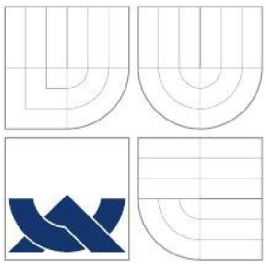
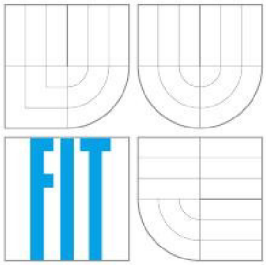


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INDEXOVÁNÍ DATABÁZÍ: SP-GIST PRO POSTGIS

DATABASE INDEXING: SP-GIST FOR POSTGIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. LUKÁŠ MATULA
AUTHOR

VEDOUCÍ PRÁCE DOC. ING. JAROSLAV ZENDULKA, CSC.
SUPERVISOR

BRNO 2016

Abstrakt

Cílem této práce je seznámení se s indexačními metodami a datovými typy prostorových objektů v databázovém systému PostgreSQL (obsahuje indexační metody GiST a SP-GiST) a vytvoření indexu SP-GiST pomocí quad stromu v rozšíření PostGIS. Toto rozšíření přináší databázi možnost pracovat s prostorovými a geografickými objekty. PostGIS obsahuje své vlastní datové typy a metody, které s nimi pracují. Obsahuje také GiST index pro práci s daty, ale chybí v něm stále realizace SP-GiST indexu. Z tohoto důvodu byla vytvořena tato diplomová práce.

Abstract

The goal of the master´s thesis is to study index methods, spatial data type objects in PostgreSQL database systems and to create SP-GiST index by quadtree in the PostGIS. The PostGIS is spatial database, which extends of PostgreSQL. PostGIS adds support for geographic and spatial objects. It is a big benefit. PostGIS has its own data types, methods and GiST index too, but there is SP-GiST index missing, therefore master's thesis was created.

Klíčová slova

Databázový systém, PostgreSQL, GiST, SP-GiST, datový typ, PostGIS, databázový index, metody, quad strom, kd strom

Keywords

Database system, PostgreSQL, GiST, SP-GiST, data type, PostGIS, database index, methods, quadtree, kdtree

Citace

MATULA, Lukáš: Indexování databází: SP-GiST pro PostGis. Brno, 2016. 57 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc Ing. Zendulka Jaroslav, CSc.

Indexování databází: SP-GiST pro PostGIS

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Jaroslava Zendulky, CSc.
Další informace mi poskytl Ing. Tomáš Volf.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Matula
31. května 2016

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce panu doc. Ing. Jaroslavu Zendulkovi, CSc. a svému konzultantovi panu Ing. Tomáši Volfovi za odborné vedení, vstřícnost, za pomoc a rady při zpracování této práce.

Obsah

1 Úvod

Rychlé vyhledávání potřebných informací ve velkém množství dat, je nezbytnou vlastností pro dnešní databázové systémy. Jedním z hlavních nástrojů, které umožňuji rychlejší provádění operací s daty jsou databázové indexy. Proto se tato diplomová práce zabývá způsobem indexace 2D prostorových dat (dále jen prostorová data) v databázovém systému PostgreSQL. PostgreSQL systém umí pracovat s různými typy dat, ale pro diplomovou práci se omezíme na prostorová data. Ta jsou zpravidla velmi rozsáhlá a zároveň je potřeba v nich rychle vyhledávat, aby informační systémy mohly s těmito daty pracovat a uživatel nemusel hodně dlouho čekat (zvlášť nyní, kdy uživatelé vyžadují co nejnižší odezvu jakéhokoliv systému).

Cílem této diplomové práce je seznámit se s prostředky pro práci s prostorovými daty, jak v databázovém systému PostgreSQL, tak i v jeho rozšíření PostGIS určeném pro prostorové databáze a naimplementovat SP-GiST index do rozšíření PostGIS.

V kapitole č. 2 je postupně popsán jak databázový systém PostgreSQL, kde jsou informace o historii PostgreSQL, tak i rozšíření PostGIS určené pro prostorové a geografické objekty. Následně jsou představeny geometrické datové typy zastupující prostorové datové typy v PostgreSQL a datové typy v rozšíření PostGIS.

Obecný přehled indexačních metod v PostgreSQL je popsán v kapitole 2.5. B-strom je základní indexační metodou, následuje méně populární hash, GIN, BRIN. GiST index je v této kapitole (2.5) popsán jen pro představení a širší popis pokračuje v kapitole č. 3. Stejně tak je v tomto přehledu (kapitoly 2.5) popsán i SP-GiST index, kterému je věnována kapitola č. 4. Poslední indexační metodou je GIN, která je stručně popsána jen v tomto přehledu.

V kapitole č. 3 GiST framework je nejprve vysvětlena struktura tohoto indexu. Jsou zde popsány obecně základní metody pro práci s vyhledávacími klíči, a také algoritmy pro vyhledávání, vkládání a odstraňování dat v indexu. Na závěr této kapitoly jsou popsány informace ohledně odlišnosti implementace tohoto indexu v rozšíření PostGIS oproti PostgreSQL.

Následuje kapitola věnována indexu SP-GiST. Zde jsou nejprve ukázány operátory, se kterými tento index pracuje, potom následuje popis parametrů externího rozhraní (určují vlastnosti stromu – kd stromu, quad stromu, atd.), kterým vývojář definuje konkrétní specifikaci vytvářeného stromu. Hlavní částí této kapitoly je popis dvou realizací SP-GiST. První z nich je realizace kd stromu i s konkrétní definicí externích parametrů a druhou realizací je quad strom. Důležitou součástí SP-

GiST jsou i interní metody, které realizují stejně jako v GiST frameworku algoritmy pro vkládání, hledání a odstraňování dat v indexu.

Po krátké podkapitole věnované implementaci SP-GiST indexu v PostgreSQL, kde je názorně ukázána a popsána struktura propojení interních a externích metod SP-GiST indexu, následuje kapitola s popisem hlavní implementační části diplomové práce.

V kapitole číslo 6 je podrobně popsána struktura implementovaného quad stromu, třída operátorů definovaná pro tuto variantu v SP-GiST indexu a implementace konkrétních externích metod. Realizace v kapitole č. 6 je postavena na základě znalostí z předchozích kapitol.

V sedmé kapitole jsou prováděná ověřování správnosti chování implementace vůči teoretickým znalostem. Ověřování správnosti zahrnuje jednak správné vytvoření quad stromu, vrácení očekávaných výsledků dotazů (na základě znalosti o použitých datech), odstranění položek z indexu po smazání položek z tabulky a nesmí chybět ani výkonnostní srovnání s GiST indexem.

V diplomové práci jsem po drobnějších úpravách teoretické části, které proběhly na základě získaných znalostí z praktické části diplomové práce a zohlednění připomínek vedoucího diplomové práce, jsem použil celý semestrální projekt, tzn. Kapitoly 1 - 4.

2 PostGIS jako rozšíření PostgreSQL

Úvod ke kapitole 2 - Databázový systém PostgreSQL

PostgreSQL je výkonný, open source objektově-relační databázový systém odvozen z balíku POSTGRES. Projekt POSTGRES pochází z Californské univerzity v Berkeley, kde byl vyvíjen pod vedením profesora Michaela Stonebrakera.[1] V dnešní době se používá název POSTGRES jako alias pro PostgreSQL a to hlavně z důvodu jednodušší výslovnosti.

PosgreSQL má více než 19 let aktivního vývoje a osvědčenou architekturu. K systému PostgreSQL existuje kvalitně zpracovaná volně dostupná dokumentace [1]. Tento databázový systém je plně v souladu s ACID vlastnostmi. Co se týká operačních systémů, tak je tento systém schopen pracovat s operačními systémy, jako jsou Windows, Linux a další Unixové systémy (včetně Mac OS X).

Má nativní programovací rozhraní pro jazyky Java, C/C++, .Net, Perl, Python a mnohé další.
V nabídce systému PostgreSQL je mnoho moderních funkcí, mezi které patří komplexní dotazy, cizí klíče, triggery, aktualizovatelné pohledy a nechybí ani integrita transakcí, která je zvláště důležitá při používání databáze v bankovnictví. Výhodou tohoto systému je, že uživatel může vytvářet nové datové typy, funkce, operátory, agregační funkce i indexační metody. Díky této možnosti bylo vytvořeno rozšíření s názvem PostGIS.

Jednou z velkých výhod databázového systému PostgreSQL je jedna ze sofistikovaných funkcí MVCC (Multi-Version Concurrency Control), která řeší kolizní problémy transakcí pracujících nad stejnými daty. PostgreSQL systém nabízí širokou podporu pro mezinárodní znakové sady, podporuje také multibytové kódovací znaky a unicode. Databázový systém rozlišuje velká a malá písmena (case-sensitivity) a poskytuje lokální podporu pro řazení.

PostgreSQL zahrnuje většinu datových typů standardu SQL:2008 (Integer, Numeric, Boolean, Char, Varchar, Date, Interval a Timestamp). Samozřejmostí je podpora pro uložení velkých binárních objektů, kterými jsou obrázky, zvuky a videa.

PostgreSQL obsahuje velké množství různých datových typů. Aby práce s tímto množstvím datových typů byla přehlednější, jsou rozděleny do skupin podle toho, pro jaká data jsou určeny (jsou to typy číselné, peněžní, znakové, binární, výčtové, geometrické, XML, síťových adres, polí, rozsahové atd.).

2.1 PostGIS

PostGIS je prostorové databázové rozšíření pro PostgreSQL objektově relační databázi. Přidává podporu pro geografické objekty. S těmito novými datovými typy se vážou i nové funkce, operátory, vylepšené indexy. Přidané funkce, operátory, indexy a datové typy zvyšují sílu PostgreSQL systému, dělají ho rychlejší, nabízí spoustu funkcí, a díky tomu se stává robustním prostorovým systémem pro řízení báze dat.

PostGIS poskytuje:

- funkce pro analýzu a zpracování jak vektorových tak i rastrových dat (spojování, rozdělení, atd.)
- algebru rastrových map pro jemnozrnné zpracování rastrů
- balík v příkazovém řádku pro import rastrových dat z mnoha standardních formátů (GeoTiff, NetCDF, PNG, JPG, atd.)
- podporu pro 3D objekty, prostorové indexy a funkce
- a mnoho dalších.[1]

Díky PostGIS je umožněno ukládat na PostgreSQL server prostorové databáze pro geografické informační systémy (GIS). Zahrnuje podporu pro prostorový R-strom index založený na obecném vyhledávacím stromu – Generalized Search Tree (GiST). Obsahuje rovněž funkce pro analýzu a zpracování GIS objektů. PostGIS nabízí mnoho funkcí, které zřídka nalezneme v jiných konkurenčních databázích, jako jsou Oracle Locator/Spatial a SQL Server. [2]

PostGIS se začal vyvíjet společností Refractions Research jako open source projekt v prostorové databázi. Nyní je uvolněn pod GNU General Public License (GPLv2[2]).

K 20. březnu 2015 byla uvolněna prozatím poslední verze PostGIS 2.1.6. Tato verze se úspěšně zaměřila na chyby a problémy výkonu. Při velkých tabulkách bodů lze dosáhnout ~50% úsporu prostoru na disku oproti předchozí verzi[5].

2.2 Prostorové datové typy v PostgreSQL

V PostgreSQL reprezentují skupinu prostorových typů geometrické typy. Jejich nevýhodou je, že mohou zastupovat pouze objekty ve dvoudimenzionálním prostoru. Toto omezení bylo jedním

1. http://postgis.net/features/
2. http://opensource.org/licenses/gpl-2.0.php

z hlavních důvodů, proč se později vytvořilo rozšíření PostGIS. Mezi geometrické typy patří těchto sedm zástupců.

Bod (point)

Tento datový typ představuje bod v 2D prostoru. Bod je základním stavebním prvkem prostoru. Ve 2D prostoru je definován dvěma souřadnicemi x, y, kde x a y jsou čísla s plovoucí desetinnou čárkou.

Přímka (line)

Datový typ přímka představuje přímku v 2D prostoru. Přímka je reprezentována lineární rovnicí ax + by + c = 0, kde a a b jsou nenulové. Hodnota tohoto typu je definována buď trojicí parametrů ({a,b,c}), nebo dvojicí bodů na přímce v následujícím formátu: ((x1, y1), (x2, y2)) .

Úsečka (lseg)

Tento datový typ zastupuje úsečku ve 2D prostoru. Tento datový typ je definován dvěma koncovými body na úsečce. Formát zápisu hodnoty typu úsečka je v následujícím tvaru:
((x1, y1), (x2, y2)).

Obdélník (box)

Tento datový typ představuje obdélník ve 2D prostoru. Obdélník je definován dvojicí bodů v protějších rozích rámce. Tvar hodnoty tohoto geometrického typu je následující:
((x1, y1), (x2, y2)).

Pro uložení hodnoty tohoto typu do databáze, systém vyžaduje vložení protilehlých rohů obdélníku z pravého horního rohu a levého dolního rohu. Jsou-li vloženy hodnoty rohů opačných, systém si je sám automaticky převede.

Cesta (path)

Datový typ cesta představuje cestu ve 2D prostoru. Tento datový typ je definován seznamem propojených bodů. Cesta je dvojího typu. Prvním je otevřená cesta (první bod se nerovná poslednímu bodu), formát hodnoty tohoto typu je: [(x1, y1), …, (xn, yn)].

Druhým typem je uzavřená cesta (první bod = poslední bod), tvar hodnoty tohoto typu je: ((x1, y1), …, (xn, yn)). Body 1...n jsou koncové body úseček, kterými je vytvořena daná cesta.

Mnohoúhelník (polygon)

Tento datový typ představuje mnohoúhelník ve 2D prostoru. Je definován seznamem propojených bodů (vrcholů mnohoúhelníku). Tvar zápisu hodnoty tohoto typu je: ((x1, y1), …, (xn, yn)), kde body 1...n jsou koncové body úseček určující hrany mnohoúhelníku.

Kružnice (circle)

Kružnice je datový typ, který reprezentuje kružnici ve 2D prostoru. Tento typ je definován pomocí jednoho bodu a poloměru. Formát jeho hodnoty je definován takto: < (x, y), r > , kde x a y jsou souřadnice středového bodu a r je poloměr kružnice

2.3 Datové typy v PostGIS

V PostGIS se vyskytují tři druhy prostorových datových typů: geometrické, geografické a ohraničující typy. Již samotné názvy vyznačují význam jednotlivých prostorových typů.

Geometrie (geometry)

Tento hierarchický prostorový datový typ představuje geometrii v n-rozměrném prostoru. Mezi jeho podtypy patří bod (Point), křivka (Surve), plocha (Surface) a kolekce geometrií (GeometryCollection). Formát hodnot jednotlivých podtypů je různý a je hlavně ovlivněn tím, v kolika dimenzionálním prostoru se nachází. Obrázek celé hierarchie typů této kategorie je v příloze č. 1.

Ohraničující obdélník (box2d)

Ohraničující obdélník představuje prostorový datový typ ve 2D. Slouží k ohraničení (výběru) části roviny, ve které se nachází jeden, nebo více objektů typu geometrie. Hodnota tohoto typu je definovaná čtyřmi hodnotami typu float xmin, ymin, xmax, ymax, které určují pozici a velikost objektu. Datový typ box (v PostreSQL) se liší tím, že je definován dvěma body typu point (protilehlé rohové body).

Ohraničující kvádr (box3d)

Tento prostorový datový typ reprezentuje ohraničující kvádr ve 3D prostoru. Slouží k výběru 3D prostoru pomocí ohraničujícího kvádru, ve kterém se vyskytují objekty typu geometrie. Hodnota tohoto prostorového datového typu je složena z šesti položek vymezujících daný prostor. Jsou to tyto hodnoty: xmin, ymin, zmin, xmax, ymax, zmax.

Úložiště geometrií (geometry_dump)

Úložiště geometrií je prostorový typ vyskytující se v n-dimenzionálním prostoru (závisí to na metodě, pro jaký prostor je vytvořena). Je složený ze dvou polí. Prvním je pole geom (je v něm uložen objekt typu geometrie) a druhým je pole path (jednodimenzionální celočíselné pole). Indexy v poli path začínají své hodnoty od 1, tzn. path[1] vrátí první položku. Pole path[] slouží k indexování objektů geometrie v poli geom. Geometrický sklad je určen pro ukládání jednotlivých objektů geometrie, které vznikají po rozdělení složitých geometrických objeků. Pro jednodušší představu uvedu krátký příklad:

```
path  |  geom
----------------------
{1}| Polygon ((0 0, 1 2, 2 2, 2 0, 0 0))
{2}| Polygon ((3 3, 3 0, 2 0, 2 3, 3 3))
```

Kde na indexu path[1] je uložen v poli geom objekt Polygon ((0 0, 1 2, 2 2, 2 0, 0 0)) typu geometrie.

Geografie (geography)

Geografie je prostorovým datovým typem ve 3D prostoru. Tento prostorový datový typ je ve tvaru elipsy. Tento tvar reprezentuje vlastnosti povrchu země, která nemá přesný tvar koule. Díky elipse je možné lépe popsat tvar povrchu země. Jeho hodnoty jsou definovány pomocí zeměpisného souřadnicového systému povrchu země (šířka/délka).

2.4 Databázové indexy v PostgreSQL

Databázové indexy slouží k rychlejšímu provádění SQL příkazů (`SELECT`, `UPDATE`, `DELETE`) nad tabulkami v databázi. V databázovém systému PostgreSQL si můžeme vybrat z více indexačních metod. Každá z nich používá jiný algoritmus, proto se hodí na jiný typ dotazu. Výběr správného databázového indexu závisí na datovém typu dat a operátorech, které chceme v dotazech používat.

PostgreSQL má ve verzi 9.5 aktuálně šest druhů indexu. V následujících podkapitolách jsou uvedeny jednotlivé indexy, které je možné použít v systému PostgreSQL.

B-strom

Indexační metoda B-strom je vhodná pro nejběžnější situace. Při vložení příkazu `CREATE INDEX` se použije jako základní index. B-strom je vhodný k použití nad datovými typy, u kterých existuje relace uspořádání (např. datum a čas, číselné typy, atd.). Používá se k datovým dotazům na rovnost a rozsah s následujícími operátory: <, <=, =, >=, >. Konstrukce, které jsou ekvivalentní s kombinacemi těchto operátorů, jako je např. `BETWEEN` a `IN`, mohou být také implementovány s indexem B-stromu. Dalšími vhodnými podmínkami pro použití B-stromu jsou `IS NULL` a `IS NOT NULL`.

Tento index je postaven na vyvážené struktuře B-stromu, kde jednotlivé uzly stromu obsahují několik klíčů s ukazateli na další uzly. Průchod stromem je prováděn směrem od kořene k listům a na jednotlivých úrovních zleva doprava.

Hash

Hash je další základní indexační metoda, která je implementována pomocí hašovací tabulky. Je vhodná pouze pro porovnání na rovnost s operátorem = (např. pro textové sloupce, kde se očekává pouze porovnání na rovnost). Důvodem, proč se hash index tolik nepoužívá ve srovnání s B-stromem, je jeho pomalejší sestavování a větší prostorová náročnost.
Pro vytvoření hash indexu je potřeba do příkazu pro vytvoření indexu vložit klíčové slovo hash:
```
CREATE INDEX name ON table USING hash (column);
```

GiST

GiST není jednoduchým druhem indexu, ale spíše infrastrukturou, ve které může být realizováno mnoho různých strategií. Standardní distribuce PosgreSQL zahrnuje třídy operátorů GiST pro několik dvoudimenzionálních geometrických datových typů. Struktura GiST se používá jak v B-stromech (pro vyhledávání podle hodnoty klíče), tak i v R-stromech (pro vyhledávání podle splnitelnosti predikátu).

SP-GiST

SP-GiST (Space Partitioned GiST – GiST index rozdělující prostor), obdobně jako GiST nabízí infrastrukturu, která podporuje různé druhy vyhledávání. Princip tohoto indexu je založen na postupném rozdělování velkého prostoru na menší části. SP-GiST umožňují implementovat široký rozsah různých nevyvážených, diskově založených datových struktur, jako jsou quad stromy, kd stromy a radix stromy. Standardní distribuce PostgreSQL obsahuje třídy operátorů SP-GiST, které jsou určené pro dvoudimenzionální body.

GIN

GIN je invertovaný index, určený pro indexování hodnot obsahující více než jeden klíč, např. pole. Stejným způsobem jako GiST a SP-GiST, tak i GIN index podporuje mnoho uživatelsky definovaných indexačních strategií. Podle těchto strategií jsou určeny operátory, které se používají s GIN indexem. Např. ve standardní distribuci PostgreSQL je GIN index definován s třídou operátorů pro jedno-dimenzionální pole (třída operátorů obsahuje operátory: <@, @>, =, &&)[3].
GIN index je vhodný pro fulltextové vyhledávání.

BRIN

BRIN je novým indexem definovaným od verze 9.5. v PostreSQL. BRIN je dalším indexem s uživatelsky definovanými strategiemi, které určují, jaké třídy operátorů se použijí. Tento index dokáže indexovat velký rozsah datových typů, mezi které patří převážně číselné typy jako integer, real, bit, byte atd. Z nečíselných datových typů podporuje datum, text, char, inet (síťový datový typ). BRIN index je určen pro zvýšení výkonu prováděných operací (vyhledávání) nad velkými tabulkami.

2.5 Indexační metody v PostGIS

B-strom

B-strom je implementován rovněž i v rozšíření PostGIS. Základní vlastností dat, které je B-strom schopen indexovat je, že mohou být řazeny podle jedné osy. Patří mezi ně čísla, datumy a znaky. Tento index není vhodný pro indexaci GIS dat (data geografických informačních systémů), které

3 https://www.postgresql.org/docs/9.2/static/functions-array.html

nelze logicky seřadit podle jedné osy (např. body (0,1) a (1,0), mezi těmito body nelze logicky určit, který je větší).

R-strom

R-strom využívá stejně jako v PostgreSQL systému rozdělení prostoru do obdélníků a podobdélníků. Tento způsob už umožňuje indexovat GIS data v prostorových databázích, ale nepoužívá se ve velké míře, protože není tak robustním indexem oproti GiST v PostGIS.

GiST

GiST, jak již bylo uvedeno v kap. 2.5.3 není klasickým indexem, ale spíše infrastrukturou. Díky prostorovým datovým typům v rozšíření PostGIS může GiST pracovat i s více než dvoudimenzionálními daty. Infrastruktura GiST je používána ve spojení s R-stromem. GiST může indexovat velký rozsah datových typů obsahující GIS data. Kromě požadavku na rychlé vyhledávání v tabulkách o tisících řádcích, je i požadavek na rychlé vytvoření indexu nad takto rozsáhlými sloupci tabulek.

2.6 Třídy operátorů

Důležitou součástí všech indexačních metod v systému PostgreSQL i v rozšíření PostGIS jsou třídy operátorů. Tyto třídy obsahují jednak název konkrétní třídy, dále jaký index používají, v jakém datovém typu jsou ukládány hodnoty do paměti, seznam operátorů, které daná třída uživateli poskytuje a seznam metod skrze které je realizováno konkrétní chování databázového indexu. V PostgreSQL systému jsou všechny tyto třídy operátorů a jejich součásti uloženy v tzv. katalogu.

Tyto třídy operátorů jsou používány, pokud si uživatel vytvoří index nad sloupcem tabulky v databázi. Už při tomto vytváření si může explicitně definovat, kterou třídu operátoru chce používat při následující práci s vytvořenou tabulkou. Pokud si však tuto třídu explicitně nezvolí, databázový systém mu defaultně jednu přidělí.

Při veškerých DML (Data Manipulation Language) operacích prováděných uživatelem nad sloupcem tabulky s vytvořeným indexem se budou volat jádrem indexu metody zaregistrované v příslušné třídě operátorů.

```
CREATE OPERATOR CLASS gist_geometry_ops_2d
   DEFAULT FOR TYPE geometry USING GIST AS
   STORAGE box2df,
   OPERATOR 1  <<,
   OPERATOR 2  &<,
   OPERATOR 3  &&,
   OPERATOR 4  &>,
   OPERATOR 5  >>,
   OPERATOR 6  ~=,
   OPERATOR 7  ~,
   OPERATOR 8  @,
   OPERATOR 9  &<|,
   OPERATOR 10 <<|,
   OPERATOR 11 |>>,
   OPERATOR 12 |&>,
#if POSTGIS_PGSQL_VERSION >= 91
   OPERATOR 13 <-> FOR ORDER BY pg_catalog.float_ops,
   OPERATOR 14 <#> FOR ORDER BY pg_catalog.float_ops,
   FUNCTION 8 geometry_gist_distance_2d(internal, geometry, int4),
#endif
   FUNCTION 1 geometry_gist_consistent_2d(internal,
geometry, int4),
   FUNCTION 2  geometry_gist_union_2d(bytea, internal),
   FUNCTION 3  geometry_gist_compress_2d(internal),
   FUNCTION 4  geometry_gist_decompress_2d(internal),
   FUNCTION 5  geometry_gist_penalty_2d(internal, internal,
   internal),
   FUNCTIO  6  geometry_gist_picksplit_2d(internal, internal),
   FUNCTION 7  geometry_gist_same_2d(geom1 geometry, geom2
geometry, internal);
```

Ukázka definice třídy operátorů v PostGIS pro geometrické objekty ve 2D prostoru indexované pomocí GiST.

3 GiST framework

GiST neboli Generalized Search Tree je vyvážená datová struktura. Slouží jako šablona, pomocí které lze implementovat indexy založené na GiST struktuře. Uživatelům umožňuje rozšiřitelnost pro nové datové typy a přístupové funkce k těmto typům. GiST sjednocuje předchozí nesourodé struktury používané v současné době pro běžné datové typy. Pomocí struktury GiST mohou být implementovány např. B-stromy nebo R-stromy.
Rozhraní GiST má vysokou úroveň abstrakce, požaduje po realizátoru přístupových metod pouze sémantickou implementaci datového typu. GiST vrstva se stará o souběžnost, logování a vyhledání ve stromové struktuře.

GiST je snadný na konfiguraci. Přizpůsobení stromu pro různé použití vyžaduje pouze registraci sedmi metod databázovým systémem (consistent, same, union, compress, decompress, picksplit, penalty). Osmá metoda (distance) je volitelná. Metody same, union a picksplit zajišťují konzistenci stromu v GiST frameworku. Efektivita (velikost a rychlost) provádění indexu závisí na metodách penalty a picksplit. Metoda consistent slouží k procházení stromu indexu. Zbývající dvě základní metody, compress a decompress, umožňují indexu mít data v interním stromu uložena v jiném datovém typu než data, se kterými index pracuje. Osmá metoda, distance, se používá při určování tzv. „vzdálenosti“ položek vzhledem k určené položce (hledání nejbližšího souseda).

Jednou z odlišností mezi GiST indexem v PostgreSQL a PostGIS jsou prostorové datové typy, pro které jsou tyto indexy určeny. Zatímco v PostgreSQL se jedná pouze o dvoudimenzionální prostorové typy (geometrické typy), tak u PostGIS to jsou především dvoudimenzionální až n-dimenzionální prostorové typy (geometrické, geografické a ohraničující typy).

GiST index podporuje (v PostgreSQL i v PostGIS) následující seznam operátorů, které je možné použít v dotazech pro vyhledávání prostorových dat:
<< … je striktně vlevo od. Př: circle'((0,0),1)' << circle'((3,0),1)'.
>> … je striktně vpravo od. Př: circle'((3,0),1)' >> circle'((0,0),1)'.
&< … přesahuje vpravo přes. Př: box'((0,0),(1,1))' &< box'((0,0),(2,2))'.
&> … přesahuje vlevo přes. Př: box'((0,0),(3,3))' &> box'((0,0),(2,2))'.
<<| … je striktně níže. Př: box '((0,0),(3,3))' <<| box '((3,4),(5,5))'.
|>> … je striktně výše. Př: box '((3,4),(5,5)) |>> box '((0,0),(3,3))'.

&<| … přesahuje nad. Př: box '((0,0),(1,1))' &<| box'((0,0),(2,2))'.

|&> … přesahuje pod. Př: box '((0,0),(3,3))' |&> box '((0,0),(2,2))'.

@> … obsahuje. Př: circle '((0,0),2)' @> point '(1,1)'.

<@ … obsažen v. Př. Point '(1,1)' <@ circle '((0,0),2)'.

~= … stejny jako. Př. Polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))'.

&& … přesahuje (je-li alespoň jeden společný bod, tak výsledek je true). Př: box'((0,0),(1,1))' && box'((0,0),(2,2))'.

GiST indexy umožňují optimalizované vyhledávání nejbližšího souseda, např: SELECT * FROM places ORDER BY location <-> [1]point '(101,456)' LIMIT 10, který hledá k danému cílovému bodu deset nejbližších míst.

3.1 Struktura GiST

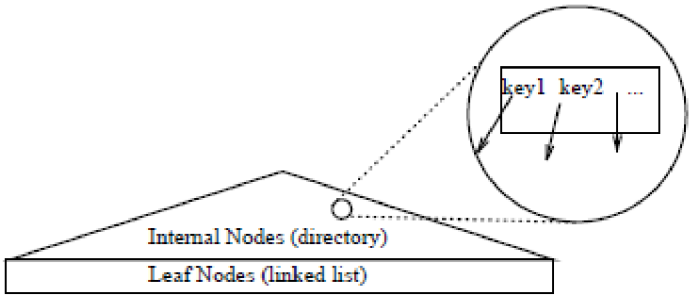
GiST je vyvážený strom proměnlivého rozvětvení mezi $k*M$ a M, kde $\frac{2}{M} \leq k \leq \frac{1}{2}$. Pro kořenový uzel platí, že může mít rozvětvení mezi *2* a M. Konstanta k je nazývána minimálním faktorem plnění stromu. M určuje maximální rozvětvení z uzlu.

Každý vnitřní uzel stromu obsahuje několik dvojic (p, ptr), kde p je hodnota (po vyhodnocení dotazovaného predikátu s touto hodnotou je výsledek True nebo False) a ptr je identifikátor n-tice v databázi (u listových uzlů), nebo je to ukazatel na jiný uzel ve stromu (u nelistových uzlů).

Vyhledávání začíná u kořenového uzlu a pokračuje směrem k listové úrovni. Pro každý ukazatel na uzel platí, že pokud je predikát q splnitelný nad hodnotou klíče daného uzlu, vyhledávání pokračuje v podstromu daného uzlu dokud všechna odpovídající data nejsou nalezena. Pokud, ale predikát q není splnitelný v daném uzlu, tzn. že pro žádnou hodnotu uloženou v podstromu od tohoto uzlu níže taky není dotazovaný predikát splnitelný. Vyhledávání (dotazovaní na splnitelnost predikátu q) tudíž pokračuje v následujícím uzlu.

Na obr. 1 je ukázán hrubý obraz databáze vyhledávacího stromu. Tento strom je složen ze dvou hlavních částí. První část je tvořena vnitřními uzly, které tvoří adresářovou strukturu (vnitřní strukturu stromu) a druhou částí jsou listové uzly, které obsahují ukazatele na aktuální data.

1 Geometrický operátor <-> znamená vzdálenost mezi objekty.



Obr. 1 Hrubý obraz vyhledávacího stromu [7].

3.2 Metody třídy reprezentující klíč

Metody třídy reprezentující klíč jsou velmi důležitým prvkem pro správnou funkci každé GiST struktury. Díky těmto metodám se mohou realizovat následně metody vkládání, vyhledávání, odstraňování položek. Samotný klíč je nejdůležitějším prvkem celého systému. Je to zapříčiněno tím, že bez klíče by nebylo možné systematicky sestavit jakýkoliv strom a samotné vyhledávání by taky nebylo možné realizovat.

Klíčem v GiST struktuře může být libovolná hodnota, která po dotazu na splnitelnost predikátového dotazu vrátí hodnotu True nebo False. V praxi klíče pochází z uživatelsky implementovaných tříd objektů, které poskytují konkrétní sadu metod požadovaných v GiST. V každé GiST implementaci je požadován výskyt následujících sedmi metod. V novějších verzích systému PostgreSQL (verze 9.1+) je nabízena ještě jedna volitelná metoda.

3.2.1 Consistent (E , q)

Na vstupu metody Consistent je položka E (dvojice p, ptr, kde p je hodnota klíče a ptr je odkaz na další uzel stromu) z aktuálního uzlu a predikát q. Funkce consistent určuje, zda predikát je s hodnotou klíče položky splnitelný nebo nesplnitelný. Metoda Consistent vrací false, pokud můžeme garantovat nesplnitelnost predikátu s hodnotou klíče p. To by znamenalo, že v podstromu této položky neexistuje žádná další položka, která by daný predikát splňovala. Díky tomu nemusíme dále prohledávat do hloubky celý podstrom dané položky.

3.2.2 Union (array)

Tato metoda sjednocuje objekty ve stromě. Na vstupu je dána sada objektů (položek), funkce union vygeneruje nový objekt typu box o minimální možné velikosti, který ohraničuje všechny objekty ze vstupního parametru array (pole objektů) [8]. Metoda union svojí implementací tvoří hlavní stavební prvek GiST indexu, protože GiST je založen na seskupování jednotlivých objektů v prostoru do jednoho velkého objektu.

3.2.3 Same (o1, o2)

Metoda same porovnává oba objekty (položky) na vstupu a vrací hodnotu true, pokud jsou objekty identické, jinak vrací hodnotu false.

3.2.4 Compress (o1) a Decompress

Metoda compress konvertuje objekt o1 určitého datového typu do datového typu vhodného pro fyzické úložiště na indexové stránce. Ve zdrojovém kódu GiST indexu v PostgreSQL se např: provádí konverze datových typů point, polygon, circle na datový typ box, který se následně ukládá do paměti. Datový typ box se používá pro ukládání do paměti, protože dokáže ohraničit všechny datové typy a výsledný objekt má vždy jednotný datový typ.

Metoda Decompress neprovádí v PostgreSQL ani v PostGIS žádnou činnost. Důvodem je, že po uložení objektů jiných datových typů než je box (box2d v PostGIS) do paměti, kdy proběhne jejich konverze na datový typ box (box2d) se již všechny operace s tímto objektem provádí v datovém typu box (box2d).

3.2.5 Penalty (newobj, oldobj)

Metoda penalty má na vstupu nový objekt newobj a původní objekt oldobj (objekty jsou typu box), který je uložený ve stromu. Vrací hodnotu indikující „cenu“ vkládaného nového objektu do konkrétní větve stromu. Objekty budou vloženy po cestě od nejmenší ceny ve stromu. Hodnoty ceny vrácené pomocí penalty by neměly být záporné. Pokud se vrátí záporná hodnota, je brána jako nula.

3.2.6 Picksplit (array)

Picksplit je metoda, která rozhoduje, zda-li se má provést rozdělení indexové stránky, a zároveň toto rozdělení realizuje. Na vstupu metody je pole objektů (datového typu box) uložených na indexové

stránce. Stejně jako funkce penalty, tak i picksplit funkce je stěžejní pro dobrý výkon indexu. V této funkci je řízena volba faktoru minimálního zaplnění stromu.

3.3 Metody třídy rezprezentující strom

Metody třídy reprezentující strom jsou další nedílnou součástí GiST struktury. Ke své činnosti tyto metody používají požadované metody ze třídy reprezentující klíč, popsané v kapitole 3.2. Stromové metody realizují algoritmy pro vkládání, vyhledávání a odstraňování položek stromů. Při průchodu stromem se provádějí dotazy na splnitelnost predikátu ze vstupu nad klíči jednotlivých uzlů stromu.

Pokud je hodnota klíče uzlu uložena v paměti v jiném datovém typu, ve srovnání s datovým typem v dotazovaném predikátu je potřeba provést konverzi datového typu hodnoty v predikátu tak, aby byl tento datový typ stejný s datovým typem hodnoty klíče v uzlech stromu.

3.3.1 Algoritmus vkládání

Algoritmus vkládání zaručuje, že GiST index zůstává vyvážený. Tento algoritmus je podobný vkládacím postupům R-stromů, které zobecňují jednodušší vkládací postupy pro B+stromy. V procesu vkládání se využívají uživatelsky definované metody Penalty z třídy reprezentující klíč. Pomocí ní se ve stromové struktuře najde vhodný podstrom pro vložení nové položky (položka je složena z hodnoty klíče a identifikátoru do paměti, kde se nachází uložený objekt ze vstupu). Při procesu vkládání položek do stromu se využívají i další metody z třídy reprezentující klíč. Jednou z nich je metoda Picksplit, která se používá pro rozdělení uzlu (v případě, že už se do něj nevejdou další položky). Další metodou, která je používaná v algoritmu vkládání nových položek je metoda Union. Jejím úkolem je propagovat změny od listových uzlů, až ke kořenu stromu.

Pseudokód:

```
Začni v kořeni stromu;
loop{
   if(je aktuální uzel listový) then

      if(je v listové stránce dostatek místa) then
         vlož novou položku ;
         volej metodu Union pro změnu hodnoty klíče;
```

```
            if(proběhlo rozdělení stránky) then
               propaguj změny směrem ke kořeni;
            end if;
         else
            volej metodu PickSplit nad listovou stránkou;
         end if;
      else
         volej metodu Penalty pro nalezení vhodného podstromu;
      end if;
}
```

3.3.2 Algoritmus vyhledávání

Pro vyhledávání existují dvě techniky. První technika je obecná, ale je hlavně méně efektivní. Může být použita na hledání jakýchkoliv datových sad s libovolným predikátem dotazu. Projde stromem tolikrát, kolikrát je nezbytně nutné pro splnění dotazu. Tato vyhledávací technika je nejobecnější a je analogická s postupem u R-stromů.

Její způsob prohledávání stromu je rekurzivním zanořováním. V pseudokódu realizuje tuto techniku metoda Search (R, q) s parametry R obsahující ukazatel na kořen stromu a predikátem q.

Pseudokód:

```
metoda Search(R, q){
   if(R není listový uzel) then
      for E in each položky uzlu R

         /*kde E je položka vnitřního uzlu obsahující klíč a ptr na
            další uzel ve stromu*/
         if(Consistent(E, q)) then
            volej Search(E.ptr, q);
         end if;
      end for;
   else     //R ukazuje na listový uzel
      for E in each položky uzlu R
         if(Consistent(E,q)) then
            vrať hodnotu položky E na výstup;
         end if;
```

```
        end for;
    end if;
}
```

Druhou (efektivnější) technikou je dotazování nad daty, která jsou lineárně uspořádané. Dotazování se provádí pomocí predikátů vymezující se na rovnost nebo rozsah. Pokud data splňují podmínku lineárního uspořádání, je efektivnější používat tuto druhou techniku, která je založena na typickém vyhledávání rozsahem jako v B+stromu. Výstupem druhé techniky jsou seznamy položek, splňující predikát zadaný na vstupu.

V pseudokódu realizují tuto techniku dvě metody. První je FindMin (R, q) s parametry R a q, kde R je odkaz na kořen stromu a q je predikát. Druhou metodou je metoda Next (R, q, E), kde R je odkaz na uzel, ve kterém je uložena položka E, q je predikát a parametr E obsahuje aktuální položku.
Pseudokód:

```
FindMin(R, q){
    if (R není listový uzel) then
        najdi první položku, splňující Consistent(E,q);
        volej FindMin(E.ptr, q);
    else
        vrať první položku E z uzlu R, která splňuje Consistent(E,q);
        pokud žádná nesplňuje Consistent(E,q) vrať NULL;
    end if;
}

/*nad vrácenými položkami z listových uzlů se volá metoda Next*/
Next(R, q, E){
    if(E není nejpravější položka v uzlu) then
        if(N je následující položka vpravo od E) then
            if(Consistent (N, q)) then vrať N;
            else vrať NULL;
            end if;
        end if;
    else
        nechť P je následující uzel vpravo od R na stejné úrovni ve
        stromu;
        if(P neexistuje) then  vrať NULL;
```

```
        else
            nechť N je nejlevější položka uzlu P;
            if(Consistent (N,q)) then vrať N;
            else vrať NULL;
            end if;
        end if;
    end if;
}
```

3.3.3 Algoritmus odstraňování

Algoritmus pro odstraňováni položek z indexu má za cíl nejen odstranit položky z indexu, ale taky musí zajistit, aby strom indexu zůstal vyvážený. Pro nalezení položek určených k odstranění se provádí vyhledání položek pomocí výše popsaných algoritmů (buď obecnou nebo efektivnější technikou). Po odstranění nalezených položek se zavolá metoda union, která provede kontrolu, zdali nelze sjednotit některé ze zbylých objektů v daném stromu.

3.4 Popis implementace GiST indexu v PostgreSQL a PostGIS

Jelikož GiST index není klasickým indexem, ale je frameworkem pro vytváření konkrétních variant indexů, proto taky jeho struktura je složena ze dvou částí (metody třídy reprezentující strom a metody třídy reprezentující klíč). Každá část je implementována samostatně. V PostgreSQL je implementováno jádro GiST indexu obsahující metody třídy reprezentující strom společně s dalšími nástroji pro správnou funkci GiST indexu. Odděleně jsou implementovány metody třídy reprezentující klíč, které realizují konkrétní implementaci indexu s použitím struktury GiST frameworku (dále to budu nazývat jako konkrétní rozšíření). Komunikace mezi konkrétním rozšířením a jádrem GiST indexu se provádí pomocí tříd operátorů definovaných nad daným rozšířením a pomocí manažeru funkcí systému PostgreSQL. Tyto třídy operátorů obsahují seznam operátorů, které dané rozšíření podporuje.

V PostGIS je realizováno pouze konkrétní rozšíření, které je určeno pro indexaci dvoudimenzionálních i vícedimenzionálních datových typů. S jádrem GiST indexu z PostgreSQL komunikuje konkrétní rozšíření skrze třídy operátorů a manažera funkcí PostgreSQL systému.

4 SP-GiST v PostgreSQL

SP-GiST (Space partitioned GiST) – indexační metoda GiST rozdělující prostor. V názvu tohoto frameworku je sice zmíněn GiST, ale implementace GiST a SP-GiST jsou zcela oddělené. Dalším rozdílem je I způsob jakým obě indexační metody pracují. GiST seskupuje jednotlivé objekty v prostoru dohromady, kdežto SP-GiST rozděluje velký prostor na menší části.

SP-GiST podporuje rozdělující vyhledávací stromy, které usnadňují rozvoj širokému rozsahu různých nevyvážených datových struktur, jako jsou quad stromy, kd stromy a radix stromy (trie). Obecná funkce těchto struktur je taková, že opakovaně rozdělují vyhledávací prostor do oddílů, které nemusí mít stejnou velikost. Samotné vyhledávání může být velmi rychlé za předpokladu, že se prostor správně rozdělil do oddílů podle pravidel.

Tyto populární datové struktury byly původně vyvíjeny pro použití v paměti. V hlavní paměti jsou obvykle navrhovány jako sady dynamicky alokovaných uzlů. To není vhodné pro přímé ukládání na disk, neboť tyto řetězce ukazatelů mohou být poměrně dlouhé a vyžadovaly by příliš mnoho přístupů na disk.

Oproti tomu, diskově založené datové struktury by měly mít vysoké rozvětvení k minimalizování I/O. Úkol, který SP-GiST řeší je mapování hledaných uzlů stromu do diskových stránek tak, aby hledání potřebovalo pouze několik diskových stránek, i když projde mnoho uzlů.

Jak GiST, tak i SP-GiST umožňuje vývoj vlastních datových typů s příslušnými přístupovými metodami.

Strukturální charakteristiky stromů rozdělující prostor, které je odlišují od ostatních stromových tříd:

1. Stromy rozdělující prostor dekomponují daný prostor rekurzivně. Pokaždé se vytvoří pevný počet disjunktních oddílů (neplatí pro radix strom).
2. Stromy rozdělující prostor jsou nevyvážené stromy.
3. Stromy rozdělující prostor trpí omezeným počtem rozvětvení z každého uzlu, např. pro quad strom je tato hodnota 4.
4. Ve stromech rozdělující prostor existují dva odlišné typy uzlů, jsou to indexové (interní) a datové (listové) uzly.

SP-GiST framework má dvě hlavní části. Interní metody indexu, které tvoří jádro celého frameworku, a externí rozhraní vytvářející konkrétní realizaci stromu společně s parametry upravující specifické rysy tohoto stromu.

V SP-GiST indexu se při dotazování na prostorová data uložená v tabulkách databáze využívá následujících šesti operátorů:
<<, >>, ~=, <@ (všechny čtyři operátory jsou již definovány v kap.3).
<^ … je níže (je povolen dotyk). Př. circle '((0,0),1)' <^ circle '((0,5),1)' .
>^ … je výše (je povolen dotyk). Př. circle '((0,5),1)' >^ circle '((0,0),1)'.

4.1 Parametry externího rozhraní

Tyto parametry slouží pro konkrétnější specifikaci podoby stromu, který je vytvářen programátorem pomocí externích metod. Jsou nastavovány ve zdrojovém kódu programátorem, nikoli uživatelem, jak je definováno v [6]. Hodnoty těchto parametrů nemusí být explicitně zadány, ale mohou vyplývat z konkrétní implementace.

NodePredicate - Tento parametr specifikuje typ predikátu, který bude použitý ve vnitřních uzlech stromu (např. u kd stromu jsou to predikáty left, right nebo blank).

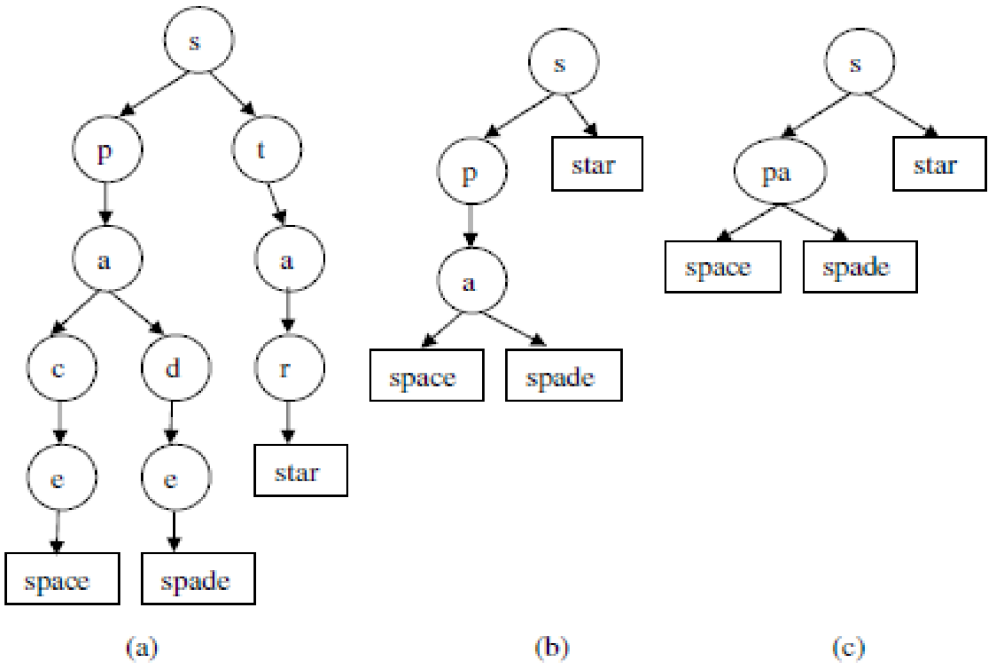
KeyType - Specifikuje datový typ dat uložených v listových uzlech (např. Point).

NumberOfSpacePartitions - Parametr určuje počet disjunktních částí poskytnutých v každé dekompozici (u kd stromu to jsou 2 a u quad stromu to jsou 4 části).

Resolution - Parametr určující maximální počet prostorových dekompozic, tzn. na jak velké části je požadováno rozdělit prostor (prostorová granularita).

ShrinkPolicy - Tento parametr udává, kolikrát se maximálně dekomponuje prostor v reakci na vložení dat. Nabývá tří možných hodnot. Pro názornou ukázku rozdílů mezi jednotlivými hodnotami jsem zvolil variantu stromu trie (Obr. 2):
Never Shrink - data jsou uložena v uzlu, který odpovídá maximálně rozloženému prostoru (to může vést k mnoha rekurzivním dekompozicím prostoru).
Leaf Shrink - data jsou vložena na první listový uzel (dekompozice nezávisí na maximálním možném rozložení).
Tree Shrink - interní uzly jsou sloučeny dohromady a tím strom eliminuje všechny jednoduché potomky interních uzlů.

BucketSize – tento parametr určuje maximální počet datových položek, které může mít datový uzel uložené. V implementaci se vyskytuje jako práh pro rozdělení datového uzlu (Split treshold). Pokud počet datových položek přesáhne hodnotu prahu, dojde k rozdělení tohoto uzlu na další datové uzly (tím vznikne i nový vnitřní uzel).



Obr. 2: Varianty stromové struktury trie a) Never shrink, b) Leaf shrink, c) Tree shrink [4]

Stromová datová struktura trie (prefixový strom) je vyhledávací strom. Vyskytuje se v realizaci se strukturou SP-GiST indexu pro práci s řetězci (zpravidla textová data). Je zde použita pro názornou ilustraci rozdílů mezi jednotlivými variantami ShrinkPolicy. Dále ji zde nebudu popisovat, protože není předmětem této diplomové práce. Více informací o struktuře trie naleznete v dokumentu viz [10].

Externí metody jsou druhou částí externího rozhraní SP-GiST, která umožňuje vývojáři specifikovat chování konkrétního stromu. Pro určení přesného tvaru těchto stromů slouží šest hlavních metod.

4.2 Externí metody

Vstupní parametry těchto externích metod jsou předávány skrze strukturu in a výstupní hodnoty metod jsou předávány skrze strukturu out.

InnerConsistent (in, out)

Tato metoda má na vstupu dva parametry in a out. Vstupní struktura obsahuje vnitřní uzel, a predikát. Na základě splnitelnosti tohoto predikátu s hodnotou klíče uloženou ve vnitřním uzlu se uloží do výstupní struktury odkaz na další vnitřní (nebo listový) uzel ve stromové struktuře.

LeafConsistent (in, out)

Metoda LeafConsistent je podobná metodě InnerConsistent s tím, že ověřuje splnitelnost predikátu v listovém uzlu. V prvním parametru je listový uzel společně s predikátem. Do výstupního parametru out se ukládá hodnota aktuálního uzlu, který je ověřován na splnitelnost predikátu. Tato metoda vrací True nebo False na základě splnitelnosti predikátu v tomto uzlu.

Obě metody Consistent určují, jakým způsobem se bude procházet stromem indexu při vyhledávání hodnot, pro které je predikát splnitelný.

Choose (in, out)

Metoda choose slouží pro výběr správného místa ve stromu indexu pro nově vkládanou hodnotu. Parametr in obsahuje aktuální vnitřní uzlel a vkládanou hodnotu. Ve výstupním parametru out se vrací odkaz na další uzel, který odpovídá vkládané hodnotě.

Config (out)

Metoda config je pouze konfigurační metodou, která nastavuje některé parametry pro strom indexu. Jedním z těchto parametrů je např. nastavení datového typu pro klíče vnitřních uzlů stromu. Tyto parametry se nastavují do výstupní struktury uložené v parametru out.

Compress (in, out)

Tato metoda slouží pro konverzi datového typu vstupní hodnoty na datový typ, který je index schopen ukládat do paměti, a zároveň je schopen s tímto datovým typem provádět potřebné operace. Parametr

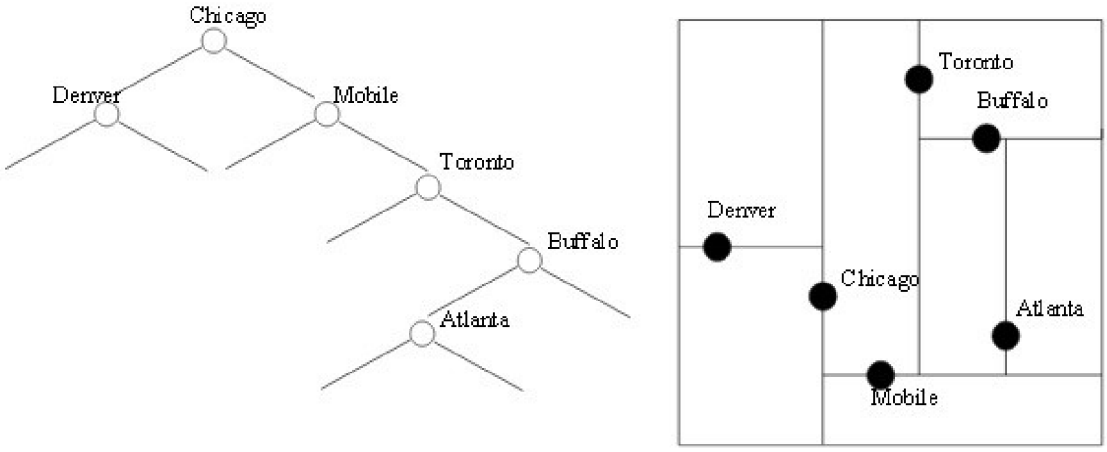
in obsahuje vstupní hodnotu, která je po konverzi převedena na datový typ bod a přes výstupní parametr out je odkazem vrácena zpátky.

PickSplit (in, out)

Metoda PickSplit rozděluje listový uzel (do kterého se nevejdou všechny požadované datové položky) na nové uzly za vzniku nového vnitřního uzlu. Metoda má na vstupu v prvním parametru seznam položek o velikosti BucketSize+1, jenž se nevejdou do jednoho uzlu. Druhým parametrem metoda PickSplit vrací nově vytvořený vnitřní uzel a jednotlivé položky rozdělené do nově vytvořených oddílů (počet oddílů dle parametru NumberOfSpacePartition), které vytvořením nového vnitřního uzlu vzniknou.

4.2.1 Princip realizace kd stromu

Kd strom je speciální druh vyhledávacího stromu vytvořený pomocí rozhraní SP-GiST. Tento strom je užitečný k zodpovězení dotazů o rozsahu množiny bodů v k-dimenzionálním prostoru. Kd strom používá datově řízenou dekompozici prostoru. Strom je konstruován rekurzivním rozdělovováním prostoru na dvě poloviny ve vztahu k jedné z dimenzí na každé úrovni stromu. Algoritmus pro dvoudimenzionální případ (kde k=2) s body v rovině x y vybírá jakýkoliv bod a rozděluje prostor pomocí pomyslné úsečky vedené tímto bodem, která je zároveň paralelní k ose y na dva podprostory (poloroviny). Následně se vybere další bod a provede se horizontální (pomyslná úsečka bude paralelní s osou x) rozdělení poloroviny, ve které leží. Obecně platí, že bod, který spadá pod region vytvořený vertikálním rozdělením, bude rozdělovat tento region horizontálně a naopak.



Obr. 3: Příklad kd stromu pro k=2 [6].

Tato realizace kd stromu je dána na základě těchto vstupních parametrů rozhraní:
ShrinkPolicy = Leaf Shrink
NumberOfSpacePartitions = 2
Node Predicate = „left“, „right“, nebo „blank“
KeyType = Point
BucketSize = 1

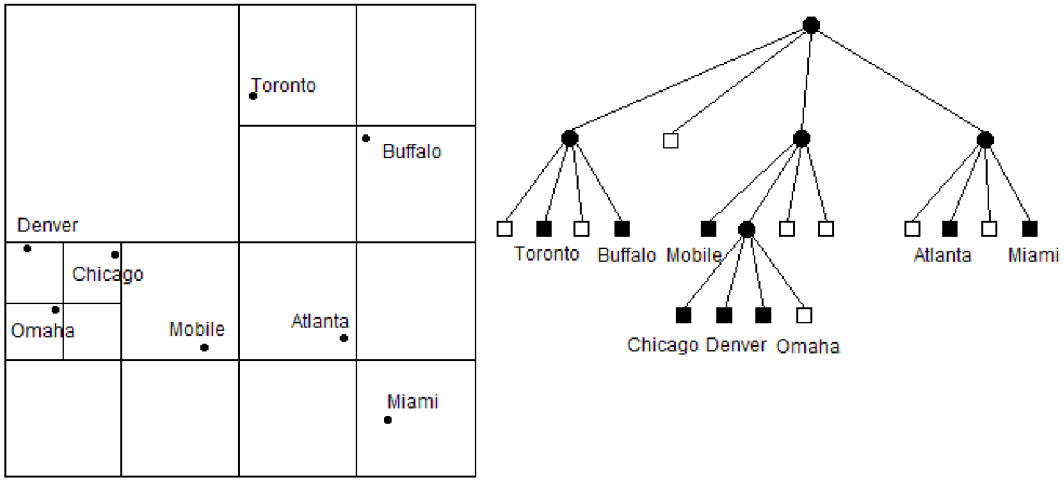
Jelikož je hodnota parametru ShrinkPolicy nastavena na Leaf Shrink, proto je každý bod ze vstupu vložen na první možné místo v závislosti na předchozích bodech. Díky BucketSize = 1 nese každý uzel pouze jeden bod. Parametr NumberOfSpacePartitions nastavený na hodnotu 2 určuje, že rozdělením vzniknou vždy jen dva nové prostory a to „left“ a „right“ definované v parametru Node Predicate.

4.2.2 Princip realizace Quad stromu

Quad strom je stromová datová struktura, ve které se každý vnitřní uzel rekurzivně rozděluje vždy na čtyři potomky. Quad strom se vyskytuje v různých variantách, které jsou určené pro více typů dat, např. pro body, obdélníky, nebo pro mnohoúhelníky. První variantou je MX (matrix) quad strom. Jedná se o variantu, kde každý bod ve stromu zastupuje čtverec o velikosti 1x1. Další variantou jsou

MX-CIF quad stromy, které slouží pro ukládání obdélníků. Pro ukládání polygonálních map slouží varianta PMR quad strom.

Varianta, která je na ukázána na obr. 4, se nazývá PR (point region) quad strom. V této variantě platí, že při rozdělování každé buňky vzniknout vždy 4 stejně velké nové kvadranty a následně je každý datový bod mapován do příslušného kvadrantu (na rozdíl od MX quad stromu, kde se body mapují do čtverců 1x1).



Obr. 4: Příklad PR Quad strom

Realizace PR Quad stromu na obrázku je definovaná těmito vstupními parametry:
ShrinkPolicy = Leaf Shrink,
NoOfSpacePartitions = 4,
Node Predicate = Kvadrant je reprezentován čtveřicí souřadnic (x1, y1, x2, y2), kde (x1, y1) jsou souřadnice levého horního rohu a (x2, y2) jsou souřadnice pravého dolního rohu.
Key Type = Point
BucketSize = 1

Pokud by v parametru ShrinkPolicy byla hodnota Never Shrink, všechny datové body by musely být na stejné úrovni. Parametr NoOfSpacePartitions obsahuje hodnotu 4, protože každý uzel musí mít buď žádného potomka, nebo právě 4 potomky.

4.3 Interní metody SP-GiST

Metody pro vkládání, odstraňování a hledání v SP-GiST jsou interní operace, které jsou implementovány uvnitř SP-GiST. Tyto metody se používají ve spojení s externími metodami pro realizaci specifických prostorově rozdělených stromů. Interní metody jsou obecné pro třídu prostorově dělitelných stromů a jejich konkrétní chování je realizováno skrze externí metody a parametry.

Algoritmus pro vkládání

Tento algoritmus je realizován několika metodami, proto jsem zvolil vysvětlení principu pomocí obecného popisu a pseudokódu. Hlavním cílem tohoto algoritmu je zachovat konzistentní stav stromu indexu v jakémkoliv okamžiku.

Pozn: V ostrých závorkách jsou umístěny odkazy pro podrobnější popis.

Pseudokód:

```
Začínám se s prvním uzlem v kořenové stránce <1>
loop {
  if(stránka je listová) then
    if(je na stránce dostatek místa) then
      vlož hodnotu na stránku a skonči;
    else
      volej funkce PickSplitFn() <2>
    end if;
  else
```

```
    switch (chooseFn())
        case MatchNode: sestup stromem přes vybraný uzel;
                break;
        case AddNode: přidej uzel a pak opakuj volání chooseFn();<3>
                break;
        case SplitTuple: rozdělení vnitřního uzlu na „prefix“ a
                „postfix“;
                volej metodu chooseFn() nad novým prefix uzlem;<4>
                break;
    end switch;
end if;
}
```

<1> Vkládáme datové položky (hodnota klíče a pozice v heap) do kořenové stránky dokud není plná, pak provedeme rozdělení (PickSplitFn()) celé datové n-tice (seznam datových položek). Po rozdělení datové n-tice vznikne nový vnitřní uzel, který se uloží do kořenové stránky. Rozdělené datové n-tice se přesunou na jiné paměťové stránky.

<2> Aktuální implementace SP-GiST indexu umožňuje provést Picksplit a vložení nové datové n-tice v jedné operaci, pokud se nový seznam datových n-tic vejde na jednu stránku. Tento princip je možný provádět pro stromy s malými uzly jako jsou quad stromy a kd stromy.

<3> Při přidání vnitřního uzlu (celé n-tice u quad stromu nebo kd stromu) do vnitřní pamětové stránky musí být kontrolován volný prostor na paměťové stránce, aby nedošlo k přetečení stránky.

<4> Pojem vytvoření „prefixu“ a „postfixu“ jsou typické pro strukturu radix stromu, ale stejný proces se provádí i při rozdělování vnitřních uzlů v quad stromu a kd stromu. Cílem této operace je vytvořit nad původním vnitřním uzlem nový vnitřní uzel (a původní uzel rozdělit na příslušný počet nových uzlů – quad strom = 4, kd strom = 2), díky kterému může index provést detailnější dělení prostoru.

Algoritmus pro vyhledávání

Stejně jako algoritmus pro vkládání začíná v kořenovém uzlu a postupně sestupuje stromem směrem k listům. Pro ověřování splnitelnosti predikátu nad vnitřními uzly používá externí metody InnerConsistent (vrací seznam adres uzlů, které jsou určené k dalšímu ověřování) a pro ověřování splnitelnosti predikátu nad listovými (datovými) uzly používá externí metody LeafConsistent (vrací seznam datových hodnot, nad kterými je predikát splnitelný).

```
Začínám v kořenovém uzlu
lock (kořenová stránka);
volej InnerConsistentFn(kořenový uzel);
vrácené adresy ulož do zásobníku;
unlock (kořenová stránka);

loop {
   if(zasobnik je prazdny) then
      ukonci cyklus;
   end if;
   vyjmi první adresu ze zásobníku;
   lock(stránka s adresou uzlu)
      if(listová stránka) then
         volej LeafConsistentFn (stránka);
      else
         volej InnerConsistentFn (stránka);
         ulož vrácené adresy na zásobník;
      end if;
   unlock(stránka s adresou uzlu);
}
```

V tomto algoritmu jsou využívány zámky stránek, na kterých je právě zpracován uzel vyhledávacím algoritmem. Stránky se zamykají proto, aby v průběhu zpracování uzlu nemohlo dojít k jeho změně, např. pomocí algoritmu pro vkládání.

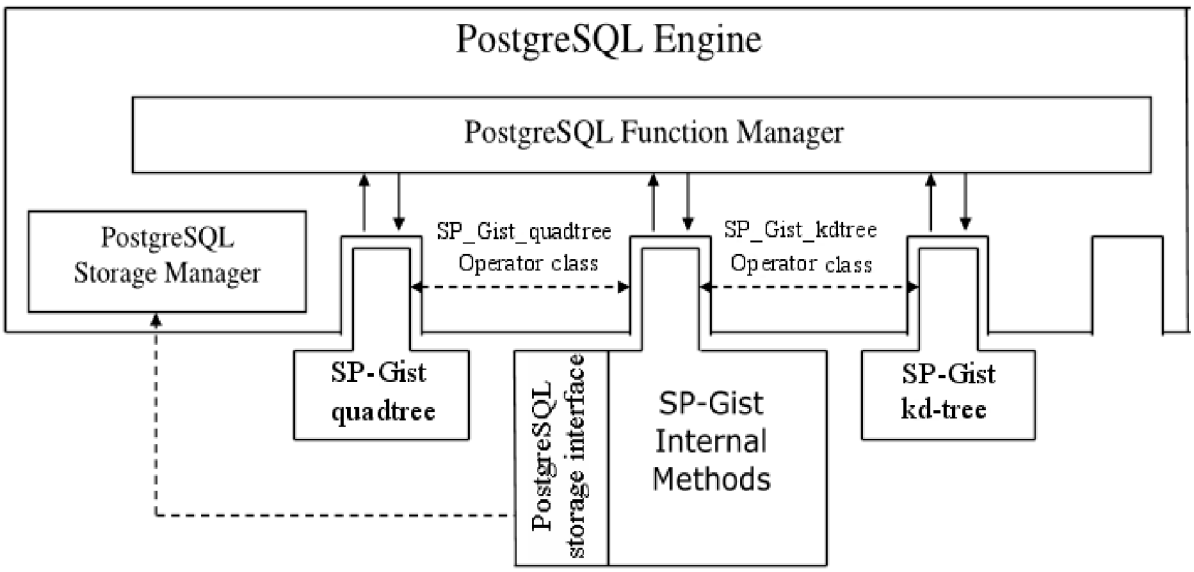
Algoritmus pro odstraňování

V SP-GiST indexu se neprovádí odstranění položky uzlu z paměti. Proces „odstraňování“ se provádí pomocí změny stavu položky z LIVE na DEAD nebo PLACEHOLDER[1]. Tím se adresa z položky do paměti (heap) zachová a umožní se její znovupoužití.

1 Tyto stavy jsou více popsány v souboru PostgreSQL/src/backend/access/spgist/README

4.4 Implementace SP-GiST indexu v PostgreSQL

Jak už bylo zmíněno, SP-GiST index je složen ze dvou hlavních částí, a to jádra indexu (obsahující vnitřní metody) a externích metod (společně s parametry) taky někdy nazýváno konkrétní rozšíření jádra. V systému PostgreSQL jsou tyto části od sebe navzájem plně izolovány. Spojení jádra a konkrétního rozšíření (externích metod) je provedeno pomocí tříd operátorů odpovídajících jednotlivým rozšířením (např. třída operátorů SP_Gist_quadtree pro rozšíření SP-GiST quad stromu). Samotná komunikace interních a externích metod je prováděna skrze Function Manager v PostgreSQL. Na obrázku obr. 5 je tato struktura propojení jednotlivých částí názorně ukázána.



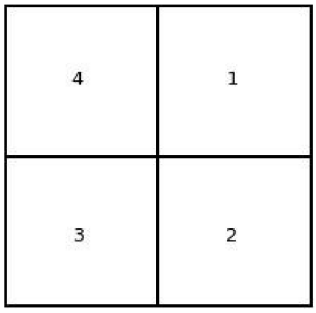
Obr. 5: Struktura propojení interních a externích metod SP-GiST indexu [6]

5 Implementace SP-GiST v PostGIS

5.1 Implementace PR quad stromu

Pro implementaci indexu SP-GiST v rozšíření PostGIS jsem zvolil konkrétně variantu PR quad strom. PR quad strom je stromová struktura se speciální vlastností. V tomto stromu mohou z každého uzlu vycházet buď právě čtyři následníci nebo žádný (tato vlastnost je určena parametrem NumberOfSpacePartitions rovnající se hodnotě čtyři). V tomto stromu se ukládají hodnoty v datovém typu bod. Implementovaný PR quad strom se vůči PR quad stromu na Obr. 4 mírně liší (kvůli efektivnějšímu řešení v implementaci). Prvním rozdílem těchto dvou quad stromů je, že při rozdělování prostoru se nevytváří vždy čtyři stejné kvadranty, ale nový dělící bod (prostřední bod) se vypočítá na základě průměru všech souřadnic bodů v daném prostoru, který se má rozdělit. Důvod, proč jsem zvolil tuto variantu, je kvůli lepšímu využití rozložení bodů do paměťových stránek (části prostoru s větším počtem umístěných bodů budou v menší oblasti, oproti části, kde je bodů méně).

Jedním z dalších rozdílů je způsob definování jednotlivých kvadrantů. Na Obr. 4 je kvadrant definován dvěma body v protilehlých rozích (levý horní a pravý dolní roh). Oproti tomu jsem v implementaci použil definování kvadrantů vůči prostřednímu bodu Obr. 6 (zdali se bod nachází v prvním až čtvrtém kvadrantu dle porovnání jednotlivých souřadnic vůči prostřednímu bodu).



Obr. 6: Rozložení kvadrantů v prostoru vůči prostřednímu bodu

Poslední zásadní rozdíl v quad stromu, který jsem implementoval, je v definování parametru velikosti BucketSize. Na Obr. 4 je nastavena na velikost 1, což by v implementaci bylo krajně neefektivní, protože by se muselo nepřetržitě provádět rozdělování prostoru. Proto jsem zvolil nejvyšší možnou

hodnotu (odpovídá maximálnímu počtu položek, které se vejdou na pamětovou stránku). Toto nastavení vede k tomu, že snížím počet provádění dělení prostoru na minimum.

5.2 Třída operátorů pro PR quad strom v PostGIS

Stejně jako v databázovém systému PostgreSQL, tak i v jeho rozšíření je potřeba definovat třídu operátorů pro každou konkrétní implementaci indexu. Jedinou výhodou vytváření třídy operátoru v PostGIS vůči PostgreSQL je to, že se celá třída definuje na jednom místě a není potřeba hledat, kde se co musí doplnit v celém rozsáhlém katalogu.

Vytvořil jsem novou třídu operátorů („spgist_geometry_ops“) využívající databázový index SP-GiST. Datový typ hodnot, se kterými bude index pracovat, jsem musel zvolit Point (bod). Třída operátorů obsahuje seznam operátorů, které jsou definovány v SP-GiST a měly by být podporovány pro danou realizaci indexu v quad stromu. Tyto operátory slouží proto, aby skrze ně uživatel mohl používat vytvořenou konkrétní realizaci indexu. Pokud jsou zařazené operátory již definovány v některé ze tříd a odpovídají i definice operandů, nemusím operátor v souboru pro definici tříd operátorů znovu definovat (v implementaci konkrétního rozšíření musím provádět definici chování operátoru). Není-li operátor ještě definován, je potřeba jej definovat (definovat symbol, nebo skupinu symbolů pro daný operátor, určit datový typ operandů a metodu, která provádí danou operaci definovanou operátorem).

Poslední částí třídy operátorů je definice metod realizující implementaci konkrétního indexu. Tento seznam obsahuje externí metody popsané již v kap. 5.2. Jednou z vlastností celého frameworku SP-GiST je, že všechny konkrétní realizace (quad strom, kd strom, trie) musí mít definované stejné rozhraní externích metod pro komunikaci s jádrem SP-GiST (stejně to funguje i u GiST frameworku), interními metodami.

Pro správné načítání vstupních hodnot do stromu indexu jsem musel vytvořit metodu Compress (popsanou v kap. 5.2), pomocí které převádím vstupní hodnoty indexu na datový typ bod (Point). Následně se provádí vložení bodu do stromu indexu. Aby se správně používala vytvořená metoda Compress musel jsem vložit do jádra SP-GiST volání této metody (to implikuje potřebu definovat metodu Compress i v ostatních konkrétních rozšíření SP-GiST).

Použití datového typu Point pro ukládání hodnot quad stromu v PostGIS

Na základě konzultace s panem Paulem Ramsey[1](jedním z vývojářů PostGIS), jsem zjistil, že aktuálně v SP-GiST není možné načítat do paměti větší objekty (mnohoúhelníky, obdélníky, atd.) než je bod, protože mohou být větší než stránka paměti a tím by došlo k přetečení paměťové stránky při vkládání objektu do indexu.

Při vkládání objektů (uložených v datovém typu geometry) do indexu, se neprovádí detekce o jaké objekty se jedná. K detekci konkrétního typu objektu v rámci datového typu geometrie se systém dostane až při provádění rozdělování uzlu, pokud jeho velikost překročí BucketSize. Toto je důvod, proč chybí implementace SP-GiST indexu v PostGIS. Proto mi pan P. Ramsey, napsal, že tento index je možné implementovat v PostGIS pouze s datovým typem point a to jen pro demonstrační účely.

5.3 Externí metody PR quad stromu

V kap. 5.2 jsou definovány jednotlivé externí metody. V implementaci jsem je použil k realizování PR quad stromu. První metodou, která je volaná jádrem SP-GiST, je metoda Config. V této metodě jsem nastavil několik parametrů pro upřesnění stromu, ale nejdůležitějším z nich je nastavení datového typu pro klíč ukládaný ve vnitřních uzlech. Tuto hodnotu jsem nastavil na datový typ Point.

5.3.1 Externí metody pro vkládání nových objektů (bodů)

Nyní už následuje načítání jednotlivých bodů (objektů datového typu geometry) a vytváření struktury quad stromu. Jelikož jsou tyto hodnoty vkládány v datovém typu geometry, musí se provést z této serializované podoby konverze do datového typu bod (Point). Konverzi provádí metoda Compress a vrací do jádra SP-GiST bod, který se následně ukládá skrze listové uzly do stránek paměti.

Jakmile se zaplní některý z listových uzlů, jádro zavolána metodu Picksplit. V Picksplit metodě se vytvoří nový vnitřní uzel nad původním listovým uzlem, ze kterého se jednotlivé položky (dvojice klíč a ukazatel) rozdělí do čtyř nově vytvořených kvadrantů (čtyři nové listové uzly). Do nově vytvořeného vnitřním uzlu se vloží hodnota klíče (bod), vypočítaná jako průměr všech bodů v původním listovém uzlu. Následuje rozdělení všech bodů (položek, původního uzlu) do příslušných kvadrantů podle klíče nového vnitřního uzlu.

1 http://blog.cleverelephant.ca/

Pokud už vytvářený quad strom obsahuje i vnitřní uzly (není zde jen kořenový uzel) je potřeba průchod stromem provádět pomocí metody Choose, která vždy porovnává hodnotu klíče vkládaného bodu vůči aktuálnímu vnitřnímu klíči uzlu. Výstupem této metody je vždy struktura ve které je obsažen jak nově vkládaný bod, tak i číslo kvadrantu, kde má pokračovat průchod stromem.

5.3.2 Externí metody pro vyhledávání hodnot splňujících požadovaný predikát

Externí metodou, která s pomocí jádra SP-GiST prochází vnitřními uzly quad stromu a tím napomáhá k vyhledávání datových hodnot splňujících predikát zadaný na vstupu uživatelem, je metoda InnerConsistent. Metoda získá na vstupu vnitřní uzel a predikát. Provádí dotaz na splnitelnost predikátu s hodnotou klíče vnitřního uzlu. Pokud je tento predikát splnitelný s hodnotou klíče, vrátí se pouze odkazy na kvadranty, které splňují podmínku v predikátu (ostatní kvadranty prohledávány nejsou). Pokud však pro predikát v aktuálním uzlu neplatí splnitelnost, funkce dále neprochází podstrom aktuálního uzlu a pokračuje v dalším uzlu.

Poslední velmi důležitou externí metodou je LeafConsistent. Tato metoda slouží pro vyhodnocování splnitelnosti predikátu v listových uzlech quad stromu. V těchto uzlech se provádí vyhodnocování predikátu nad konkrétními hodnotami, které jsou hledány. Proto pokud je predikát splněn s danou hodnotou klíče, je následně tato datová položka vrácena z metody. Není-li však predikát splnitelný nad hodnotou klíče, metoda nevrací žádnou datovou položku.

6 Ověřování chování PR quad stromu

Popis zařízení

Ověřování správnosti chování implementovaného PR quad stromu bylo prováděno na notebooku Thinkpad Edge E520 s dvou-jádrovým procesorem o frekvenci 2,1 Ghz a RAM pamětí o velikosti 4 GB. Na tomto zařízení je nainstalován databázový systém PostgreSQL verze 9.5.0 s rozšířením PostGIS ve verzi 2.2.1. Komunikace databázového systému s uživatelem byla prováděna skrze interaktivní terminálový program *psql*, který PostgreSQL systém nabízí pro práci s databází. Program psql umožňuje interaktivní vkládání, úpravu a provádění SQL příkazů.

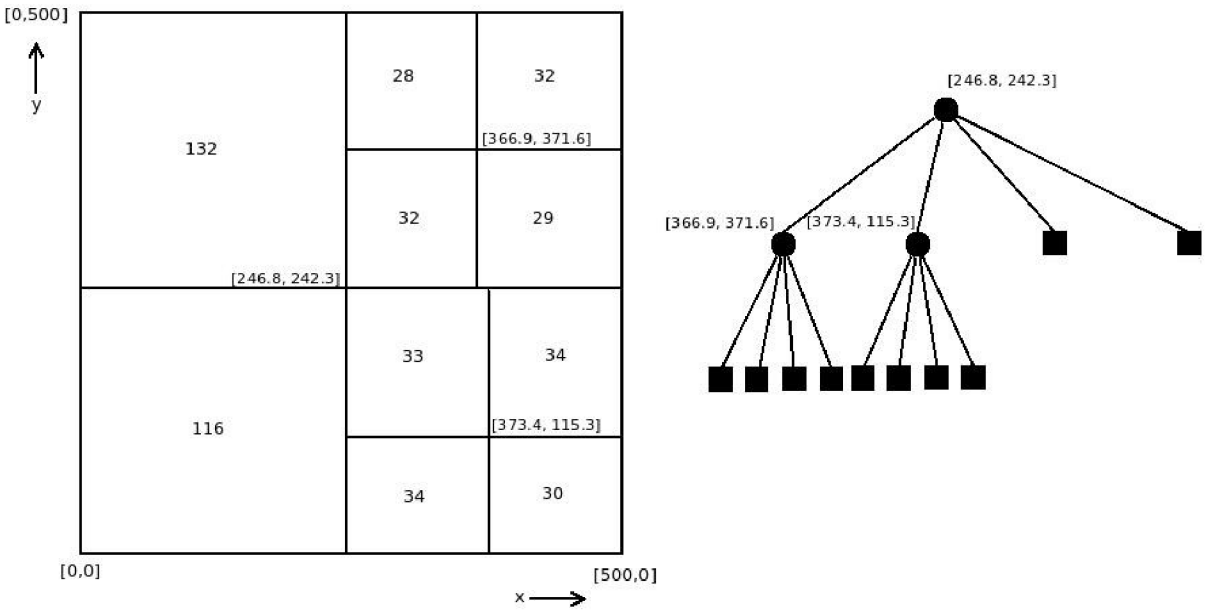
6.1 Ověřování chování PR quad stromu při vytváření indexu

Proces ověřování správného vytváření quad stromu indexu jsem prováděl v několika krocích. V prvním kroku jsem vytvořil tabulku „tab1_body“ se dvěma sloupci. První sloupec s názvem id je určen pro identifikační celočíselné hodnoty jednotlivých záznamů číslované od 0. Druhý sloupec „body“ je vytvořen pro ukládání objektů bodů datového typu geometrie. Následně jsem tabulku naplnil tisíci záznamy (pseudonáhodně vygeneroval z definovaného rozsahu). Body se nachází ve dvoudimenzionálním prostoru ohraničeném intervaly na ose x i y <0, 500>.

Následně jsem vložil příkaz pro vytvoření databázového indexu pojmenovaného „body_spgist“ nad tabulkou „tab1_body“ a sloupcem „body“.

```
CREATE INDEX body_spgist ON tab1_body USING spgist (body
spgist_geometry_ops);
```

Vytvořila se struktura quad stromu s pěti sty datovými body a třemi vnitřními (interními) body.



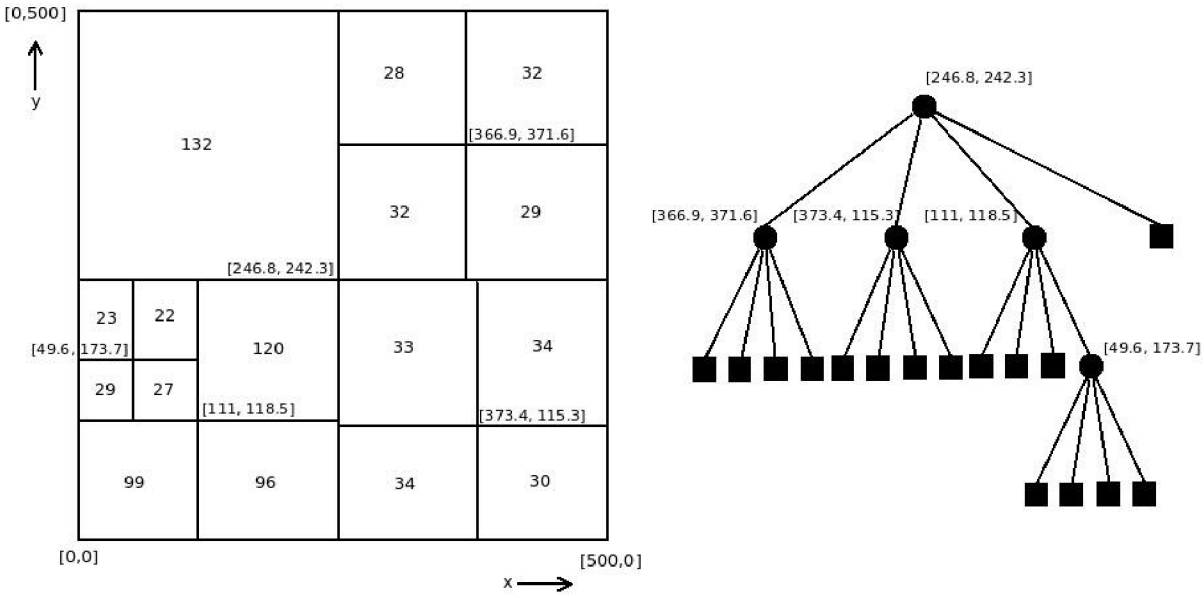
Obr. 7: Quad strom SP-GiST indexu body_spgist

V levé části Obr. 7 (rozdělení prostoru) jsou znázorněny vnitřní body, které byly vytvořeny na základě průměru všech bodů v původním prostoru při naplnění velikosti vyhrazeného pro kvadrant

(BucketSize). Čísla vepsaná v jednotlivých kvadrantech udávají počet bodů, které jsou uloženy v daném kvadrantu. Listové uzly v quad stromu (pravá část obr. 7) zastupují celý kvadrant (všechny body v příslušném kvadrantu).

Při vytváření indexačního stromu jsem si nasimuloval ukládání bodů do jednotlivých částí prostoru (pomocí výpisů z kódu). Následně jsem ověřil počty bodů v jednotlivých částech prostoru pomocí sql dotazů na vymezený prostor.

Dalším krokem ověřování správného chování SP-GiST indexu v PostGIS je, zdali strom správně změní svoji strukturu, pokud se do tabulky, nad kterou již existuje databázový index, vloží nové hodnoty (300 nových bodů).



Obr. 8: Quad strom SP-GiST indexu body_spgist po vložení 300 nových bodů

Při vkládání nových položek do tabulky se tyto položky zároveň vkládají do quad stromu na příslušnou pozici. Na Obr. 8 je vidět, jak se změnila struktura rozdělení prostoru po vložení dalších tři sta nových bodů v rozsahu na ose x <0, 246.8> a ose y <0, 242.3> vůči obr. 7. Na místo původního třetího kvadrantu (třetí kvadrant v kořenovém vnitřním uzlu), který obsahoval jen listový uzel s množinou bodů, bylo potřeba vytvořit nový vnitřní uzel s hodnotou klíče [111, 118.5]. Jelikož, ani po tomto rozdělení na čtyři nové kvadranty se všechny body nevešly na jednu stránku paměti, bylo potřeba znovu rozdělit čtvrtý kvadrant na čtyři nové kvadranty a vypočítat novou hodnotu klíče pro vnitřní uzel (nový bod [49.6, 173.7]).

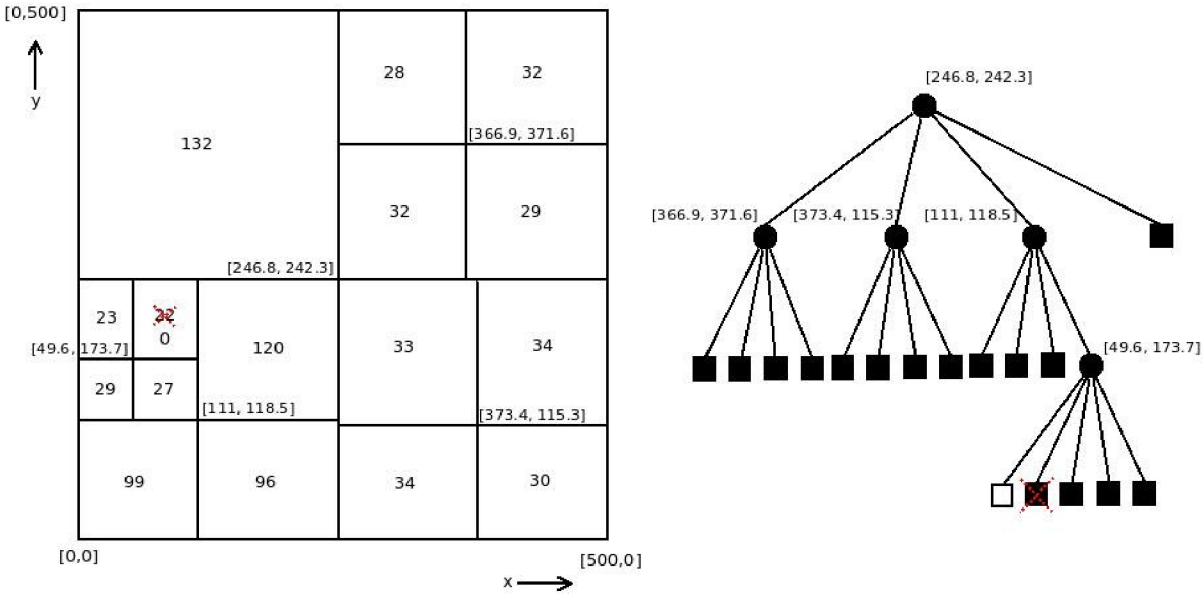
Tímto postupem jsem si ověřil (provedl jsem i ruční kontrolu počtu bodů), že implementovaný quad strom tvoří strukturu quad stromu správným způsobem podle popisu v kap 6.1, a to nejen na již vložená data v tabulkách, ale i na data, která jsou vkládána do tabulky až po vytvoření indexu nad touto tabulkou.

Sql skripty, pomocí kterých jsem prováděl toto ověřování, jsou uloženy na přiloženém CD.

6.2 Ověřování chování indexu při odstraňování položek z indexu SP-GiST

Pro ověření chování indexu při odstraňování záznamů z tabulky (tím i z quad stromu SP-GiST indexu) jsem použil tabulku „tab1_body“ s 800 záznamy. Nad touto tabulkou je vytvořený SP-GiST index pomocí quad stromu. Provedení operace odstranění záznamů z tabulky „tab1_body“ jsem provedl následujícím příkazem.

```
DELETE FROM tab1_body WHERE body << 'point(111 118.5)'::geometry AND
body >> 'point(49.6 173.7)'::geometry AND
body <^'point(246.8 242.3)'::geometry AND
body >^ 'point(49.6 173.7)'::geometry;
```



Obr. 9: Quad strom SP-GiST indexu body_spgist po odstranění 22 bodů z 1. kvadrantu vnitřního bodu [49.6, 173.7]

Po odstranění všech 22 položek z 1. kvadrantu vnitřního uzlu s hodnotou klíče [49.6, 173.7] se nastaví ve vnitřním uzlu odkaz na listový uzel na hodnotu null. V grafovém stromovém znázornění quad stromu se projeví tento prázdný prostor tím, že místo plného čtverečku (znázorňující výskyt minimálně jednoho bodu v prostoru) prázdný čtvereček (znázorňující prázdný kvadrant).

6.3 Ověřování správnosti chování operátorů z třídy operátorů spgist_geometry_ops

V této části jsem provedl ověření chování šesti operátorů definovaných v třídě operátorů spgist_geometry_ops, která byla vytvořena s pomocí PR quad stromu SP-GiST indexu. Jako datový zdroj jsem použil tabulku tab1_body, kde její rozložení bodů v prostoru je zobrazeno na obr. 8.

Prvním ověřovaným byl operátor pojmenovaný striktně vpravo „>>“. Aby se dalo snadno ověřovat správnost vráceného počtu bodů v daném prostoru vůči Obr. 8 zvolil jsem hraniční bod [246.8, 242.3], kde počet všech bodů vpravo od tohoto bodu je 252. Pro jednodušší kontrolu, kolik mi systém vrátil záznamů, jsem použil vestavěnou metodu COUNT() nad sloupcem id z tabulky

tab1_body. Ta spočítá počet vrácených záznamů a toto číslo vypíše na výstup terminálu. Operátor „>>“ jsem ověřil následujícím dotazem:

```
SELECT COUNT(id) AS pocet FROM tab1_body
WHERE body >> 'point(246.8 242.3)'::geometry;

 pocet
-------
   252
```

Na základě korektního chování z předcházejícího ověření jsem navázal rozšířením podmínky s operátorem pojmenovaným je výše „>^“. Do klauzule WHERE jsem přidal další podmínku, tentokrát omezující prostor ze spodní strany bodem [246.8, 242.3]. Výstupem by mělo být 121 bodů (podle Obr. 8).

```
SELECT COUNT(id) AS pocet FROM tab1_body
WHERE body >> 'point(246.8 242.3)'::geometry AND
      body >^ 'point(246.8 242.3)'::geometry;

 pocet
-------
   121
```

Dalším operátorem, který jsem přidal do podmínky pro ověření správnosti chování, je operátor striktně vlevo „<<“ (je operátor s opačným chováním vůči operátoru „>>“). S pomocí tohoto operátoru jsem omezil prostor s body z pravé strany bodem [366.9, 371.6]. Podle Obr. 8 by výsledná hodnota měla být 60.

```
SELECT COUNT(id) AS pocet FROM tab1_body
WHERE body >> 'point(246.8 242.3)'::geometry AND
      body >^ 'point(246.8 242.3)'::geometry AND
      body << 'point(366.9 371.6)'::geometry;

 pocet
-------
    60
```

Poslední strana, ze které ještě nebyl prostor omezen, bylo shora. Proto jsem přidal operátor je níže „<^“. V podmínce jsem s tímto operátorem definoval omezení prostoru pomocí bodu [366.9, 371.6]. Očekávaným výsledkem byla hodnota 32.

```
SELECT COUNT(id) AS pocet FROM tab1_body
WHERE body >> 'point(246.8 242.3)'::geometry AND
      body >^ 'point(246.8 242.3)'::geometry AND
      body << 'point(366.9 371.6)'::geometry AND
      body <^ 'point(366.9 371.6)'::geometry;

 pocet
-------
  32
```

Operátorem pro vyhledávání konkrétního bodu v prostoru je v SP-GiST tato dvojice symbolů „~=“. Jako vyhledávaný bod jsem si vybral např. bod [308, 258].

```
SELECT id, ST_ASTEXT(body)AS body FROM tab1_body
WHERE body ~= 'point(308 258)'::geometry;
 id  |      body
-----+----------------
 126 | POINT(308 258)
```

Posledním operátorem definovaným ve třídě spgist_geometry_ops je operátor – obsahuje „@“ (v dokumentaci je sice popsán symboly „<@“, ale v implementaci je definován jen pomocí „@“). Tento operátor je implementován tak, aby mohl zpracovat jako dotazovaný objekt nejenom bod (point), ale i další objekty z datového typu geometrie (geometry). V tomto případu ověření správnosti chování operátoru „@“ jsem použil objekt mnohoúhelník (polygon) s pěti body (z nichž první a poslední bod je stejný). Pomocí mnohoúhelníku jsem ohraničil stejný prostor jako v třetím případu ověřování. Měl jsem tedy získat i stejný výsledek, tj. 60.

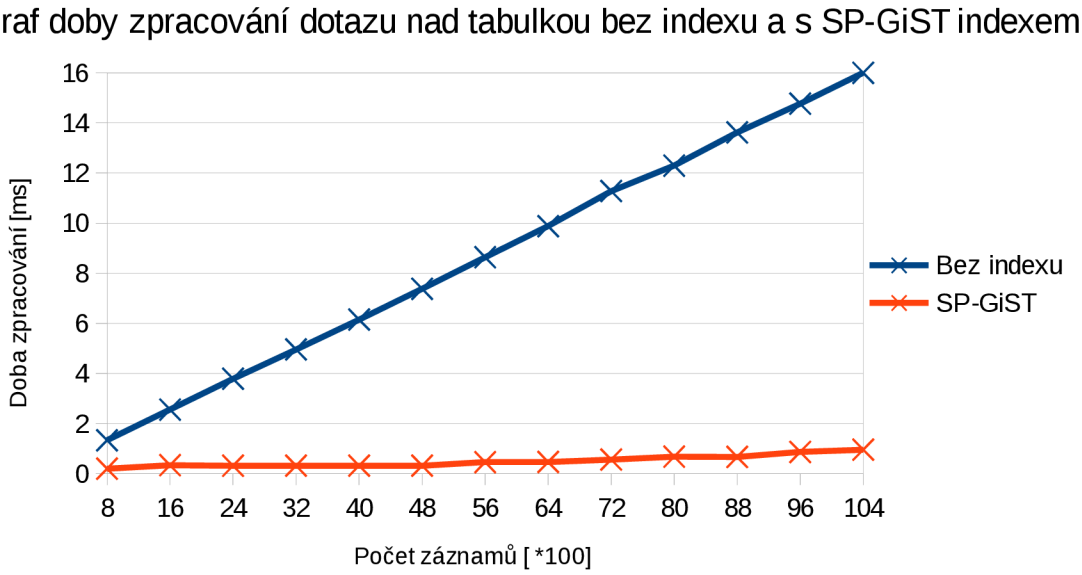
```
SELECT COUNT(id) FROM tab1_body
WHERE body @ 'polygon((246.8 242.3, 246.8 500, 366 500, 366.9 242.3,
246.8 242.3 ))'::geometry;
 pocet
-------
   60
```

6.4 Srovnání rychlosti zpracování databázových dotazů bez použití indexu a s indexem SP-GiST v PostGIS

Pro srovnávání rychlosti zpracování databázových dotazů nad tabulkou jsem použil opět agregační databázový dotaz:

```
SELECT COUNT(id) FROM tab1_test
WHERE body @ 'polygon((246.8 242.3, 246.8 500, 366 500, 366.9 242.3,
246.8 242.3 ))'::geometry;
```

Před prováděním samotného měření doby provádění dotazu jsem si vytvořil třináct tabulek pojmenovaných tab1_test až tab13_test. V tabulce tab1_test bylo 800 záznamů, v tabulce tab2_test 1 600 záznamů, atd. až v tabulce tab13_test bylo 10 400 záznamů (v každé tabulce od 1 do 13 bylo pseudonáhodně vygenerováno 800 nových záznamů). Nad každou z tabulek jsem provedl nejprve 1000 krát databázový dotaz bez vytvořeného indexu (prováděl jsem tedy sekvenční prohledávání tabulek) a následně opět 1000 krát tím stejným dotazem nad tabulkami už s vytvořeným indexem (implementovaným PR quad stromem). Z těchto získaných hodnot jsem vždy vypočítal průměrnou hodnotu a tu jsem následně zanesl do grafu. V grafu jsem zvolil pro osu x počty záznamů v jednotlivých tabulkách místo názvů jednotlivých tabulek (názvy tabulek nejsou tak důležité jako množství záznamů uvnitř tabulek pro tento graf) a na ose y je čas v milisekundách. Nezbytnou podmínkou bylo, aby dotaz vždy vrátil správnou množinu hodnot, což bylo dodrženo.



Graf 1: Porovnání doby zpracování agregačního dotazu nad tabulkami s SP-GiST indexem a bez indexu

Tabulka s daty pro graf č. 1 je v příloze č. 2. Podle průběhu modré barvy (provádění dotazů bez databázového indexu) jde vidět, že s rovnoměrným nárůstem množství záznamů v tabulkách, které musí systém projít pro nalezení hledaných hodnot, se rovnoměrně zvyšuje doba zpracování dotazu. Zatímco průběh oranžové barvy, znázorňující provádění dotazů nad sloupcem tabulky s databázovým indexem SP-GiST nad daným množstvím dat téměř neroste v porovnání s druhým průběhem.

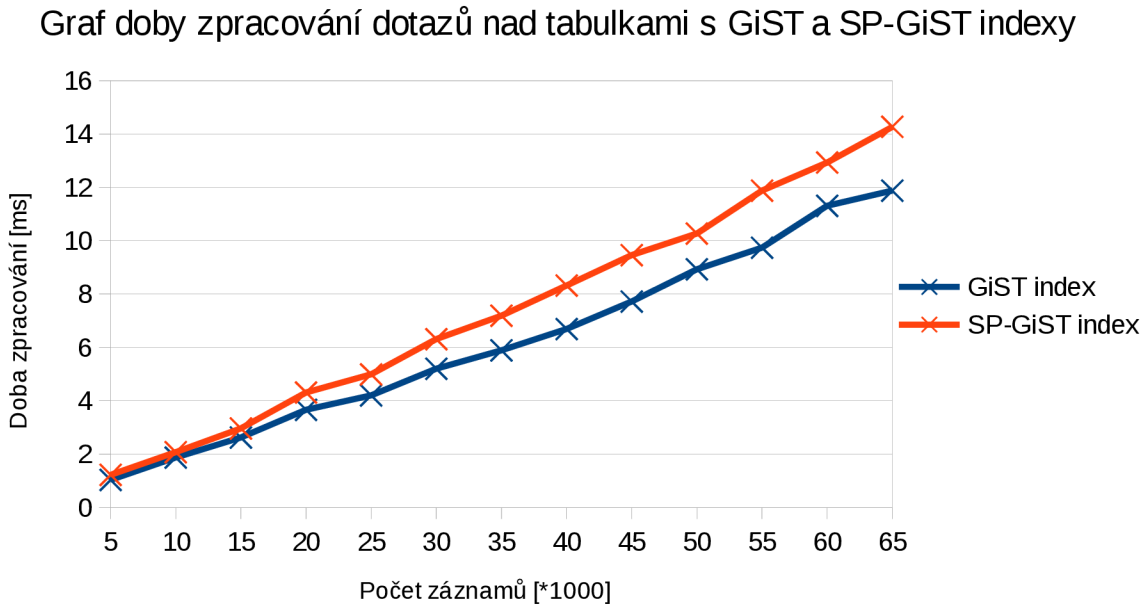
6.5 Srovnání doby zpracování indexů SP-GiST a GiST v PostGIS

K provedení tohoto srovnání jsem si vytvořil třináct tabulek pro SP-GiST index a třináct pro GiST index. První tabulka měla 5000 záznamů, druhá měla 10 000 záznamů, atd. až poslední třináctá tabulka měla 65 000 záznamů. Tabulky pro oba indexy měly stejný obsah. Tabulky obsahovaly vždy dva sloupce, kde prvním byl id obsahující celá čísla od 1 do maximálního počtu záznamů a druhým sloupcem byl body (typu geometrie). Body v tabulkách byly generovány pro souřadnice x a y v rozsahu 0 – 1 000. Tento rozsah byl stejný jak pro tabulku s 5 000 body, tak i pro tabulku s 65 000

body. Jako databázový dotaz jsem zvolil opět agregační dotaz s definováním prostoru pro výběr pomocí mnohoúhelníku, ale oproti dotazu v minulém příkladu jsem upravil prostor mnohoúhelníku.

```
SELECT count(id) FROM test2_tab1
WHERE body @ 'polygon((246 200, 246 800, 566 800, 566 200,
246 200 ))'::geometry
```

Pro osu x jsem zvolil rozsah hodnot od 5 000 až po 65 000 záznamů (zastupující tabulky jedna až třináct). Na osu y jsem vynesl dobu zpracování v milisekundách. Ověřování jsem prováděl tak, že pro každou tabulku jsem provedl 1000 krát výše definovaný dotaz (měnil jsem pouze název tabulky) a pomocí nástroje explain s parametrem analyze jsem měřil čas provádění dotazu. Následně jsem provedl průměr ze získaných časů a ten zanesl do grafu.



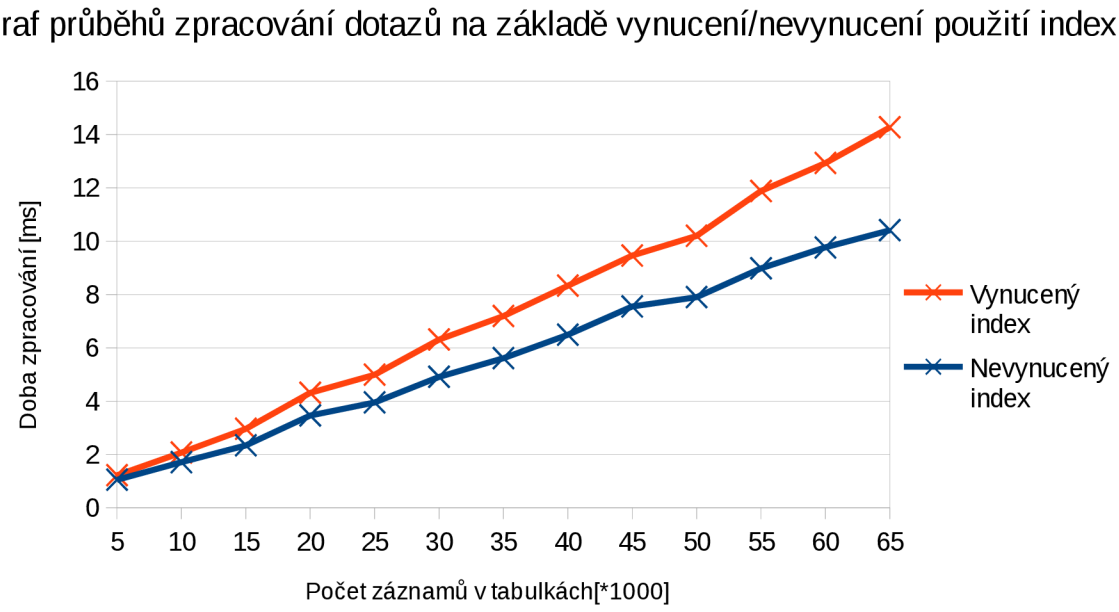
Graf 2: Srovnání doby zpracování dotazů nad tabulkami s různým počtem položek a indexy GiST a SP-GiST

Zdroj dat pro průběhy v tomto grafu jsou umístěny v tabulce v příloze č. 3. Průběhy v grafu č. 2 ukazují, že databázové dotazy nad tabulkami s GiST indexem v porovnání s databázovými dotazy nad tabulkami s naimplementovaným SP-GiST indexem nad danými daty, dosahují menších časů při zpracování.

Pro všechny databázové dotazy, které byly dosud provedeny, bylo nastaveno vynucené použití indexu, pokud byl nad dotazovanou tabulkou (sloupcem) definován. Toto nastavení nevedlo v mnoha případech k nejefektivnější variantě průchodu záznamů v tabulkách. Názorná ukázka je v následujícím příkladu.

6.6 Srovnání doby zpracování databázových dotazů na základě vynucení a nevynucení použití indexu

Pro tento příklad jsem použil datový průběh SP-GiST z předchozího příkladu (modrý průběh v grafu č. 3) a abych získal druhý průběh zrušil jsem vynucení použití indexu (v tomto konkrétním případě indexu SP-GiST) při prohledávání záznamů v tabulkách na základě databázového dotazu. Hodnoty pro vykreslení oranžového průběhu jsem získal prováděním stejného dotazu jako v předchozím příkladu, nad tabulkami rovněž z předchozího příkladu.



Graf 3: Průběhy zpracování dotazů na základě vynucení nebo nevynucení použití indexu v prováděcím plánu

Klíčovým prvkem, který ze dvou stejných databázových dotazů provedených nad naprosto shodnými tabulkami získá různé doby zpracování těchto dotazů, je optimalizátor (pomocí něho se provádí optimalizace zadaného dotazu). Pomocí nastavení parametrů (enable_seqscan, random_page_cost [1]) optimalizace lze ovlivnit vynucení použití indexu pro procházení tabulkou, nad kterou daný index existuje. Výsledný průběh s indexem (modrý průběh) dosahuje tedy delší doby zpracování databázových dotazů z toho důvodu, že databázový systém PostgreSQL nepodporuje klastrované indexy (indexy, které zaručují, že pořadí záznamů v indexovaném sloupci tabulky bude zachováno i ve fyzickém úložišti v paměti). Tento fakt má za důsledek, že po nalezení hledaného uzlu (podle klíče) se musí ještě znovu provést hledání hodnoty podle pozice v datovém souboru v režimu náhodného přístupu. Tato režie, která se musí provádět, má za následek v mém případě horší dobu zpracování dotazů. Průběh s lepšími výsledky (oranžový průběh) využívá pro procházení dat v tabulce plán nazvaný „bitmap index scan“[2]. Tento plán je složen ze dvou částí. Nejprve se pomocí indexu naleznou oblasti řádků, které splňují podmínku indexu a následně se tyto oblasti řádků načtou z hromady a hledají se jednotlivé záznamy, splňující podmínku dotazu.

1 https://www.postgresql.org/docs/8.0/static/runtime-config.html

2 https://www.postgresql.org/docs/8.1/static/performance-tips.html

7 Závěr

Výsledkem diplomové práce je implementace SP-GiST indexu pomocí PR quad stromu v PostGIS. Tato implementace je fyzickým výsledkem této práce, ale mnohem důležitějším přínosem je zjištění, z jakých důvodů tento index stále chyběl v oficiální veřejné verzi PostGIS, která je dostupná pod open source licencí. Jádro problému tkví už v samotném databázovém systému PostgreSQL, kde je implementováno jádro SP-GiST indexu. Problémem je způsob, jak ukládat objekty prostorových datových typů do paměťových stránek. Tento problém se týká hlavně datového typu geometrie, který může obsahovat objekty bod, mnohoúhelník, obdélník, křivka atd.. Tyto objekty mohou mít různou velikost (kromě bodu) a tím, že systém nedokáže tuto velikost nějak kontrolovat (hrozí, že jeden objekt bude větší, než celá paměťová stránka), nelze zaručit stabilitu tohoto indexu. Tento problém jsem konzultoval s jedním z hlavních vývojářů PostGIS panem P. Ramsey.

Proto se implementoval SP-GiST index v PostGIS systému pouze pro bod a tuto možnost jsem využil i pro implementaci v diplomové práci. Implementace SP-GiST indexu, kterou jsem v rámci diplomové práce provedl, slouží tedy jen pro demonstraci chování tohoto indexu v rozšíření PostGIS. Protože reálné využití SP-GiST indexu v PostGIS s možností indexovat pouze objekty typu bod není velkým přínosem, a zároveň není možné omezit, aby se uživatelé nepokoušeli ukládat i jiné objekty (což povede k neočekávanému chování indexu), nedoporučuji tuto implementaci zahrnovat do oficiální verze PostGIS pro běžné užívání veřejností.

Jakmile se tento problém v PostgreSQL systému vyřeší, bude tato indexační metoda velkým přínosem.

Literatura

[1] *PostgreSQL: Global Development Group* [online]. California: Regents of the University of California, 2016 [cit. 2016-05-30]. Dostupné z: http://www.postgresql.org

[2] PostGIS: Feature List. *PostGIS: Spatial and Geographic objects for PostgreSQL* [online]. Beaverton, Oregon Unated States, 2012 [cit. 2016-05-30]. Dostupné z: http://postgis.net/features/

[4] ELTABAKH, Mohamed, Walid AREF a Ramy ELTARRAS. *Space-partitioning Trees in Postgr eSQL: Realization and Perf ormance: The 22nd International Conference of Data Engineering* [online]. Atlanta: Purdue University, 2006, p.100-111 [cit. 2016-05-30]. DOI: 10.1109/ICDE.2006.146. Dostupné z: https://www.cs.purdue.edu/spgist/papers/icde06.pdf

[5] RAMSEY, Paul. PostGIS 2.1.6 Released. In: *PostGIS: Spatial and Geographic objects for PostgreSQL* [online]. 2015 [cit. 2016-05-30]. Dostupné z: http://postgis.net/2015/03/20/postgis-2.1.6/

[6] AREF, Walid a Ihab ILYAS. *A Framew ork for Supporting the Class of Space Partitioning Trees* [online]. Purdue University, West Lafayette, 2001, 22 s. [cit. 2016-05-30]. Dostupné z: http://www.sai.msu.su/~megera/postgres/gist/papers/sp-gist.pdf

[7] HELLERSTEIN, Joseph M. - Avi PFEFFER - NAUGHTON, Jeffrey F. (ed.). Generalized Search Trees for Database Systems. In *Proceedings of the 21st International Conference on Very Large Data Bases* [online]. San Francisco: Morgan Kaufmann Publishers, 1995 [cit. 2016-05-30]. ISBN 15-586-0379-4. Dostupné z: http://db.cs.berkeley.edu/papers/vldb95-gist.pdf

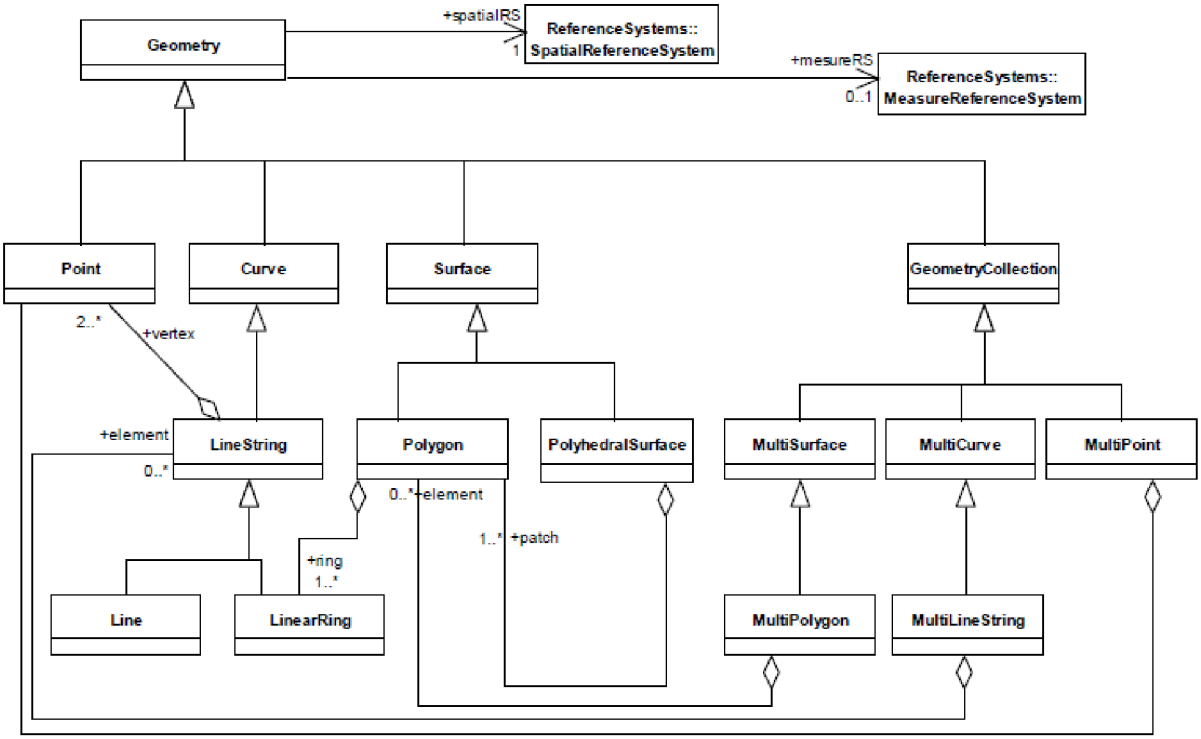
[8] *PostgreSQL: PostgreSQL Global Development Group*. ©1996-2016. Dostupné také z: http://doxygen.postgresql.org/gistproc_8c_source.html

[9] HERRING, John R. (ed.). *OpenGIS ® Implementation Standard for Geographic information: Simple feature access - Part 2: SQL option*. 2010. ref. Num. OGC 06-104r4 Dostupné také z: http://portal.opengeospatial.org/files/?artifact_id=25355

[10] PITTS, Robert I. *Trie: A kind of tree for efficient retrieval using keys* [online]. Boston, 2000 [cit. 2016-05-30]. Dostupné z: http://www.cs.bu.edu/teaching/c/tree/trie/. Teaching material. Boston University.

Seznam příloh

Příloha č. 1: Obrázek SQL hierarchie prostorového typu geometrie [9]



Příloha č. 2: Datový zdroj pro Graf č.1

	Doba zpracování databázového dotazu	
Tabulka/Počet záznamů	**Bez indexu [ms]**	**SP-GiST index[ms]**
tab1_test / 800	1.331	0.197
tab2_test / 1600	2.564	0.325
tab3_test / 2400	3.792	0.317
tab4_test / 3200	4.966	0.319
tab5_test / 4000	6.158	0.318
tab6_test / 4800	7.386	0.319
tab7_test / 5600	8.640	0.459
tab8_test / 6400	9.880	0.466
tab9_test / 7200	11.284	0.563
tab10_test / 8000	12.301	0.679
tab11_test / 8800	13.624	0.667
tab12_test / 9600	14.777	0.868
tab13_test / 10400	16.014	0.958

Příloha č. 3: Datový zdroj pro Graf č.2

	Doba zpracování databázového dotazu	
Tabulka / Počet záznamů	**GiST index [ms]**	**SP-GiST index [ms]**
test2_tab1 / 5 000	1.037	1.228
test2_tab2 / 10 000	1.873	2.076
test2_tab3 / 15 000	2.622	2.963
test2_tab4 / 20 000	3.667	4.315
test2_tab5 / 25 000	4.203	4.990
test2_tab6 / 30 000	5.208	6.315
test2_tab7 / 35 000	5.894	7.196
test2_tab8 / 40 000	6.695	8.332
test2_tab9 / 45 000	7.726	9.466
test2_tab10 / 50 000	9.135	10.270
test2_tab11 / 55 000	9.648	11.889
test2_tab12 / 60 000	11.342	12.934
test2_tab13 / 65 000	11.878	14.279

Příloha č. 4: Datový zdroj pro Graf č. 3

	Doba zpracování databázového dotazu	
Tabulka / Počet záznamů	**Bez vynucení indexu [ms]**	**s vynucením SP-GiST indexu [ms]**
test2_tab1 / 5 000	1.054	1.228
test2_tab2 / 10 000	1.717	2.076
test2_tab3 / 15 000	2.348	2.963
test2_tab4 / 20 000	3.462	4.315
test2_tab5 / 25 000	3.956	4.990
test2_tab6 / 30 000	4.917	6.315
test2_tab7 / 35 000	5.615	7.196
test2_tab8 / 40 000	6.492	8.332
test2_tab9 / 45 000	7.556	9.466
test2_tab10 / 50 000	7.907	10.270
test2_tab11 / 55 000	8.988	11.889
test2_tab12 / 60 000	9.766	12.934
test2_tab13 / 65 000	10.419	14.279

Obsah CD

- postgis/ - obsahuje celé rozšíření PostGIS s implementací SP-GiST
- postgresql/ - obsahuje databázový systém PostgreSQL, připravený pro práci s SP-GiST indexem v PostGIS
- patch/ - obsahuje diff soubory pro možnost vyzkoušení implementace na původních, neupravených verzích systému PostgreSQL a PostGIS
- skripty/ - obsahuje sql skripty pro vytváření tabulek, indexů a sql příkazy pro ověření chování indexů