



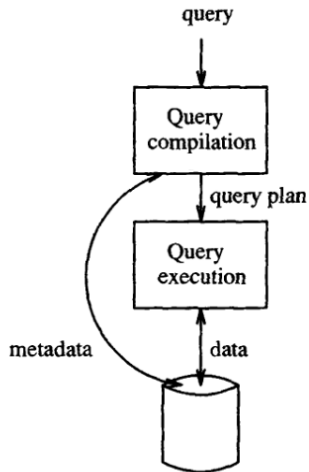
Databázové technologie  
**Provádění dotazů**

Petr Krajča

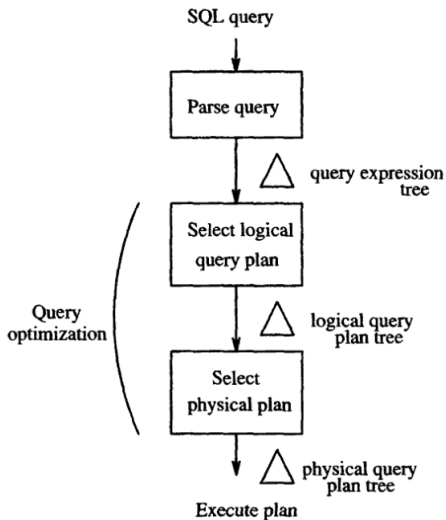


Katedra informatiky  
Univerzita Palackého v Olomouci

- query processor – klíčová část SŘBD
- dotazy zpracovávají ve dvou krocích
  - 1 dotaz je přeložen na prováděcí plán
  - 2 data jsou zpracována podle prováděcího plánu



- dotazy překládány v několika krocích
  - 1 parsování – text je převed na syntaktický strom
  - 2 prvotní sestavení prováděcí plánu – ze syntaktického stromu je vytvořen **logický prováděcí plán** (blízký rel. algebře)
  - 3 vytvoření **fyzického prováděcího plánu** (operace pracující s daty)
- interní reprezentace jako strom operací
- využití algebraických vlastností operací
- nutné brát v potaz efektivitu jednotlivých operací (zavisí na datech, jejich velikosti, dostupné paměti, indexech, hodnotách atributu, ...)



- praktické relační databázové systémy se odchyľují od teoretického modelu
  - umožňují duplicitní řádky
  - přidávají operace, které nemají oporu/smysl v rel. modelu (řazení, eliminace duplicitních řádků)
- alternativní teoretický základ
  - relace je multimnožina (bag) ntic
  - přidány nutné a praktické operace
- reálně se používá množinový i multimnožinový pohled (současně)

## Multimnožina

- jednotlivé prvky se v množině mohou vyskytovat vícekrát
- tomu uzpůsobené operace (počty prvků se sčítají, odčítají, atp.)
- pro  $R = \{A, B, B\}, S = \{C, A, B, C\}$ 
  - $R \cup S = \{A, A, B, B, B, C, C\}$  (součet, liší se od standardní operace  $\cup$  na multimnožinách)
  - $R \cap S = \{A, B\}$  (minimum)
  - $R - S = \{B\}$  (rozdíl)
- alternativní pohled na relace  $\mathcal{D} : \prod_{y \in R} D_y \rightarrow \mathbb{N}_0$

- většina operací rel. algebry snadno převoditelná i na multimnožiny
- $\sigma, \pi, \bowtie, \times, \cap, \cup, -$
- další operace plynoucí z vlastností SQL

## Aggregace a seskupení

- tyto dvě operace z pohledu SQL spolu úzce souvisí
- pro relaci  $\mathcal{D}$  na relačním schématu  $R$
- budeme  $\gamma_L(\mathcal{D})$  značit operaci seskupení, kde  $L$  obsahuje
  - atributy z  $R$  (dle kterých dochází k seskupení), nebo
  - dvojice  $f_A(y_i) \rightarrow y'_i$ , kde  $y_i \in R$ ,  $y'_i \in Y$  a  $f_A$  je agregační funkce (min, max, sum, ...)

## Eliminace duplicit

- operaci eliminující duplicitní ntice v relaci budeme značit  $\delta(\mathcal{D})$
- redundantní, lze nahradit  $\gamma$

## Řazení ntic

- pro řazení ntic se zavádí operace  $\tau_L(\mathcal{D})$ , kde  $L$  je posloupnost atributů z  $R$



## table scan

- nejzákladnější operací je přečtení všech dat dané tabulky
- předpokládá se, že data jsou v blocích (rozděleny podle tabulek)
- data načítána po jednotlivých blocích

## index scan

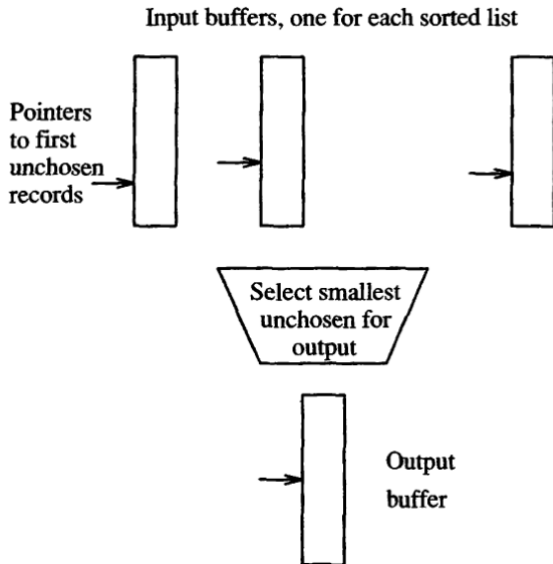
- pokud je k dispozici index, je možné získat řádky splňující danou podmínku
- získat řádky v seřazené podobě vzhledem k danému indexu

## řazení při čtení dat

- buď součástí dotazu (ORDER BY) nebo jako předpoklad jiné operace
- několik možností
  - 1 pro malá data přímo v paměti
  - 2 s využitím indexu
  - 3 vhodný řadící algoritmus (dvoufázový, vícecestný mergesort)



- vhodný v situacích, kdy dostupná paměť je menší než data
- fáze 1: rozdělení dat na menší části a jejich seřazení
  - 1 do dostupné paměti je načtená část bloků tabulky
  - 2 řádky tabulky jsou seřazené (v paměti)
  - 3 seřazené části jsou uloženy na disk
  
- fáze 2: sloučení seřazených částí
  - 1 pro každou samostatně seřazenou část se načte do paměti jeden blok
  - 2 je vybrán nejmenší dosud neuvažovaný řádek
  - 3 ten je zapsán na první volné místo výstupního bloku (bufferu)
  - 4 pokud je výstupní buffer zaplněn, je zapsán na disk a v paměti vynulován
  - 5 pokud je blok dané části zcela zpracován, načte se další (pokud je k dispozici)
  - 6 opakuje se od kroku 2, dokud nejsou zpracovány všechny řádky







- pro hodnocení náročnosti jednotlivých operací je důležité zavést model
- určuje, který fyzický prováděcí plán bude zvolen
- nejnáročnější I/O operace (operace v paměti budeme zanedbávat)
- budeme předpokládat:
  - všechna vstupní data jsou na disku (v praxi nemusí platit)
  - I/O operace pro výstup lze zanedbat (obvykle předány dál)
- základní odhad náročnosti operací lze udělat na základě
  - počtu bloků dané tabulky, budeme značit  $B(R)$
  - dostupného počtu bufferů v paměti, které lze použít pro zpracování dotazu, budeme značit  $M$  (buffer má stejnou velikost jako blok)
- dále lze využít počet:
  - počet řádek tabulky  $T(R)$
  - počet různých hodnot daného atributu v tabulce  $V(R, a)$



- table scan, řazení tabulky v paměti:  $B(R)$
- index scan: více než  $B(R)$  (samostatné vyhledání řádků, může být rychlejší)
- dvoufázový, více cestný mergesort:  $3 \cdot B(R)$ 
  - načtení dat v první fázi:  $B(R)$
  - uložení dat v první fázi:  $B(R)$
  - postupné načítání dat v druhé fázi:  $B(R)$
  - uložení výsledku: zanedbáváme



- operátory typicky realizovány formou iterátorů
- operace:
  - `open` – inicializace iterátoru
  - `getNext` – vrátí další řádek, nebo signalizuje konec dat
  - `close` – ukončí práci s iterátorem (typicky použije operaci `close` na své argumenty)
- v ideálním případě umožňují proudové zpracování dat
- „data tečou” z listů ke kořenu (uzly představují operace s daty)
- někdy nutné logiku operátoru z velké části realizovat v operaci `open` (např. řazení)

# Table scan jako iterátor



```
def open(R):
    B := prvni blok R
    T := prvni radek v bloku B
    Found := true

def getNext(R):
    If (T ukazuje za posledni radek v B):
        B := dalsi blok R
        If (dalsi blok neexistuje):
            Found := false
            Return None
        Else: // dalsi blok nacten
            T := prvni radek bloku B
    U := T
    T := posun na dalsi radek v bloku B
    return U

def close(R):
    pass
```



- široká paleta operátorů pro práci s daty
- liší se použitým stylem práce s daty
- počtem průchodů

## dle typu práce s daty

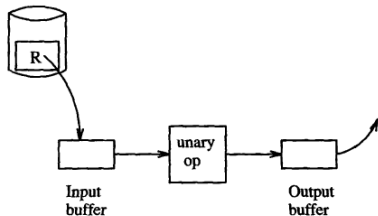
- algoritmy využívající řazení (sorting-based)
- algoritmy využívají hash tabulky (hash-based)
- algoritmy využívají indexy (index-based)

## dle počtu průchodů

- jednopřechodové – vhodné pro operace  $\sigma$ ,  $\pi$  nebo pokud se data vejdou do paměti
- dvouprůchodové – data jsou načtena, částečně zpracována a uložena zpět na disk, aby mohla být načtena transformována do finální podoby
- víceprůchodová – obvykle zobecnění dvouprůchodových algoritmů pro větší data

## 1 unární operace postupně zpracovávající jednotlivé řádky

- restrikce a projekce nevyžadují mít v paměti celou tabulku (neřeší se duplicity)
- lze načítat jednotlivé bloky a v nich zpracovávat řádky (přímočará implementace)
- vyžadují jeden paměťový buffer
- náročnost operace dána čistě čtením dat



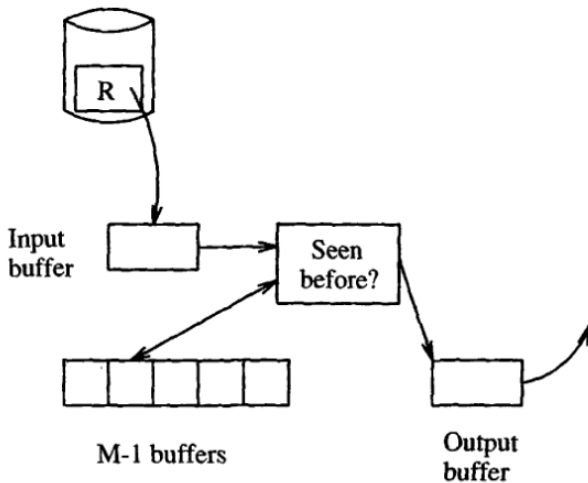
## 2 unární operace pracující s celou tabulkou

- omezení na tabulky  $B(R) \leq M$
- operace seskupení/agregace, eliminace duplicitních řádků

## 3 binární operace pracující s celou tabulkou

- zbývající operace (množinové operace, spojení, ...)
- vyžadují, aby velikost alespoň jednoho argumentu byla menší nebo rovna  $M$

- 1 Načte se jeden řádek ze vstupu,
  - 2 pokud se řádek na vstupu vyskytl poprvé zapíšeme jej na výstup a uložíme si tuto informaci,
  - 3 pokračuje se dalším řádkem.
- jeden paměťový buffer potřebujeme pro načtení vstupu,  $M - 1$  bufferů můžeme využít pro evidenci zpracovaných řádků
  - vhodná datová struktura? (hash tabulka, vyvážený vyhledávací strom)
  - tyto struktury mají svou režii (můžeme zanedbat)
  - pro velikost by mělo platit  $B(\delta(R)) < M$ , jinak hrozí thrashing







- parametr  $L$  operace  $\gamma_L$  udává žádný, jeden nebo více atributů, dle kterých má dojít k seskupení
- pro každou skupinu řádků evidujeme v paměti jeden záznam, tj. hodnoty atributů + průběžné hodnoty jednotlivých agregačních funkcí (např. počet hodnot, minimální/maximální hodnotu, ...)
- nelze zapsat data na výstup, dokud nejsou zpracovány všechny řádky vstupu
- nezapadá do konceptu iterátorů
- není snadné určit velikost dat v paměťovém bufferu



- s výjimkou sjednocení multimnožin je nutné načíst menší operand do paměti
- tj.  $\min(B(R), B(S)) \leq M$ , I/O operace:  $B(R) + B(S)$
- v popisu operací budeme předpokladat, že  $B(R) > B(S)$

## Sjednocením množin

- do  $M - 1$  paměťových bufferů se načte  $S$  a vytvoří se vhodná vyhledávací datová struktura (kde klíčem jsou celé řádky)
- obsah  $S$  je zapsán na výstup
- do zbývajících bufferů se postupně načítají jednotlivé bloky  $R$
- pro každý řádek v  $R$  se ověří, zda je v  $S$ , a pokud ne, je zapsán do výstupního bufferu

## Sjednocením multimnožin

- lze spočítat prostým překopírováním operandů na výstup
- postačuje  $M = 1$



## Průnik množin

- do  $M - 1$  paměťových bufferů se načte  $S$  a vytvoří se vhodná vyhledávací datová struktura (kde klíčem jsou celé řádky)
- do zbývajících bufferů se postupně načítají jednotlivé bloky  $R$
- pro každý řádek v  $R$  se ověří, zda je v  $S$ , a pokud ano, je zapsán do výstupního bufferu

## Průnik multimnožin

- do  $M - 1$  uložíme unikátní řádky z  $S$  a počet jejich výskytů
- do zbývajících bufferů se postupně načítají jednotlivé bloky  $R$
- pro každý řádek  $t$  z  $R$  buď
  - $t$  není v  $S$ , v takovém případě jej ignorujeme,
  - $t$  je v  $S$  a má kladný počet výskytů, pak jej zapíšeme do výstupního bufferu a snížíme počet výskytů o jedna,
  - $t$  je v  $S$  a má 0 výskytů, a ignorujeme jej.



## Rozdíl množin

- nekomutativní operace
- (pro obě varianty) do  $M - 1$  paměťových bufferů se načte  $S$  a vytvoří se vhodná vyhledávací datová struktura (kde klíčem jsou celé řádky)
- pro  $R - S$ : pro každý řádek v  $R$  se ověří, zda je v  $S$ , pokud ne, je uložen do výstupního bufferu, jinak je ignorován
- pro  $S - R$ : pro každý řádek v  $R$  se ověří, zda je v  $S$ , pokud ano, je z  $S$  odstraněn; po zpracování všech řádků je obsah bufferů s  $S$  překopírován do výstupního bufferu.

## Rozdíl multimnožin

- (pro obě varianty) do  $M - 1$  uložíme unikátní řádky z  $S$  a počet jejich výskytů
- pro  $S - R$ : pro každý řádek v  $R$  v  $S$  se sníží počet výskytů o 1, po zpracování všech řádků jsou řádky s kladným počtem výskytů překopírovány právě tolikrát do výstupního bufferu.
- pro  $R - S$ : pro každý řádek  $R$  se ověří, zda je v  $S$ , pokud ne nebo je počet výskytů 0, je zapsán do výstupního bufferu. V opačném případě se počet výskytů sníží o 1.



## Kartézský součin

- není potřeba žádná vyhledávací datová struktura
- do  $M - 1$  paměťových bufferů se načte  $S$
- pro každý řádek  $t$  z  $R$  se vytvoří spojení řádků z  $S$  s  $t$  a zapíše se do výstupního bufferu

## Přirozené spojení

- do  $M - 1$  paměťových bufferů se načte  $S$ , vytvoří se vhodná vyhledávací datová struktura (kde klíčem jsou společné atributy)
- pro každý řádek  $t$  z  $R$  se vyhledají spojitelné řádky (shodující se v hodnotách společných atributů) a spojení těchto řádků se zapíše do výstupního bufferu



Všechny obrázky:

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