SQL

Optimalizace III

Postgre SQL 13.2

- Typy indexů:
  - B-tree,
  - hash,
  - GiST,
  - SP-GiST,
  - GIN,
  - BRIN
  - by default, the CREATE INDEX command creates B-tree indexes, which fits the most common situations

Btree

- B-trees can handle equality and range queries on data that can be sorted into some ordering. In particular, the PostgreSQL query planner will consider using a B-tree index whenever an indexed column is involved in a comparison using one of these operators:
- <, <=, =, >=, >
- Constructs equivalent to combinations of these operators, such as BETWEEN and IN, can also be implemented with a B-tree index search.
- IS NULL or IS NOT NULL condition on an index column can be used with a B-tree index.
- The optimizer can also use a B-tree index for queries involving the pattern matching operators LIKE and ~ if the pattern is a constant and is anchored to the beginning of the string
  - for example, col LIKE 'foo%' or col ~ '^foo',
  - but not col LIKE '%bar'.
  - It is also possible to use B-tree indexes for ILIKE and ~*, but only if the pattern starts with non-alphabetic characters, i.e., characters that are not affected by upper/lower case conversion.
- B-tree indexes can also be used to retrieve data in sorted order. This is not always faster than a simple scan and sort, but it is often helpful.

Hash

- Hash indexes can only handle simple equality comparisons.
- The query planner will consider using a hash index whenever an indexed column is involved in a comparison using the = operator.

GiST indexes

- GiST indexes are not a single kind of index, but rather an infrastructure within which many different indexing strategies can be implemented.
- GiST stands for Generalized Search Tree. It is a balanced, tree-structured access method, that acts as a base template in which to implement arbitrary indexing schemes.
- B-trees, R-trees and many other indexing schemes can be implemented in GiST.
- One advantage of GiST is that it allows the development of custom data types with the appropriate access methods, by an expert in the domain of the data type, rather than a database expert.
- Accordingly, the particular operators with which a GiST index can be used vary depending on the indexing strategy (the operator class).
- As an example, the standard distribution of PostgreSQL includes GiST operator classes for several two-dimensional geometric data types, which support indexed queries using these operators:
  - <<, &<, &>, >>, <<|, &<|, |&>, |>>, @>, <@, ~=, &&
- GiST indexes are also capable of optimizing "nearest-neighbor" searches, such as

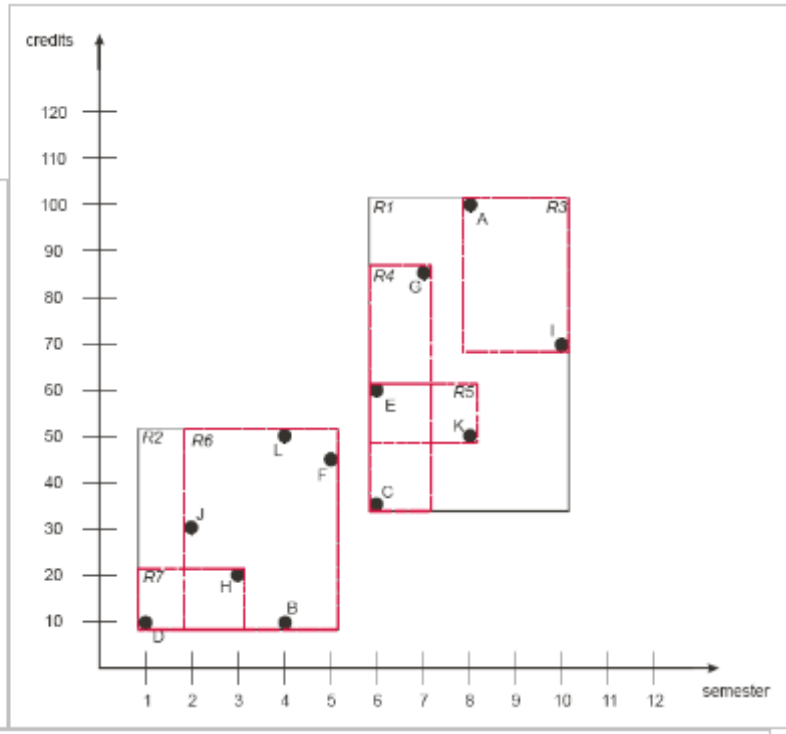
```
SELECT * FROM places ORDER BY location <-> point '(101,456)' LIMIT 10;
```

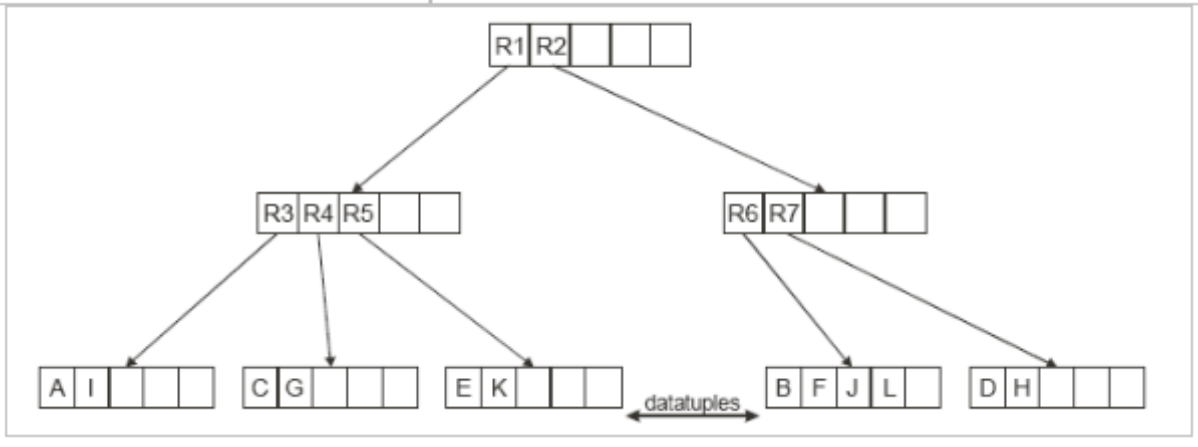
R-tree

- R-Tree mohou být využity k organizaci jakýchkoli multidimenzionálních dat tak, že taková data representují jako minimální čtverec, který takové doby odhraničí.
- Každý uzel ohraničuje své potomky a každý objekt může mít libovolný počet potomků
- Výška stromu je vždy vyvážena na log n

Příklad na R-Tree

name	semester	credits
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50





Převzato z :http://lacot.org/public/enst/bda/img/schema1.gif

Operace

- Vyhledávání: najdi všechny uzly, které mají průnik a pak pokračuj rekurzivně do těch uzlů
- Vložení: najdi místo pro vložení pomocí operací vyhledání a vložení
  - Pokud je uzel plný pak je rozděl
- Smazání: uzel se může stát nevyplněným, vybalancuj pomocí nového vložení.

SP-GiST

- SP-GiST indexes offer an infrastructure that supports various kinds of searches.
- SP-GiST permits implementation of a wide range of different non-balanced disk-based data structures, such as quadtrees, k-d trees, and radix trees (tries).
- As an example, the standard distribution of PostgreSQL includes SP-GiST operator classes for two-dimensional points, which support indexed queries using these operators:
  - <<, >>, ~=, <@, <^, >^

GIN

- GIN indexes are inverted indexes which can handle values that contain more than one key, arrays for example.
- Like GiST and SP-GiST, GIN can support many different user-defined indexing strategies and the particular operators with which a GIN index can be used vary depending on the indexing strategy.
- As an example, the standard distribution of PostgreSQL includes GIN operator classes for one-dimensional arrays, which support indexed queries using these operators:
  - <@, @>, =, &&

Optimalizace dotazů v RMD

Zpracování dotazů

- Mimo jiné překlad jazyka vyšší úrovně do výrazů implementovatelných na úrovni zpracování fyzických souborů
- Optimalizační techniky
- Vyhodnocování dotazů
  - Např. překlad SQL do rozšířené relační algebry (tam máme dost prostředků pro jejich fyzickou implementaci) není jednoznačný
- Optimalizace je založena především na heuristických algoritmech – tedy odhadech

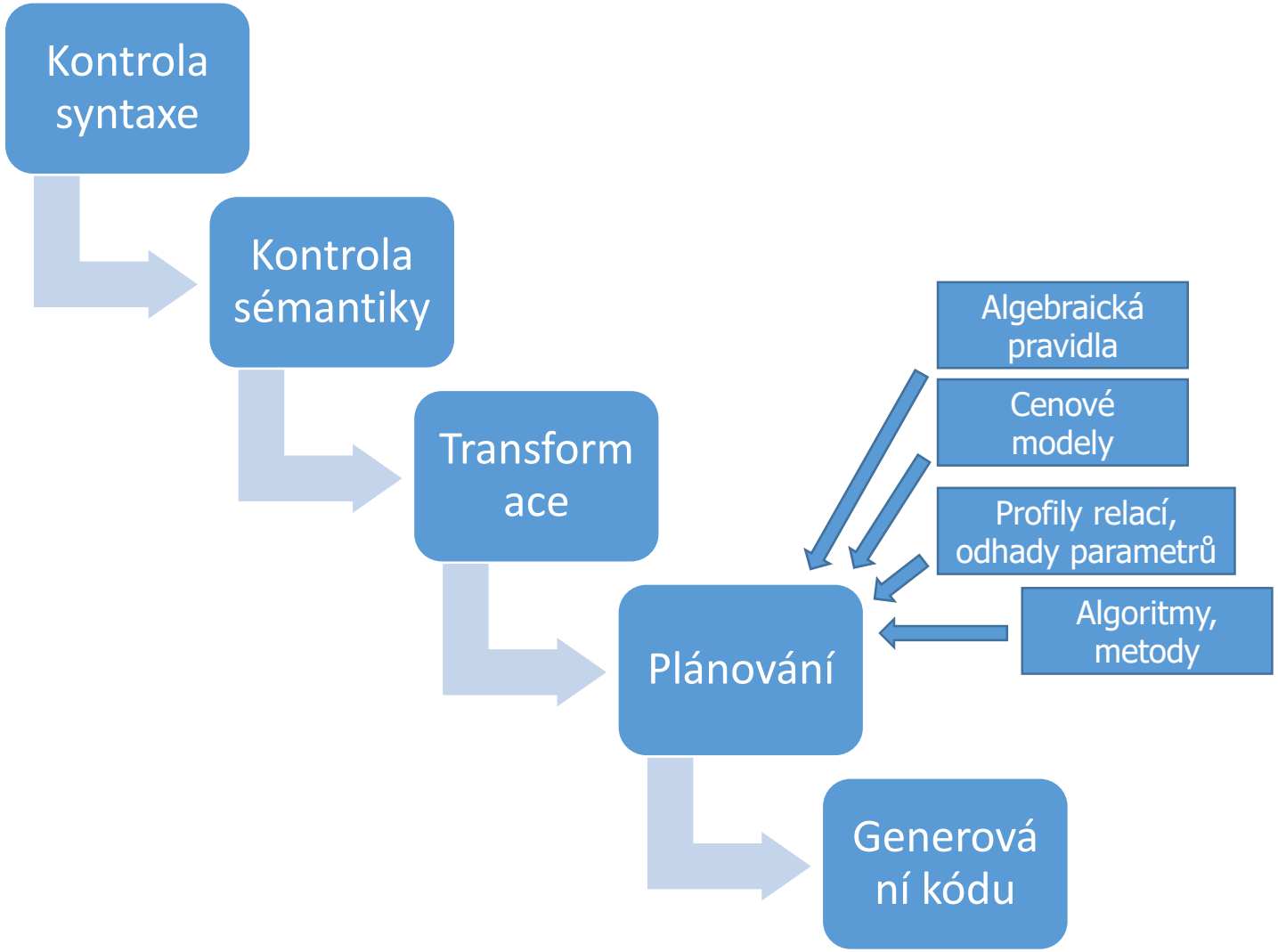
Plán vyhodnocení dotazu

- Posloupnost operací spolu s přiřazenými algoritmy
- Cena plánu
  - Časová složka
    - počet I/O operací
  - paměťová složka
    - počet stránek vnitřní nebo vnější paměti

Optimalizace dotazu

- Výběr toho nejefektivnějšího plánu
- Máme ale pouze odhad ceny
- Optimalizátor využívá statistických informací
- **Současně je ale optimalizace silně ovlivněna návrhem databáze (včetně indexů) a jejího fyzického uložení**

Zpracování dotazu



Relační operace

- I/O operace je přečtení stránky z vnější paměti do vnitřní paměti počítače

Metody pro výpočet **spojení**

- Nejkritičtější místo v relačním SŘBD (plus operace kartézský součin)
- Jaká je spojitost mezi operací kartézský součin a spojení, případně dalšími operacemi?

Metody pro výpočet **spojení**

- Mějme např. dvě relace A a B
- Každá má 100 000 řádků s 10 řádky na stránce
- Výsledek kartézského součinu pak bude obsahovat 10 000 000 000 řádku a tedy jeho uložení by mohlo trvat 1 000 000 000 I/O operací
- Při 30 I/O za sekundu to odpovídá cca 385 dnům

Metody pro výpočet **spojení**

- Základní metody pro výpočet spojení:
  - Hnízděné cykly (nested loops) a indexované hnízděné cykly
  - Vyhledávání (look-up)
  - Setříděné slévání (sort-merge)
  - Hašovaná spojení (hash joins)

- V případě kdy nejsou k dispozici indexy se považuje za nejrychlejší algoritmus setříděného slévání, který ale velmi závisí na množství vnitřní paměti, která je k dispozici

Hnízděné cykly

- Základní algoritmus, když nejsou k dispozici další pomocné sktruktury

```
for i=1 to pA do //pA number of pages of relation A
  begin read A[i] page to bufferA
    for j=1 to pB do
      begin read Bj page to bufferB
        for each a in bufferA do
          begin for each b in bufferB do
            if join_condition holds begin
              create join of a and b
              write results
            end
          end
      end
  end
```

Varianta se selekcí (look-up varianta)

- U operace spojení máme i další podmínku, kterou lze využít k zefektivnění operace
- Např.
- `SELECT * FROM A,B WHERE A.a = B.b AND A.aa = 10`
- Za předpokladu, že aa je prim. klíčem v relaci A a b je prim. klíčem v relaci B. Relace A i B mají 10 000 řádků v 1 000 stránkách a indexy jsou pouze na dvou úrovních ve vnitřní paměti.
- Cena pak bude v nejlepším případě pouze 2*1/30 sekundy.
- Nelze použít vždy
  - Např. Relace A má 10 000 řádků v 1000 stránkách a relace B má 10 řádků v 5 stránkách přičemž každá hodnota A.aa je rovna 10
  - Pro čtení řádků z A je třeba 1 000 I/O operací, pak je třeba 10 000 porovnání vždy s jednou přečtenou stránkou z B, což je 10 000 I/O operací. Celkově tedy 11 000 I/O což je 5,1 minuty

Varianta se spojovacím indexem

- Spojovací index je datová struktura (např. B strom), kde s každým klíčem K jsou uvedeny identifikátory n-tic spojovacích relací, pro které hodnota A nabývá K
- Máme tedy novou strukturu ,kde ke každé hodnotě primárního klíček je odpovídající hodnota v tabulce s cizím klíčem
- Čísla stránek, které je třeba číst jsou zakódována v identifikátorech n-tic.

Setříděné slévání

- Každá relace se nejprve setřídí a pak se provede slévání na uložených datech

Hašovaná spojení

- Ve vnitřní paměti se alokuje prostor , do kterého lze hašovat celou relaci R.
- Po zhašování R do vnitřní paměti se sekvenčně čte relace S, hašuje se hodnota s.A pro každý prvek z S a k ní se nachází přímým přístupem odpovídající n-tice z R uložené ve vnitřní paměti.
- Je-li splněna podmínka spojení, pak se produkuje odpovídající n-tice výsledku
- Pokud se hash relace R nevejde do paměti, pak se vytvářejí pouze hash z části relace a vytvářejí se vhodné disjunktní množiny a ty se pak spojí

Optimalizace dotazů

- Algebraická optimalizace
- Založená na indexech a velikostech relací
- Optimalizace řízená redukčním faktorem
- Statisticky řízená optimalizace
- Syntaxí řízená optimalizace

Algebraická optimalizace

- Získání ekvivalentního výrazu relační algebry je umožněno platností některých zákonů vyplývajících z definice relačních operací.

1. Komutativita spojení a kartézského součinu (např.
   1. E1 x E2 ≡ E2 x E1
   2. E1 * E2 ≡ E2 * E1
2. Asociativita spojení a kartézského součinu
   1. (E1 x E2)xE3 ≡ E1x(E2 x E3)
   2. (E1 * E2) *E3 ≡ E1 * (E2 * E3)
3. Tato pravidla např. umožní použít takové pořadí, které zajistí co nejmenší mezivýsledky

Algebraická optimalizace

3. Komutativita selekce a projekce
4. Distribuce selekce a kartézského součinu
   - Pokud jsou všechny atributy z podmínky (ϕ) se vyskytují nejvýše v E1
     - (E1 x E2)(ϕ) ≡ E1(ϕ)xE2
   - Jestliže ϕ = ϕ1 Λ ϕ2, ϕ1 využívá pouze E1 a ϕ2 pouze E2 pak
     - (E1 * E2)(ϕ) ≡ E1(ϕ1)*E2(ϕ2)
5. Distribuce selekce a sjednocení
   - (E1 ∪ E2)(ϕ) ≡ E1(ϕ) ∪ E2(ϕ)
6. Distribuce selekce a rozdílu
   - (E1 – E2) (ϕ) ≡ E1(ϕ) - E2(ϕ)
7. Distribuce projekce a kartézského součinu
8. Distribuce projekce a sjednocení
9. Degenerace kaskády projekcí
   - E[X][Y]….[Z] ≡ E[Z]

Shrnutí, pravidla

1. Proveď selekce co nejdříve
2. Proveď projekce co nejdříve
3. Jestliže se jako argument selekce vyskytuje kartézský součin a selekce zahrnuje atributy obou relací, transformuj kartézský součin na spojení
4. Jestliže je dána posloupnost selekcí a/nebo projekcí, použij kombinaci komutativity a redukci kaskády selecí a projekcí v pořadí selekce následovaná projekcí a aplikuj je současně na každou vybranou n-tici
5. Použij asociativitu spojení, kartézského součinu, sjednocení a průniku k seskupení větví stromu dotazu tak, že selekční operace, která produkuje menší relaci bude volána nejdříve
6. Jestliže se nějaký podvýraz vyskytuje vícekrát ve stromu dotazu a výsledek, který produkuje, není příliš velký, spočítej ho jednou do pomocné relace

Statisticky řízená optimalizace

- Statistické informace o jednotlivých relacích jsou ukládány ve formě histogramů v systémovém katalogu
- Pak se jich využívá při odhadu ceny dotazů

Syntaxí řízená optimalizace

- Záleží jak uživatel napíše dotaz, takže je například rozdíl mezi

```
Select * from users where salary < 1000 AND name
= “Roman”
```

- a

```
Select * from users where name = “Roman” AND
salary < 1000
```

Syntaxí řízená optimalizace

- Zajímavý příklad:
- Mějme dotaz:

```
SELECT * FROM movies WHERE (year >= 1990 AND year <= 2000) OR actor = “Tom”
```

- Optimalizátor se pravděpodobně bude snažit projít relaci movies sekvenčně i když jsou k dispozici indexy pro atributy year a actor.
- Je to dáno tím, že od určité složitosti podmínky WHERE optimalizátor podmínku dále neanalyzuje
- Můžeme ale dotaz přepsat takto a to se již využijí indexy dokonce pro každý pod dotaz

```
(SELECT * FROM movies WHERE year >= 1990 AND year <= 2000)
UNION
(SELECT * FROM movies WHERE actor = “Tom”)
```

Děkuji za pozornost