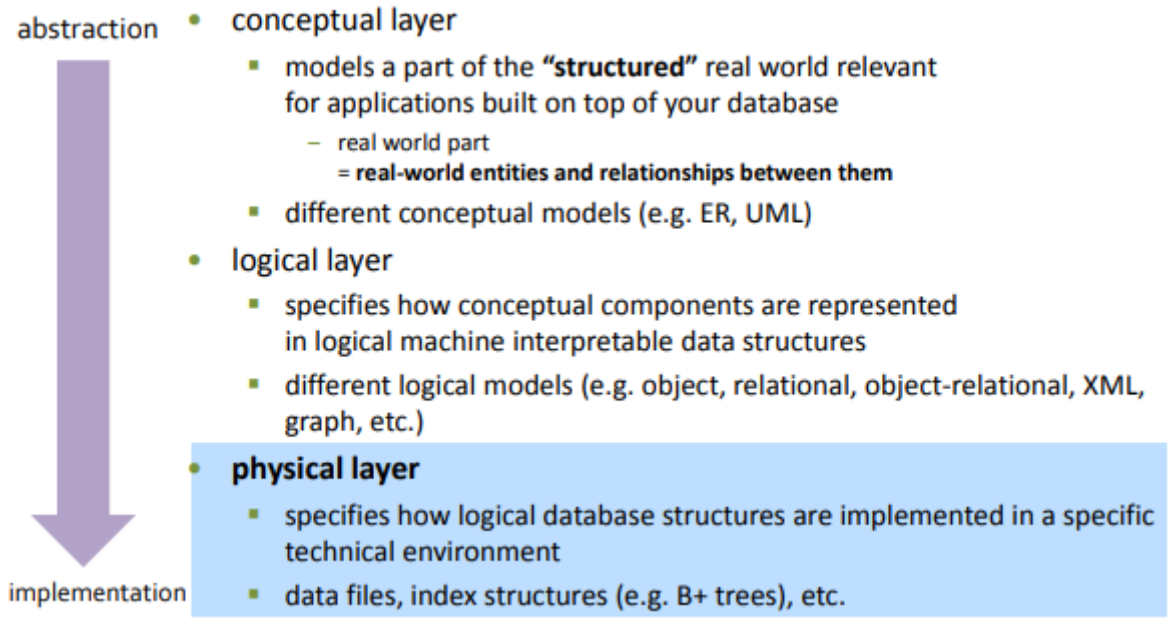
Physical Layer (8)



- relace/tabulky jsou uloženy v souborech na disku
- potřebujeme organizovat tabulkové záznamy uvnitř souboru
    - … a k tomu efektivní ukládání, aktualizaci a přístup
    - př.: Employees (name char(20), age integer, salary integer)

Paging (stránkování)

- záznamy jsou uloženy ve **stránkách** disku (**disk page**) pevné velikosti (pár kB)
    - kvůli HW, organizace dat se tomu musí přizpůsobit
    - firmware HW může přistupovat jen k celým stránkám
    - **reálný čas pro I/O operace** = *čas hledání + rotační zpoždění + čas přenosu dat*
    - sekvenční přístup ke stránkám je mnohem rychlejší než náhodný
        - hledání a rotace nejsou potřeba

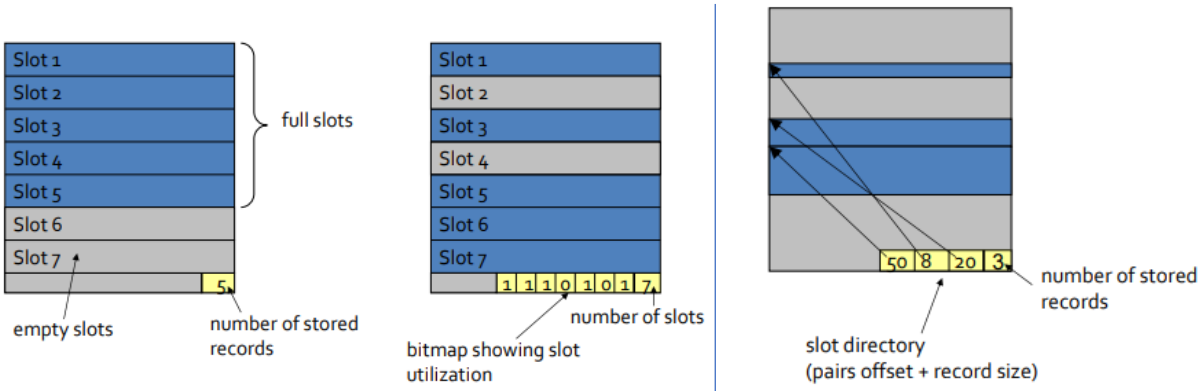
- I/O je jednotkou time cost
- stránka je rozdělena do **slotů**, kde se ukládají **záznamy**
    - record je identifikován pomocí "rid" = page id + slot id

- záznam může být uložen:
    - **do vícera stránek** (lepší využití místa, ale potřeba více I/O pro manipulaci)
    - **do jedné stránky** (část stránky může být nevyužita, ale potřeba méně I/O)

- ideální by bylo, kdyby záznam zabral celou jednu stránku

- jsou-li v záznamech použity jen **datové typy s pevnou velikostí** → pevná velikost záznamu
    - pevná velikost záznamů = pevná velikost slotů

- jsou-li tam i **typy s proměnnou velikostí** (varchar(X),…) → proměnná velikost záznamů
    - proměnná velikost záznamů = potřeba adresáře pro slot v hlavičce stránky



1 - organizace stránky s pevnou velikostí

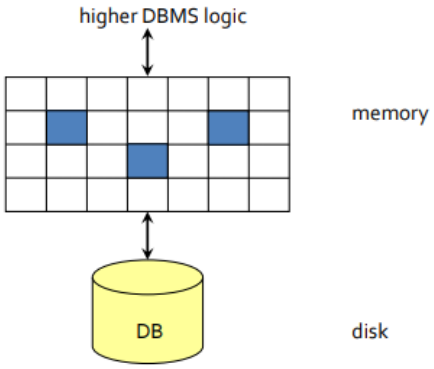
2 – s proměnnou velikostí

Buffer manager

- **buffer** = kus hlavní paměti pro dočasné ukládání diskových stránek
  - stránky jsou mapovány 1:1 do paměťových rámců (memory frame)
  - každý frame má **2 flagy**:
    - **pin_count** = počet referencí na stránku v něm
    - **dirty** = indikace modifikovaného záznamu

- **buffer manager** – implementuje R/W operace pro vyšší DBMS logiku
  - **čtení** – vrací stránku z bufferu + zvýší pin_count
    - pokud tam není, načte ji z disku
  - **zápis** – vloží stránku do bufferu + nastaví dirty

- je-li buffer plný, některé stránky se nahradí
  - různé strategie, např. LRU
  - je-li nahrazená stránka dirty, musí být uložena



Organizace DB souborů

- **data files** – obsahují tabulková data
- **index files** – zrychlují zpracování dotazů
- **systémový katalog** – obsahuje metadata (schémata, integritní omezené, jména indexů,…)

Data files

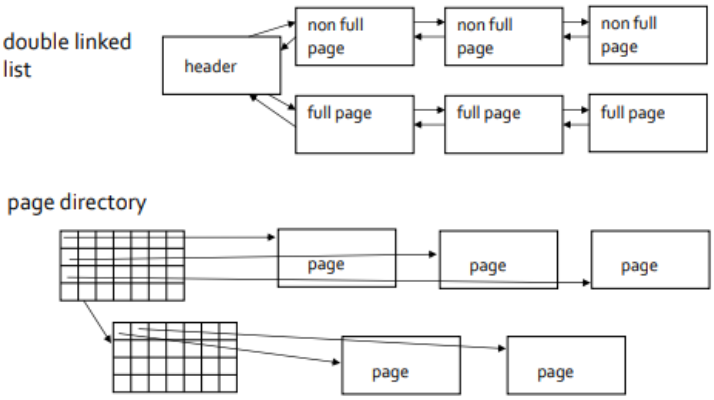
- sledujeme průměrnou I/O cenu jednoduchých operací
- cost model: **N = počet stránek**, **R = záznamů /stránku**

Heap file

- záznamy ve stránkách nejsou seřazené
- jsou ukládány v pořadí vkládání
- hledání může být jen přes sekvenční skenování
- rychlé vložení záznamu (na konec souboru)
- problémy s mazáním (díry – nevyužité místo)

- sequential reading of pages
  SELECT * FROM Employees
- searching on equality
  SELECT * FROM Employees WHERE age = 40
- searching on range
  SELECT * FROM Employees
  WHERE salary > 10000 AND salary < 20000
- record insertion
  INSERT INTO Employees VALUES (...)
- record deletion based on rid
  DELETE FROM Employees WHERE rid = 1234
- record deletion
  DELETE FROM Employees WHERE salary < 5000

- **údržba prázdných stránek haldy:**
  - double linked list (hlavička + seznam plných a ne-plných stránek)
  - stránkový adresář (linked list adresářů, každý prvek odkazuje na stránku s daty)



- **ceny jednoduchých operací:**
  - sekvenční přístup = N
  - hledání dle rovnosti = N
  - hledání dle rozsahu = N
  - vložení záznamu = 1
  - smazání záznamu…
    - 2, při ceně rid-vyhledávání 1 I/O
    - N či 2*N, při vyhledávání dle rovnosti či rozsahu

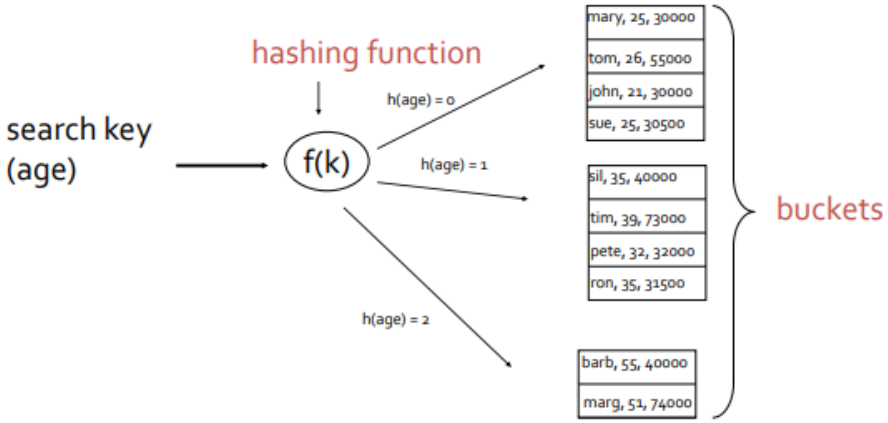
Sorted file

- záznamy ve stránkách jsou řazeny dle vyhledávacího klíče (1+ atributů)
- souvislá údržba stránek – žádné díry
- rychlé vyhledávání dle rovnosti či rozsahu
- pomalé vkládání či mazání (posun zbytku stránek)
- v praxi:
  - seřazený soubor na začátku
  - každá stránka má overhead, kam se dá vkládat
  - je-li overhead plný, použijí se updatované stránky (linked list)
  - čas od času potřeba reorganizace

- **ceny jednoduchých operací:**
  - sekvenční přístup = N
  - hledání dle rovnosti = $\log_2 N$
  - hledání dle rozsahu = $\log_2 N + M$ (počet relevantních stránek)
  - vložení záznamu = N
  - smazání záznamu = $\log_2 N + N$ (dle klíče)

Hashed file

- organizováno do K bucketů (rozšiřitelných do vícera stránek)
- záznam je vložen/čten do/z bucketu dle hashovací funkce f aplikované na vyhledávací klíč
  - bucked id = f(key)
- je-li bucket plný, alokují se nové stránky a nalinkují na bucket (linked list)
- rychlé vyhledávání a mazání dle rovnosti
- větší overhead, problémy s řetězenými stránkami



- **ceny jednoduchých operací:**
  - sekvenční přístup = N
  - hledání dle rovnosti = N/K (best case, K = počet bucketů)
  - hledání dle rozsahu = N
  - vložení záznamu = N/K (best case)
  - smazání záznamu = N/K + 1 (best case)

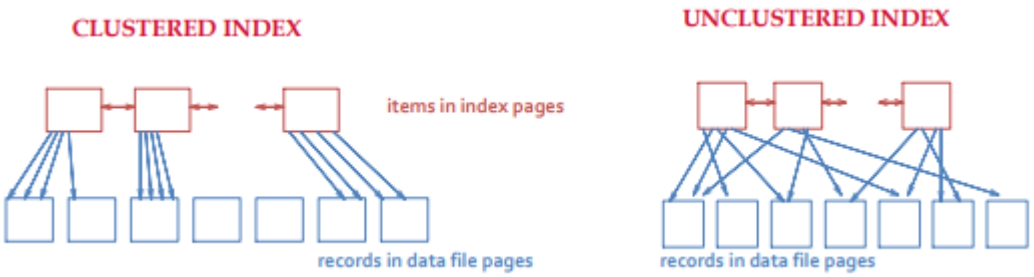
Indexing

- **index** = pomocná struktura poskytující rychlé vyhledávání dle vyhledávacího klíče
- organizace do diskových stránek (podobné data files)
- většinou obsahuje jen vyhledávací klíče a odkazy na příslušné záznamy (rid)
- potřeba mnohem méně místa než data files (třeba 100x)

- **index item může obsahovat:**
  - celý záznam (index a data file jsou stejné)
  - pár <klíč, rid>
  - pár <klíč, rid-list>, kde rid-list je seznam linků na záznamy se stejným vyhled. klíčem

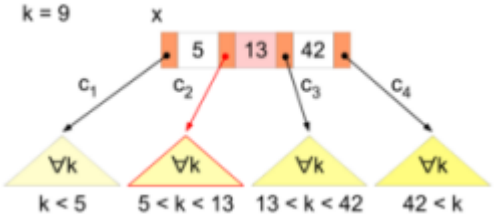
- **clustered index** – řazení index itemů je (skoro) stejné jako řazení v data file
  - stromový index, celý záznam, hashed index,…
  - primární klíč = vyhledávací klíč užitý v clustered indexu
  - výhoda – vysoká rychlost při vyhledávaní v rozsahu (sekvenční čtení)
  - nevýhoda – velký overhead kvůli zachování pořadí

- **unclustered index** – řazení vyhledávacích klíčů není zachováno

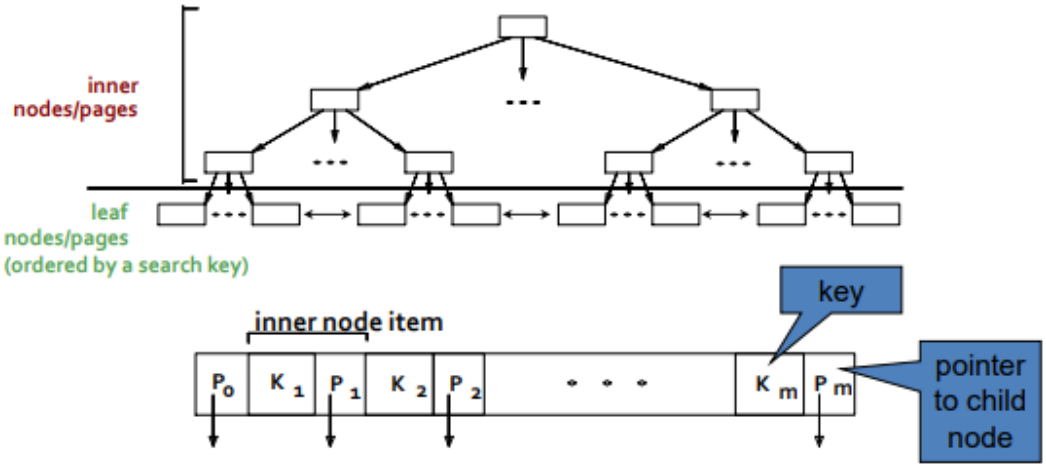


B⁺-strom

- rozšiřuje B-strom (vyvážený stromový index):
  - všechny klíče jsou v listech – vnitřní nody obsahují indexované intervaly
  - linkuje listové stránky pro efektivní rozsahové dotazy



- zajišťuje logaritmickou složitost pro vkládání, vyhledávání/mazání dle rovnosti (bez duplikátů)
- garantuje 50% využití nodů (stránek)



Hashed index

- podobný hashovanému data file (buckety + hashovací funkce)
- buckety obsahují jen hodnoty klíčů s rid
- stejné výhody a nevýhody

Bitmapy

- vhodné pro indexování atributů datových typů s nízkou kardinalitou
  - např.: atribut FAMILY_STATUS = {single, married, divorced, widow}

- pro každou hodnotu „**h**“ indexovaného atributu „**a**“ je vytvořena bitmapa (binární vektor), kde 1 na i-té pozici znamená, že hodnota „**h**“ se vyskytuje i-tém záznamu (v atributu „**a**“)
  - bitové OR = 1 (každý atribut má hodnotu)
  - bitové AND = 0 (hodnoty atributu jsou deterministické)

Name	Address	Family status
John Smith	London	single
Rostislav Drobil	Prague	married
Franz Neumann	Munich	married
Fero Lakatoš	Malacky	single
Sergey Prokofjev	Moscow	divorced

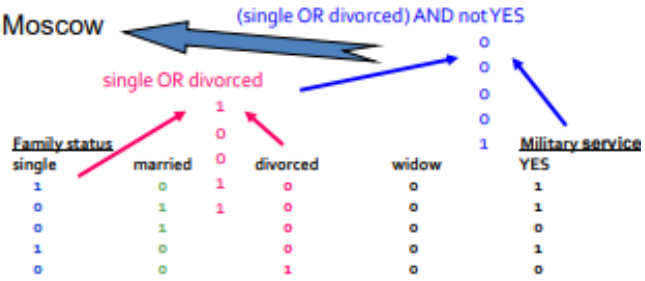
single	married	divorced	widow
1	0	0	0
0	1	0	0
0	1	0	0
1	0	0	0
0	0	1	0

- **vyhodnocení dotazu:**
  - bitové operace s bitmapami atributů
  - výsledná bitmapa označuje dotazované záznamy

  - **příklad**:
    - *Which single or divorced people did not complete the military service?* (bitmap(single) **OR** bitmap(divorced)) **AND not** bitmap(YES)

answer: Sergey Prokofjev, Moscow

Name	Address	Military service	Family status
John Smith	London	YES	single
Rostislav Drobil	Prague	YES	married
Franz Neumann	Munich	NO	married
Fero Lakatoš	Malacky	YES	single
Sergey Prokofjev	Moscow	NO	divorced



- **výhody:**
  - efektivní ukládání, může být kompresováno
  - rychlé zpracování dotazů, bitové operace jsou rychlé
  - snadná paralelizace

- **nevýhody:**
  - vhodné jen pro atributy s malou doménou atributů
  - rozsahové dotazy se zpomalují lineárně s počtem hodnot v rozsahu (musí se zpracovat bitmapy pro všechny hodnoty)