Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Indexování vícerozměrných dat v relačních databázových systémech

Multidimensional Data Indexing in Relational Database Systems

2015 **Patrik Pjontek**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Patrik Pjontek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Indexování vícerozměrných dat v relačních databázových systémech
Multidimensional Data Indexing in Relational Database Systems

Zásady pro vypracování:

Kromě tradičních jednorozměrných indexů typu B-strom se v dnešních relačních databázových systémech využívají i vícerozměrné datové struktury jako je R-strom. Úkolem této práce je testování vícerozměrných datových struktur používaných v dnešních databázových systémech a jejich porovnání.

Bakalář musí v rámci bakalářské práce:
1. Nastudovat vícerozměrné indexy implementované v softwarovém rámci QuickDB vyvíjeném na Katedře informatiky.
2. Prostudovat vícerozměrné datové struktury existujících klient-server databázových systémů (Oracle, MSSQL, MySQL, PostgreSQL, AsterixDB,....) a porovnat jejich možnosti a výkon se softwarovým rámcem QuickDB.
3. Prostudovat vícerozměrné datové struktury existujících embedded databázových systémů (SQLite, Embedded MySQL, ...) a porovnat jejich možnosti a výkon se softwarovým rámcem QuickDB.
4. Vytvořit webové rozhraní pro vizualizaci testovaných databázových systémů.
5. Porovnat výkon vícerozměrných datovych struktur testovaných databázových systémů a softwarového rámce QuickDB.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Peter Chovanec**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave dňa: *7.* mája *2015*

.....
podpis študenta

Poďakovanie

Rád by som poďakoval **Ing. Petrovi Chovancovi** za odbornú pomoc a konzultácie pri vytváraní tejto bakalárskej práce.

Abstrakt

Účelom tejto práce je porovnanie možností a testovanie výkonu jednorozmerných (s viacerými atribútmi) a viacrozmerných databázových indexov v relačných databázových systémoch. Práca zahŕňa vytvorenie softwaru, ktorý vizualizuje výsledky tohto testovania vybraných klient - server, embedded databázových systémov a softwarového rámcu Radegast (predtým pomenovaný ako QuickDB), vyvíjaného na Katedre informatiky VŠB – Technickej univerzity v Ostrave. Práca obsahuje všeobecne zhrnutie dátových štruktúr používaných na indexovanie viacrozmerných dát v dnešných relačných databázových systémoch.

Kľúčové slová

viacrozmerný index, priestorový index, indexovanie, databáza. vyhľadávanie

Abstract

The purpose of this bachelor thesis is to compare options and test performance of one-dimensional (with more attributes) and multidimensional database indexes in relational database systems. Thesis involves the creation of software, which visualizes results of tests selected client - server, embedded database systems and software framework Radegast (earlier named as QuickDB), developed by Department of Computer Science at the VŠB – Technical University of Ostrava. The thesis contains general summary of data structures used to indexing multidimensional data in today's relational database systems.

Key words

multidimensional index, spatial index, indexing, database, searching

Obsah

Úvod

Index je dátová štruktúra, ktorá slúži k zlepšeniu výkonu pri získavaní dát z databáz. Vyhľadávanie vo veľkom množstve dát bez použitia indexu trvá spravidla nezanedbateľné množstvo času. V dnešných relačných databázových systémoch existuje mnoho metód efektívneho vyhľadávania, ktoré tieto indexy využívajú. Pri vhodnom výbere typu indexu a vhodnom výbere dát, na ktorých ma byť tento index vytvorený sa čas potrebný na vyhľadanie potrebných záznamov značne zredukuje. Nevýhodou indexov je potrebná réžia pre ich správnu funkciu a navýšenie potrebného miesta na disku, preto ich použitie nie je vždy prínosom. V úvode tejto práce budú priblížené najpoužívanejšie metódy, ktoré sú v indexoch používané. Sústredenie bude však na vyhľadávanie tzv. viacrozmerných dát. Pod pojmom viacrozmerné dáta rozumieme dáta, ktoré nie je možné vyhľadať na základe jednej vlastnosti (atribútu). Pri indexovaní takýchto dát je potrebné použiť viacrozmerné dátové štruktúry. V práci budú prezentované možnosti použitia týchto štruktúr v závislosti od jednotlivých databázových systémov. Bude vytvorený software, ktorý umožní nad vybranou kolekciou dát otestovať výkon rôznych typov indexov u rôznych databázových systémov. Tieto výsledky budú prezentované formou potrebného času pre určitý počet operácií. Testy sú vykonávané nad dátami, ktoré sú celočíselného dátového typu.

1 Databázový index

Databázový index je definovaný výberom tabuľky, jedného alebo viac stĺpcov nad ktorými je žiaduce zvýšenie výkonu získavania dát a typom indexu. Vytvorenie indexu spôsobí zvýšenie nároku databázového serveru na operačnú pamäť a diskový priestor v prípade ukladaniu vyhľadávacích štruktúr, avšak veľkosť samotných dát nie je ovplyvnená. Pri manipulovaní s dátami je potrebné priebežne aktualizovať aj štruktúru indexu. Inými slovami, ukladanie dát je kvôli týmto procesom spomalené, na druhej strane čítanie značne urýchlené.

Získavanie dát z tabuľky a stĺpcov, ktoré majú vytvorený index, sa od tabuľky bez indexu líši v tom, že dáta nie sú prehľadávané sekvenčne, ale podľa informácií uložených v pamäťovom priestore indexu je pristupované priamo k relevantným riadkom tabuľky. Dá sa povedať, že index funguje podobne ako register v knihe, kde namiesto postupného listovania sa najprv nahliadne do registra knihy a potom sa pristúpi k relevantným stranám knihy.

Databázové indexy delíme do rôznych druhov podľa toho, čo chceme pri prístupoch k primárnym dátam príslušnej databázovej tabuľky optimalizovať. Hlavné druhy indexov:

Primárny index

Primárny index (kľúč) je tvorený jedným alebo viacerými atribútmi, ktorých hodnoty jednoznačné identifikujú záznam v relácii. Kombinácia hodnôt týchto atribútov je v rámci tabuľky vždy jedinečná. Atribúty primárneho kľúča nemôžu mať nulovú hodnotu. Tento typ indexu sa v tabuľke môže vyskytnúť iba raz.

Sekundárny index

Sekundárny index je tvorený jedným alebo viacerými atribútmi, ktorých hodnoty na rozdiel od primárneho indexu nemusia byť v rámci tabuľky unikátne. Tento typ indexu sa v tabuľke môže vyskytovať opakovane.

Unikátny index

Unikátny index zabezpečuje podobne ako primárny index jednoznačnosť záznamu podľa unikátnej hodnoty kľúča, tzn. nie je možné vložiť do tabuľky záznam s rovnakou hodnotou atribútov na ktorých je unikátny index vytvorený. Pri pokuse o vloženie takéhoto záznamu nastane chyba. Rozdiel medzi unikátnym a primárnym indexom je, že unikátny index sa v tabuľke môže vyskytovať opakovane.

Architektúra indexov [1]:

Zhlukovaný

Táto architektúra spočíva v udržiavaní dát v tabuľke zoradených podľa ich kľúčovej hodnoty. Kľúčovými hodnotami sú myslené hodnoty atribútov na ktorých je index vytvorený. Nad tabuľkou môže existovať len jeden index tohto typu, pretože dáta môžu byť zoradené len jedným spôsobom.

Nezhlukovaný

Index tejto architektúry má štruktúru oddelenú od reálnych dát tabuľky. Tento index obsahuje kľúčové hodnoty indexovaných atribútov a ku každej tejto hodnote je navyše pridaný ukazovateľ:

- v prípade nezoradenej tabuľky, ukazovateľ na pozíciu záznamu v tabuľke s touto kľúčovou hodnotou
- v prípade zoradenej tabuľky, ukazovateľ na kľúčovú hodnotu v zhlukovanom indexe

2 Techniky indexovania

V súčasnosti, kedy sme z každej strany obklopovaní veľkým množstvom dát sú sústavne vyvíjané techniky pre čo najrýchlejšie vyhľadávanie a orientáciu v týchto dátach. Táto kapitola pokrýva techniky vyhľadávania viacrozmerných dát používané v dnešných databázových systémoch [2]. Existujú mnohé metódy rýchleho získavania dát, ktorých vhodnosť je závislá na mnoho faktoroch akými sú množstvo dát, dimenzia dát a pod.

Vlastnosti indexových štruktúr

Indexová štruktúra pomáha pri identifikovaní relevantných objektov v databáze. Existujú 4 žiaduce vlastnosti indexových štruktúr:

1. **Efektívny prístup:** Indexová štruktúra by mala zabezpečiť efektívne vyhľadávanie a dopytovanie. Musí to byť rýchlejšie než sekvenčný prechod databázou, inak je na zamyslenie ich použitie.

2. **Malá aktualizácia:** Ak je indexová štruktúra použitá pre dynamické databázy, malo by byť zabezpečené jednoduché vkladanie a upravovanie objektov. Ak je upravovanie náročné (napr. ak je potrebné index kompletne pre usporiadať), tak je vhodné index použiť len pre statické dáta, ktoré nebudú menené.

3. **Malá veľkosť:** Indexová štruktúra by mala mať čo najmenšiu veľkosť. Prednostne by mala mať vlastnú pamäť, takže ukazovatele na reálne dáta v databáze by mali byť z indexovej štruktúry ľahko dostupné.

4. **Korektnosť:** Posledná, ale pravdepodobne najviac dôležitá vlastnosť je korektnosť, tzn. objekty získavané za použitia indexovej štruktúry by mali odpovedať dotazu. Inými slovami, ak nie je použitá indexová štruktúra a databáza je prehľadávaná sekvenčne, nemal by byť rozdiel vo výsledku pri použití indexu.

2.1 B-Strom a B+-Strom

B-Strom navrhol R. Bayer v roku 1970 [2]. B-Strom je vyvážená hierarchická dátová štruktúra slúžiaca na organizovanie kľúčov v sekundárnej pamäti (na disku). Je optimalizáciou štruktúry vyváženého binárneho vyhľadávacieho stromu. B-Strom rádu *n* má nasledovné vlastnosti:

- strom je vyvážený, tzn. všetky listové uzly sú na rovnakej úrovni
- koreň obsahuje aspoň jeden kľúč
- všetky ostatné uzly obsahujú *n* až *2n* kľúčov
- uzol majúci *k* kľúčov obsahuje *k+1* ukazovateľov na potomkov
- listové uzly nemajú potomkov

Táto štruktúra nedovoľuje viacnásobne uchovávanie rovnakých kľúčových hodnôt, pretože ukazovatele na záznamy odpovedajúce kľúčovej hodnote sú uložené priamo s kľúčovou hodnotou.

B-Strom je zovšeobecnením perfektne vyvážené binárneho vyhľadávacieho stromu, kde namiesto dvoch potomkov môže mať vnútorný uzol viac potomkov založených na viacnásobných kľúčoch.

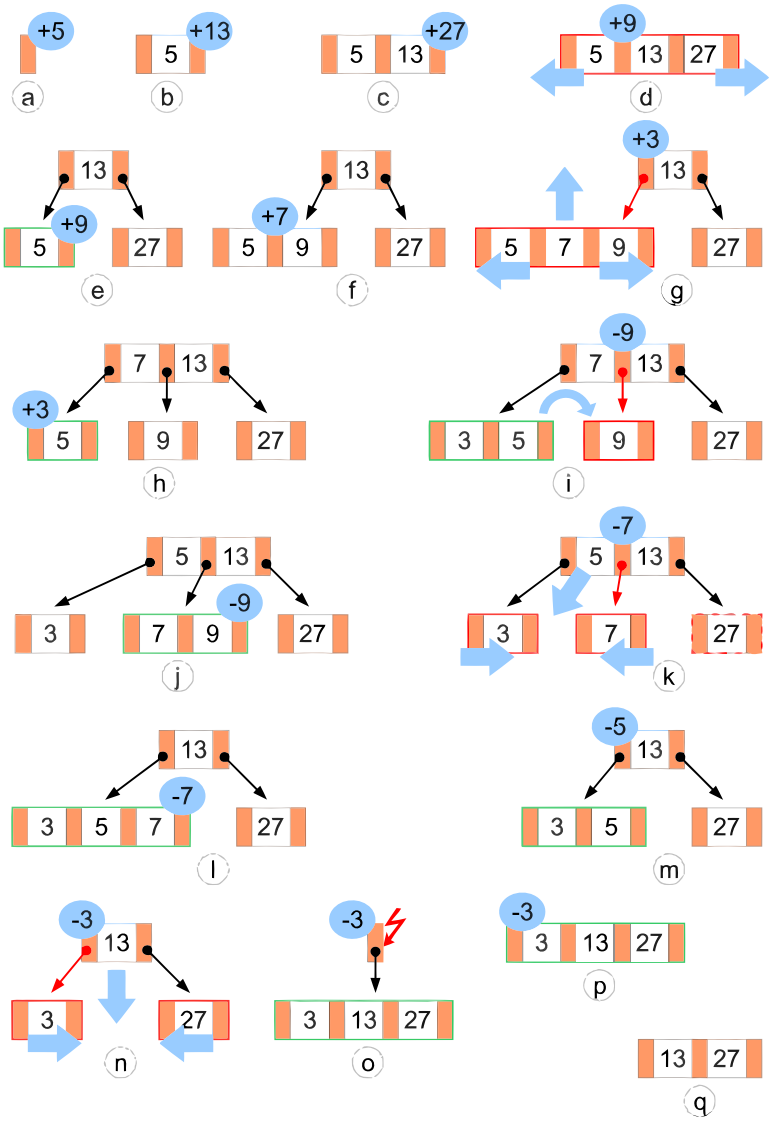
Najdôležitejšia varianta B-Stromu, ktorý je významne používaný na dáta uložené na disku je B+-Strom. Vnútorné uzly v B+-Strome uchovávajú len kľúče (nie ukazovatele na záznamy). Ukazovatele na záznamy odpovedajúce kľúču sú uchovávané v listových uzloch. Zvyšná štruktúra B+-Stromu je rovnaká ako štruktúra B-Stromu. Inými slova B+-strom je varianta B-Stromu, kde sú všetky dáta uchovávané v listoch a vnútorné uzly obsahujú len kľúče. Aj keď je možné dosiahnuť hľadaný kľúč v B-Strome rýchlejšie, tým že nie je potrebné prechádzať celý strom až k úrovni listov, je kvôli veľkému množstvu kľúčov uložených na úrovni listov vyhľadávanie v priemere rýchlejšie v B+-Strome, pretože počet úrovni stromu je menší. Taktiež vkladanie, mazanie a upravovanie je pre B-strom komplikovanejšie ako pre B+-strom.

Vyhľadávanie v B-strome prebieha nasledovným spôsobom [3]. Označme kľúče v uzle symbolmi k1,k2,...,k_m. V prípade, že hľadaná hodnota *x* sa nerovná žiadnej z kľúčov k_i, pokračujeme prehľadávaním uzlu potomka, ktorého určíme takto:

1. Ak $k_i < x < k_{i+1}$ pre $1 \leq i < m$, pokračujeme spracovaním uzlu nasledovníka p_i
2. Ak $k_m < x$, vyhľadávanie pokračuje na uzle p_m
3. Ak $x < k_1$, vyhľadávanie pokračuje na uzle p_0

Ak nasledovník vybraný týmto spôsobom neexistuje, položka s kľúčom *x* sa v strome nevyskytuje.

Na obr. 2.1 (prevzaté z http://commons.wikimedia.org/wiki/Category:B-Trees) je vidieť princíp vkladania a mazania záznamov v B-Strome.



Obrázok 2.1: Vkladanie a mazanie záznamov v B-Strome

2.2 B-Strom so zloženým kľúčom

Vyššie uvedená varianta B-Stromu nie je použiteľná pre indexovanie viacrozmerných dát, je však možné rozšíriť B-Strom zložením jednotlivých atribútov (kľúčov), ktoré budú pomyselné predstavovať jeden kľúč. Jedna sa o klasické zreťazenie, kde usporiadanie takéhoto stromu závisí od prvého atribútu v zloženom kľúči (v prípade duplicít od atribútu nasledujúceho), preto je vhodné dávať na prvé miesta v zloženom kľúči atribúty s najmenšou doménou. Doménou rozumieme množinu všetkých možných hodnôt atribútu. Na druhej strane s množstvom týchto duplicít rastie počet uzlov, ktoré je potrebné prehľadať, čo neprispieva efektivite vyhľadávania. Takéto usporiadanie je výhodne pri vyhľadávaní práve pomocou tohto prvého atribútu, nie však pre vyhľadávanie na základe ostatných atribútov. Inak povedané B-Strom so zloženým kľúčom je efektívny pri vyhľadávaní na základe všetkých atribútov a navyše v určenom poradí. Keď sa pozrieme na štruktúru dotazov v reálnych databázových systémoch, zistíme že takýchto ideálnych dotazov je málo. Preto nie je ani toto rozšírenie B-Stromu výhodné pre indexovanie viacrozmerných dát.

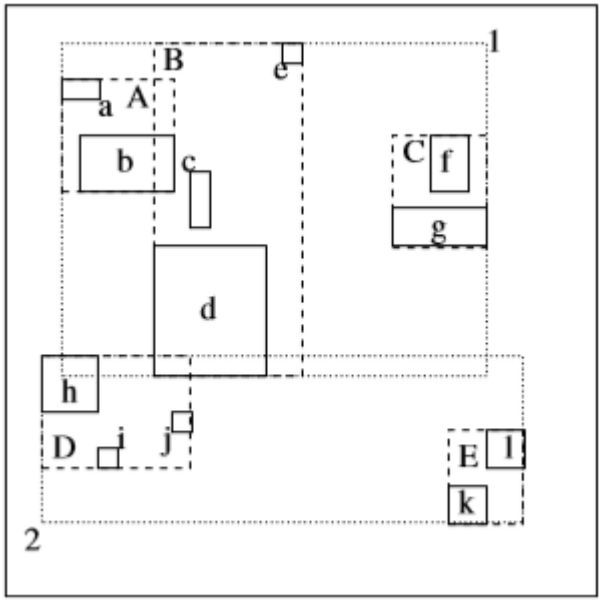
Možnosťou by bolo vytvorenie B-Stromu so zloženým kľúčom pre každé možné poradie atribútov v zloženým kľúči, to však pri narastajúcom počte dimenzií rapídne rastie (u 5 dimenzií by to bolo 120 indexov). Takýto nárast v počte indexov je neprípustný.

2.3 R-Strom

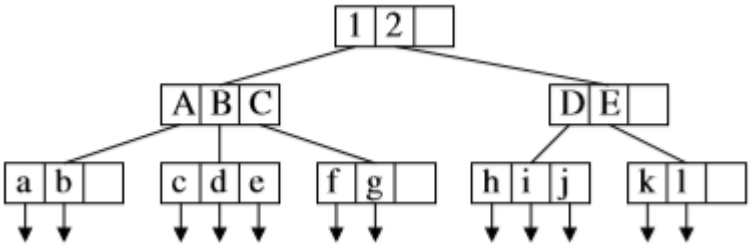
R-Strom je po B+-Strome najviac úspešná indexová štruktúra, ktorá bola kedy navrhnutá pre databázové aplikácie [2]. Bola predstavená A. Guttmanom v roku 1984. Je míľnikom v oblastí databázového indexovania a je silne zodpovedná za vývoj v tejto oblasti. Aj po uplynutí troch dekád od jej vzniku je stále používaná pre mnoho moderných úloh indexovania a je dobrým konkurentom mnohým špecializovaným štruktúram. Za uplynulé roky od jej vzniku inšpirovala celú rodinu indexových štruktúr. R-Strom má podobnú štruktúru ako B+-Strom, aj keď je určený pre viacnásobné dimenzie. Je univerzálny kvôli tomu, že umožňuje ukladať nie len objekty typu bod, ale tiež iné útvary (napr. čiary). Tieto objekty sú uzatvorené v minimálnom ohraničujúcom obdĺžniku, tzv. MBR (**m**inimum **b**ounding **r**ectangle). Objekt typu bod je tiež reprezentovaný ako MBR s nulovým objemom, kde najnižšie súradnice v každej dimenzií sú zároveň najvyššie.

2.3.1 Štruktúra

R-Strom je vyvážená štruktúra. Vnútorné uzly obsahujú regióny, ktoré pokrývajú všetky MBR svojich potomkov. Pre *d*-dimenzionálný priestor je tento obdĺžnik špecifikovaný *2*d* parametrami, dvomi v každej dimenzii na určenie minimálnych a maximálnych hodnôt danej dimenzie. Listové uzly obsahujú MBR indexovaných objektov (vždy objektov celých, tzn. objekty nie sú fragmentované). V R-Strome nie je striktne dodržané pravidlo o polovičnom naplnení uzlov v najhoršom prípade ako v B-Strome.



Obrázok 2.2: Dvojdimenzionálny priestor



Obrázok 2.3: R-Strom odpovedajúci obrázku 2.2

Obrázky 2.2 a 2.3 sú príkladom R-Stromu, kde je celý priestor rozdelený na dva MBR znázornené v obrázku 2.3 ako uzly 1 a 2. Tieto MBR neindexujú tzv. mŕtvy priestor bez dát. Uzol 1 obsahuje tri vnútorné uzly A, B a C. Týmto zostupným postupovaním sa dostaneme na úroveň listov. MBR reprezentujúce listové uzly obsahujú ukazovatele na objekty a na ich detailné informácie (ostatné atribúty).

2.3.2 Vyhľadávanie

Vstupom vyhľadávania je hľadaný obdĺžnik (MBR), kde vyhľadávanie je podobné vyhľadávaniu v B+-Strome. Hľadanie začína v koreňovom uzle. Každý vnútorný uzol obsahuje skupinu obdĺžnikov a ukazovateľov na nich a každý listový uzol obsahuje obdĺžniky objektov. Pre každý obdĺžnik v uzle je potrebné rozhodnúť či prekrýva hľadaný obdĺžnik. Ak áno je potrebne prehľadať relevantných potomkov tohto uzlu. Vyhľadávanie je ukončené keď sú prehľadané všetky obdĺžniky, ktoré prekrývajú hľadaný obdĺžnik. Keď je dosiahnutá úroveň listových uzlov, obdĺžniky sú znovu skontrolované na pokrytie obdĺžnika hľadaného a ich objekty (pokiaľ nejakú sú) sú vrátené ako výsledok dotazu.

2.3.3 Vkladanie

Pri vložení objektu je strom rekurzívne prejdený od koreňového uzlu. Pri každom kroku sú preskúmané všetky obdĺžniky v aktuálnom uzle a kandidáti na vloženie sú hľadaní na základe čo najmenšieho potrebného rozšírenia obdĺžnika pre vloženie nového záznamu. V prípade rovnosti potrebného rozšírenia je rozhodujúca výsledná plocha obdĺžnika. Táto výsledná plocha by mala byť opäť čo najmenšia. Snaha o čo najmenšie rozšírenie plochy je z dôvodu eliminácie mŕtveho priestoru, tzn. priestoru neobsahujúceho žiadne dáta. Hľadanie postupuje až do listového uzlu. Ak uzol nie je plný, objekt je vložený. Naopak ak je listový uzol plný (pretečie), musí byť rozdelený ešte pred vložením záznamu. Pridaním novovytvoreného uzlu na predošlú úroveň v strome, môže tento uzol znovu pretiecť a toto pretečenie môže byť pre propagované až do koreňového uzlu. Ak pretečie aj koreňový uzol, je vytvorený nový koreňový uzol a výška stromu je zvýšená. Delenie obdĺžnikov je dôležité z hľadiska efektivity vyhľadávania, preto existujú rôzne algoritmy (najznámejšie sú úplný, kvadratický a lineárny algoritmus), ktoré k deleniu pristupujú rôznymi spôsobmi.

2.3.4 Mazanie

Mazanie záznamov z R-Stromu môže vyžiadať aktualizáciu obdĺžnikov rodičovských uzlov. Keď nastane tzv. podtečenie (nedostatočné vyváženie stromu) je potrebné zmazanie podtečeného uzlu a znovu vloženie všetkých objektov spadajúcich pod tento uzol. Ak bude počas tohto procesu koreňový uzol obsahovať len jeden prvok, výška stromu bude zmenšená.

2.3.5 R*-Strom

Úspech konceptu R-Stromu viedol k vytvoreniu mnoho jeho variánt. Jedným z najúspešnejších nástupcov je R*-Strom [2]. Vylepšuje tri algoritmy pôvodného R-Stromu:

1. Pri vkladaní, v úrovni nad listovými uzlami je záznam vložený do toho uzlu, kde výsledný obdĺžnik bude minimalizovať presah s ostatnými obdĺžnikmi. Pri zhode je vybraný uzol s najmenším potrebným rozšírením plochy.

2. Pri delení, výsledná suma objemov dvoch obdĺžnikov vytvorená po delení by mala byť čo najmenšia. Zároveň je snaha o minimalizáciu obvodu obdĺžnikov. Presah dvoch obdĺžnikov by mal byť čo najmenší.

3. *Násilne znovuvloženie* je zmena oproti R-Stromu v zmene postupu pri pretečení uzlu. Pri pretečení uzlu, namiesto okamžitého delenia uzlu sú znovu vložené vybrané MBR pretečeného uzlu. Kandidáti na znovu vloženie sú vybraní podľa vzdialenosti stredu týchto MBR od stredu MBR pretečeného. Najlepšie výsledky boli dosiahnuté pri znovu vložení 30% najvzdialenejších, prípadne 30 % najbližších MBR. Po vybraní vhodných MBR sú tieto obdĺžniky z pretečeného uzlu odobrané a následne znovu vložené. Opätovné vloženie je vykonané len raz, z dôvodu vyhnutia sa možnej nekonečnej slučke. Keď nenastane v strome zmena ani po vykonaní tohto znovu vloženia, pristúpi sa na metódu delenia uzlov.

2.3.6 R+-Strom

Pri tejto variante majú uzly R+-Stromu [2] nulový presah aj za cenu rozdelenia jedného objektu do viacerých MBR. Tým je zabezpečené, že strom je prehľadávaný práve jednou cestou. Problémom je, že pri snahe o splnenie podmienky nulového presahu rastie počet regiónov. Operácie vkladania a mazania sú v tomto strome v porovnaní s R-Stromom omnoho komplikovanejšie, to je dôvod prečo nie je táto varianta až tak populárna.

2.3.7 Hilbertov R-Strom

Hilbertov R-Strom [18] bol navrhnutý ako mix B+-Stromu a R-Stromu. Je to v podstate B+-Strom s geometrickými objektmi charakterizovanými vypočítanou hodnotou (tzv. Hilbert value, v texte používanou už len ako "hodnota") na základe ich ťažiska. Štruktúra je založená na Hilbertovej plochu - vyplňujúcej krivke [2]. Táto krivka veľmi dobre zachováva priestorovú blízkosť priestorových objektov. Položky vnútorných uzlov sú rozšírené pridaním najväčšej Hilbertovej hodnoty potomkov týchto uzlov, tzn. uzly obsahujú trojicu údajov:

- MBR, ktorý pokrýva všetkých potomkov daného uzlu
- najväčšiu Hilbertovu hodnotu podstromu daného uzlu

- ukazovateľ na ďalšiu úroveň

Položky v listových uzloch sú rovnaké ako v R-Strome, R+-Strome a R*-Strome, tzn. obsahujú MBR a identifikátor daného objektu. Spôsob vyhľadávania je rovnaký ako v R-Strome a R*-Strome. Vyhľadávanie začne v koreňovom uzle, zostúpi na uzly, ktorých MBR pretína hľadaný MBR. Keď je dosiahnutá úroveň listov, všetky relevantné objekty sú vrátené ako výsledok. Vkladanie do Hilbertovho R-Stromu sa však líši od ostatných variánt. Pri vložení nového objektu je na základe ťažiska jeho MBR vypočítaná Hilbertová hodnota. Táto hodnota je následne použitá pri procese vloženia. Získaná hodnota objektu je porovnaná s každou hodnotou alternatívnych uzlov a je vybraný ten uzol, ktorý má najmenšiu a zároveň väčšiu hodnotu ako je hodnota vkladaného objektu.

Dôležitou vlastnosťou Hilbertovho R-Stromu, ktorá chýba u ostatných variánt, je usporiadanie uzlov na každej úrovni stromu. Toto zoradenie dovoľuje definovať "súrodencov" každého uzlu. Existencia súrodencov dovoľuje oddialiť okamžité delenie uzlu, v prípade jeho pretečenia. Namiesto okamžitého rozdelenia uzlu po jeho pretečení, je uskutočnený pokus o vloženie uzlov do súrodeneckých uzlov. Delenie nastane až vtedy, keď pretečú aj všetky súrodenecké uzly. Táto vlastnosť Hilbertovho R-Stromu značne pomáha pri efektívnom využití pamäte a predchádza nepotrebnej operácii delenia.

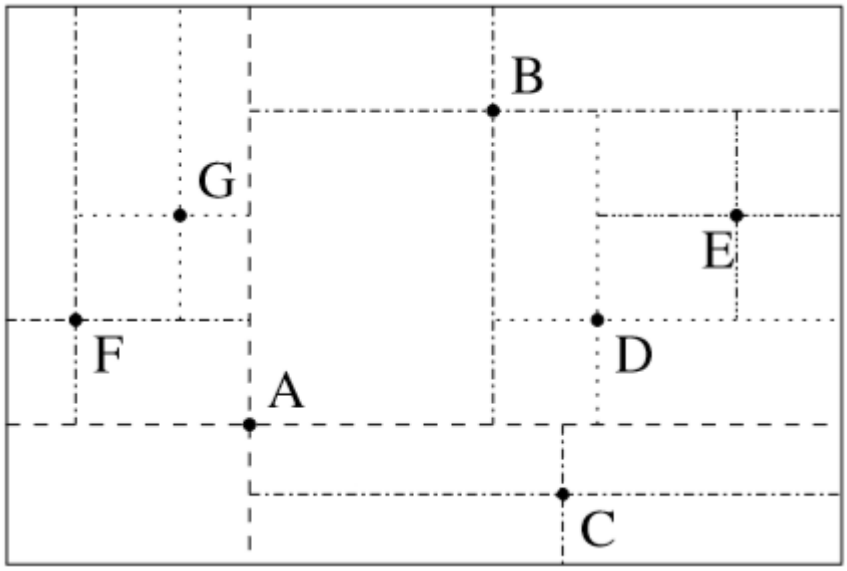
Je evidentné, že Hilbertov R-Strom pracuje ako B+-Strom pri vkladaní a ako R-Strom pri dopytovaní. Podľa testovaní autora, bol Hilbertov R-Strom označený ako najlepšia dynamická verzia R-Stromu od čias jeho publikovania [18]. Avšak s rastúcou dimenziou priestoru nie je adekvátne zachovaná priestorová blízkosť objektov daná Hilbertovov krivkou, čo vedie k nárastu prekrývania MBR vo vnútorných uzloch stromu.

2.4 Kvadratický strom

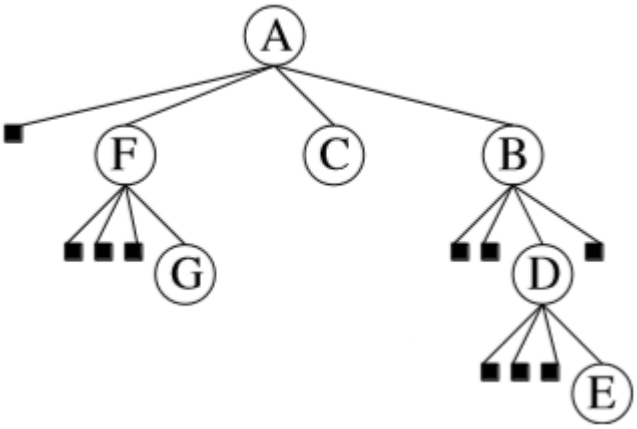
Táto štruktúra je zovšeobecnením binárneho vyhľadávacieho stromu v 2 dimenziách [2]. Každý uzol delí rozsah jeho hodnôt na dve polovice. Prvú polovicu tvoria hodnoty menšie alebo rovné hodnote kľúča a druhú polovicu hodnoty väčšia ako je hodnota kľúča. Ak sa však jedná o 2 dimenzie (x, y) je priestor delený na 4 časti nasledovným spôsobom:

1. ($\leq x, \leq y$)
2. ($\leq x, > y$)
3. ($> x, \leq y$)
4. ($> x, > y$)

Každý uzol ma 4 potomkov, ktorí reprezentujú 4 kvadranty získané rozdelením podľa bodu, ktorý predstavuje začiatok uzlu.



Obrázok 2.4: Body v dvojdimenzionálnom priestore



Obrázok 2.5: Príslušný kvadratický strom pre obr. 2.4

Obr. 2.5 predstavuje kvadratický strom, ktorý odpovedá dvojdimenzionálnym bodom na obr. 2.4. Neexistujúci potomkovia sú v tomto prípade označený ako malé čierne štvorce.

2.4.1 **Vkladanie**

Predpokladajme, že body na obr. 2.5 boli vkladané v abecednom poradí. Prvý bod A je koreňom. Nasledujúci bod B sa stal potomkom bodu A a bol vložený do 4. kvadrantu, pretože jeho súradnice na oboch osách sú väčšie ako súradnice bodu A. Podobne bod C bol vložený ako

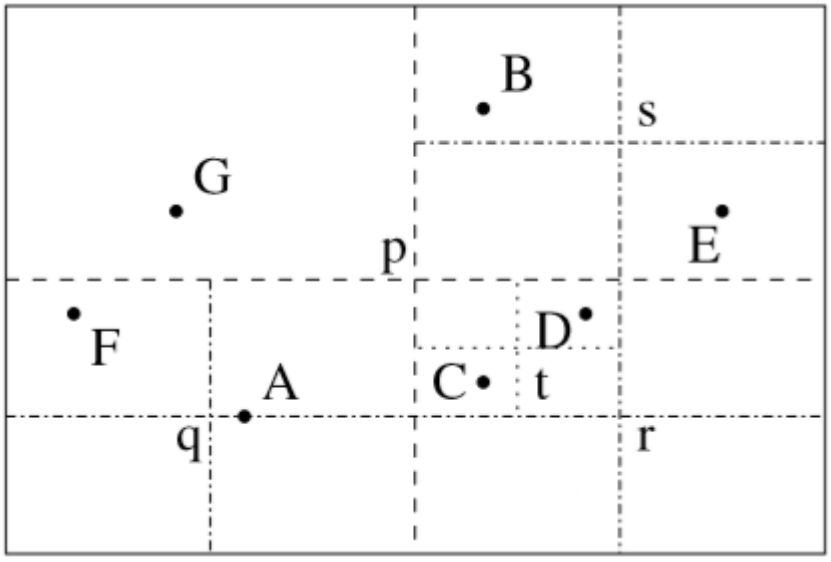
potomok bodu A. Avšak pri vkladaní bodu D sa tento bod nachádza v rovnakom kvadrante ako bod B, takže bod D sa stal potomkom bodu B. Rovnakým spôsobom boli do stromu vložené ostatné body. Na obr. 2.5 je výsledná podoba stromu po vložení všetkých bodov. Aj keď je proces vkladania pomerne jednoduchý, výsledná podoba stromu je silne závislá na poradí vkladaných bodov. Podobne ako binárny vyhľadávací strom je kvadratický strom nevyvážený, pretože nie je snaha o jeho priebežne vyváženie počas procesu vkladania.

2.4.2 **Vyhľadávanie**

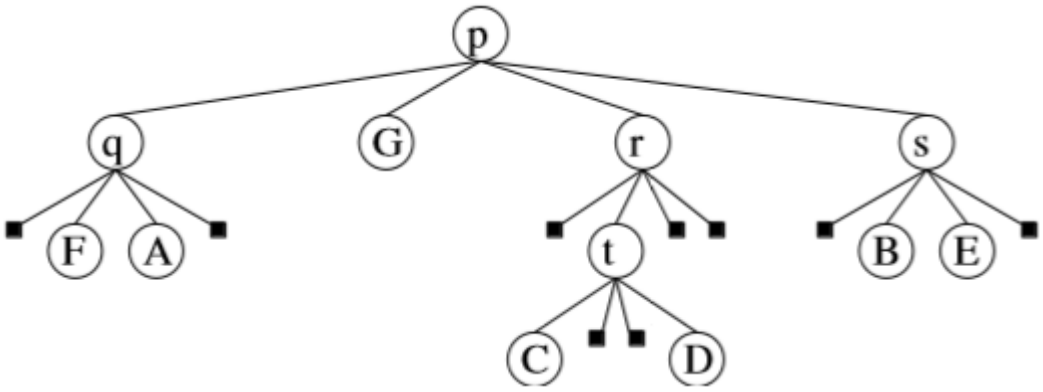
Vyhľadávanie je podobne procesu vkladania. Na začiatku vyhľadávania je na základe súradníc koreňa určené v ktorom kvadrante hľadaný bod leží. Následne je prehľadaný určený kvadrant. Aj keď je priemerný čas hľadania rovný O (log *n*) pre dáta s veľkosťou *n* objektov, najhorší možný čas aj pre vyvážený kvadratický strom je O ($\sqrt{n}$).

2.4.3 **Varianty a rozšírenia**

Najpoužívanejšia verzia kvadratického stromu je verzia popísaná vyššie, ktorá je známa tiež ako bodový kvadratický strom, pretože práve body delia dvojdimenzionálny priestor do kvadrantov. Varianta tohto stromu je regionálny kvadratický strom. Tento strom má sklon byť viac vyvážený v porovnaní s bodovou variantou.



Obrázok 2.6: Body v dvojdimenzionálnom priestore



Obrázok 2.7: Príslušný regionálny kvadratický strom

Na obr. 2.7 je znázornený regionálny kvadratický strom pre body na obr. 2.6. Body označené malými písmena v skutočnosti nie sú uložené. Tieto body sú vytvorené umelo a predstavujú stredy jednotlivých regiónov. Všetky skutočné dáta sú uložené ako listy.

2.5 Bitmapový index

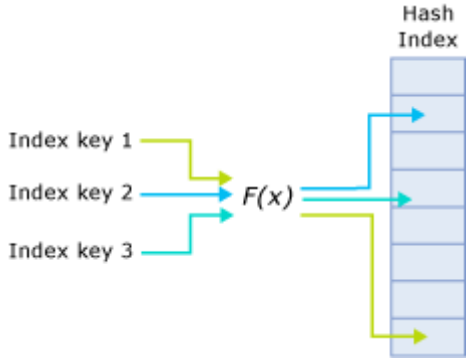
Relačné databázové systémy používajú bitmapové indexy pre rozsahové vyhľadávanie atribútov s malou doménou. Doména atribútu predstavuje rozsah možných hodnôt daného atribútu. V bitmapovom indexe je vytvorený reťazec bitov pre každý atribút. Bitová dĺžka tohto reťazca je rovná veľkosti domény atribútu. Každá možná hodnota atribútu odpovedá jednému bitu v reťazci. Pre každý záznam v tabuľke je vytvorený reťazec pre všetky atribúty bitmapového indexu a bit na príslušnej pozícii je nastavený na hodnotu "1", v závislosti od hodnoty atribútu. Následný dotaz prejde sekvenčne týmto bitmapovým indexom.

Výhodou týchto indexov je kompaktná štruktúra, ktorá ich robí rýchlymi pri čítaní a spracovaní.

Nevýhodou je náročná réžia pre ich udržiavanie. Keďže vloženie novej hodnoty vyžaduje kompletné obnovenie indexu, je tento typ indexu vhodný prevažne pre databázy, kde nie sú často vkladané nové záznamy.

2.6 Hash index

Hash index obsahuje kolekciu "košov" organizovanú ako pole [19]. Hašovacia funkcia mapuje kľúče do odpovedajúcich košov v hash indexe. Obr. 2.8 znázorňuje mapovanie troch kľúčov do troch rôznych košov v hash indexe. Pre ilustráciu je hašovacia funkcia pomenovaná ako *F(x)*.



Obrázok 2.8: Mapovanie kľúčov do "košov"

Hašovacia funkcia má nasledujúce vlastnosti:

- je deterministická, tzn. rovnaké kľúčové hodnoty sú vždy mapované do rovnakých košov v hash indexe
- zložené kľúče môžu byť mapované do rovnakého koša
- hašovacia funkcia je vyvážená, tzn. mapovanie kľúčov do košov dodržiava určité pravidlo mapovania (napr. Poisson distribution)

Ak sú dva kľúče mapované do rovnakého koša hash indexu, nastáva kolízia. Veľký počet takýchto kolízií môže mať vplyv na výkon operácií čítania.

Každý "kôš" v poli ukazuje na svoj prvý záznam. Každý záznam v koši ukazuje na nasledujúci záznam, čoho výsledkom je reťazec záznamov v každom koši hash indexu.

3 Indexovanie viacrozmerných dát v klient - server databázových systémoch

Táto kapitola prezentuje spôsoby a možnosti indexovania viacrozmerných dát vo vybraných klient - server databázových systémoch. Každý z týchto databázových systémov podporuje indexovanie dát B-Stromom (prípadne jeho variantou) pre jednotlivé stĺpce (kapitola 2.1) a so zloženým kľúčom (kapitola 2.2).

3.1 Microsoft SQL Server

SQL Server 2008 a neskoršie verzie podporujú priestorové dáta delené do dvoch kategórii [4]:

- geometrický dátový typ (**geometry**) - napr. body, čiary a polygóny
- geografický dátový typ (**geography**)- reprezentujú geografické objekty na povrchu Zeme, napr. kus plochy

Zoznam objektov, na ktorých je možné použiť priestorový index [20]: Point, MultiPoint, LineString, CircularString. MultiLineString, CompoundCurve, Polygon, CurvePolygon, MultiPolygon a GeometryCollection.

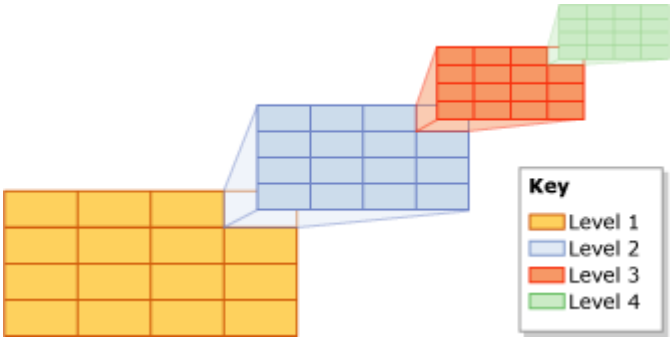
Priestorový index vytvorený na geografickom objekte mapuje geografické dáta do dvojdimenzionálneho priestoru. Priestorový index je definovaný len na stĺpci tabuľky, ktorý obsahuje priestorové dáta. Každý priestorový index sa odkazuje na konečný priestor. Napr. index vytvorený na stĺpci reprezentujúcom geometrický útvar odkazuje na používateľom definovanú obdĺžnikovú plochu, tzv. ohraničujúci box.

SQL Server 2008 dovoľuje vytvorenie priestorového indexu len nad dátami, ktoré sú geometrického alebo geografického dátového typu (max. dimenzia je 4). Na jednom stĺpci je možné vytvoriť viacero priestorových indexov. Táto možnosť je výhodná z hľadiska použitia rôznych parametrov u týchto indexov. Priestorový index môže byť vytvorený len na tabuľke, ktorá obsahuje primárny kľúč. Maximálny počet stĺpcov primárneho kľúča je 15. Maximálna veľkosť záznamov indexu je 895 bajtov. Je možné vytvoriť až 249 indexov na akomkoľvek stĺpci priestorového dátového typu v tabuľke.

3.1.1 Hierarchia mriežky

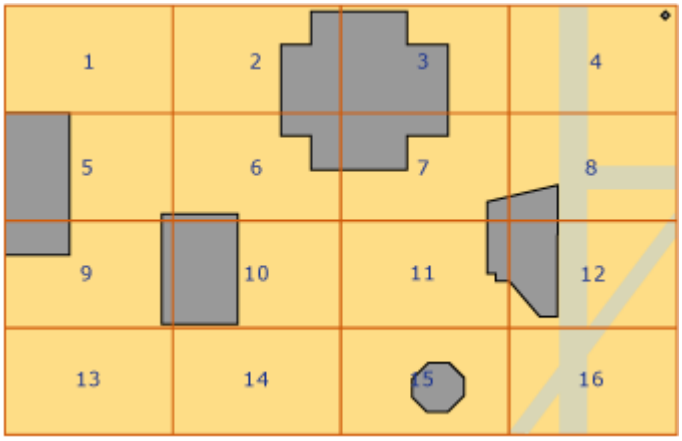
V SQL Serveri 2008 sú priestorové indexy vytvorené použitím B-Stromov, čo znamená že indexy musia reprezentovať dvojdimenzionálne priestorové dáta v lineárnom poradí B-Stromov. Pred načítaním dát do priestorového indexu vytvorí SQL Server hierarchickú dekompozíciu priestoru. Pri vytvorení indexu je indexovaný priestor rozložený do

štvorúrovňovej mriežkovej hierarchie. Tieto úrovne sú označené ako úroveň 1 (najvyššia úroveň) až úroveň 4. Každá nasledujúca úroveň mriežky rozkladá úroveň nad ňou, takže každá bunka (časť mriežky) vo vrchnej úrovni odpovedá mriežke na nižšej úrovni. Mriežka na danej úrovni má rovnaký počet buniek na oboch osách (napr. 4x4, 8x8) a všetky bunky majú rovnakú veľkosť. Obr. 3.1 znázorňuje dekompozíciu pravej vrchnej bunky na všetkých úrovniach, kde každá úroveň má veľkosť 4x4. V skutočnosti sú týmto spôsobom rozložené všetky bunky, tzn. dekompozícia priestoru do 4 úrovni s veľkosťou každej úrovne 4x4 vytvorí 65 536 buniek.



Obrázok 3.1: Hierarchia mriežky v SQL Serveri

Bunky v mriežke sú očíslované lineárne použitím varianty Hilbertovej plochu - vyplňujúcej krivky [2]. Pre ilustráciu je na obr. 3.2 použité jednoduché číslovanie namiesto číslovania, ktoré je vytvorené Hilbertovou krivkou. Na obr. 3.2 je vidieť niekoľko polygónov a čiar znázorňujúcich budovy a ulice, ktoré sú dekomponované do mriežky o veľkosti 4x4. Bunky sú očíslované číslami 1 až 16 so začiatkom vo vrchnej ľavej bunke.



Obrázok 3.2: Ukážka jednej úrovne mriežky

Hustota mriežky je daná počtom buniek, kde čím väčší počet buniek, tým väčšia hustota. SQL Server dovoľuje používateľovi pri vytvorení indexu definovať rôznu hustotu mriežky na rôznych úrovniach. Na výber sú 3 možnosti: nízka hustota (4x4), stredná hustota (8x8) a vysoká hustota (16x16). S**tredná** hustota mriežky je automaticky prednastavená na všetkých úrovniach**.**

3.1.2 **Mozaikovanie**

Po dekompozícii indexovaného priestoru do hierarchie mriežky, prečíta priestorový index riadok po riadku dáta z indexovaného stĺpca a vloží hodnoty do vytvorenej mriežky. Tento proces sa nazýva mozaikovanie. Výstupom procesu je uloženie buniek do priestorového indexu. Odkazovaním sa na uložené bunky umožňuje priestorový index umiestniť objekt blízky ostatným objektov v indexovanom stĺpci.

Pre minimalizovanie dotknutých buniek (objektom), proces mozaikovania aplikuje niekoľko pravidiel:

- pravidlo pokrytia - ak objekt kompletne pokrýva bunku, táto bunka nie je ďalej mozaikovaná a je uložená do indexu
- pravidlo počtu buniek na objekt - toto pravidlo limituje počet buniek pokrytých jedným objektom na všetkých úrovniach s výnimkou najvyššej úrovne (prednastavená hodnota tohto parametru je 16 s možnosťou zmeny používateľom)
- pravidlo najhlbšej bunky - toto pravidlo určuje uloženie do indexu len najhlbších buniek v mriežke, ktoré boli mozaikované pre daný objekt a tým je zároveň zabezpečené najlepšie priblíženie objektu

Správanie priestorového indexu sčasti závisí na schéme mozaikovania. Existujú dve schémy mozaikovania, pre geometrické dátové typy a pre geografické dátové typy. Tieto schémy sú nastavené automatický podľa dátového typu.

Ohraničujúci box určuje konečný priestor pre priestorový index. Tento box je v tvare obdĺžnika. Pri vytvorení priestorového indexu na geometrických dátach je potrebné nastaviť veľkosť tohto boxu. Ohraničujúci box predstavujú štyri hodnoty, pre každý roh obdĺžnika jedna hodnota. Priestor nepokrytý týmto ohraničujúcim boxom je v indexe považovaný za jednu bunku s číselnou hodnotou 0. Po definovaní ohraničujúceho boxu dekomponuje priestorový index tento priestor do mriežky, kde najvyššia úroveň mriežky pokrýva celý ohraničujúci box. K dosiahnutiu najlepšej efektivity priestorového indexu je potrebné definovať ohraničujúci box tak, aby pokrýval všetky alebo väčšinu objektov indexovaného stĺpca tabuľky.

3.2 Oracle

Každý viacrozmerný index môže byť štruktúry R-Strom (automaticky), kvadratický strom alebo oboje [5]. Každý z týchto indexov je vhodnejší v rôznych situáciách. Je možné mať na jednom stĺpci vytvorené obidva indexy a pri dopytovaní vybrať, ktorý index zabezpečí vyhľadanie. Pri rozhodovaní, ktorú štruktúru zvoliť, je dobré poznať výhody a nevýhody oboch štruktúr:

R-Strom

- horšie laditeľný (používa definovaný ohraničujúci box)
- vytvorenie indexu a jeho údržba je jednoduchšia
- zaberá menej miesta
- vhodný pri častých dotazoch typu "najbližší sused" (nájde najbližšie útvary daného útvaru) a "útvary v dosahu" (vracia útvary v danom rozsahu)
- pri veľkom množstve úprav na danom stĺpci nie je najlepšou voľbou
- dovoľuje indexovať až 4 dimenzie

Kvadratický strom

- ľahšie laditeľný (pomocou nastaviteľných parametrov)
- údržba indexu je komplikovanejšia
- zaberá väčšia množstvo miesta
- pri veľkom množstve úprav na danom stĺpci je vhodnou voľbou
- dovoľuje indexovať len 2 dimenzie

Oracle dovoľuje vytvoriť priestorový index na dátach tohto typu: Points and point clusters, Line strings, *n* - point polygons, Arc line strings, Arc polygons, Compound polygons, Compound line strings, Circles, Optimized rectangles.

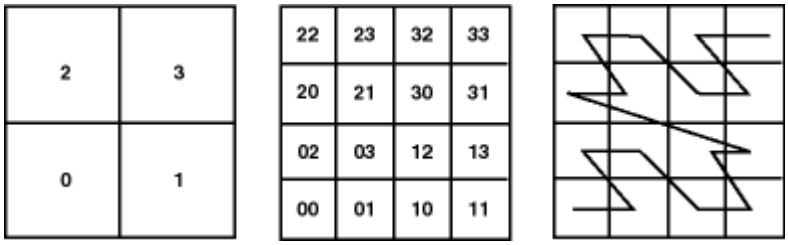
3.2.1 R-Strom

Štruktúra indexu tohto typu je detailnejšie popísaná v kapitole 2.3. Oracle poskytuje niekoľko funkcií a procedúr spojené s udržiavaním správnej funkcie indexu, napr. či je potrebná obnova štruktúry indexu a ohodnotenie správnej funkcie indexu.

3.2.2 Kvadratický strom

Štruktúra indexu tohto typu je detailnejšie popísaná v kapitole 2.4. Podobne ako u SQL Serveru aj tu je využívaný proces mozaikovania**,** kedy je každý objekt správnym spôsobom vložený do stromu. Plocha kde sú postupne vložené všetky objekty je znázornená obdĺžnikom a

rozdelená do častí, tzv. *dlaždíc*. Uzly sú delené do 4 kvadrantov až kým nie je splnená podmienka pre ukončenie tohto procesu. Používateľom je možné definovať pevne daný počet dlaždíc, do ktorých bude celkový priestor rozdelený alebo maximálny počet dlaždíc, ktorý nesmie byť prekročený. Dlaždice na danej úrovni môžu byť zoradené podľa poradia, ktoré je definované plochu - vyplňujúcou krivkou (obr. 3.3). Na zistenie ideálnej veľkosti dlaždíc, ktoré je veľmi dôležité pre správnu funkciu indexu poskytuje Oracle funkcie, ktoré používateľom pomôžu so správnym nastavením parametrov pre danú kolekciu dát.



Obrázok 3.3: Zoradenie dlaždíc v kvadratickom strome

3.3 MySQL

MySQL poskytuje priestorový index len pre tabuľky typu MyISAM [6, 7, 8, 9]. Stĺpce na ktorých je vytvorený priestorový index musia obsahovať nenulové hodnoty. Pri vytvorení priestorového indexu je vytvorený R-Strom s kvadratickým delením. Štruktúra R-Stromu a jeho delenie priestoru do MBR je detailnejšie popísané v kapitole 2.3. Úložiska (MySQL ponúka rôzne úložiska), ktoré poskytujú nepriestorové indexy pre priestorové dáta, vytvárajú na týchto dátach B-Strom. Index typu B-Strom je v týchto prípadoch vhodnejší pre vyhľadávanie konkrétnych hodnôt v indexovanom stĺpci namiesto rozsahového vyhľadávania. Pre vyhľadávanie v stĺpcoch na ktorých je vytvorený priestorový index je vhodné používať funkcie MySQL GIS.

MySQL podporuje vytvorenie priestorového indexu na týchto dátových typoch [21]: Geometry, Point, Linestring, Polygon, Multipoint, Multilinestring, Multipolygon a Geometrycollection.

3.4 PostgreSQL

PostgreSQL ponúka pre indexovanie viacrozmerných dát indexy typu GiST, SP-GiST a GIN [10, 11, 12]. Každý z týchto indexov nie je index len jedného druhu, ale skôr infraštruktúra

v ktorej môžu byť implementované rôzne stratégie indexovania (Kvadratické stromy, R-Stromy, K-D stromy a pod.). Výber stratégie je určený podľa typu vyhľadávania (vyhľadávanie konkrétnych hodnôt, rozsahové vyhľadávanie a pod.).

PostGIS je rozšírenie o priestorové databázy pre PostgreSQL. Zahŕňa niekoľko dátových typov, funkcií, indexov a rôznych ďalších rozšírení. PostGIS ponúka mnoho vlastností, ktoré sa zriedka vyskytujú v ostatných priestorových databázach iných firiem.

Podporované dátové typy, na ktorých je možné vytvoriť priestorový index [22]: Point, Line, Lseg, Box, Path, Polygon, Circle a niektoré špeciálne dátové typy (napr. Cube).

3.4.1 **GiST**

GiST (Generalized Search Tree) je vyvážená stromovo-štruktúrovaná prístupová metóda, ktorá slúži ako základná šablóna pre implementáciu indexov. GiST dovoľuje implementáciu mnohých indexových štruktúr (napr. B-Strom, R-Strom a pod) pre určené dátové typy. Jednou z výhod GiST je, že dovoľuje expertom vývoj vlastných dátových typov spolu s vhodnými prístupovými metódami. Samotný GiST zahŕňa už niekoľko vytvorených indexových štruktúr. Pre indexovanie geometrických dátových typov obsahuje štruktúru R-Strom. Viacrozmerné dáta (vyhľadávané na základe viacerých atribútov) je možné za použitia GiST reprezentovať pomocou multidimenzionálnej kocky. Dátový typ tejto multidimenzionálnej kocky je v GiST nazvaný *cube*. Mimo tohto dátového typu obsahuje GiST niekoľko ďalších dátových typov, ktoré sú vhodné pre uloženie rôznych druhov dát. Takisto obsahuje k týmto dátovým typom indexy, ktoré sú špecializované práve na dané dáta a z toho dôvodu je mnohokrát ich efektivita nezanedbateľná. Pre využívanie všetkých výhod tejto štruktúry je potrebné doinštalovať rozšírenia, ktoré PostgreSQL vo svojej základnej verzii niekedy neobsahuje.

3.4.2 **Cube**

Ako už bolo spomenuté dátový typ kocka reprezentuje multidimenzionálnu kocku. Do tohto dátového typu je možné uložiť bod v n - dimenzionálnom priestore. Hodnoty, resp. parametre tejto kocky sú uložené ako 64-bitové desatinné čísla, tzn. čísla s viac ako 16 platnými číslicami sú skrátené. Na dátach tohto typu je vytvorený index typu R-Strom (detailnejší popis v kapitole 2.3), ktorý sa zdá byť pre indexovanie n - dimenzionálnych bodov v priestore najefektívnejší. Dátový typ kocka je prednastavene obmedzený na maximálny počet dimenzií 100. Na rozdiel od predchádzajúcich klient - server databázových systémov je vďaka tomuto dátovému typu možné vytvoriť index typu R-Strom aj na dátach, ktoré nie sú geometrické, prípadne geografické útvary.

3.4.3 SP-GiST

SP-GiST je skratka pre priestor - deliaci (**s**pace - **p**artitioning) GiST [26]. SP-GiST podporuje deliace vyhľadávacie stromy, čo uľahčuje vývoj širokej škály rôznych nevyvážených dátových štruktúr (kvadratické stromy, k-d stromy a pod.).

Vlastností štruktúry priestor - deliacich stromov, ktoré ich odlišujú od ostatných stromových štruktúr sú:

- rekurzívna dekompozícia priestoru; vždy je vytvorený pevne daný počet častí, ktoré nemusia byť rovnakej veľkosti
- predstavujú nevyvážené stromy
- sú obmedzené určeným limitom vetvenia, napr. kvadratické stromy delia priestor na 4 časti
- obsahujú dva druhy uzlov, sú to vnútorné a listové uzly

Podobne ako GiST, aj SP-GiST dovoľuje vývoj vlastných dátových typov s príslušnými prístupovými metódami.

3.4.4 GIN

GIN (Generalized Inverted Index) [25] bol navrhnutý pre situácie, kde indexované dáta predstavujú skupinu položiek a dotazy nad týmito dátami vyhľadávajú hodnoty týchto položiek v danej skupine. Napríklad, skupinu položiek môžu predstavovať dokumenty a vyhľadané majú byť práve tie dokumenty, ktoré obsahujú špecifické slová.

Podobne ako GiST a SP-GiST, aj GIN dovoľuje vytvorenie vlastných dátových typov s príslušnými prístupovými metódami.

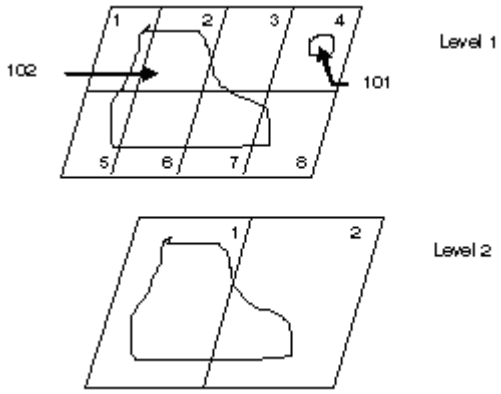
3.5 DB2

Spatial Extender (priestorové rozšírenie) je rozšírenie pre databázu DB2, ktoré obsahuje priestorové dátové typy a indexy pre indexovanie dát tohto typu [13, 14]. Toto rozšírenie nie je vždy v základnej výbave DB2, preto ho je niekedy potrebné doinštalovať. Medzi priestorové dátové typy, ktoré je dovolené indexovať priestorovým indexom patria: Point, LineString, Polygon, MultiPoint, MultiLineString a MultiPolygon. Pre indexovanie priestorových dát používa DB2 tzv. *grid index*, ktorý začlení priestorové dáta do štvorcovej mriežky. Čiary a polygóny môžu pretínať viac než len jednu bunku mriežky, kým body môžu pretínať maximálne jednu bunku. Pri definovaní veľkostí buniek mriežky je potrebné urobiť kompromis, kde bude minimalizovaný počet buniek pretínaných objektom a zároveň minimalizovaný počet objektov v jednej bunke. K efektívnemu mapovaniu objektov s rôznymi veľkosťami je možné

špecifikovať až tri veľkosti buniek (tri vrstvy). Tieto bunky sú následne indexované použitím B-Stromu.

Určenie veľkostí buniek nemusí byť celkom jasné. Vo všeobecnosti platí, že pri určovaní veľkosti bunky pre polygóny a čiary by veľkosť bunky mala byť o niečo väčšia ako priemerná veľkosť týchto objektov. Pre body platí, že veľkosť bunky by mala byť 1/10 šírky typického dotazu na tieto objekty a mala by byť použitá len prvá úroveň. DB2 ponúka dva nástroje pre pomoc so správnym definovaním veľkosti bunky, sú to *gseidx index advisor* a *Java index advisor*.

Úrovne mriežky sa stupňujú s veľkosťou bunky, tzn. druhá úroveň mriežky musí mať bunku väčšej veľkosti ako prvá a rovnako tretia úroveň musí mať bunku väčšej veľkosti než druhá úroveň. Prvá úroveň mriežky je povinná, druhú a tretiu úroveň je možné zablokovať nastavením hodnoty 0.



Obrázok 3.4: Dve úrovne mriežky

Obr 3.4 (prevzaté z http://jiangsuimage.com/arcgis/Manager/Help/index.htm#geodatabases/creating-95168347.htm) znázorňuje dve úrovne mriežky a dva objekty v tejto mriežke (101 a 102).

DB2 konštruuje priestorový index nasledovne:

1. Každý objekt je obsiahnutý v mriežke začínajúc na prvej úrovni.
2. Ak objekt pretína menej ako štyri bunky mriežky na prvej úrovni, sú identifikátor objektu a identifikátory buniek pretínaných týmto objektom uložené do indexu a nasleduje indexovanie ďalšieho objektu.
3. Ak objekt pretína viac než štyri bunky je postúpené na druhú úroveň mriežky, kde by vzhľadom na väčšiu veľkosť buniek mal objekt pretínať menej buniek. Ak nie je povolená druhá úroveň mriežky, sú identifikátor objektu a identifikátory buniek pretínaných týmto objektom uložené do indexu. V prípade, že objekt pretína viac buniek aj na druhej úrovni, je postúpené na

tretiu úroveň kde sú bez ohľadu na počet pretínajúcich buniek objektom uložené informácie o objekte a pretínaných bunkách ako v predchádzajúcich prípadoch.

Pri vyhľadávaní s použitím indexu sú pri dotaze vybrané (na základe súradníc) častí mriežky, ktoré pretínajú rozsah hľadania. Následne index získa zoznam objektov, ktoré tieto vybrané častí mriežky obsahujú a sú z neho vyradené tie objekty, ktorých obdĺžnikové ohraničenie nepretína hľadaný rozsah. Pri poslednom kroku porovnávania súradníc objektu s hľadaným rozsahom sú priamo porovnávané záznamy objektov v tabuľke. Počet týchto porovnaní, tzn. prístupov do tabuľky je však značne zredukovaný, kvôli vybraným relevantným častiam mriežky.

4 Indexovanie viacrozmerných dát v embedded databázových systémoch

Táto kapitola prezentuje spôsoby a možnosti indexovania viacrozmerných dát vo vybraných embedded databázových systémoch. Každý z týchto databázových systémov podporuje indexovanie dát B-Stromom (prípadne jeho variantou) pre jednotlivé stĺpce (kapitola 2.1) a so zloženým kľúčom (kapitola 2.2).

4.1 SQLite

SQLite používa pre indexovanie viacrozmerných dát štruktúru R*-Strom (detailnejšie v kapitole 2.3) [15]. Táto štruktúra je vylepšením štruktúry R-Strom. Modul podporujúci indexovanie pomocou R-Stromu je v SQLite prednastavene zablokovaný, preto je potrebné pre jeho využívanie modul povoliť. Index typu R*-Strom je implementovaný ako virtuálna tabuľka. Virtuálna tabuľka predstavuje z pohľadu SQL príkazov reálnu tabuľka s pridanými obmedzeniami (napr. nemožno vytvoriť trigger, dodatočné indexy a pod). Každý tento index predstavuje virtuálnu tabuľku s nepárnym počtom stĺpcov v rozmedzí 3 až 11. Prvý stĺpec je vždy primárny kľúč reprezentovaný 64-bitovým znamienkovým celočíselným dátovým typom. Ostatné stĺpce predstavujú páry, jeden pár pre každú dimenziu obsahujúci minimálne a maximálne hodnoty objektu pre danú dimenziu. Tieto hodnoty môžu byť 32-bitové desatinné, prípadne celočíselné dátové typy v závislosti od danej situácie. Pri pracovaní výhradne s celočíselnými súradnicami je možnosť zadať túto podmienku už pri vytvorení virtuálnej tabuľky. Dvojdimenzionálny R*-Strom by teda obsahoval 5 stĺpcov (1. stĺpec pre primárny kľúč a 2. až 5. stĺpec pre minimálne a maximálne hodnoty dvoch dimenzií). Z obmedzenia na 11 stĺpcov je jednoduché zistiť, že R*-Strom v SQLite podporuje maximálne 5 dimenzií. Na rozdiel od bežných tabuliek v SQLite môže virtuálna tabuľka implementujúca R*-Strom uchovávať len 32-bitové desatinné a celočíselne dátové typy. V prípade, že súradnice objektu nemôžu byť dostatočne reprezentované 32-bitovým desatinným číslom, sú dolné súradnice zaokrúhlené nadol a horné súradnice nahor, tzn. ohraničujúci box daného objektu bude väčší, ale nikdy nie menší. Vonkajšie zaokrúhlenie súradníc môže spôsobiť, že dotazu bude odpovedať viac objektov, nikdy však nebudú stratené objekty, ktoré majú reálne dotazu odpovedať.

Jediné informácie o objekte, ktoré index typu R*-Strom uchováva je jeho identifikátor a ohraničujúci box. Dodatočné informácie o objekte musia byť uložené v oddelenej tabuľke a prepojené s objektom v R*-Strome pomocou primárneho kľúča (identifikátora objektu).

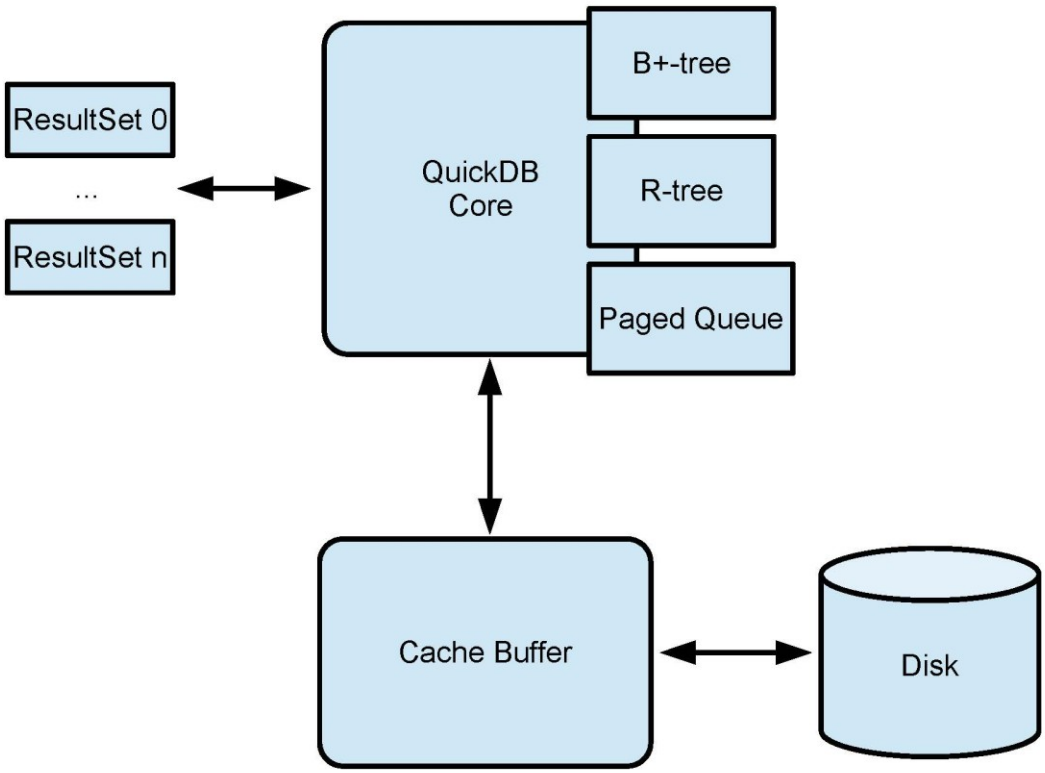
Použitím štandardných SQL príkazov v podmienke sa je možné dopýtať na všetky objekty, ktoré sú v určenom vzťahu (napr. pretína, prekrýva a pod.) s určeným ohraničujúcim boxom. Použitím špeciálnych operátorov sa je možné dopýtať na objekty v R*-Strome, ktoré sú v tomto vzťahu nie len s určeným obdĺžnikom, ale s rôznym regiónom alebo tvarom.

4.2 SQL Server Compact

Hoci SQL Server Compact podporuje geometrické a geografické dátové typy, nepodporuje indexovanie týchto dát priestorovým indexom [16]. Viacrozmerné dáta, ktoré môžu byť reprezentované číselnými hodnotami bez primárneho kľúča, však môžu byť indexované nezhlukovaným indexom. Vytvorením nezhlukovaného indexu sú dáta v indexe logicky usporiadané a z toho dôvodu je vo väčšine prípadov vyhľadávanie efektívnejšie.

5 Radegast Database

Radegast (predtým pomenovaný ako QuickDB) je prototyp DBMS (systém riadenia bázy dát) vyvíjaný skupinou Database Research Group (http://db.cs.vsb.cz) [17]. Zahŕňa širokú škálu dátových štruktúr a mnohokrát poskytuje vyšší výkon v rôznych prípadoch v porovnaní s dnešnými DBMS. Poskytuje rôzne možnosti ako sú indexy rôznych štruktúr, tabuľku typu halda a pod. Je podporovaný na oboch (32-bit a 64-bit) platformách.



Obrázok 5.1: Architektúra Radegastu (predtým pomenovaný QuickDB)

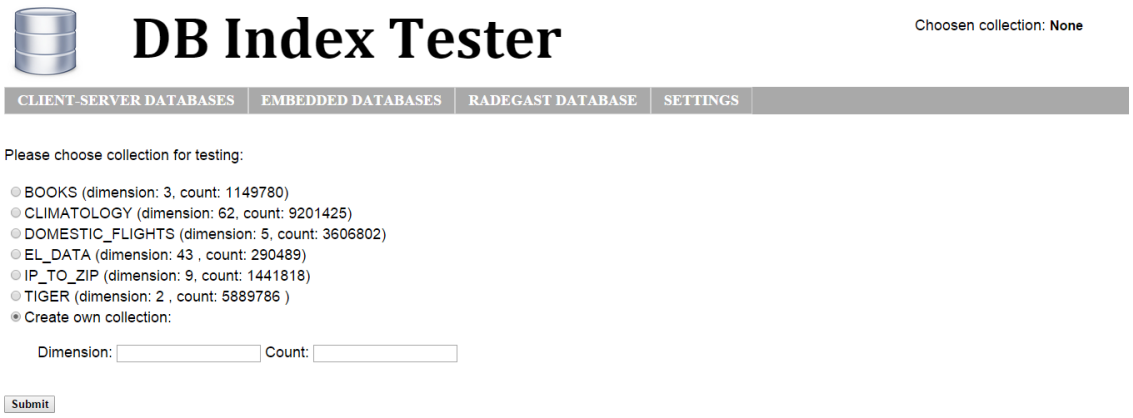
Na obr. 5.1 (QuickDB je predchádzajúcim pomenovaním Radegastu) môžeme vidieť architektúru Radegastu. Vyrovnávacia pamäť uchováva stránky dátovej štruktúry pre zabránenie prístupu na disk v prípade, že je táto stránka potrebná. Veľkosť stránky v pamäti je približne o 20% väčšia ako jej veľkosť na disku, tzn. v prípade 8 kB stránky je jej veľkosť v pamäti 9 830 B. Keď operácia dátovej štruktúry vráti výsledok, napr. pri rozsahovom vyhľadaní, je tento výsledok uložený v ResultSet a následne vrátený používateľovi. Keď používateľ uzavrie ResultSet, je vrátený Radegastu.

V Radegaste existujú implementácie B-Stromu (kapitola 2.1), R-Stromu (kapitola 2.3) a stránkový front, ktoré využívajú jadro Radegastu. Cieľom Radegastu je poskytnúť maximálny

výkon namiesto implementácie bohatej funkcionality ako napr. podpory metód uložených v databáze a pod.

6 Testovacie webové rozhranie

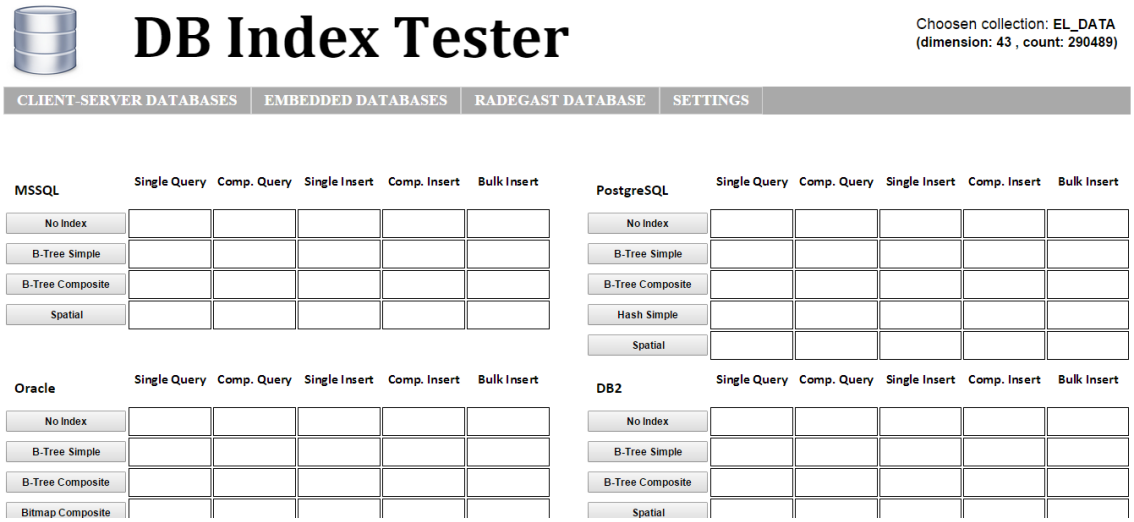
Okrem výsledkov testovania je výstupom práce aj vytvorená webová aplikácia, ktorá slúži na opakované testovanie indexových štruktúr v jednotlivých databázových systémoch na rôznych kolekciách. Obr. 6.1 predstavuje úvodnú stránku, ktorá zobrazí zoznam kolekcií v používateľom definovanej zložke. Základné menu aplikácie obsahuje nasledujúce položky: klient - server databázové systémy, embedded databázové systémy, databázový systém Radegast a položku Nastavenia, ktorá je totožná s úvodnou stránkou (obr. 6.1). Používateľ si zo zoznamu vyberie kolekciu na testovanie a stlačí tlačidlo "Submit", ktoré ho presmeruje na stránku indexov klient - server databázových systémov (obr. 6.2). V prípade, že požadovanému testovaniu nevyhovuje žiadna z vytvorených kolekcií, je možné vytvoriť vlastnú kolekciu a testovanie realizovať na nej. Zadaním dimenzie a počtu záznamov kolekcie, sa v priečinku aplikácie vytvorí kolekcia, kde sú všetky hodnoty reprezentované celými 32-bitovými kladnými číslami.



Obrázok 6.1: Úvodná stránka aplikácie

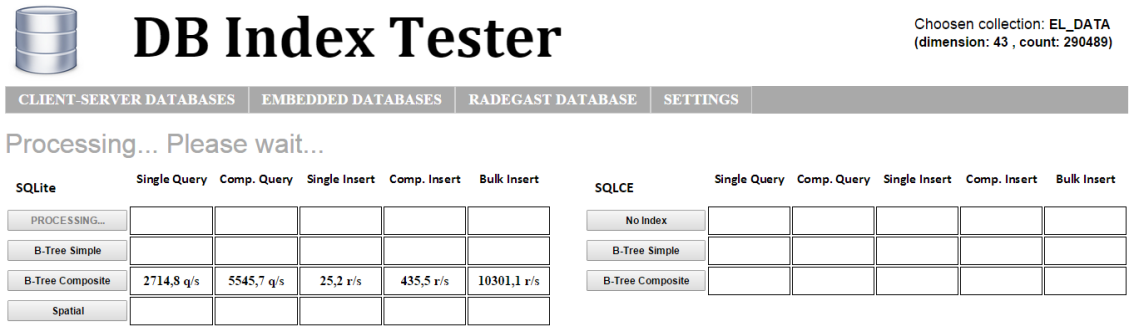
V pravom hornom rohu je zobrazená zvolená kolekcia. Po stlačení tlačidla "Submit" je sekvenčne prejdená celá vybraná kolekcia, pre určenie minimálnych a maximálnych hodnôt každej dimenzie. Tieto hodnoty sú potrebné pre vytvorenie priestorových indexov. Rovnako je overené, či kolekcia obsahuje len celé čísla. Navyše je náhodne vybraný určený počet záznamov, ktoré sú počas testovania vyhľadávané a vkladané.

Obr. 6.2 obsahuje tlačidla, reprezentujúce každý testovaný index v jednotlivých databázových systémoch a tabuľku reprezentujúcu výsledky testovania.



Obrázok 6.2: Testovanie klient - server databázových systémov

Na obr. 6.3 je znázornený prebiehajúci test na tabuľke bez implementovaného indexu v databázovom systéme SQLite. Zároveň obrázok obsahuje výsledok testu zloženého B-Stromu v piatich kategóriách. Záložka "Radegast Database" je štruktúrou podobná obr. 6.2 a 6.3. V položke "Settings" je možné zmeniť testovanú kolekciu. V prípade chyby počas testovania, je táto chýba pre propagovaná k používateľovi formou chybovej hlášky.



Obrázok 6.3: Príklad výsledku testovania a prebiehajúceho testu

7 Testovanie

Pri testovaní výkonu indexových dátových štruktúr bolo testované, tzv. bodové vyhľadávanie (použitie operátora "=") a vkladanie záznamov do tabuľky s už vytvoreným indexom, prípadne pre porovnanie rozdielu vo výkone s použitím a bez použitia indexu aj do tabuľky, ktorá nemá implementovaný index. Netestované indexy sú v tabuľke označené znakom "-". Kvôli potrebe plne naplnenej tabuľky pre správne testovanie vyhľadávania boli pre rýchle naplnenie tabuľky dátami použité rôzne možnosti hromadného naplnenia (tzv. bulk insert) v závislosti od jednotlivých databázových systémov. Rýchlosť (potrebný čas pre vloženie určeného počtu záznamov) hromadného vloženia záznamov je zahrnutá vo výsledkoch testovania, je ale potrebné podotknúť, že v niektorých prípadoch existujú efektívnejšie spôsoby hromadného naplnenia tabuľky, preto nie je vhodné považovať tieto výsledky za najlepšie možné. Zložený dotaz/vloženie v tabuľke výsledkov predstavuje vyhľadanie/vloženie 100 záznamov jedným príkazom zo strany aplikácie, na rozdiel od jednoduchého dotazu/vyhľadávania, kde je jedným príkazom vyhľadaný/vložený práve jeden záznam. Vyhľadávanie je realizované spôsobom dopytovania sa na konkrétnu hodnotu (operátor "=") každého atribútu v tabuľke, kde platí, že hľadaný záznam tabuľka reálne obsahuje.

Detaily implementácie jednotlivých indexov v jednotlivých databázových systémoch:

- bez indexu - tabuľka bez implementovaného indexu
- jednoduchý B-Strom - vytvorenie indexu typu B-Strom (prípadne jeho varianty v závislosti od databázového systému) na každý stĺpec tabuľky zvlášť, tzn. počet vytvorených indexov sa rovná počtu stĺpcov tabuľky
- zložený B-Strom - vytvorenie indexov typu B-Strom (prípadne jeho varianty v závislosti od databázového systému) na skupine stĺpcov, kde maximálny počet stĺpcov v jednom indexe je prednastavene obmedzený v každom databázovom systéme nasledovne: MSSQL - 16, Oracle - 32, MySQL - 16, PostgreSQL - 32, DB2 - 63, SQLite - 2000, SQLCE - 16
- zložený Bitmap - vytvorenie indexu typu bitmap na skupine stĺpcov, kde maximálny počet stĺpcov v jednom indexe je prednastavene obmedzený v každom databázovom systéme, rovnako ako je tomu u zloženého B-Stromu
- jednoduchý Hash - vytvorenie indexu typu hash na každý stĺpec tabuľky zvlášť, tzn. počet vytvorených indexov sa rovná počtu stĺpcov tabuľky
- zložený Hash - vytvorenie indexu typu hash na skupine stĺpcov, kde maximálny počet stĺpcov v jednom indexe je prednastavene obmedzený v každom databázovom systéme, rovnako ako je tomu u zloženého B-Stromu
- priestorový - vytvorenie priestorového indexu v rámci ponúkaných možností v jednotlivých databázových systémoch nasledovne:
    - MSSQL - vytvorenie grid indexu s definovaným ohraničujúcim boxom skutočného rozsahu hodnôt jednotlivých dimenzií danej kolekcie a

použitie prednastavenej hustoty mriežky SQL Serverom (stredná hustota)
- Oracle - vytvorenie R-Stromu s definovaným ohraničujúcim boxom skutočného rozsahu jednotlivých dimenzií a s nastavenou toleranciou na hodnotu 0,99
- MySQL - vytvorenie indexu typu R-Strom bez manuálne nastavených parametrov
- PostgreSQL - vytvorenie indexu typu GiST (v tomto prípade implementujúci R-Strom) na dátovom type Cube (taktiež súčasťou GiST), do ktorého sú uložené záznamy kolekcie
- DB2 - vytvorenie grid indexu s použitím len prvej úrovne mriežky a manuálne nastavenou hodnotou veľkosti bunky na hodnotu 1
- SQLite - vytvorenie virtuálnej tabuľky s použitím R*-Stromu
- SQLCE - nepodporuje priestorový index
- Radegast - vytvorenie R-Stromu

7.1 Kolekcia TIGER

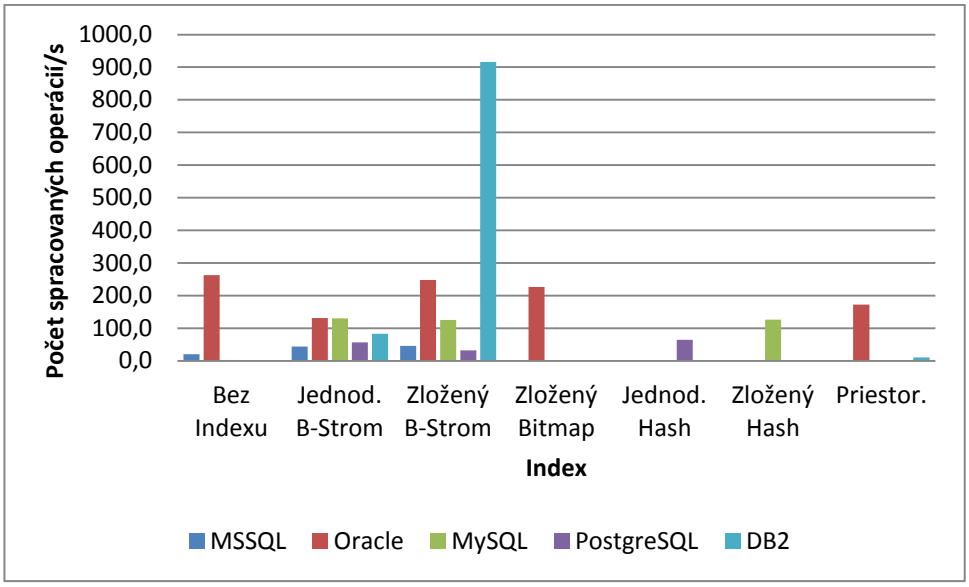
Kolekcia TIGER predstavuje tabuľku s 5 889 786 riadkami (záznamami) a 2 stĺpcami (dimenzia). Hodnoty v stĺpcoch sú celočíselného dátového typu. Táto kolekcia obsahuje 4,285,056 a 4,285,991 odlišných hodnôt v 1. a 2. stĺpci tabuľky.

Príslušné grafy zobrazujú miestami skreslené výsledky, kvôli veľkým výsledkovým rozdielom v jednotlivých databázových systémoch.

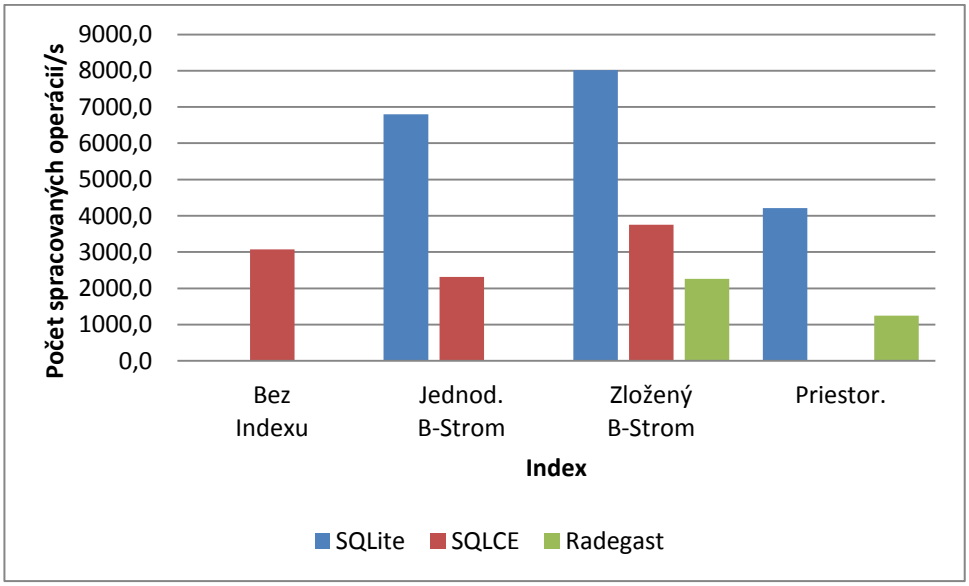
7.1.1 Jednoduché vyhľadávanie

Tabuľka 7.1: Jednoduché vyhľadávanie v kolekcii TIGER [počet spracovaných operácií/s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	20,3	43,8	46,3	-	-	-	0,1
Oracle	262,7	131,1	248,6	226,6	-	-	326,3
MySQL	2,9	130,1	126,0	-	-	127,1	1,1
PostgreSQL	1,4	56,9	32,2	-	64,9	-	1,4
DB2	2,5	83,8	916,0	-	-	-	11,3
SQLite	1,6	6801,6	8011,6	-	-	-	4213,8
SQLCE	3072,5	2309,4	3755,9	-	-	-	-
Radegast	-	-	2260,9	-	-	-	1244,9



Obrázok 7.1: Jednoduché vyhľadávanie v klient - server databázových systémoch (kolekcia TIGER)



Obrázok 7.2: Jednoduché vyhľadávanie v embedded databázových systémoch a v Radegast databáze (kolekcia TIGER)

V tabuľke 7.1 a obr. 7.1 a 7.2 je vidieť, že najpočetnejšie zastúpenie a najlepšie výsledky dosahuje index typu zložený B-Strom takmer vo všetkých databázových systémoch. Dôvodom môže byť ideálna forma dotazu pre tento typ indexu.

7.1.2 Zložené vyhľadávanie

Tabuľka 7.2: Zložené vyhľadávanie v kolekcii TIGER [počet spracovaných operácií /s]

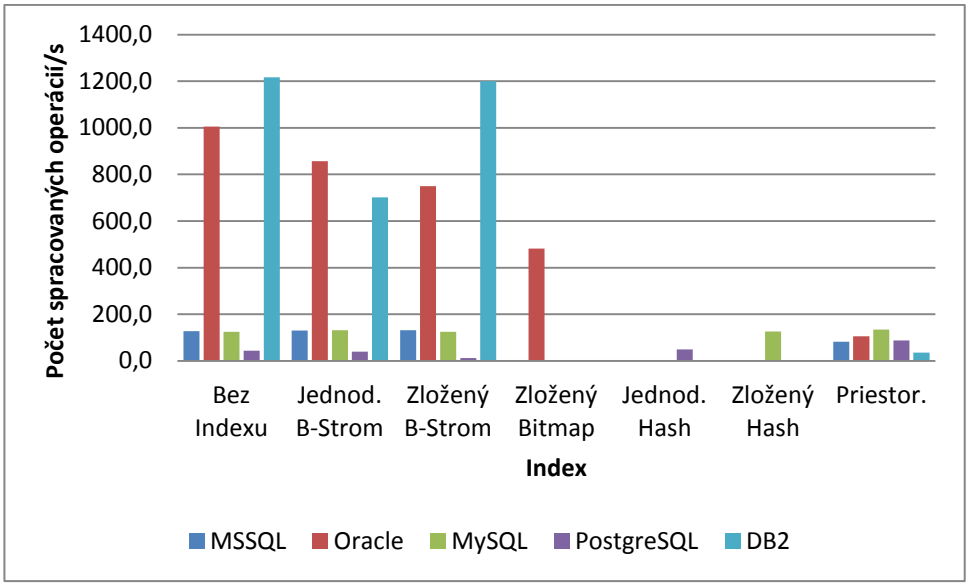
	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	35,5	978,4	2161,2	-	-	-	0,1
Oracle	303,6	142,4	286,5	262,0	-	-	193,8
MySQL	3,0	4682,2	5523,2	-	-	5520,5	1,1
PostgreSQL	1,5	2013,2	6185,5	-	7814,0	-	1,5
DB2	2,5	1860,8	2010,3	-	-	-	15,6
SQLite	5394,7	83042,7	84868,0	-	-	-	56844,0
SQLCE	10851,4	8106,2	7373,4	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.2 je vidieť rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Dôvodom je rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami mnohonásobne lepšie výsledky, pravdepodobne z dôvodu ich hlavných výhod.

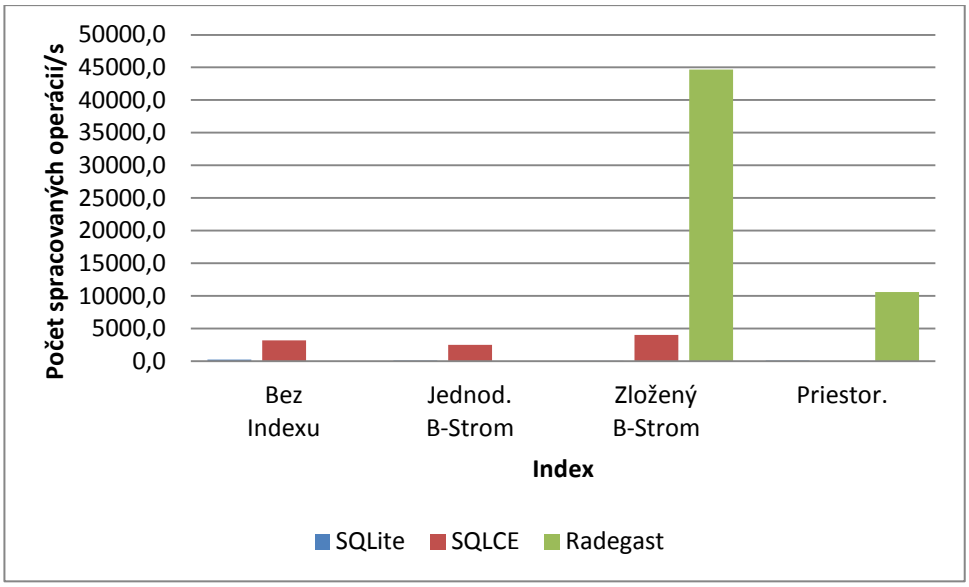
7.1.3 Jednoduché vkladanie

Tabuľka 7.3: Jednoduché vkladanie záznamov kolekcie TIGER [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	127,6	130,6	132,2	-	-	-	81,9
Oracle	1005,0	857,5	750,0	482,1	-	-	105,8
MySQL	125,8	132,0	125,2	-	-	126,3	134,4
PostgreSQL	43,8	39,4	12,4	-	49,7	-	88,1
DB2	1217,4	701,7	1199,7	-	-	-	35,5
SQLite	228,1	99,7	44,4	-	-	-	126,4
SQLCE	3176,5	2512,4	4049,3	-	-	-	-
Radegast	-	-	44661,9	-	-	-	10621,9



Obrázok 7.3: Jednoduché vkladanie v klient - server databázových systémoch (kolekcia TIGER)



Obrázok 7.4: Jednoduché vkladanie v embedded databázových systémoch a v Radegast databáze (kolekcia TIGER)

V tabuľke 7.3 a obr. 7.3 a 7.4 je možné vidieť, že operácia vkladania je najefektívnejšia (aj keď nie s veľkým rozdielom) v tabuľke bez implementovaného indexu takmer vo všetkých prípadoch. Dôvodom je nepotrebná réžia pri obnove štruktúry indexu. Databázový systém Radegast dosahuje mnohonásobne lepšie výsledky aj pri vkladaní do tabuľky s indexom.

7.1.4 Zložené vkladanie

Tabuľka 7.4: Zložené vkladanie záznamov kolekcie TIGER [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	7379,5	6761,7	7095,2	-	-	-	751,2
Oracle	3857,4	3197,5	3319,9	3283,5	-	-	128,0
MySQL	12294,1	11776,2	11476,4	-	-	11546,4	11282,5
PostgreSQL	3172,0	3146,8	1590,6	-	5867,3	-	2417,5
DB2	32096,5	27965,8	29859,7	-	-	-	33,4
SQLite	35732,2	3738,8	2136,4	-	-	-	8337,7
SQLCE	25463,4	20279,0	17369,9	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.4 je vidieť rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Opäť je dôvodom rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami mnohonásobne lepšie výsledky, pravdepodobne z dôvodu ich hlavných výhod.

7.1.5 Hromadné vkladanie

Tabuľka 7.5: Hromadné vkladanie záznamov kolekcie TIGER [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	377699,6	50165,6	113867,8	-	-	-	6108,0
Oracle	151535,2	31662,0	49969,8	48602,2	-	-	3570,1
MySQL	1459523,8	504196,6	744025,8	-	-	744948,5	77919,8
PostgreSQL	60310,9	33750,2	55250,2	-	17694,3	-	13564,4
DB2	43097,3	20075,5	25262,3	-	-	-	10719,8
SQLite	27533,2	14051,0	17621,2	-	-	-	12331,1
SQLCE	27135,9	16586,7	18186,9	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.5 je vidieť výsledky hromadného vkladania záznamov v jednotlivých systémoch. Pri tomto testovaní neboli vždy použité najlepšie spôsoby ponúkané jednotlivými systémami, preto nie je namieste ich porovnávať. Je však vidieť pokles efektivity pri hromadnom vkladaní do tabuliek s indexom, z dôvodu už spomenutej potrebnej réžie indexov.

7.2 Kolekcia DOMESTIC_FLIGHTS

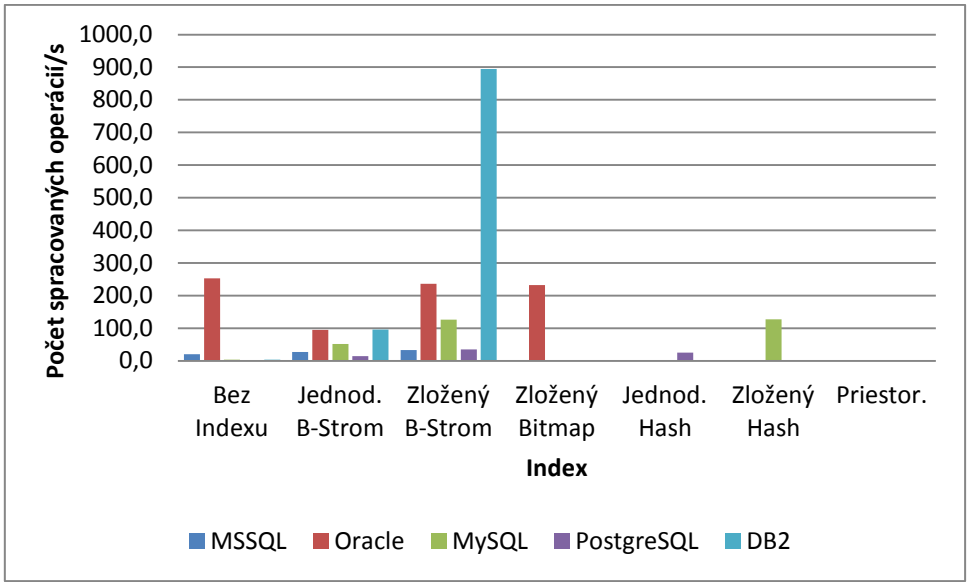
Kolekcia DOMESTIC_FLIGHTS predstavuje tabuľku s 3 606 802 riadkami (záznamami) a 5 stĺpcami (dimenzia). Hodnoty v stĺpcoch sú celočíselného dátového typu. V tabuľkách, ktoré prezentujú výsledky testovania je vidieť, že priestorový index pre dimenziu 5 je podporovaný len v PostgreSQL, SQLite a Radegast.

Príslušné grafy zobrazujú miestami skreslené výsledky, kvôli veľkým výsledkovým rozdielom v jednotlivých databázových systémoch.

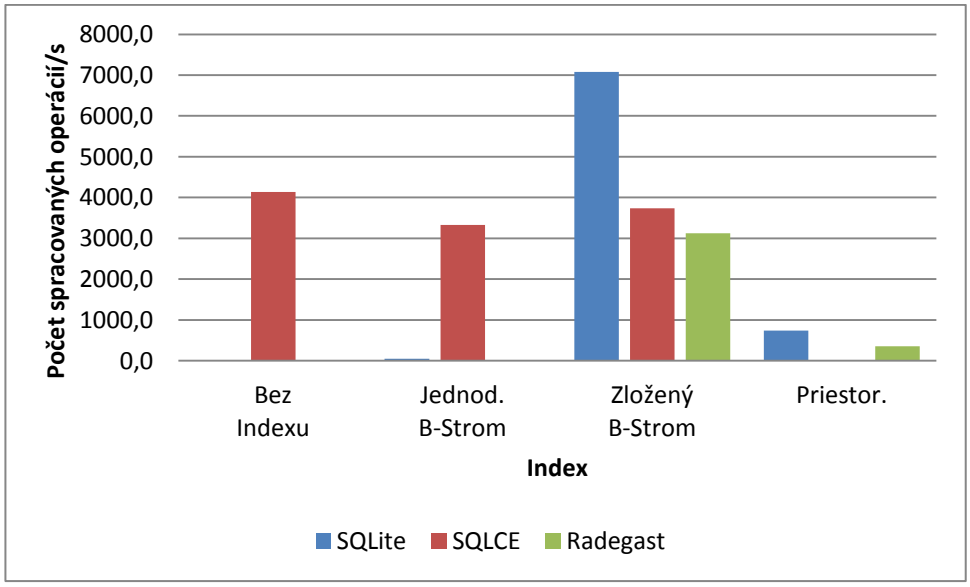
7.2.1 Jednoduché vyhľadávanie

Tabuľka 7.6: Jednoduché vyhľadávanie v kolekcii DOMESTIC_FLIGHTS [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	21,0	27,5	33,9	-	-	-	-
Oracle	253,6	95,6	236,4	232,5	-	-	-
MySQL	4,0	52,5	126,8	-	-	127,9	-
PostgreSQL	2,0	15,1	35,0	-	26,1	-	2,0
DB2	3,6	95,8	894,8	-	-	-	-
SQLite	2,0	51,2	7077,9	-	-	-	736,8
SQLCE	4138,7	3329,3	3737,3	-	-	-	-
Radegast	-	-	3125,8	-	-	-	350,7



Obrázok 7.5: Jednoduché vyhľadávanie v klient - server databázových systémoch (kolekcia DOMESTIC_FLIGHTS)



Obrázok 7.6: Jednoduché vyhľadávanie v embedded databázových systémoch a v Radegast databáze (kolekcia DOMESTIC_FLIGHTS)

V tabuľke 7.6 a obr. 7.5 a 7.6 je vidieť, podobne ako pri testovaní jednoduchého vyhľadávania na predchádzajúcej kolekcii dát, že najpočetnejšie zastúpenie a najlepšie výsledky dosahuje index typu zložený B-Strom takmer vo všetkých databázových systémoch. Dôvodom môže byť ideálna forma dotazu pre tento typ indexu.

7.2.2 **Zložené vyhľadávanie**

Tabuľka 7.7: Zložené vyhľadávanie v kolekcii DOMESTIC_FLIGHTS [počet spracovaných operácií /s]

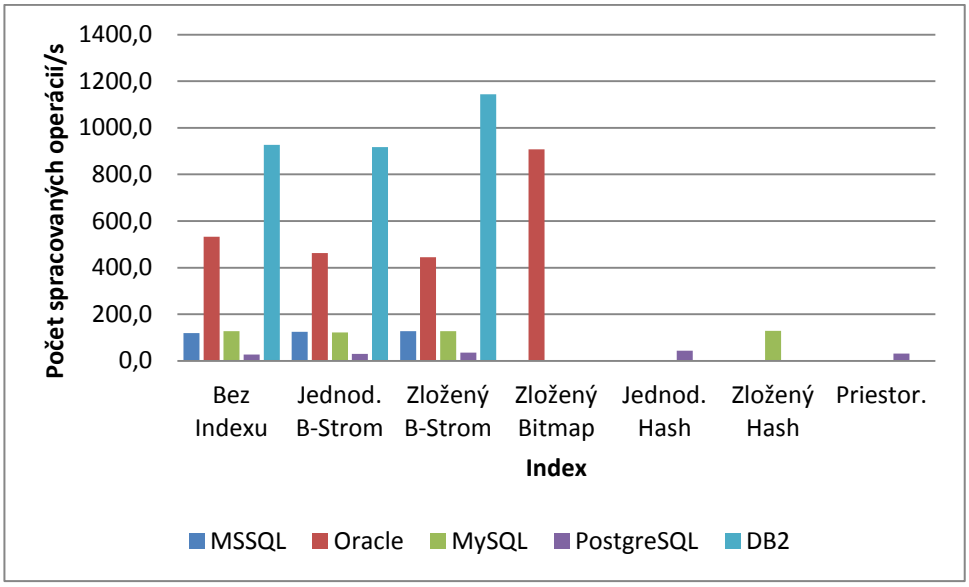
	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	49,8	172,2	1397,2	-	-	-	-
Oracle	282,7	99,0	266,4	260,5	-	-	-
MySQL	4,2	91,2	4987,2	-	-	4990,0	-
PostgreSQL	2,2	33,3	3798,5	-	80,6	-	2,2
DB2	3,3	101,3	1076,3	-	-	-	-
SQLite	35929,9	49704,3	54347,8	-	-	-	32512,9
SQLCE	5075,2	3984,1	4566,3	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.7 je vidieť podobne ako v predchádzajúcom prípade rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Dôvodom je rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami mnohonásobne lepšie výsledky, pravdepodobne z dôvodu ich hlavných výhod.

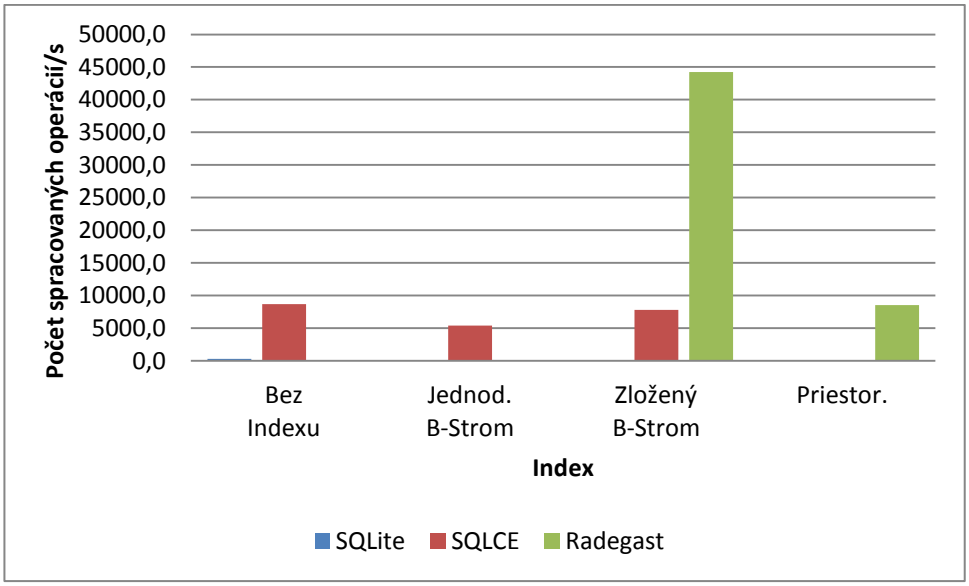
7.2.3 **Jednoduché vkladanie**

Tabuľka 7.8: Jednoduché vkladanie záznamov kolekcie DOMESTIC_FLIGHTS [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	119,1	125,6	127,2	-	-	-	-
Oracle	532,9	462,4	444,8	908,1	-	-	-
MySQL	128,2	122,8	128,1	-	-	128,7	-
PostgreSQL	27,6	29,9	36,0	-	44,7	-	31,7
DB2	926,5	917,0	1143,3	-	-	-	-
SQLite	293,4	104,6	38,7	-	-	-	67,6
SQLCE	8678,8	5422,9	7812,3	-	-	-	-
Radegast	-	-	44248,0	-	-	-	8556,1



Obrázok 7.7: Jednoduché vkladanie v klient - server databázových systémoch (kolekcia DOMESTIC_FLIGHTS)



Obrázok 7.8: Jednoduché vkladanie v embedded databázových systémoch a v Radegast databáze (kolekcia DOMESTIC_FLIGHTS)

V tabuľke 7.8 a obr. 7.7 a 7.8 je na rozdiel od predchádzajúcej testovanej kolekcie možné vidieť, že operácia vkladania nedosahuje vždy najlepšie výsledky v tabuľke bez implementovaného indexu. Dôvodom môže byť vyššia dimenzia kolekcie. Databázový systém Radegast dosahuje mnohonásobne lepšie výsledky aj pri vkladaní do tabuľky s indexom.

7.2.4 Zložené vkladanie

Tabuľka 7.9: Zložené vkladanie záznamov kolekcie DOMESTIC_FLIGHTS [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	3867,2	3612,3	3968,6	-	-	-	-
Oracle	2864,1	1614,3	2118,4	2632,5	-	-	-
MySQL	12554,9	9244,6	11857,9	-	-	12020,5	-
PostgreSQL	2124,4	3026,0	2113,3	-	1309,8	-	1902,9
DB2	21091,7	17521,1	20588,8	-	-	-	-
SQLite	34300,6	1473,6	8065,4	-	-	-	6670,3
SQLCE	20033,3	13365,9	17452,6	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.9 je vidieť podobne ako v predchádzajúcom prípade rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Dôvodom je rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami mnohonásobne lepšie výsledky, pravdepodobne z dôvodu ich hlavných výhod.

7.2.5 Hromadné vkladanie

Tabuľka 7.10: Hromadné vkladanie záznamov kolekcie DOMESTIC_FLIGHTS [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	313596,1	26809,6	71195,1	-	-	-	-
Oracle	67568,7	14221,4	37193,0	38485,5	-	-	-
MySQL	985522,2	198147,5	488503,7	-	-	484305,7	-
PostgreSQL	46219,7	23404,4	45832,1	-	2574,9	-	11304,5
DB2	5499,5	19135,5	18008,0	-	-	-	-
SQLite	25188,9	9665,3	19050,6	-	-	-	11178,2
SQLCE	15192,8	6733,5	11285,8	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.10 je vidieť výsledky hromadného vkladania záznamov v jednotlivých systémoch. Pri tomto testovaní neboli vždy použité najlepšie spôsoby ponúkané jednotlivými systémami, preto nie je namieste ich porovnávať. Je však vidieť pokles efektivity pri hromadnom vkladaní do tabuliek s indexom, z dôvodu už spomenutej potrebnej réžie indexov.

7.3 Kolekcia XMARK

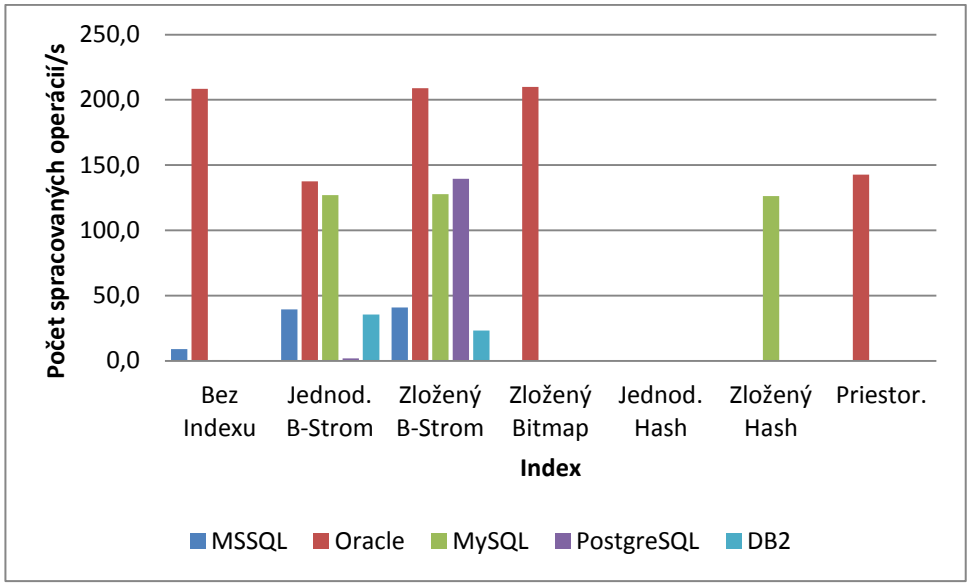
Kolekcia XMARK predstavuje tabuľku s 20 532 805 riadkami (záznamami) a 3 stĺpcami (dimenzia). Hodnoty v stĺpcoch sú celočíselného dátového typu. Pri tejto kolekcii neboli testované: priestorového indexy v databázových systémoch MySQL (nepodporovaný pre dimenziu 3), DB2 a hash index v systéme PostgreSQL, z dôvodu problémov počas testov.

Príslušné grafy zobrazujú miestami skreslené výsledky, kvôli veľkým výsledkovým rozdielom v jednotlivých databázových systémoch.

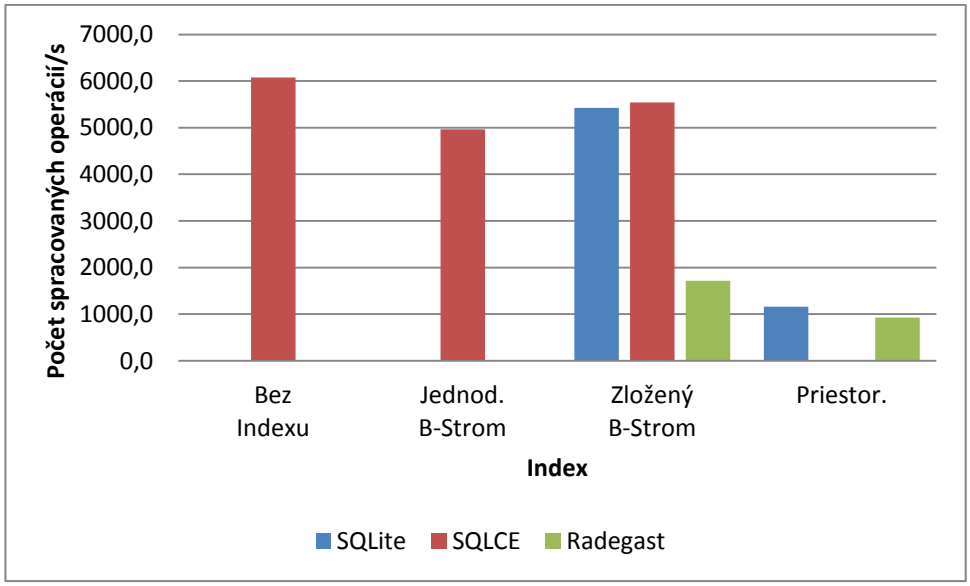
7.3.1 Jednoduché vyhľadávanie

Tabuľka 7.11: Jednoduché vyhľadávanie v kolekcii XMARK [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	9,0	39,5	40,9	-	-	-	< 0,1
Oracle	208,4	137,6	209,0	209,9	-	-	142,7
MySQL	0,7	127,0	127,7	-	-	126,4	-
PostgreSQL	0,4	2,0	139,6	-	-	-	0,4
DB2	0,3	35,7	23,3	-	-	-	-
SQLite	0,6	0,4	5424,1	-	-	-	1162,2
SQLCE	6076,1	4966,5	5541,2	-	-	-	-
Radegast	-	-	1719,7	-	-	-	930,9



Obrázok 7.9: Jednoduché vyhľadávanie v klient - server databázových systémoch (kolekcia XMARK)



Obrázok 7.10: Jednoduché vyhľadávanie v embedded databázových systémoch a v Radegast databáze (kolekcia XMARK)

Z tabuľky 7.11 a obr. 7.9 a 7.10 je vidieť, podobne ako pri testovaní jednoduchého vyhľadávania v predchádzajúcich prípadoch, že najpočetnejšie zastúpenie a najlepšie výsledky dosahuje index typu zložený B-Strom takmer vo všetkých databázových systémoch. Dôvodom môže byť ideálna forma dotazu pre tento typ indexu. Tento náskok sa však zmenšil so zvýšeným počtom záznamov v tabuľke. Výraznejšie vystupuje databázový systém Oracle, ktorý

dosiahol najlepšie výsledky vo všetkých testovaných indexoch, ktorých sa zúčastnil. Za zmienenie stojí aj výkon priestorového indexu v systéme Oracle.

7.3.2 Zložené vyhľadávanie

Tabuľka 7.12: Zložené vyhľadávanie v kolekcii XMARK [počet spracovaných operácií /s]

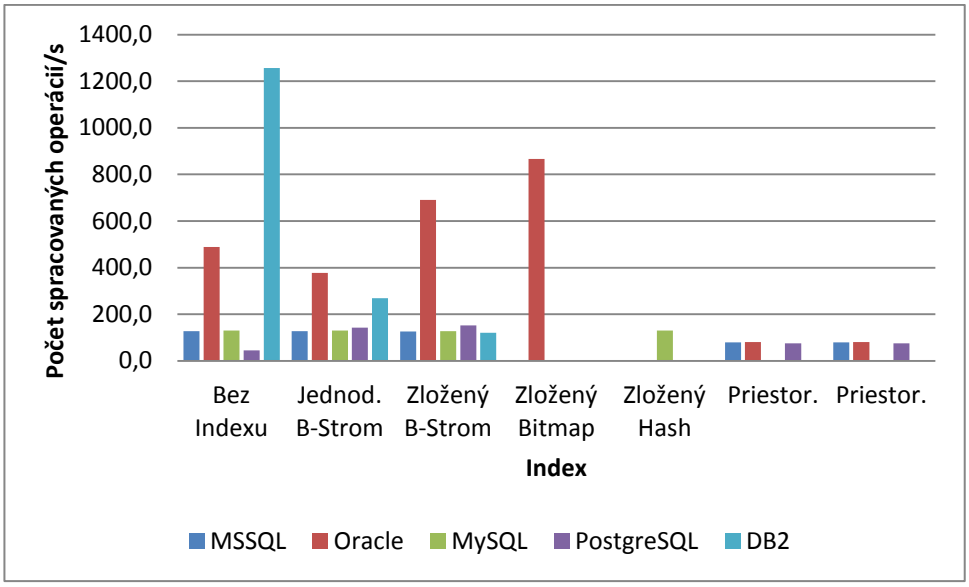
	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	11,1	730,4	1901,9	-	-	-	< 0,1
Oracle	232,4	149,6	237,5	226,2	-	-	171,1
MySQL	0,7	4806,6	5321,2	-	-	5394,4	-
PostgreSQL	0,4	2,1	9883,8	-	-	-	0,4
DB2	0,2	1332,7	1575,9	-	-	-	-
SQLite	2222,6	1537,6	69749,6	-	-	-	24294,8
SQLCE	8606,1	6997,0	7705,7	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.12 je vidieť opätovný rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Dôvodom je rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami vo väčšine prípadov lepšie výsledky, pravdepodobne z dôvodu ich hlavných výhod.

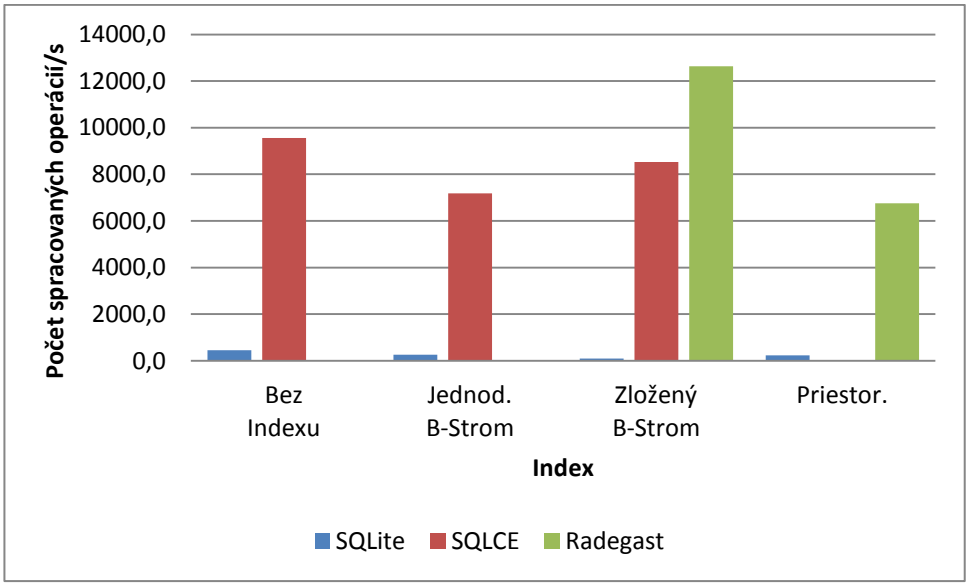
7.3.3 Jednoduché vkladanie

Tabuľka 7.13: Jednoduché vkladanie záznamov kolekcie XMARK [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	127,6	127,4	126,8	-	-	-	80,2
Oracle	488,6	377,8	690,7	866,4	-	-	80,8
MySQL	130,7	130,4	127,5	-	-	130,0	-
PostgreSQL	44,9	142,4	152,1	-	-	-	75,5
DB2	1256,9	268,9	121,4	-	-	-	-
SQLite	448,1	258,4	92,1	-	-	-	241,4
SQLCE	9562,6	7188,1	8531,9	-	-	-	-
Radegast	-	-	12638,4	-	-	-	6755,6



Obrázok 7.11: Jednoduché vkladanie v klient - server databázových systémoch (kolekcia XMARK)



Obrázok 7.12: Jednoduché vkladanie v embedded databázových systémoch a v Radegast databáze (kolekcia XMARK)

V tabuľke 7.13 a obr. 7.11 a 7.12 je na rozdiel od prvej testovanej kolekcie možné vidieť, že operácia vkladania nedosahuje vždy najlepšie výsledky v tabuľke bez implementovaného indexu. Dôvodom môže byť vyšší počet záznamov kolekcie. Databázový systém Radegast dosahuje najlepšie výsledky z testovaných systémov.

7.3.4 **Zložené vkladanie**

Tabuľka 7.14: Zložené vkladanie záznamov kolekcie XMARK [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	5519,5	5028,5	5588,2	-	-	-	742,0
Oracle	3497,6	2612,1	3078,2	2754,7	-	-	130,0
MySQL	12675,2	11801,5	12092,2	-	-	12154,4	-
PostgreSQL	50746,0	29332,4	6212,9	-	-	-	2778,7
DB2	27835,0	5105,0	25013,1	-	-	-	-
SQLite	41181,1	4730,9	9000,8	-	-	-	12726,7
SQLCE	25020,0	17656,3	22359,4	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.14 je vidieť podobne ako v predchádzajúcich prípadoch rapídny nárast počtu spracovaných operácií (za sekundu) vo väčšine prípadov. Dôvodom je rozdiel v spracovaní všetkých operácií len jedným príkazom zo strany aplikácie. Embedded databázové systémy dosahujú v porovnaní s klient - server systémami mnohonásobne lepšie výsledky, pravdepodobne z dôvodu ich už spomenutých hlavných výhod.

7.3.5 **Hromadné vkladanie**

Tabuľka 7.15: Hromadné vkladanie záznamov kolekcie XMARK [počet spracovaných operácií /s]

	Bez Indexu	Jednod. B-Strom	Zložený B-Strom	Zložený Bitmap	Jednod. Hash	Zložený Hash	Priestor.
MSSQL	359933,0	41726,8	62889,4	-	-	-	6571,4
Oracle	106659,9	36041,8	57062,9	60458,2	-	-	3780,2
MySQL	951894,0	350181,4	657582,1	-	-	556094,2	-
PostgreSQL	55203,9	25512,7	34834,3	-	-	-	11415,5
DB2	34445,0	25628,5	31001,5	-	-	-	-
SQLite	26800,1	15674,7	19846,2	-	-	-	11808,1
SQLCE	17914,9	8761,9	9851,9	-	-	-	-
Radegast	-	-	-	-	-	-	-

V tabuľke 7.15 je vidieť výsledky hromadného vkladania záznamov v jednotlivých systémoch. Pri tomto testovaní neboli vždy použité najlepšie spôsoby ponúkané jednotlivými systémami, preto nie je namieste ich porovnávať. Je však vidieť pokles efektivity pri hromadnom vkladaní do tabuliek s indexom, z dôvodu už spomenutej potrebnej réžie indexov.

Záver

Indexovanie viacrozmerných dát v dnešných relačných databázových systémoch je značne obmedzené pri použití priestorových indexov. Väčšina databázových systémov podporuje vytvorenie priestorového indexu len na kolekcii dát priestorového dátového typu (body, polygóny, čiary a pod.). Možnosť použitia priestorového indexu (často R-Stromu) je silne závislá na počte atribútov danej kolekcie dát, z dôvodu častého použitia indexu tohto typu len na objekty v 3D priestore. Samozrejme v niektorých prípadoch je dovolené použitie priestorového indexu aj na špeciálne dátové typy, ktorými je možné prezentovať multidimezionálnu kolekciu dát. Z kapitoly 7, v ktorej je reálne testovaná efektivita použitia priestorového indexu v porovnaní s často využívaným B-Stromom, vykazuje priestorový index nižšiu efektivitu, je však potrebné podotknúť, že bolo testované len tzv. bodové vyhľadávanie (použitie operátora "="). Štruktúra R-Strom (častá pre priestorové indexy) bola však navrhnutá pre rozsahové vyhľadávanie [15].

Okrem preskúmania možností a porovnanie skutočného výkonu indexových štruktúr na rôznych kolekciách je výsledkom práce aj software, ktorý umožňuje podobné testovanie realizovať opakovane, na používateľom vybranej kolekcii dát.

Databázový systém Radegast podporuje vytvorenie indexu aj na objekty, ktoré nepredstavujú 3D objekty v reálnom priestore. Tým je možné indexovať kolekcie dát vyšších dimenzií. Efektivita ukladania dát (vloženie jedného záznamu jedným príkazom) je v Radegaste v porovnaní s testovanými systémami bezkonkurenčná. Dôvodom tejto výhody môže byť použitie tzv. rýchlej pamäte namiesto neustáleho priebežného ukladania dát na disk. Vyhľadávanie za pomocí indexových štruktúr je v Radegaste výkonnejšie v porovnaní s klient - server databázovými systémami. Z embedded databázových systémov vykazuje lepšie výsledky len SQLite, ktorý podobne ako Radegast podporuje viacrozmernú dátovú štruktúru R-Strom (konkrétne jeho modifikáciu R*-Strom).

Z pohľadu budúceho rozšírenia práce sa je možné zamerať na implementáciu netestovaných indexových štruktúr a na testovanie rôznych typov vyhľadávania (rozsahové, najbližšie objekty v určenom rozsahu daného objektu a pod.) a zistiť ich výkon a možností.

Použitá literatúra

[1] Clustered and Nonclustered Indexes Described. Dostupné z: http://msdn.microsoft.com/en-us/library/ms190457.aspx

[2] BHATTACHARYA Arnab. Fundamentals of Database Indexing and Searching. 1. vyd. Boca Raton: Chapman and Hall/CRC, 2014, 280 s. ISBN 978-1-4665-8255-2

[3] Nelineární datové struktury IV. Dostupné z: http://www.cs.vsb.cz/kratky/courses/2004-05/udp/presentation/udp-10_6.pdf

[4] Spatial Indexing Overview. Dostupné z: http://technet.microsoft.com/en-us/library/bb964712(v=sql.105).aspx

[5] Spatial Concepts. Dostupné z: http://docs.oracle.com/cd/B10501_01/appdev.920/a96630/sdo_intro.htm#i877656

[6] Creating Spatial Indexes. Dostupné z: https://dev.mysql.com/doc/refman/5.0/en/creating-spatial-indexes.html

[7] CREATE INDEX Syntax. Dostupné z: https://dev.mysql.com/doc/refman/5.0/en/create-index.html

[8] Optimizing Spatial Analysis. Dostupné z: https://dev.mysql.com/doc/refman/5.0/en/optimizing-spatial-analysis.html

[9] SCHWARTZ Baron, ZAITSEV Peter, TKACHENKO Vadim. High Performance MySQL. 3. vyd. Sebastopol: O'Reilly Media, 2012, 828 s. ISBN 978-1-449-31428-6

[10] GiST: A Generalized Search Tree for Secondary Storage. Dostupné z: http://gist.cs.berkeley.edu/gist1.html

[11] Cube. Dostupné z: http://www.postgresql.org/docs/9.4/static/cube.html

[12] PostGIS. Dostupné z: http://www.postgis.net

[13] Manage spatial data with IBM DB2 Spatial extender, Part 1: Acquiring spatial data and developing applications. Dostupné z: http://www.ibm.com/developerworks/data/tutorials/dm-1202db2spatialdata1/

[14] Spatial indexes generated by the DB2 Spatial Extender. Dostupné z: http://edndoc.esri.com/arcobjects/9.2/NET_Server_Doc/manager/geodatabase/administering_a-557706548/spatial-586982968.htm

[15] The SQLite R*Tree Module. Dostupné z: http://www.sqlite.org/rtree.html

[16] Differences Between SQL Server Compact and SQL Server. Dostupné z: https://technet.microsoft.com/en-us/library/bb896140(v=sql.110).aspx

[17] R. Bača, P. Chovanec, M. Krátký, P. Lukáš: QuickDB - Yet another Database Management System?. In Proceedings of DATESO 2014, CEUR Workshop Proceedings, Volume 1139, pp. 91-99, 2013. Recorded in Scopus.

[18] Dynamic Versions of R-trees. Dostupné z: http://www.bowdoin.edu/~ltoma/teaching/cs340/spring08/Papers/Rtree-chap2.pdf

[19] Hash Indexes. Dostupné z: http://msdn.microsoft.com/en-us/dn133190.aspx

[20] Spatial Data Types Overview. Dostupné z: http://msdn.microsoft.com/en-us/library/bb964711.aspx

[21] Spatial Data Types. Dostupné z: http://dev.mysql.com/doc/refman/5.0/en/spatial-datatypes.html

[22] Geometric Types. Dostupné z: http://www.postgresql.org/docs/9.4/static/datatype-geometric.html

[23] SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees. Dostupné z: http://www.cs.purdue.edu/spgist/papers/W87R36P214137510.pdf

[24] Spatial Data Types. Dostupné z: http://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.spatl/src/tpc/spatl_db2sb41.dita?lang=en

[25] GIN Indexes. Dostupné z: http://www.postgresql.org/docs/9.4/static/gin-intro.html

[26] SP-GiST Indexes. Dostupné z: http://www.postgresql.org/docs/9.2/static/spgist-intro.html

Zoznam príloh

Súčasťou BP je CD.

Adresárová štruktúra priloženého CD:

\src Zdrojový kód aplikácie.

\doc Dokument BP.