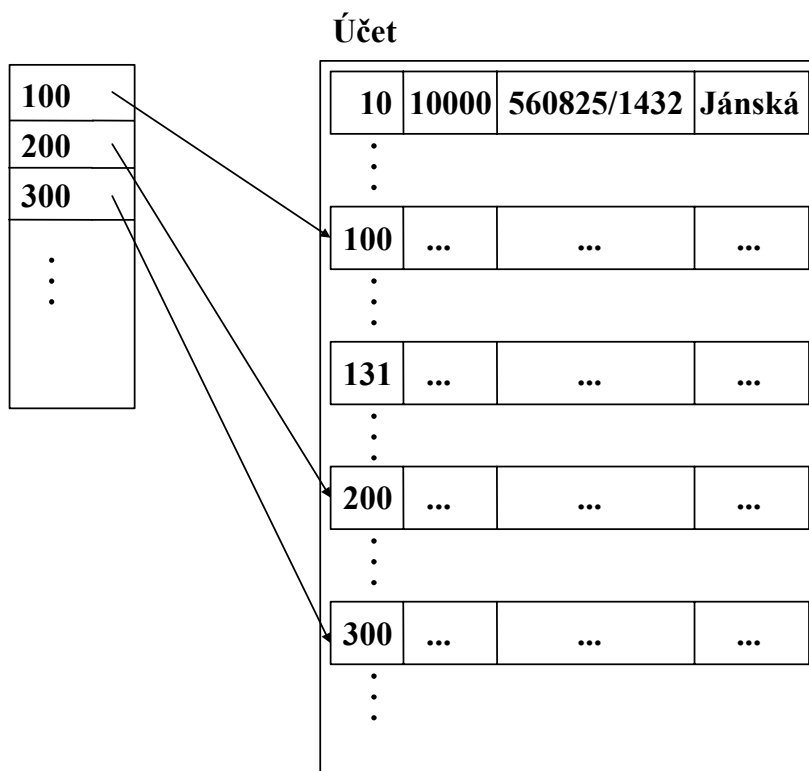
Sekundární vyhledávací klíč – vyhledávací klíč, který není primární.

Uspořádaný index – založený na seřazení podle hodnot vyhledávacího klíče.

Hašovaný index – založený na rovnoměrném rozložení hodnot vyhledávacího klíče v sektorech (bucket) záznamů primárního

souboru (typicky blok na disku). Sektor pro záznam s danou hodnotou vyhledávacího klíče je určen *hašovací funkcí*.

6.6. Uspořádané indexy



Primární index (také shlukující) – index pro prim. vyhledávací klíč.

Sekundární index – index pro sekundární vyhledávací klíč.

Indexsekvenční soubor - uspořádaný primární soubor + primární index.

Položka indexu – záznam obsahující hodnotu vyhledávacího klíče a ukazatele na jeden nebo několik záznamů primárního souboru s touto hodnotou primárního klíče, případně ukazatel na sektor ukazatelů.

Hustý index – index s položkami obsahujícími všechny hodnoty vyhledávacího klíče vyskytující se v primárním souboru.

Řídký index – index s položkami obsahujícími jen některé hodnoty vyhledávacího klíče vyskytující se v primárním souboru. Primární řídký index typicky obsahuje jednu položku pro každý blok primárního souboru.

Víceúrovňový index – index, u kterého je každá úroveň, s výjimkou poslední, řídkým primárním indexem úrovně následující. Poslední úroveň je jednoúrovňovým indexem primárního souboru.

Index s unikátními hodnotami (unique index) – index pro vyhledávací klíč, jehož hodnoty jsou v primárním souboru unikátní.

- Operace nad indexem

- Vložení hodnoty (pro nový záznam v primárním souboru)

- Hustý index

- Najdi v indexu položku s vkládanou hodnotou vyhledávacího klíče;

- IF položka existuje THEN

- Vlož ukazatel na příslušný záznam primárního souboru do položky indexu

- ELSE

- Vytvoř novou položku s vkládanou hodnotou vyhledávacího klíče a ukazatelem na příslušný záznam primárního souboru;

- **Řídký index (předpoklad – položka indexu ukazuje na blok)**

```
IF je vytvořen nový blok primárního souboru THEN
    Vytvoř novou položku indexu s vkládanou
    hodnotou vyhledávacího klíče a ukazatelem na
    příslušný blok
```

```
ELSE
```

```
    IF nově vložený záznam má nejmenší hodnotu
    vyhledávacího klíče v bloku THEN
        Aktualizuj hodnotu vyhledávacího klíče
        v položce indexu ukazující na daný blok;
```

- **Zrušení hodnoty**

- **Hustý index**

```
Najdi v indexu položku s rušenou hodnotou
vyhledávacího klíče;
```

```
IF položka obsahuje pouze ukazatel na rušený
záznam THEN
```

```
    Zruš položku indexu
```

```
ELSE
```

```
    Zruš ukazatel položky na rušený záznam;
```

- **Řídký index**

```
Najdi v indexu položku s rušenou hodnotou
vyhledávacího klíče;
```

```
IF položka existuje THEN
```

```
    IF položka indexu ukazuje na rušený záznam THEN
```

```
        IF rušený záznam není jediným záznamem
        v daném bloku primárního souboru THEN
```

```
            Aktualizuj položku indexu pro následující
            záznam bloku primárního souboru
```

```
        ELSE
```

```
            Zruš položku indexu;
```

- **Nevýhody indexsekvenčních souborů**

- hlavní nevýhodou je pokles výkonnosti s rostoucí velikostí souboru (více záznamů, více záznamů indexu i primárního souboru v přetokových blocích)

6.7. B⁺ strom

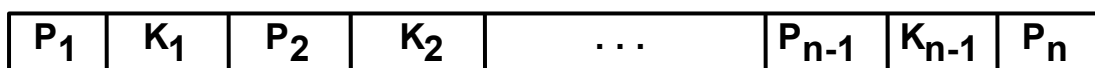
Nejpoužívanější indexová struktura v databázových systémech

• Struktura B⁺ stromu

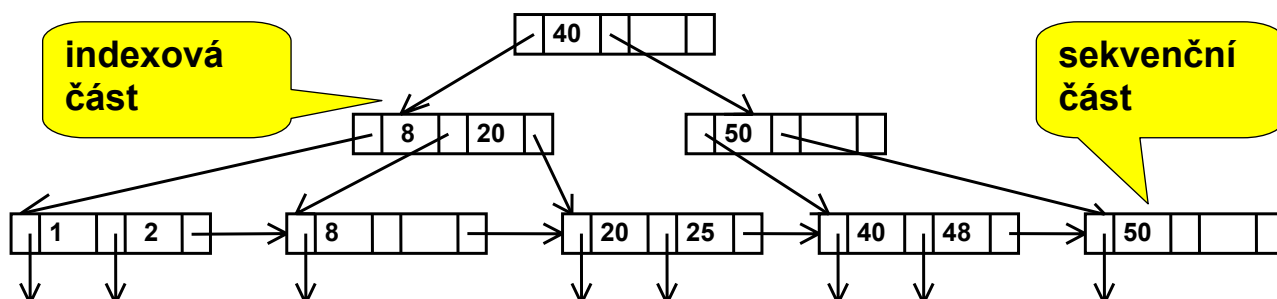
- víceúrovňový index ve tvaru vyváženého n-nárního stromu
- uzel stromu má typicky velikost jednoho diskového bloku
- každý uzel s výjimkou kořene a listu má $\lceil n/2 \rceil$ až n následníků,
- kořen, není-li současně listem, má nejméně 2 následníky
- list obsahuje $\lceil (n-1)/2 \rceil$ až $n-1$ hodnot vyhledávacího klíče
- listové uzly tvoří jednosměrný uspořádaný seznam
- pro K hodnot vyhledávacího klíče není cesta od kořene k listu delší než $\lceil \log_{\lceil n/2 \rceil} (K) \rceil$

Pozn.: Výraz $\lceil x \rceil$ značí nejmenší celé číslo větší nebo rovno x a $\lfloor x \rfloor$ značí největší celé číslo menší nebo rovno x .

Typický tvar uzlu:



Př) Struktura B⁺ stromu pro $n=3$



- Operace na B+ stromu

Uvažujeme pro jednoduchost index s unikátními hodnotami

➤ Vyhledání

Nastav kořen jako aktuální uzel;

WHILE aktuální uzel není listem DO

BEGIN

 Najdi nejmenší hodnotu vyhledávacího klíče K_i
 v uzlu, která je větší než hledaná hodnota;

 IF existuje THEN

 Nastav jako aktuální uzel, na který ukazuje P_i

 ELSE

 Nastav jako aktuální uzel, na který ukazuje
 poslední ukazatel v aktuálním uzlu

END

IF existuje v aktuálním uzlu hodnota K_i rovná hledané
hodnotě THEN

 Ukazatel P_i ukazuje na hledaný záznam

ELSE

 Záznam s hledanou hodnotou neexistuje;

➤ Vkládání

Najdi listový uzel U , který by měl obsahovat novou
položku indexu;

IF je v uzlu U místo pro vložení položky THEN

 Vlož položku indexu do uzlu U

ELSE

BEGIN /* Rozštěpení uzlu */

 Alokuj nový uzel U' indexu;

 Rozděl hodnoty štěpeného uzlu U včetně vkládané
 hodnoty na dvě „stejně velké“ skupiny. Jedna
 zůstane v uzlu U a druhá se přesune do uzlu U' ;

REPEAT

 Rekurzivně pokračuj s vkládáním nové položky
 indexu do uzlu předchůdce ve stromu s první

```

    hodnotou vyhledávacího klíče uzlu  $U$ 'a
    ukazatelem na tento uzel
UNTIL nedošlo k dalšímu štěpení nebo byl rozštěpen
kořen
END;
IF byl rozštěpen kořen THEN
    Vytvoř nový kořen s ukazateli na dva uzly vzniklé
    rozštěpením původního kořene;

```

➤ Rušení

```

Najdi listový uzel  $U$ , který obsahuje rušenou položku
indexu;
Zruš položku indexu v uzlu  $U$ ;
IF kleslo zaplnění uzlu  $U$  pod  $\lceil (n-1)/2 \rceil$  THEN
    IF lze přesunout obsah do levého souseda uzlu  $U$ 
    THEN /* slévání uzlů*/
        Přesuň obsah uzlu  $U$  a zruš prázdný uzel  $U$ ;
    REPEAT

```

```

    Rekurzivně pokračuj s rušením položky indexu
    s ukazatelem na uzel  $U$  v uzlu předchůdce ve
    stromu
UNTIL nedošlo k dalšímu slévání nebo bylo
dosaženo kořene;
IF kořen má jen jednoho následníka THEN
    Zruš kořen, novým kořenem se stane následník
ELSE
BEGIN /* redistribuce hodnot*/
    Redistribuuuj hodnoty a ukazatele sousedních
    uzlů tak, aby oba splňovaly podmínku
    minimálního zaplnění;
    Aktualizuj hodnotu vyhledávacího klíče v uzlu
    předchůdce ve stromu
END

```

- Organizace souborů jako B+strom

- primární soubor tvoří sekvenční část indexu
- při vkládání a rušení se uvažuje větší počet sourozeneckých uzlů (úspora prostoru) → lepší využití ($\lfloor 2n/3 \rfloor$), vyšší režie

Př) Uvažujme index pro primární klíč tabulky, která má 15000 řádků ve tvaru B+ stromu. Blok primárního souboru obsahuje až 20 záznamů na blok.

Jaký bude počet bloků a úrovní indexu, obsahuje-li blok indexu maximálně 50 hodnot primárního klíče?"

	Min. počet ukazatelů	Max. počet ukazatelů
Indexová část	$\lceil 51/2 \rceil = 26$	51
Sekvenční část	$\lceil (51-1)/2 \rceil = 25$	50

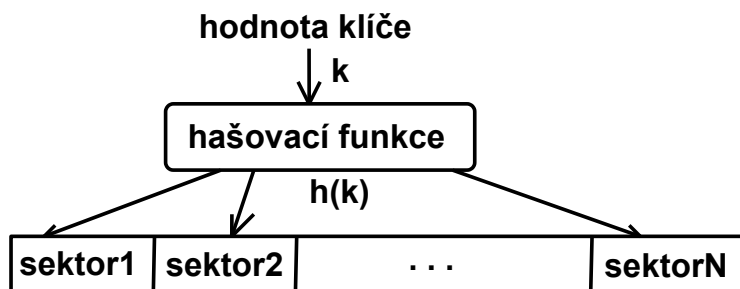
Úroveň	Minimální zaplnění (maximum bloků)	Maximální zaplnění (minimum bloků)
Sekvenční	$\lfloor 15000/25 \rfloor = 600$ bloků	$\lceil 15000/50 \rceil = 300$ bloků
Indexová X	$\lfloor 600/26 \rfloor = 23$ bloků	$\lceil 300/51 \rceil = 6$ bloků
Indexová X-1	1	1

Index bude mít 3 úrovně. Sekvenční část indexu bude mít 300 až 600 bloků a indexová bude tvořena kořenem a jednou úrovní s 6 až 23 bloky.

6.8. Hašování

- Hašované soubory (přímé hašování)

Hašovací funkce převádí hodnotu vyhledávacího klíče na adresu sektoru záznamů (typicky blok).



➤ Požadavky na hašovací funkci

- rozložení hodnot je rovnoměrné
- rozložení hodnot je náhodné

Př) Hašovaný soubor se záznamy tabulky Ucet

- a) pro vyhledávací klíč pobočka (velká banka s množstvím poboček).

- 27 sektorů s rozdělováním podle prvního písmene názvu
- jednoduché, ale rozdělení nebude rovnoměrné

b) pro vyhledávací klíč stav

- rozdělení na intervaly, např. 0 – 100000, 101000 – 200000, ...
- jednoduché, ale rozdělení nebude rovnoměrné

➤ Typické hašovací funkce

- $\text{součet_bin_reprezentace_znaků_řetězce} \text{ MOD počet_sektorů}$

➤ Přetečení sektorů

- nedostatečný počet sektorů

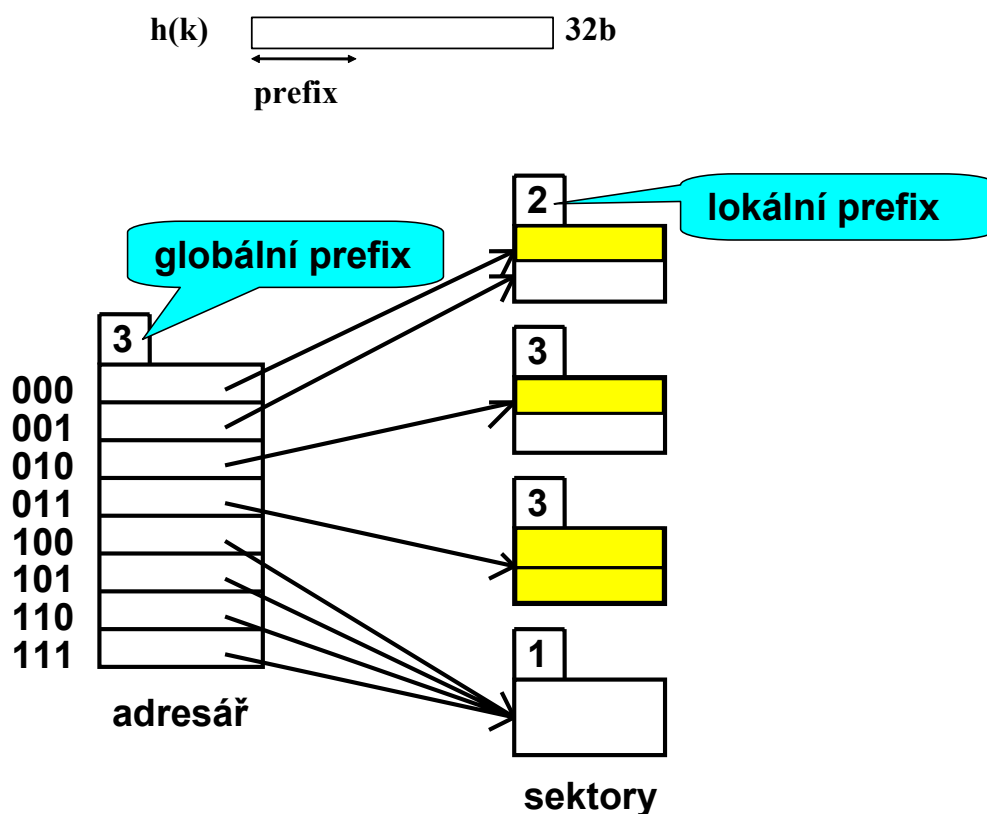
- musí platit: $n_s > n_z/z_s$, kde n_s je počet sektorů n_z je celkový počet záznamů a z_s je počet záznamů sektoru

- přetížení některých sektorů (skew)

- nerovnoměrné rozdělení hodnot vyhledávacího klíče nebo nerovnoměrné rozdělení počtu hodnot se stejnou hodnotou vyhledávacího klíče

- počet sektorů se volí $(n_z/z_s)^*(1+d)$, kde d je faktor korekce (fudge), typicky 0.2
- řešení řetězením přetokových sektorů
- Hašované indexy (nepřímé hašování)
 - soubor může být hašovaný pouze podle jednoho vyhledávacího klíče
 - u hašovaného indexu jsou v sektorech položky s hodnotou vyhledávacího klíče a ukazatelem do souboru (obdoba sekundárních indexů)
- Nevýhoda statického hašování
 - hašovací funkce se musí zvolit při implementaci, resp. definici hašovaného souboru a nelze ji snadno změnit se změnou velikosti tabulky
- Dynamické hašování
 - dynamická modifikace hašovací funkce tak, aby odrážela změny velikosti tabulky

➤ Rozšířitelné hašování



6.9. Shlukování (clustering)

- Podstata

- umístění záznamů se stejnými vlastnostmi (hodnotou tzv. klíče pro shlukování (cluster key)) fyzicky blízko u sebe
- přístup k záznamům prostřednictvím indexu (v listové části jsou přímo záznamy) nebo hašováním (obyčejné přímé hašování)
- v souboru mohou být záznamy jedné nebo několika tabulek

Př) shlukování v Oracle (viz Dodatek B)

6.10. Bitmapové indexy

- Podstata

- uložení informace o tom, ve kterých záznamech tabulky se vyskytuje daná hodnota vyhledávacího klíče, ve formě bitových vektorů → velikost bitmapového souboru je výrazně menší

- Předpoklad

- očíslování jednotlivých řádků tabulky
- ve sloupci vyhledávacího klíče je relativně málo různých hodnot

Př) Klient

číslo záznamu	r_cislo	jmeno	prijmova skupina	mesto
0	440726/0672	Jan Novák	S2	Brno
1	530610/4532	Petr Veselý	S3	Brno
2	601001/2218	Ivan Zeman	S1	Brno
3	510230/048	Pavel Tomek	S2	Brno
4	580807/9638	Josef Mádr	S5	Brno
5	625622/6249	Jana Malá	S2	Vyškov

Bitmapy pro příjmovou skupinu

S1	001000
S2	100101
S3	010000
S5	000010

Bitmapy pro město

Brno	111110
Vyškov	000001

- **Použití**

- dotazy na hodnoty několika vyhledávacích klíčů spojené logickými spojkami

Př) Najdi klienty z Brna z příjmové skupiny S2.

$111110 \wedge 100101 = 100100 \rightarrow$ vyberou se odpovídající záznamy

- agregační funkce COUNT (sloupec_vyhledávacího_klíče)

Př) Kolik klientů z Brna patří do příjmové skupiny S2?

$\text{počet_jedniček}(111110 \wedge 100101) = \text{počet_jedniček}(100100) = 2$

- **Implementace**

- využití instrukcí pro logické operace po bitech
- uložení tzv. existenční bitmapy pro informaci o zrušených záznamech
- pole pro převod bitového vektoru (např. byte) na počet jedniček v něm

6.11.Fyzický návrh databáze

- Zdroje

- nefunkční (výkonnostní požadavky)
- pochopení zátěže

- Popis zátěže

- Seznam dotazů a jejich četnosti

- které tabulky se čtou
- které atributy jsou vybírány (klauzule SELECT)
- atributy se vyskytují v podmínkách selekce nebo spojení (klauzule WHERE) a jak selektivní tyto podmínky jsou

- Seznam aktualizací a jejich četnosti

- atributy se vyskytují v podmínkách selekce nebo spojení (klauzule WHERE) a jak selektivní tyto podmínky jsou
- typ aktualizace (INSERT, DELETE, UPDATE) a aktualizovaná tabulka

- pro UPDATE sloupce, které jsou modifikovány

- Výkonnostní požadavky pro každý typ dotazu a aktualizace

- Základní rozhodování fyzického návrhu

- Kdy a jaké přístupové metody použít

- pro které tabulky a které sloupce, resp. kombinace sloupců
- typ přístupové metody

Základní zásady pro výběr indexů:

1. Zda indexovat

Nevytvářej index, pokud nepřispěje ke zrychlení dotazů, resp. přístupu při aktualizaci. Důležité kritérium – selektivita dotazu,

2. Volba vyhledávacího klíče

Sloupce v klauzulích WHERE a výrazech s JOIN jsou kandidáty na vyhledávací klíč. Pro porovnání na rovnost je efektivnější hašování, pro rozsahové dotazy je nutné použít B+ stromy.

3. Složené vyhledávací klíče

Složené vyhledávací klíče zvažuj v těchto situacích:

- a) Klauzule WHERE obsahuje podmínky pro více než jeden sloupec tabulky.
- b) V případě, kdy bude možné použít pro vyhodnocení dotazu strategii založenou jen na indexu.

4. Zda použít shlukování záznamů (clustering index)

Pro tabulku lze použít nejvýše jedno shlukování. Hlavní přínos je pro rozsahové dotazy. Může-li dotaz být řešen strategií založenou jen na indexu, cluster nic nepřináší.

5. Hašování vs. B+strom

```
SELECT a
FROM R
WHERE a = c;
```

```
SELECT a
FROM
WHERE a BETWEEN c1 AND c2;
```

B+stromy jsou obecnější (rozsahové dotazy). Hašování je výhodnější v těchto situacích:

- a) Pro podporu přirozeného spojení a spojení na rovnost metodou „hash join“ (pro).
- b) Existuje-li velmi významný dotaz na rovnost a nejsou rozsahové dotazy se sloupci vyhledávacího klíče.
- c) Lze-li určit potřebný prostor pro statické hašování.

6. Vyvážení režie údržby indexu

Po vytvoření seznamu žádoucích indexů zvažuj dopad na aktualizace.

- a) Pokud údržba indexu zpomalí významné aktualizací operace, vyřaď ho ze seznamu.
- b) Pamatuj ale, že přidání indexu může významně urychlit i aktualizací operace (prohledávací UPDATE a DELETE).

- Zda modifikovat konceptuální schéma z důvodu vyšší výkonnosti
 - Alternativní normalizované schéma, je-li více možností
 - Denormalizace
 - omezení spojování tabulek vs. nebezpečí nekonzistence a prostor.
 - Vertikální rozčlenění tabulky
 - jsou-li významné dotazy jen na některé sloupce rozsáhlé tabulky.
 - Replikace dat
- Zda přepsat často prováděné dotazy a transakce, aby byly rychlejší

Literatura

- 1. Silberschatz, A., Korth H.F, Sudarshan, S.: Database System Concepts. Fourth Edition. McGRAW-HILL. 2001, str. 393 – 492.**
- 2. Pokorný, J.: Databazová abeceda. Science, Veletiny, 1998, str. 27 – 34, 81 – 88.**
- 3. Ramakrishnan, R.: Database Management Systems. WCB/McGraw-Hill, 1998, str. 436 – 468.**