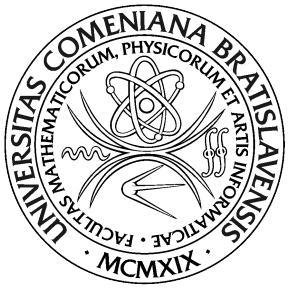
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



AUTOMATICKÝ MANAŽMENT INDEXOV PRE POSTGRESQL

Diplomová práca

2019 Bc. Pavel Rajčok

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

AUTOMATICKÝ MANAŽMENT INDEXOV PRE POSTGRESQL

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Alexander Šimko, PhD

Bratislava, 2019 Bc. Pavel Rajčok

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Pavel Rajčok
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Automatický manažment indexov pre PostgreSQL
Automatic index management for PostgreSQL

Anotácia: Aby databázový systém fungoval efektívne, je ho potrebné nakonfigurovať. Súčasťou tejto konfigurácie je aj vytváranie vhodných indexov, ktoré typicky robí databázový špecialista. Vzľadom aj na finančné náklady je zaujímavé túto činnosť plne automatizovať. Problém automatického výberu indexov v databázových systémoch je NP-ťažký a jeho riešeniu bolo venované značne množstvo výskumu. Pre komerčné databázové systémy existujú nástroje, ktoré tento problém riešia, pre open-source databázový systém PostgreSQL takéto nástroje chýbaju, s výnimkou veľmi mladého projektu Dexter. Cieľom práce je nadviazať na výskum v tejto oblasti, navrhnúť a implementovať open-source nástroj pre automatický výber a manažovanie indexov v open-source databázovom systéme PostgreSQL. Súčasťou práce bude aj experimentálne vyhodnotenie relevantných parametrov vytvoreného nástroja.

Literatúra:
- Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2008. Prentice Hall Press.
- Kołaczkowski. Rybiński. Automatic Index Selection in RDBMS by Exploring Query Execution Plan Space. In: Advances in Data Management. Studies in Computational Intelligence. 2009. Springer.
- TCP Benchmarks. http://www.tpc.org/information/benchmarks.asp
- Dexter. https://github.com/ankane/dexter

Vedúci: Ing. Alexander Šimko, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 02.10.2017

Dátum schválenia: 09.10.2017 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

. . . . . .

Bratislava, 2019 Bc. Pavel Rajčok

Poďakovanie

Touto cestou by som sa chcel v prvom rade poďakovať môjmu školiteľovi Ing. Alexander Šimko, PhD za jeho cenné rady a usmernenia, ktoré mi veľmi pomohli pri riešení tejto diplomovej práce. Takisto sa chcem poďakovať všetkým mojím kamarátom a celej mojej rodine za podporu počas môjho štúdia.

Abstrakt

TODO

Kľúčové slová: TODO

Abstract

TODO

Keywords: TODO

Obsah

Kapitola 1

Úvod

TODO

Kapitola 2

Motivácia

TODO

Kapitola 3

Prehľad problematiky

V tejto kapitole prevedieme čitateľa od pojmu databázový systém, dátové modely, pojmy ako sú pohľady a indexy, až po teóriu pre výber správneho indexu. Informácie, ktoré sú zhromaždene v tejto kapitole budeme čerpať zo šiestej edície knihy „Database Systems Concepts“[4] a z knihy s názvom „Database Systems – The Complete Book“[3]. Kniha nás prevedie teoretickými poznatkami, ktoré sú dlhoročne zaužívané a ustálene.

3.1 Databázový systém

Databázový systém alebo aj systém riadenia bazy dát je súbor súvislých dát a množiny programov, ktoré nám umožňujú pracovať s nimi. Súbor dát má v slovenskom jazyku ustálene pomenovanie databáza. Hlavnou úlohou databázového systému je zapisovať a čítať dáta z databázy. Takéto systémy sú navrhnuté a prispôsobené na správu veľkého množstva informácií. Ak máme byť detailnejší, ide o preddefinované štruktúry na ukladanie informácií a mechanizmy pre pristupovanie k nim.

Samozrejme to nie je všetko a databázový systém musí zaručiť ochranu uložených dát bez ohľadu na to, či sa systém pokúša niekto zneužiť, alebo dôjde ku mechanickému či systémovému poškodeniu. Taktiež musí zabezpečiť neporušenosť a totožnosť informácií aj v prípade, že ku informáciám pristupuje a manipuluje s nimi viacero užívateľov súčasne.

3.2 Databáza

Aby sme mohli pokračovať v definícií ďalších pojmov, popíšeme si aspoň jednou vetou, čo je to databáza. V podstate databáza nie je nič iné než súhrn informácií, ktoré existujú daný čas.

3.3 Dátový model

Základnou štruktúrou databázy je dátový model. Bližšie ho môžeme definovať ako komplex, ktorý charakterizuje dáta, vzťahy medzi nimi, ich sémantiku a obmedzenia, ktoré im prináležia. Práve tento model nám pomáha opísať návrh, či lepšie povedané koncepciu, samotnej databázy.

Poznáme viacero typov týchto modelov. Za zmienku stojí hlavne relačný, entitno-relačný a objektovo-orientovaný model. V našom prípade si bližšie rozpíšeme relačný a objektovo-orientovaný model. Následne nás bude zaujímať ich kombinácia.

3.3.1 Relačný model

Relačný model využíva súbor tabuliek, respektíve relácií, na zobrazenie dát a vzťahu medzi nimi. Každá tabuľka má viacero stĺpcov, ktoré majú

jednoznačné pomenovanie. Tabuľky obsahujú záznamy špecifického typu. Takéto záznamové typy bližšie definujú presné hodnoty atribútov, ktoré tabuľka obsahuje. Stĺpce v tabuľkách sú vlastne totožné spomínaným atribútom. Relačný dátový model je najviac rozšíreným modelom spomedzi ostatných. Množstvo databázových systémov je založená práve na relačnom dátovom modeli.

3.3.2 Entitno-relačný model

Entitno-relačný dátový model využíva základné objekty, nazvané aj entity, a vzťahy medzi nimi. Entita je doslova objekt v reálnom svete, ktorý sa líši od ostatných, je jedinečný. S takýmto modelom sa môžeme častokrát stretnúť pri návrhoch databáz.

3.3.3 Objektovo-orientovaný dátový model

V tomto prípade začneme z opačného konca. Objektovo orientované programovanie sa stalo dominantnou metodológiou vo svete programovania. Tento jav sa prejavil aj v problematike dátových modelov. Obohatenie entitno-relačného modelu o zapuzdrenie, metódy a objektovú identitu nám prinieslo nový rozmer v dátových modeloch.

3.3.4 Objektovo-relačný model

Objektovo-relačný model je kombináciou predchádzajúcich modelov a obsahuje ich charakteristické črty a vlastnosti. Dalo by sa povedať, že základ tohto modelu tvorí relačný model, ktorý je rozšírený o prvky objektovo-orientovaného modelu. Rozšírenie obsahuje zložené typy atribútov, metódy, identifikátory n-tíc a referencie.

3.4 Pohľady a indexy elementárne

Na začiatok sa oboznámime s pojmom VIEW alebo aj voľne preložené do slovenčiny - pohlaď. Definícia pohľadu nám pomôže ľahšie uviesť čitateľa do problematiky indexov, ktorá je pre našu prácu kľúčová.

3.4.1 Pohľad

Najskôr si ale uveďme pojmy na správnu mieru. V programátorskej komunite sa často používa výraz tabuľka namiesto výrazu relácia. Dôležité je ale vedieť, že tabuľka je relácia uložená v pamäti a pohľad je virtuálna relácia.

Poznáme dva typy pohľadov a tými sú virtuálne a materializované pohľady. Poďme si teda najskôr predstaviť virtuálne pohľady. Sú to virtuálne relácie, ktoré sú definovane dopytom prostredníctvom iných relácií uložených v pamäti. Nie sú fyzicky uložené v databáze. Procesor pre dopyty nahradí pohlaď jeho definíciou za účelom spracovania dopytu. Virtuálne relácie sú jednou z viacerých tried SQL relácii, ktoré neexistujú fyzicky v pamäti. V podstate sú len definovane výrazom, lepšie povedané dopytom. Čiže pohľady môžu byť reálne dopytované akoby existovali, a v niektorých prípadoch môžu byť dokonca aj modifikované.

Na druhej strane, pohľady môžu byť aj materializované a teda fyzicky uložené v pamäti. Takto ukladané pohľady môžu urýchliť spúšťanie dopytov. Touto informáciou sa dostávame ku dôvodu, prečo sme si najprv objasnili, čo sú to pohľady. Veľmi dôležitým typom materializovaných pohľadov je index, uložená dátová štruktúra, ktorej zmyslom je urýchliť prístup ku špecifickým n-ticiam jednej z uložených relácií v pamäti.

3.4.2 Index

Na úvod tejto podkapitoly si povieme, čo sa skrýva za indexmi a prečo sú také dôležité. Množstvo dopytov má za úlohu vyselektovať len malú podmnožinu záznamov, ktoré sú uložené v reláciách. Čo značí o tom, že je častokrát veľmi neefektívne prechádzať každý záznam v relácii, ak chceme len ten, ktorý spĺňa zadanú podmienku v dopyte. Databázový systém by preto mal byť schopný nájsť takéto záznamy priamo. Preto sú v systémoch implementované dátové štruktúry, ktoré môžu byť priamo prepojené s každým záznamom v relácii.

Index by sme mohli prirovnať ku indexu v knihe. Je to jednoduché prirovnanie, ak chceme v knihe nájsť konkrétnu tému podľa slova či frázy, máme možnosť ju vyhľadať v indexe na záverečných stranách knihy. V indexe prislúcha hľadanému reťazcu istá množina strán, na ktorých sa hľadaný reťazec vyskytuje. Následne si už len prečítame informácie na konkrétnych stranách a tak niekoľkonásobne urýchlime proces, než by to bolo v prípade nutnosti čítania celej knihy. Slová v takomto indexe sú v utriedenom poradí a tým uľahčujú hľadanie.

Indexy v databáze zastávajú rovnakú rolu. Uvažujme, že chceme selektovať záznam na základe identifikačnej hodnoty. Databázový systém vyhľadá index, ktorý nám dá informáciu o mieste na disku, kde sa žiadaný záznam nachádza. Miestom na disku máme na mysli konkrétny blok disku, z ktorého nasledovne urobí výber a vyselektuje hľadaný záznam.

Avšak indexovanie v prípade tisícov záznamov nie je efektívne. Keďže samotný index by bol veľmi veľký. Existujú sofistikovanejšie techniky inde-

xovania, ktoré zohľadňujú faktory, ako sú typ prístupu, množstvo potrebného času na prístup, vkladanie a mazanie, či nadmerné využitie miesta.

Indexy teda zvyšujú výkonnosť databázy a urýchľujú proces hľadania záznamov.

3.5 Indexy formálne

V danej relácii máme nastavený index na atribúte A, ktorý je dátovou štruktúrou pre zefektívnenie vyhľadávania n-tíc, ktoré majú nemennú hodnotu. Takto definovaný index si môžeme predstaviť ako binárny vyhľadávací strom o dvojiciach kľuč(a) a hodnota(h). Kľuč a chápeme ako jednu z hodnôt, ktorú by atribút A mohol obsahovať. Kľuč a je asociovaný s hodnotou h, ktorá je množinou umiestnení n-tíc, ktoré obsahujú kľuč a v prvku atribútu A.

Takýto index by mohol byť nápomocný v prípade dopytov, ktoré porovnávajú atribút A s nejakou konštantou. Napríklad porovnanie A = 3, poprípade A <= 3. Vďaka tomuto porovnaniu si môžeme uvedomiť, že kľuč do indexu môže byt hocijaký atribút, dokonca množina atribútov. Samozrejme, nemusí to byť taký kľuč, pre ktorý je index nastavený. Na atribúty indexu budeme poukazovať ako na indexovaný kľuč, v prípade nutnosti odlíšenia.

3.6 Indexy na príklade

V prípade početných relácií je prehľadávanie všetkých n-tíc výpočtovo veľmi náročne. Daná podmienka pre vyhľadávanie častokrát spôsobí, že vý-

sledkom je malá zhodnosť n-tíc, teda výpočet zbytočne prehliada nepotrebné prvky n-tíc. Napríklad uvažujme nasledovne:

```
SELECT *
FROM movies
WHERE studioName = 'Disney'
    AND year = 1990;
```

Pre vhodnosť príkladu relácia Movie obsahuje zhruba 10 000 n-tíc a je testovaná dopytom pozostávajúcim z WHERE klauzuly. Vieme, že podmienku year = 1990 spĺňa len 200 prvkov/záznamov. Naivný spôsob, ako riešiť tento dopyt je zobrať všetkých 10 000 n-tíc a vyhodnocovať prvok po prvku s danou podmienkou. Samozrejme tento prístup nie je tak efektívny, ako by bol v prípade, ak by sme hneď dostali konkrétnych 200 n-tíc s atribútom year = 1990 a následne už len porovnávali, či spĺňa druhú zadanú podmienku studioname = 'Disney'. Efektívnejší prístup by bol už len v prípade, ak by sme boli schopní priamo získať prvky, ktoré súhlasia obom častiam podmienky súčasne - year by bol rovný číslu 1990 a studioname by bolo rovne reťazcu 'Disney'. O tom si ale ďalej povieme v časti o viac-atribútovom indexe. Tak isto sú indexy využiteľné aj v príkazoch, ktoré spájajú tabuľky. Uvažujme nasledujúci príklad.

```
SELECT name
FROM movies, movieExec
WHERE title = 'Star Wars'
    AND producerCert = cert;
```

Ide o selektovanie mena režiséra, ktorý režíroval film 'Star Wars'. Ak existuje index pre stĺpec title, v tom prípade vieme použiť index, aby sme získali

n-ticu pre 'Star Wars'. Z tejto n-tice vieme vyňať údaj zo stĺpca producerCert. Predpokladajme, ze existuje aj index pre stĺpec cert. Potom vieme využiť záznam o producerCert na vyhľadanie n-tice z tabuľky movieExec pre režiséra 'Star Wars'. Následne z tohto záznamu vyjmeme režisérove meno. Všimnime si, že za pomoci týchto dvoch indexov sme sa pozreli len na dve n-tice. V prípade, že by neexistovali takéto indexy, bolo by nutné prejsť každú n-ticu v každej relácii.

3.7 Štruktúra indexov

Na základe predošlých faktov o indexoch v databázovom systéme už nemusíme o ich dôležitosti ani polemizovať. Index je dátová štruktúra, ktorá uchováva hodnotu jednej alebo viacerých položiek záznamu a na základe takto definovanej hodnoty je systém schopný nájsť záznam s danou hodnotou oveľa rýchlejšie. Vďaka tomu sa potreba prehľadávania celého záznamu obmedzí len na malú časť. Spomínaný pojem hodnota je takzvaný vyhľadávací kľúč, resp. len kľúč.

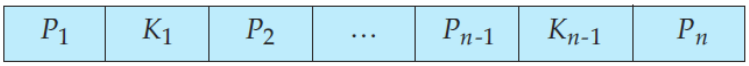
Technika implementácie indexov na veľkých reláciách je dôležitou súčasťou systémov na správu databázy. Typickým využitím pre indexy v databáze je dátová štruktúra B-strom, čo vo všeobecnosti hovorí o využití vyváženého binárneho stromu. Existujú aj ďalšie typy dátových štruktúr, ktoré nebudeme v našej práci spomínať.

3.7.1 Štruktúra B+ -strom

Keďže typy štruktúr indexov sú veľmi rozmanitá téma, v našej práci sa zameriame len na koncept indexu typu binárny strom, inak aj b-strom. Na-

priek tomu, že aj iné typy indexov sú vhodné a nápomocné, štruktúra b-strom je najviac rozšírená v databázových systémoch. Konkrétne ide o implementáciu b+-stromu. Je to forma vyváženého stromu a navyše má vlastnosť, že všetky cesty od koreňa po listy sú rovnako dlhé. Nelistové vrcholy majú od $\lceil n/2 \rceil$ po n potomkov. N je fixná hodnota pre konkrétny strom.

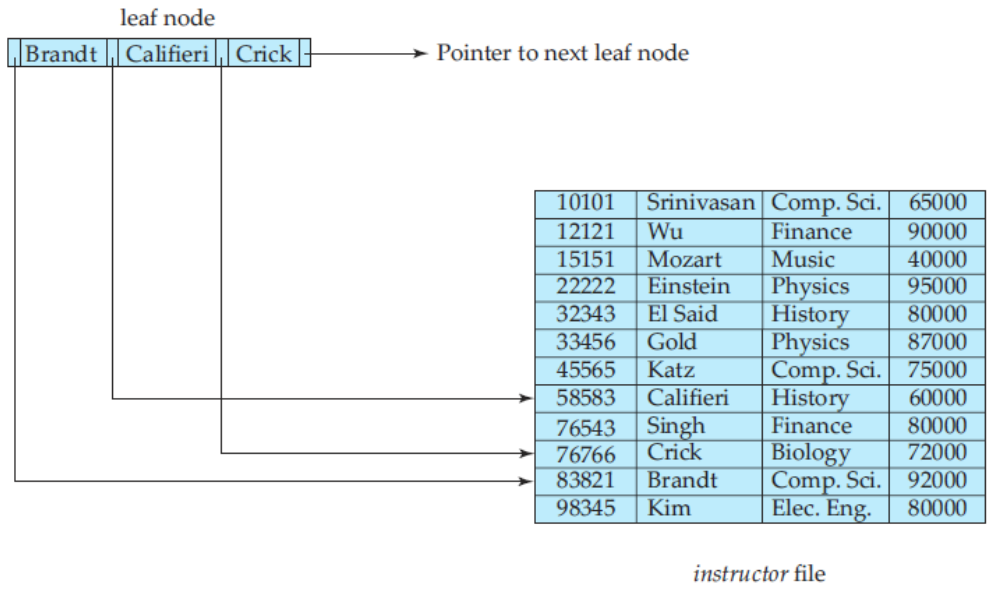
Ako môžeme vidieť na obrázku 3.1, ide o typický vrchol b+-stromu. Obsahuje n-1 hodnôt vyhľadávacieho kľúča K_1, K_2, až K_{n-1} a n smerníkov P_1, P_2 až P_n. Hodnoty kľúča vo vrchole sú v usporiadanom poradí, teda ak i < j, tak $K_i < K_j$.



Obr. 3.1: Všeobecný zápis pre vrchol B+-stromu.

Začnime štruktúrou listov, pre ktoré platí i = 1, 2, 3 až n-1, smerník P_i ukazuje na záznam s hodnotou kľúča K_i. Podstatu smerníka P_n si vysvetlíme v ďalších riadkoch.

Na obrázku 3.2 je zobrazený jeden listový vrchol b-stromu pre reláciu Inštruktor, kde n je 4 a vyhľadávací kľúč je meno.



Obr. 3.2: Listový vrchol B+-stromu.

Teraz, keď už vieme, ako vyzerá list b+-stromu, poďme sa pozrieť, že ako to je v prípade konkrétnych vrcholov. Každý list môže obsahovať od $\lceil (n-1)/2 \rceil$ až po n-1 hodnôt. V našom prípade b+-stromu sa n = 4, takže každý list musí obsahovať aspoň 2 hodnoty a najviac 3.

Teraz si môžeme vysvetliť podstatu smerníka P_n. Takýto smerník používame na prepojenie listových vrcholov.

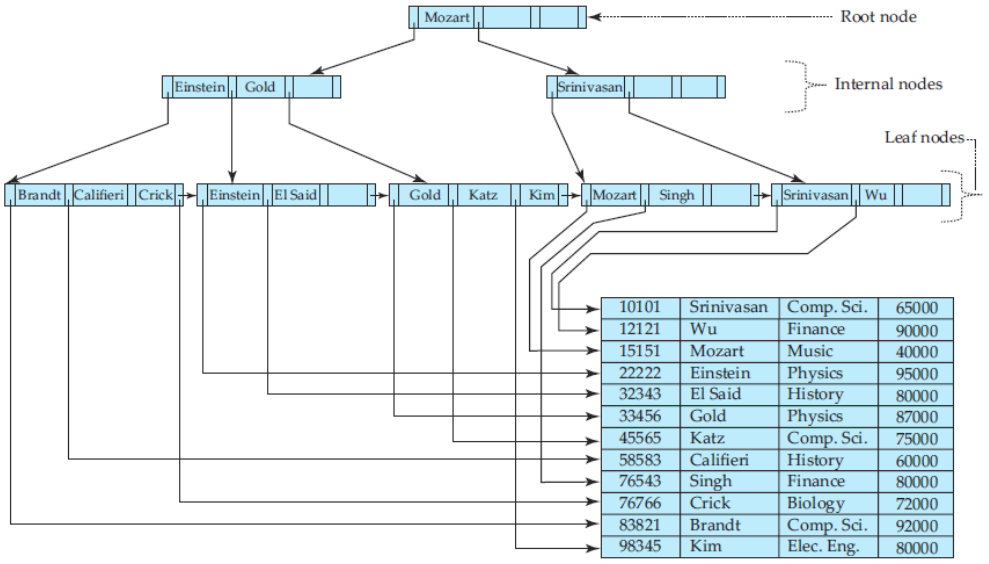
Štruktúra nelistových vrcholov je zhodná so štruktúrou listových vrcholov. Až na jednu výnimku a tou je, že všetky smerníky sú smerníkmi do iných vrcholov. Takéto vrcholy môžu obsahovať n smerníkov, no ich minimálny počet je $\lceil n/2 \rceil$. Typ takýchto smerníkov vo vrchole sa nazýva aj rozvetvenie vrcholu.

Uvažujme vrchol, ktorý obsahuje m smerníkov, kde m <= n. Pre i=2, 3, ..., m-1. Smerník P_i ukazuje na podstrom, ktorý pozostáva z kľúčových hodnôt menších ako Ki a väčších alebo rovných ako K_{i-1}. Smerník Pm poukazuje

na časť podstromu, ktorá obsahuje kľúčové hodnoty väčšie alebo rovné ako K_{m-1}. Smerník P_1 ukazuje na časť podstromu, ktorá obsahuje také kľúčové hodnoty, ktoré sú menšie ako K_1.

Na rozdiel od vrcholov, koreň môže obsahovať menej ako $\lceil n/2 \rceil$ smerníkov. Napriek tomu ale musí mať aspoň dva smerníky, odhliadnuc na to, že strom obsahuje len jeden vrchol.

Obrázok 3.3 znázorňuje kompletný b-strom pre súbor inštruktor, n = 4.



Obr. 3.3: B+-strom.

3.8 PostgreSQL

V knihe s názvom „Database Systems Concepts“[4] sme sa dočítali o stručnej charakteristike Postgresql. Je to voľne šíriteľný objektovo relačný databázový systém, ktorý bol vyvinutý profesorom Michaelom Stonebrakerom na Univerzite Barkeley v Kalifornii. Názov Postgres bol odvodený od revoluč-

ného relačného databázového systému Ingres. Nie je náhoda, že spomínaný systém má spoločného autora.. PostgreSQL podporuje množstvo aspektov a vlastnosti SQL jazyka. Navyše má možnosť rozšírenia nových dátových typov, funkcii, metód indexov a iné.

3.8.1 Indexy v PostgreSQL

Indexy v Postresql využívajú obvykle dátové štruktúry B-strom a hešovanie, ktoré sú súčasťou väčšiny databázových systémov. Taktiež podporujú ďalšie typy indexov, ktoré sú jedinečne pravé pre Postgres a tými sú "the Generalized Search Tree", “the Space-partitioning Generalized Search Tree”, "the Generalized Inverted Index", „the Block Range Indexes“, ktoré môžeme poznať aj pod skratkami GiST, SP-GiST, GIN a BRIN. Všetky spomenuté typy indexov môžu byť jednoducho prispôsobené užívateľom.

Len pre obraz čitateľa si stručne objasníme nasledujúcich pár termínov, ktoré použijeme v ďalšom definovaní indexov.

- Súborová organizácia - databáza je namapovaná do niekoľkých odlišných súborov, za ktoré zodpovedá prislúchajúcich operačný systém. Takéto súbory sú nastálo uložené na disku. Súbor je logicky organizovaný ako sekvencia záznamov. Tieto záznamy sú namapované do diskových blokov. Súbor môžeme nazvať aj ako základný prvok v operačnom systéme. Každý súbor je logicky rozdelený do diskových blokov, sú to časti pamäte s fixne danou veľkosťou. Takéto bloky môžu obsahovať množstvo záznamov.

- Haldová súborová organizácia [Heap file organization] - existuje viacero spôsobov ako organizovať záznamy do súborov. V našej práci sa však

stretneme len s touto konkrétnou organizáciou. A tej konceptom je, že ktorýkoľvek záznam môže by uložený kdekoľvek v súbore, kde je dostatok priestoru na jeho uloženie. V tomto type súborovej organizácie záznamy nie sú utriedené. Vo všeobecnosti platí, že každá relácia má samostatný súbor.

- Prístupové metódy – metódy v Postgres, ktoré získavajú dáta z disku - sekvenčný skener haldových súborov, indexový skener, indexový skener bitmáp.

- Predikát - je to výraz, ktorý vyhodnotí True, False alebo Unknown. Predikáty sa používajú v klauzule WHERE, v klauzule HAVING, v spájacej podmienku pre FROM klauzuly a ďalšie konštrukcie, v ktorých je Booleovská hodnota vyžadovaná. Predikát je SQL výraz, ktorý vyhodnotí podmienku, ktorá je buď True, False alebo Unknown. True znamená, že výraz je správny. False znamená, že výraz je nesprávny a je teda opakom True. Unknown znamená, že výraz nie je True ani False. Všetky SQL hodnoty použité v predikátoch musia byť kompatibilné pre porovnávanie.

Ako sme už viackrát spomínali, index v Postgresql je implementovaný ako dátová štruktúra, ktorá dynamicky mapuje vyhľadávané predikáty na postupnosti n-tíc id atribútov konkrétnej tabuľky(relácie). Návratové hodnoty n-tíc sú určené porovnávať "the search predicates". Aj keď v niektorých prípadoch musia byť predikáty prehľadané v "heap file". Ako bolo už spomenuté, v prípade Postgres ide o niekoľko typov indexov zahŕňajúc aj užívateľom rozšírené. Aj keď metóda prístupu môže používať rozdielny formát úseku, všetky dostupné indexy používajú "slotted-page"formát.

- Základným typom indexu v Postgresql je už viackrát spomínaný index

typu b-strom. Tieto indexy sú založene na B+-strom dátovej štruktúre. Poskytujú vysokú súbežnosť vykonávania operácii.

- Hašovacie indexy sú implementáciou lineárneho hašovania. Sú vhodne hlavne pri vykonávaní jednoduchých operácii rovnakého druhu. V Postgresql majú indexy tohto typu preukázateľne horšiu výkonnosť vyhľadávania, než je to v prípade B-strom indexov. Majú však porovnateľnú veľkosť a náročnosť na údržbu. Nevýhodou týchto indexov je, že ako jedine z množiny Postgresql indexov nepodporujú ich obnovu po poruche systému. Z toho nám vyplýva, že takmer vždy je uprednostnené využívanie už veľakrát spomínaných B-strom indexov.

- Postgresql poskytuje vysoko rozšíriteľne indexy s názvom "Generalized Search Tree", teda GiST. Ide o spôsob prístupu k dátam, ktorý je bližšie špecifikovaný ako vyvážený strom. Príkladom takýchto indexov gist, ktoré obsahujú B-strom a R-strom indexy. Zaujímavým faktom na tomto type indexu je, že pôvodná implementácia R-stromov bola nahradená pravé GiST z dôvodu použitia nových funkcionalít, ktoré však nebudeme hlbšie analyzovať.

3.9 Automatizovaný výber indexov

Množstvo komerčných databázových systémov v dnešnej dobe ponúka nástroje, ktoré pomáhajú databázovým administrátorom ku správnemu výberu indexov. No nevýhodou je to, že ide zväčša o platené systémy. V našej práci sa však zaoberáme voľne dostupným databázovým systémom, ktorý, žiaľ, nepodporuje túto funkcionalitu. Existuje riešenie, ktoré je nezávislou aplikáciou, no o nej sa dočítame neskôr v kapitole o existujúcich riešeniach. V

tejto podkapitole si ozrejmíme na akom princípe funguje automatizovaný výber indexov. Takýto proces musí byť už vykonaný pred finálnym nasadením systému do prevádzky. Zvyčajne sa implementuje ako nástroj, ktorý ponúka viacero alternatív. Jeho neodmysliteľnou súčasťou je spolupráca s nástrojom na optimalizáciu dopytov, ktorý je súčasťou databázových systémov. Vďaka nemu získa informácie ako je výpočtová náročnosť a výhody výberu jednej z ponúkaných alternatív. Nanešťastie, počet alternatív môže byť veľmi veľký, samozrejme ako aj zaťaženie systému pri tomto procese. Preto si musíme dávať pozor, akú techniku použijeme pri tomto procese.

Princíp ako fungujú takéto nástroje si rozpíšeme v nasledujúcich bodoch.

1. Nástroj sa zaoberá záťažou systému, v angličtine sa môžeme stretnúť aj s pojmom workload. Analyzuje históriu príkazov typu SELECT a UPDATE. História obsahuje množinu záznamov, zaznamenané sú príkazy spomínaného typu počas stanoveného časového úseku.

2. Nástroj vykoná nad takouto množinou kompresiu, t.j. vyberie len vhodných adeptov pre ďalší proces. Množina podobných príkazov bude reprezentovaná jedným všeobecným, ktorý má zhodný charakter a prináleží celkovému počtu volaní tejto množiny. Následne príkazy, ktoré sa vyskytujú v minimálnej miere a nezaťažujú systém, nástroj vynechá. Príkazy, ktoré zaťažujú systém najviac by mali byť uprednostnené na analýzu. Kompresia je nevyhnutná hlavne v prípade veľkej zaťaženosti systému.

3. Za pomoci nástroja na optimalizáciu dopytov by mala byť vytvorená množina indexov, ktoré znížia celkové zaťaženie systému. Problém nastáva, ak takáto množina obsahuje množstvo alternatív vytvorenia in-

dexov.

4. Riešenie sa následne realizuje za pomoci správneho výberu heuristiky, ktorá vhodným spôsobom znižuje počet možných alternatív vytvorenia indexov.

Geedy heuristiky, voľne preložené do slovenčiny ako hladné či pažravé, pre výber indexu fungujú nasledovne.

1. Odhadnú užitočnosť vytvorenia indexov za pomoci nástroja na optimalizáciu dopytov.

2. Následne vyberú tie, ktoré poskytnú čo najväčší úžitok, čo sa týka výkonu alebo využitia priestoru. Výber musí samozrejme obsiahnuť aj výpočtovú náročnosť údržby pre konkrétny index.

3. Proces sa opakuje, dokým sa dostupný priestor pre indexy nevyčerpá alebo výpočtová náročnosť ich udržiavania nie je vyššia než ich samotné dopytovanie.

V praxi to ale funguje inak. Techniky pre výber indexu využívajú síce aj niektoré elementy spomínanej heuristiky, no využívajú aj ďalšie na zlepšenie výsledkov. O ktorých si povieme v kapitole ...

Kapitola 4

Predchádzajúce riešenia

4.1 TODO Kapitola

4.1.1 TODO Podkapitola

Kapitola 5

Návrh Modelu

TODO

1. TODO
2. TODO

5.1 TODO Kapitola

- TODO
- TODO

5.1.1 TODO Podkapitola

Kapitola 6

Implementácia

TODO

6.1 TODO Kapitola

6.1.1 TODO Podkapitola

Kapitola 7

Výsledky

TODO

1. TODO
2. TODO

7.1 TODO Kapitola

- TODO
- TODO

7.1.1 TODO Podkapitola

Kapitola 8

Záver

TODO

Literatúra

[1] Jozef Kratica, Ivana Ljubic, and Dusan Tosic. A genetic algorithmfor the index selection problem. In Günther R. Raidl, Jean-ArcadyMeyer, Martin Middendorf, Stefano Cagnoni, Juan J. Romero Car-dalda, David Corne, Jens Gottlieb, Agnès Guillot, Emma Hart,Colin G. Johnson, and Elena Marchiori, editors,EvoWorkshops,volume 2611 ofLecture Notes in Computer Science, pages 280–290.Springer, 2003.

[2] Sérgio Lifschitz and Marcos Antonio Vaz Salles. Autonomic indexmanagement. InICAC, pages 304–305. IEEE Computer Society,2005.

[3] Hector Molina.Database systems : the complete book. PearsonPrentice Hall, Upper Saddle River, N.J, 2009.

[4] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan.Databasesystem concepts. McGraw-Hill, New York, 6 edition, 2010.

Zoznam obrázkov