TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M2612 – Elektrotechnika a informatika

Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Databázové systémy s fulltextovým indexem

Diplomová práce

Autor: **Štěpán Farkašovský**

Vedoucí DP práce: Mgr. Jiří Vraný

V Liberci 25. 10. 2006

Originál zadání

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Anotace

Cílem diplomové práce Databázové systémy s fulltextovým indexem je zmapovat fungování fulltextových indexů, využívaných pro textové vyhledávání, v open source databázích. Dále se seznámit s metodikami měření výkonnosti databází. Na základě těchto informací navrhnout vlastní princip měření výkonnosti open source databází s fulltextovými indexy.

Popsané systémy jsou v práci zdokumentovány a slouží spíše než k návrhu nové metodiky k zamyšlení se nad diferencemi mezi potřebami uživatelů a fungujícími principy pro měření výkonu. Další možností pro analýzu dat je využití třetí strany pro testování. Bohužel tyto třetí strany jsou na našem trhu zastoupeny formou konzultantských společností což znamená, že ne každý podnik si může dovolit jejich služby z finančního hlediska.

Výsledkem práce je skript pro měření výkonnosti na konkrétním databázovém skladu. Čímž také demonstruje nevhodnost využití obecně používaných metodik pro potřeby menších podniků. Tyto testy jsou totiž spíše testy syntetické výkonnosti. Měří výkon databáze, ale nezohledňují okolní faktory, které mohou mít vliv na výkon. Těmi může být například hardware na kterém se měří, struktura dat na kterých se měří nebo třeba na způsobu přistupování k datům. Proto se tato práce zabývá asi dnes nejpoužívanějším způsobem přístupu k datům a to přes webové rozhraní pomocí PHP. Zmiňované vlivy jsou odstraněny tím, že testy si uživatel může provést na svých datech a na svém hardwaru. Konečným výsledkem není říci zda je ta či ona databáze rychlejší či výkonnější, ale dát uživateli nástroj kterým zjistí zda pro něj je výhodnější ta či ona. Protože pokud vyhlásím vítěze nutně to nemusí znamenat, že by pro všechny uživatele byl tento sytém nejvýkonnější.

Databáze

Open source

Fulltextové indexy

Benchmarking

Abstract

The goal of this diploma thesis “Database Systems with Fulltext Index” is to map operation of fulltext indexes using for the text searching in open sources databases, then to acquaint with the methods for measuring of database efficiency. Finally on base of the gained information to project own principle for efficiency measuring of open source databases with fulltext indexes.

Described systems are documented in the thesis and they are used rather than a proposal for a new methodology more to the reflection of some differences between users’ needs and functioning principles for efficiency measuring. Next eventuality for the data analysis is using of the third party to test. Unfortunately the third parties are represented by a row of consulting companies that means not all companies can allow this service from financial reason.

Result of the thesis is a script for efficiency measuring on the concrete database storage that demonstrates some unsuitability for application of generally using methodologies for needs of small companies. These tests are rather tests of synthetic efficiency. They measure database efficiency but they do not perceive surrounding factors, which can influence the efficiency. Those factors can be e.g. hardware, data structure on which the measurement is done; or on the kind of approach the data. Therefore the thesis deals with the most widely used kind of approach data, namely through the web boundary with PHP. Mentioned effects are removed with that user can do tests on his/ her data and hardware. The final result is not to say which database is quicker or more efficient but to provide user a tool by which he/she finds out the better database for him/her. If I announce the winner, it needn’t necessarily mean, the system is the most efficient for all users.

Database

Open source

Fulltext Index

Benchmarking

Obsah

Úvod

Co říci úvodem? Databáze v nejrůznějších formách dnes můžete nalézt v nespočetném množství nasazení. Proto dnes mnoho podniků přemýšlí nad jejich optimalizací při vlastní implementaci. Jedním z důležitých faktorů pro výběr do podniku z našich lokalit je finanční stránka a výkonnost databáze. Proto se zde zabývám oblastí open source databází a jejich uplatněním. Na následujících listech se čtenář dočte co vůbec pojem open source znamená, jaké zástupce na tomto poli můžeme nalézt a jaké vlastnosti mají. Okrajově se zmiňuji o webovém serveru Apache s modulem PHP. O PHP protože PHP používám jako jazyk pro skript, kterým testuji rychlost databáze. Proto, abych mohl používat PHP potřebuji jej na nějakém webovém serveru provozovat. Webový server jako takový slouží právě jako základna pro poskytování webových služeb (interpretace nejrůznějších skriptovacích jazyků). Webový server lze nadneseně považovat za jakousi „šedou eminenci“ stojící v pozadí toho, že si můžete v poklidu prohlížet nejrůznější internetové stránky. Objasňuji problematiku fulltextových indexů (dnes jsou největším zástupcem dat, data textová). Vysvětluji proč je nemožné porovnávat výsledky s výsledky jiných společností na testování výkonnosti. Hlavně však proč je toto porovnání bezpředmětné. Dávám čitateli jakousi rukoveť (administrátorovi či implementátorovi) obsahující soubor informací o daném téma, protože dnes není problém jakékoliv informace získat na internetu, ale je to však velmi časově náročné. Provedené testování je pouze informativní a slouží spíše k ukázání odlišností mezi známými technikami a mým přístupem. Testování bylo provedeno na open source zástupcích, kteří poskytují fulltextové indexy. Jedná se o MySQL a PostgreSQL. Na ostatních testování neprobíhala ať už z důvodu neexistence fulltextového indexu nebo se nejednalo o open source licenci.

1. Pojem Open Source

Jelikož se v celé diplomové práci budu z větší části zabývat open source, je vhodné objasnit některé termíny a to nejen tento. V dnešní době hraje open source software poměrně značnou roli na poli IT (Information Technologies). Hlavní definici toho co je open source software udává OSI (Open Source Initiative). Ta předepisuje soubor pravidel, podle kterých lze konkretní licenci (a tím i pod ní šířený produkt) považovat za "Open Source". Domovské stránky organizace naleznete na **opensource.org**. Pro tuto práci není významné

překládat celou definici doslovně, a proto z ní vyzdvihnu jen některé důležité body:

-volné rozšiřování licence nesmí omezovat prodej nebo jinou distribuci programu jako součásti programového balíku obsahujícího software z různých zdrojů; licence by za takový prodej neměla vyžadovat autorský nebo jiný honorář

-zdrojový kód produkt musí obsahovat zdrojový kód a musí umožňovat distribuci jak ve zdrojové, tak v binární ("zkompilované") podobě; pokud program není šířen včetně zdrojových kódů, musí být dobře popsána možnost jejich získání, a to za přiměřený poplatek (pokrývající náklady); zdrojový kód nesmí být zamlžen; přechodné formy (např. výstup preprocesoru nebo překladače) nejsou dovoleny

-odvozené práce licence musí umožňovat tvorbu odvozených prací a musí jim umožnovat, aby byly šířeny pod stejnou licencí jako původní produkt

-integrita (celistvost) autorova zdrojového kódu licence může omezovat distribuci změněné formy zdrojového kódu pouze v případě, že je umožněno šíření tzv. záplat (patch files) spolu se zdrojovým kódem; licence musí výslovně povolit šíření programu přeloženého ze změněného zdrojového kódu; licence může vyžadovat, aby odvozené práce nesly jméno nebo verzi odlišné od původního programu

-diskriminace vůči osobám a skupinám licence nesmí diskriminovat osoby nebo skupiny osob

-diskriminace sfér užití licence nesmí omezovat použití programu v určité sféře; nesmí například omezovat použití programu v komerčním prostředí nebo v genetickém výzkumu

-šíření licence práva přiložená k programu musí platit pro všechny, bez nutnosti dalších přídavných licencí

-licence nesmí záviset na programovém produktu práva přiložená k programu nesmí záviset na existenci programu v určitém programovém balíku; pokud je program z balíku vyřazen a je používán nebo šířen v souladu s licencí, všichni, ke kterým se program dostane, by měli mít

stejná práva jako ti, kteří dostanou program jako součást programového balíku

-licence nesmí ovlivňovat ostatní programy licence nesmí klást omezení na software, který je šířen společně s licencovaným programem; licence nesmí například trvat na tom, aby všechny programy distribuované na stejném médiu splňovaly podmínky Open Source software

Zřejmě nejznámějšími "Open Source" licencemi jsou:

GNU General Public License (GPL)
GNU Library Public License (LGPL)
BSD license

Aktuální seznam všech licencí splňujících OSD (Open Source Definition) se nachází na stránkách OSI. [1]

2. Porovnání komerčního a open source softwaru

V dnešní době se vedou sáhodlouhé diskuze o výhodých a nevýhodách (a nejen ekonomických) komerčního softwaru a open source softwaru. Není proto cílem se tímto problémem zabývat v této diplomové práci. Jen uvedu několik vlastních poznatků, které mne napadají s touto problematikou.

Většina populace se dnes domnívá, že open source je jen pro odborníky a IT specialisty. Jejich domněnky jsou bohužel mylné. Velké množství firem dnes využívá open source software nejen jako softwarové řešení pro servery, ale také pro desktopy svých zaměstnanců (jako příklad za všechny mohu uvést firmu Baťa). Dalším faktem je, že většina správců IT systémů se domnívá, že struktura postavená na open source bázi je jediná správná a bezpečná. To dnes již také neplatí. Záleží jen a pouze na šikovnosti daných správců, aby se zapříčinili o hladký chod firemní sítě.

3. Open Source databáze

V této diplomové práci se budu zabývat pouze jedním z odvětví open source, a to databázovými systémy. V tuto chvíli jsou na trhu dostupné zhruba dva tucty různých open source databází. Z těchto mohu vyjmenovat jako nejznámější asi tyto – MySQL, PostgreSQL, Firebird, Ingres, Interbase. (pro testování jsem použil pouze první dva zástupce – MySQL a PostgreSQL – protože zbylé zatím neumožňují fulltextové indexy a pokud ano tak za peníze, čímž nesplňují podmínku pro open source licenci) Jejich rozsah se samozřejmě velmi liší, od speciálních nástrojů až po plnohodnotné systémy, které mohou konkurovat svým komerčním bratránkům. Zvláštnost databázových systémů na bázi open source spočívá v tom, že na rozdíl například od operačních systémů nebo dalších často využívaných aplikací (viz. Linux + Apache), jsou databáze se svým často velmi cenným obsahem daleko choulostivější záležitostí. Nutno dodat, že při pojmu open source DB si většina lidí představí vazbu na Linux. To ovšem nemusí být pravidlem (dnes některé open source systémy jsou schopny fungovat ve velmi rozmanitém prostření).

4. Základní popis nejdůležitějších hráčů na poli open source DB

4.1. MySQL:

MySQL je multiuživatelský, multithreadový SQL (SQL neříká jak se s daty má pracovat, ale co se s nimi má udělat) databázový server. Hlavním cílem MySQL je rychlost, robustnost a jednoduchost. Vývoj se datuje od roku 1996 se zaměřením na schopnost správy velkého množství dat na levném hardwaru. Systém poskytuje rychlý přístup i k velkému množství uložených dat. Jeho hlavní výhodou je snadná integrovatelnost především při vývoji webovských aplikací společně například s PHP a Apache. Drobnou nevýhodou je malá podpora systému například na nejrozšířenějších platformách Windows (pro kterou je dokonce nutné zakoupit licenci), což MySQL přesouvá především do sféry serverových aplikací. Z tohoto důvodu je i relativně obtížné psát pro tuto databázi obslužný software, pokud nechceme používat ODBC ovladače. Dalším problémem je ne úplně triviální instalace, což je další překážkou rozšířenějšího používání. Integrace velkého množství datových typů dává možnost přesněji definovat, jak má být struktura tabulek definována. Určitým plusem je také poměrně široká paleta funkcí pro operaci s textovými řetězci a dalšími datovými

typy umožňující snadněji vybírat různé záznamy z tabulek a pracovat s nimi již na databázové úrovni.

Databázový systém MySQL je v každém případě velmi vydařený projekt umožňující provozovat kvalitní a rychlou databázi i na relativně slabším hardwaru. Její licencování je jednoznačně orientováno ve prospěch Linuxu a podobných operačních systémů a společně s dalšími aplikacemi typu Apache a PHP vytváří velmi silný vývojářský nástroj pro tvorbu například internetových aplikací.

- ukládání procedur/funkcí
- triggery (programové moduly, které se aktivují v požadovaném momentě, například při vložení záznamu)
- dotazová cache (SQL dotazy jsou ukládány do mezipaměti pro případ, že by byl potřeba následně dotaz stejný a tím pádem se vykonal rychleji)
- podporuje od verze 4.1 poddotazy v SQL dotazu
- umožňuje poměrně dobrou lokalizaci (jazyk pro sloupec, tabulku či celou databázi)

4.2. Interbase:

Tento, všem vývojářům v Delphi asi známý systém měl až do nedávné minulosti značný problém, a to byla jeho komerčnost. Při vývoji aplikace na nekomerční bázi bylo v podstatě od začátku jasné, že se takový program nemůže příliš rozšířit, poněvadž jak všichni dobře víme, morálka zakupování drahých zahraničních softwarových produktů je u nás poměrně dosti nízká. Co ovšem bývalo nevýhodou je nyní velkou výhodou, neboť k překvapení asi všech se z komerční Interbase stala open source Interbase. Právě tato předchozí komerční "zkušenost" zanechala velké množství velmi dobře zpracovaných utilit vhodných ke správě takovéto databáze. Rovněž nesporně velká zkušenost s komerčním provozem se blahodárně promítla do relativně bezchybné současné podoby.

Interbase je transakční relační databázový systém na bázi Client-Server a poskytuje všechny výhody standardního RDBMS. Následující body ukazují některé klíčové vlastnosti systému INTERBASE:

- Podpora síťových protokolů (Interbase podporuje TCP/IP na všech platformách. Interbase na všech platformách Windows podporuje NetBEUI/named pipe, IPX/SPX)
- Přizpůsobení SQL92 standardu
- Současný přístup do více databází
- Multigenerační architektura
- Optimistic Row-level blocking (Server blokuje pouze záznam, se kterým se zrovna pracuje namísto zablokování celé stránky.)
- Optimalizace dotazů
- Datové typy s dynamickou velikostí (např. na grafiku…)
- Uložené procedury
- Triggery
- Event alerter (umožňuje databázi předávat zprávy aplikaci, která jí obsluhuje)
- UDF - User defined functions (Programové moduly, které běží na serveru)
- Outer joins (Relační konstrukce mezi dvěma tabulkami, která umožňuje komplexní operace)
- Transakční management (umožňuje dokonalou kontrolu nad prováděním transakcí)
- Vícenásobný přístup k datům (klient čtoucí tabulku neblokuje ostatní od provádění té samé činnosti)
- Multidimenzionální pole
- Automatické dvoufázové zapsání dat (Server před zápisem dat dohlédne na to, aby se změny projevily ve všech zúčastněných databázích)
- Podpůrné aplikace (Distribuce interbase obsahuje aplikace pro snadnou správu dat a zálohovaní)

4.3. PostgreSQL:

Databázová platforma PostgreSQL je v současné době jednou z nejpokrokovějších open source databází. Historie vývoje PostgreSQL sahá až do roku 1986 a na univerzitu v Berkeley. Mezi hlavní historické události patří:

- první demoverze byla představena v roce 1986
- v roce 1992 byl kód PostreSQL zkomercializován společností Illustra Information Technologies (jež později splynula se společností Informix)
- v průběhu roku 1994 Andrew Yu a Jolly Chen, autoři verze 1.0.1., implementovali do PostgreSQL interpret jazyka SQL a roce 1995 byl na web uvolněn Postgres95, aby si, jakožto open-source a předchůdce originálního kódu vzniklého na univerzitě v Berkley, našel cestu mezi databázovými open-source produkty
- koncem roku 1996 byl Postgres95 přejmenován na PostgreSQL Mezi hlavní vlastnosti a přednosti PostgreSQL patří:
    - open-source šířený pod BSD licencí umožňující vlastní úpravy a jejich další šíření
    - objektově-relační SQL server
    - na rozdíl od jiných open-source databázových serverů (např. MySQL) podporuje zamykání dat na úrovni záznamu a nikoli na úrovni celé tabulky, tzn., že záznamy je možno číst i při jejich současné aktualizaci aniž by došlo k již zmiňovanému uzamčení celé tabulky i pro čtení
    - zabudována podpora všech operačních systémů včetně os/2 a Novell
    - PostgreSQL je též podporován několika programovacími, popř. skriptovacími jazyky: PHP, Perl, Python, Embeded SQL pro C/C++, Java…
    - uživatelem definované funkce (UDF), které je možno psát v několika programovacích jazycích; UDF lze též využít k návrhu vlastních datových typů

4.4. Firebird:

Databázová platforma **Firebird** vyvíjená skupinou nezávislých vývojářů byla na začátku svého vývoje velmi úzce svázána s jiným systémem řízení báze dat – InterBase. Na začátku roku 2000 uvolnila společnost Borland zdrojové kódy InterBase. Vzhledem k tomu, že v té době byla podpora dalšího vývoje InterBase ze strany Borlandu velmi nejistá, ujali se uvolněného kódu přední odborníci na tuto platformu a založili projekt Firebird.

Tento systém podporuje standard dotazovacího jazyka SQL odpovídající vstupní úrovni normy SQL92, transakčního zpracování a velmi rozsáhlých databází (jako jedna z největších instalací je uváděna aplikace zpracovávající data o objemu cca 600 gigabajtů).

Českou republiku (potažmo uživatele z České republiky) jistě potěší funkční implementace českého třídění včetně znaku "ch", které i dnes znamená pro některé databázové platformy neřešitelný problém. Firebird podporuje znaky UNICODE a databáze v různých znakových sadách. Pokud by nastala situace, že server nepodporuje požadovanou funkčnost, je v mnoha případech poměrně snadné ji naprogramovat pomocí tzv. uživatelsky definovaných funkcí.

Zanedbatelná není ani podpora ze strany vývojových nástrojů, která je patrná zejména v případě Delphi – díky kompatibilitě s InterBase mohou vývojáři využít celou řadu léty praxe prověřených komponent.

Jednou z věcí v kterých Firebird zaostává je zpracování grafického, administrativního rozhraní. Trendem posledních let je dodávat s každou databázovou platformou kvalitní administrátorské nástroje s precizně provedeným grafickým uživatelským rozhraním. Firebird – v základních distribucích toho moc pro grafické rozhraní nenajdete. Naštěstí existují produkty třetích stran.

Platforma Firebird je dostupná pro unixové operační systémy (především pro linuxové klony) i pro operační systémy společnosti Microsoft (databázový server je možné provozovat na obou vývojových řadách, tedy jak na 9x, tak i na NT).

4.5. Ingres:

Tento produkt od výrobce Computer Associates Platforma Ingres je ve skutečnosti skupina produktů, kam patří i databázový server dříve známý pod obchodním názvem OpenIngres a později Ingres II.

U této platformy mají její uživatelé například k dispozici nejen podporu XML, ale také podporu distribuovaného zpracování a replikačních mechanismů, tvorby vícevrstvých databázových aplikací. Za nespornou výhodu v dnešním mobilním světě je i možnost mobilního přístupu. V této době je také hojně skloňována 64bitová architektura, kterou Ingres podporuje. Samozřejmostí je implementace základních databázových technologií, kam patří SQL, transakční mechanismy, referenční integrita apod.

Velmi přehledná je administrace celého prostředí, ke které je možné využít nástroj s grafickým uživatelským rozhraním nazvaný Advantage Ingres Visual DBA. Uvedený nástroj lze použít i jako základní klientskou aplikaci pro zadávání příkazů jazyka SQL. Díky aplikačnímu rozhraní architektury Advantage Ingres Management Architecture je dokonce možné vytvářet vlastní administrátorské nástroje přizpůsobené konkrétním požadavkům.

Produkt je dostupný pro celou řadu operačních systémů, včetně linuxových klonů a vyspělých komerčních unixů (HP-UX, Solaris apod.) Navíc platforma r3 byla nedávno uvolněna jako open source.

5. Popis Apache

Jestliže se v jedné větě zmíníme o open source databázích tak hned druhou větou musíme doplnit zmínku o PHP a Apache. Jak jsem již napsal v úvodu, webový server slouží k poskytování nejrůznějších služeb (v mém případě PHP) pro uživatelovo poklidné procházení internetem. Apache je přední webový server, s více než 60 % podílu na trhu podle přehledů společnosti Net-craft. Některé klíčové faktory, které se podílely na úspěchu Apache:

- Apache je distribuován s licencí podobnou BSD, která umožňuje komerční i nekomerční využití
- Modulární architektura. Uživatelé Apache mohou jednoduše přidávat funkce nebo rozvíjet

- Přenositelnost: Apache běží na skoro všech unixových (a linuxových) systémech, Windows, BeOS, sálových počítačích...
- Robustnost a bezpečnost.

Mnozí komerční dodavatelé vytvořili řešení založená na Apache, včetně Oracle, Red Hat a IBM. Sílu tohoto webového serveru mohou demonstrovat i tito jeho uživatelé -Amazon.com, Yahoo!, W3 Consortium, Network solutions, MP3.com. Apache vznikl jako modifikace webového serveru NCSA, jednoho z vůbec prvních webových serverů. Projekt Apache se rozšířil z vytváření pouze webového serveru na další důležité technologie na straně serveru, jakými jsou např. Java nebo XML. Všechny tyto projekty zastřešuje Apache Software Foundation. Existují dvě hlavní verze serveru Apache, řada 1.3 a řada 2.0. Obě řady jsou vhodné pro produkční nasazení, liší se však v architektuře a v možnostech:

- Apache 1.3 byl přenesen na řadu různých unixových platforem a jde o nejrozšířenější webový server na Internetu. Apache 1.3 na Unixu je procesově orientovaný server. Při startu spustí několik potomků – rodičovský proces vytvoří několik svých identických kopií. Každý z potomků obsluhuje požadavky nezávisle na ostatních. Toto řešení je výhodné kvůli zvýšené stabilitě. Pokud některý z procesů nefunguje dobře (nelze jej ovládat nebo spotřebovává mnoho paměti), lze jej ukončit bez vlivu na další procesy. Zvýšení stability s sebou přináší snížení výkonu. Na většině unixových systémů je vytvoření procesu a přepnutí kontextu náročná operace. Protože procesy jsou vzájemně nezávislé, nemohou jednoduše sdílet kód a data, což vede ke zvýšeným nárokům na systémové prostředky. Apache 1.3 je první verze, která pracuje i ve Windows, i když na této platformě není považován za tak stabilní jako na unixových platformách. Je to dáno tím, že server byl navrhován pro unixové platformy a přenositelnost na Windows byla doplněna později a není dokonale integrována. Jak jsem výše uvedl Apache pracuje s modulární architekturou. Zapínáním a vypínáním modulů můžu do serveru přidávat a odebírat funkce. Mohu jej upravit tak, aby byla zvýšena jeho bezpečnost a výkon. Kromě modulů dodávaných přímo se

serverem je k dispozici celá řada modulů třetích stran, které doplňují spoustu různých funkcí.

- Apache 2.0 je poslední a nejlepší verzí serveru. Jeho architektura byla oproti řadě 1.3 výrazně vylepšena. V řadě 2.0 je architektura pro zpracování požadavků oddělena do samostatných modulů, tzv. Multi Processing modulů (MPM). Apache tak lze nastavit jako čistě procesově orientovaný server, jako čistě vláknový server, nebo jako kombinaci obojího. Vlákna jsou součástí jednoho procesu a běží paralelně. Na rozdíl od procesů mohou vlákna sdílet kód a data. Vlákna jsou tak „odlehčenější“ než procesy a v případě extrémních požadavků na výkon tak vláknový server obvykle funguje lépe než čistě procesový server. Nevýhodou je menší spolehlivost serveru, protože v případě chyby ve vlákně může dojít k ovlivnění kódu a dat jiných vláken. Obsluha protokolů byla v Apache 2.0 rovněž oddělena do vlastní vrstvy. Znamená to, že je možné napsat moduly pro obsluhu jiných protokolů, než HTTP, například POP3 pro výběr pošty nebo FTP pro přenos souborů. Tyto moduly mohou využívat výhod základních funkcí serveru a dalších modulů, například autentizace a dynamického generování obsahu. Znamená to například, že můžu provádět autentizaci uživatelů POP3 proti stejné databázi, jakou Apache používá pro ověření přístupu k webovým stránkám, nebo že obsah FTP adresářů můžu dynamicky generovat pomocí PHP, CGI nebo jiných technologií. Apache 2.0 používá stejně jako 1.3 modulární architekturu, kterou navíc rozšiřuje o další mechanismus – o filtry. Filtry modulům umožňují modifikovat data generovaná jinými moduly. Mohou tak šifrovat, provádět virovou kontrolu nebo komprimovat nejen statické soubory, ale i dynamicky generovaný obsah. Bohužel, i když je modulární API obou verzí podobné, moduly pro verzi 1.3 je nutné pro použití na nové architektuře upravit. Řada hlavních modulů, například PHP nebo mod_perl, existují i pro verzi 2.0 a některé jiné, například mod_dav a mod_ssl, jsou už přímo součástí distribuce serveru. Spouštění modulů v architektuře založené na vláknech si vyžaduje jejich specifickou úpravu. Moduly dodávané se serverem Apache byly takto upraveny a jejich provoz na vláknové architektuře je

považován za bezpečný. U modulů a knihoven třetích stran to nemusí platit. Pokud bych potřeboval nějaké takové použít, budu moci Apache provozovat výhradně v režimu procesově orientovaného serveru. Díky knihovně Apache Portable Runtime (APR) nyní Apache funguje stejně dobře na Unixu i ve Windows. Tato knihovna zajišťuje abstrakční vrstvu pro funkce, které se na obou platformách liší, například pro souborové nebo síťové funkce. Přenos Apache na jinou platformu obnáší pouze přenesení této knihovny. Knihovna navíc implementuje platformně závislé optimalizace. [2]

6. PHP modul pro Apache

Nejznámější modul pro Apache je bezesporu PHP modul. PHP může být použito přímo v Apache, jako externí CGI nebo v jiných webových serverech. Je nezávislé na platformě a běží na většině unixových systémů i systémů Windows. Pokud uživatelé pracují na platformě Windows, pravděpodobně používají Internet Information Server s ASP (Active Server Pages) a server MS-SQL. Podobnou náhradou ve světě Unixu pro tuto trojici je Apache s PHP a MySQL. Ale o PHP se zmíním níže.

6.1. PHP

Jazyk PHP neboli „Hypertextový Preprocesor“ je skriptovací jazyk běžící na straně serveru.

Na počátku zrodu PHP stál v roce 1994 Rasmus Lerdorf. Původně měl tento systém sloužit pro evidování přístupů k jeho stránkám, postupem času se ale stal oblíbeným stále většímu počtu uživatelů, kteří ho začali používat. Autor proto systém rozšířil, doplnil o dokumentaci a uvolnil jej pod názvem „Page Construction Kit“. Celosvětovou proslulost si získal systém PHP/FI 2.0. Tento systém vznikl spojením předchozího programu a nástroje, který umožňoval začleňování SQL-dotazů do stránek. Další verzí bylo PHP 3.0. Tato novější verze byla rozšířena o mnoho dalších funkcí a zároveň přepracována tak, aby celý systém byl rychlejší. Nejnovější verzí je pak PHP 4.0, jehož nové jádro nese název Zend.

Výhodou PHP je možnost integrace s mnoha databázovými systémy a podpora protokolů jakými jsou IMAP, SNMP, NNTP, POP3, HTTP a mnoha

dalších. Jazyk PHP je nezávislý na jednotlivých platformách. Dnes jsou k dispozici verze PHP pro Windows, Unix a Macintosh. Dalším plusem jazyka, je jeho volná šiřitelnost, která ho favorizuje před konkurenčním programovacím jazykem ASP (komerční produkt firmy Microsoft).

7. K čemu slouží indexy

Jelikož dnes mezi nejpoužívanější databáze patří bezpochyby různé varianty SQL budu se proto v dalším textu zabývat popisem používání indexů v open source zástupci SQL – MySQL (čistě pro určitost výkladu).

Indexy slouží k nalezení řádku s konkrétní sloupcovou hodnotou. Bez indexování by MySQL muselo začít na prvním řádku a poté číst přes celou tabulku záznam po záznamu k nalezení relevantního řádku. To znamená, že výpočetní náročnost roste s rozměry tabulky. Pokud má tabulka pro sloupec index, který figuruje v dotazu, MySQL se může rychleji rozhodnout pro pozici pro hledání, třeba i od prostřed dat bez potřeby prohledávat celá data. Pokud tedy má tabulka 1000 řádků je takovéto prohledávaní nejméně stokrát rychlejší než čtení sekvenční. Pokud ovšem potřebujeme přístup k více řádků vyplatí se sekvenční čtení (z důvodu sekvenčního fungování disku). Proto je vhodné používat indexy u sloupců, podle kterých se vyhledává, třídí nebo podle kterých se tabulky spojují. Organizace toho jak jsou prvky v indexu provázané podle svého pořadí při řazení (ať už se jedná o číselné, nebo řetězcové sloupce), tak indexy umožňují také rychlé seřazení tabulky podle sloupců, nad kterými je index definován. Tím pádem umožňují i rychlé vybrání minima a maxima. Indexy nad řetězcovými sloupci umožňují také rychlejší vyhledávání pomocí operátoru LIKE, avšak pouze v případě, kdy je znám začátek hledaného výrazu - tedy např. X LIKE 'text%' využití indexu dovoluje, X LIKE '%text%' ne. Jelikož správa indexů má určitou režii při každém vkládání záznamu nebo jeho mazání, měli bychom se indexům vyhnout u tabulek, do kterých se převážně vkládá a jen zřídka kdy čte. Příkladem mohou být nejrůznější logy.

8. K čemu slouží fulltextové indexy

Tím, že se jedná o indexy slouží především k urychlení vyhledávání. Tím, že jde o fulltextové indexy, tak slouží k urychlení fulltextového vyhledávání. Už z podstaty je jasné, že index lze vytvořit nad sloupci s typy dat TEXT, CHAR nebo VARCHAR. Proč ale vytvářet fulltextové indexy, když text lze vyhledávat i bez

nich? Je to především proto, že pokud chci vyhledávat pomocí normálního vyhledávaní (klausule LIKE) je to dosti náročné. V případě složitého textu k vyhledání (nebo různých kombinací slov – chci najít to kde je toto slovo, ale zároveň kde není jiné slovo, apod.) se dostáváme do dosti krkolomných formulací dotazu. To odstraňují fulltextové indexy a vyhledávání pomocí nich (viz níže). Ale toto by se ještě dalo obejít vhodným psaním dotazů. Hlavní devizou fulltextových indexů je schopnost dotazu podat informaci o relevantnosti výsledku (nejrůznější ukazatele – např. četnosti výskytu). Ale o tomto si můžete přečíst níže.

9. Fulltextové indexy v MySQL

Jednou ze základních přidaných hodnot MySQL databáze je možnost využití fulltextových indexů. Hned na úvod je třeba říci, že tento index bude fungovat pouze u tabulek typu MyISAM. Což není problém, protože MyISAM je nejpoužívanější formát uložiště dat (storage engine) v databázovém systému MySQL. Je následovníkem formátu ISAM (Indexed Sequential Access Method). Databázová tabulky typu MyISAM se skládá ze tří souborů:

.frm - definice tabulky

.MYD - datový soubor

.MYI - indexový soubor

Jedná se také o standardní formát při vytváření tabulky, takže pokud cíleně toto nastavení nezměníme při vytváření tabulky tak používáme právě formát MyISAM. [3]

Již výše bylo popsáno k čemu slouží fulltextové indexy. Pro vyhledání se používá následující syntaxe:

SELECT * FROM *tabulka* WHERE MATCH(*sloupec1, sloupec2, …*) AGAINST('*výraz*');

Toto vyhledávání je ovšem v několika směrech omezeno. Lze vyhledávat pouze celá slova. Slovo musí být delší než tři znaky. A pokud je hledané slovo na více než polovině řádků tak výsledek není zobrazen. Pokud je hledaných slov více jsou výsledky řazeny dle relevantnosti. Relevantnost se vytváří jako další, dočasný sloupec, který lze klasickou SQL syntaxí zobrazit. Z výsledků jsou také

vyřazena slova z tzv. „Stopwords“. Jde o seznam slov, která jsou také vyřazena z výsledků hledání.

Další možností fulltextového vyhledávání je tzv. Boolean vyhledávání. Jde o jakousi nadstavbu, která vylepšuje vyhledávání o možnost logického vyhledávání, respektive za pomoci logických (booleanovských) operátorů. MySQL rozezná tento druh vyhledávání podle přidané klauzule IN BOOLEAN MODE v části AGAINST. Konkrétně AGAINST(‘*výraz*’ IN BOOLEAN MODE). Hlavní rozdíl proti prvnímu způsobu vyhledávání je v tom, že zde není uplatněno 50% omezovací pravidlo, tj. jsou zobrazeny i ty výrazy, které se vyskytují ve více jak polovině řádků. Tím, že se jedná o booleanovskou logiku jsou zde využívány následující operátory:

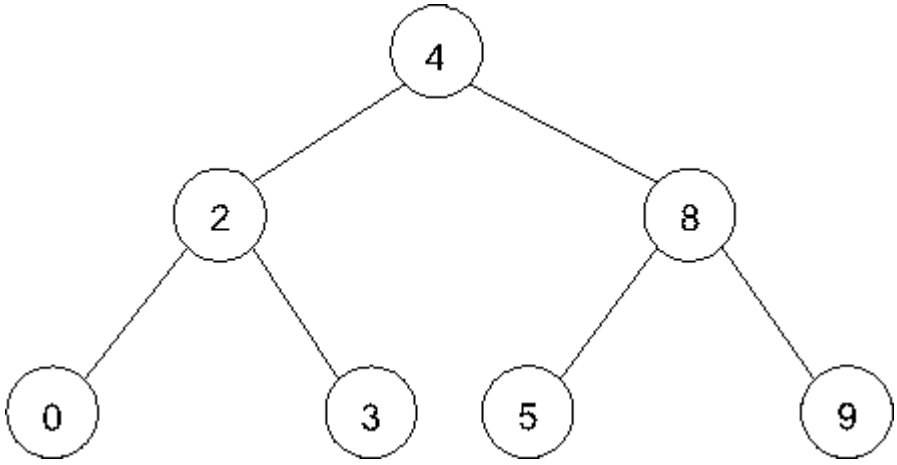
Operátor	Význam
+	slovo musí být přítomno
-	slovo nesmí být přítomno
*	doplňuje slovo o libovolný počet znaků, může být použito pouze na konci hledaného slova
( )	slučovací operátor do podvýrazů
" "	hledá celou frázi
>	zvyšuje důležitost slova
<	snižuje důležitost slova
~	negování váhy slova

[4]

10. B-Stromy

Většina MySQL indexů (konkrétně PRIMARY KEY, UNIQUE, INDEX a FULLTEXT) je uložena v tzv. B-stromu

Výše jsem uvedl pojem B-strom, tj. Binární strom je tedy na místě ho vysvětlit. Pojmem binární vyhledávací strom označujeme binární strom, jehož uzly obsahují vzájemně porovnatelné klíče a jenž je uspořádaný takovým způsobem, že klíče všech uzlů levého podstromu jsou menší nebo rovny klíči aktuálního uzlu a klíče všech uzlů pravého podstromu jsou větší než klíč aktuálního uzlu (viz obr.níže). Pokud nepřipouštíme vícenásobný výskyt klíčů, pak na obou stranách bude mezi klíči platit ostrá nerovnost. [5]



Obr. 1 – příklad B-stromu

11. Fulltextové indexy v PostgreSQL

Výše uvedené principy jsou používány v MySQL. Dále se budu zabývat způsobem implementace fulltextových indexů v PostgreSQL. U této databáze je totiž fulltextové vyhledávání provedeno odlišným způsobem než u MySQL. Tímto problémem se zabývá projekt TSearch2. Jedná se o modul, který je schopen jako dovolit rozšířit v PostgreSQL databázích používání fulltextových indexů. Hlavní myšlenkou TSearch2 je ušetřit čas při zpracování „textového“ dotazu tím že si tabulku předzpracujeme. Toto předzpracování se skládá z těchto částí:

rozdělení dokumentu na slova

lingvistické úpravy slov pro získání lexémů (viz níže)

uložení dokumentu v optimalizované formě pro hledání

TSearch2 poskytuje operátorům fulltextového hledání dva nové datové typy, které reprezentují dokument a dotaz- tsvector a tsquery. TSearch2 modul poskytuje indexování dokumentu podle slov, které obsahuje k poskytnutí velice výkonného hledacího mechanismu v dokumentu, který obsahuje kombinace hledaných slov. Příprava dokumentu zahrnuje dva kroky:

- Vytvoření seznamu slov, které dokument obsahuje. Redukovat dokument do tsvector což je seznam slov, která se v dokumentu objevují. Tento proces má mnoho voleb, protože není dán jasný předpis jak má vektor přesně vypadat, tj. slova se nemusí do vektoru kopírovat přesně tak jak jsou reprezentována

v dokumentu. To dává značný prostor pro redukci velikosti indexu. Jako příklad může sloužit třeba nezařazování frekventovaných slov do indexu, respektive jejich zařazení do stop words. Stejně tak je vhodnější ukládat do vektoru pouze základní tvary slov (neukládá se slovo v třeba v jiném pádu). Protože jsou možné různé modifikované formy slov, forma uložená ve vektoru se v anglicky mluvicích nazývá lexemes, tedy česky lexém (základní jednotka slovní zásoby).

- Vytvoření indexu dokumentu pomocí lexémů.

Pokud je dokument indexovaný vyhledávací mechanismus zahrnuje:

- Redukci hledaných výrazů na lexémy. Pokud chceme použít tsquery k vyhledání výrazu musíme tento výraz vyjádřit jako lexém případně jako booleovskou kombinaci lexémů. Proč je toto pro TSearch2 důležité je například to, že i rozdílné velikosti písmen vyhodnotí modul jako diferenci. Pokud je např. slovo lední reprezentováno v tsvector lexémem led je třeba při vyhledání slova lední upravit toto slova na lexém, protože by jinak nedošlo k vyhledání.

- Získání dokumentu, který odpovídá dotazu. Běžící SELECT…WHERE query @@vector na tabulce se sloupcem vector vrátí dokument, který odpovídá dotazu.

- Prezentace výsledku. Tato konečná etapa nabízí opět několik možností jak změnit dokumenty do vektorů. Je možné setřídit dokument podle toho jak moc dokument odpovídá hledanému výrazu-relevance. Vytvářet nadpisy-titulky- pro každý dokument ukazující některou frázi z dokumentu obsahující hledaný výraz. Omezovat počet získávaných výsledků.

Tento poslední bod lze provádět dvěma způsoby. Buďto necháme vykonání na samotné databázovém serveru nebo tuto část převedeme na nějaký jiný stroj. To už záleží na designérovi samotného datového systému.

Při vytváření vektoru z dokumentu jsou vynechány mezery a interpunkční znaménka. Slova jsou ukládána ve formě lexémů (tedy základních slovních

jednotek). Ve vektoru jsou seřazena dle abecedy a u každého lexému je uložena jeho pozice v původním dokumentu. Uchování pozice slova v původním dokumentu je ve vektoru volitelné. Uchování této pozice je důležité pokud chceme využívat řadící funkce modulu TSearch2. Tyto funkce jsou pak používány k seřazení dokumentů podle důležitosti na základě četnosti výskytu hledaného termínu v dokumentu. Ale pokud nechcete využívat řazení dokumentů podle důležitosti nebo budeme využívat vlastní procedury lze ušetřit místo ve vektoru pomocí příkazu strip - SELECT strip(to_tsvector('Text')). Stejná direktiva bez strip vytvoří vektor s pozicí, tato vytvoří vektor bez pozic.

Protože uživatel chce číst dokument a ne vektor musíme mu poskytnout nějakou cestu k celému textu kteréhokoliv přístupného dokumentu. Buď ukládáním vstupujícího textu do databáze nebo ukládáním nějakého identifikátoru (např. URL, jméno souboru). Jak je poznamenáno výše, výsledky by měly být poskytnuty uživateli setříděné dle relevance. Pokud vynecháme ve vektoru pozici slova v dokumentu tak můžeme buď použít řadící funkci PostgreSQL ORDER BY nebo můžeme odvodit vlastní vektor a provést vlastní setřídění. Pokud se rozhodneme vynechat ve vektoru pozici slova potom je na našem vlastním rozhodnutí se o relevanci, užitím buď celého textu nebo jiné informace o každém dokumentu které máme k dispozici. Každý dokument, který je vrácen jako výsledek vyhledávání budeme nejspíše chtít zobrazit jako jakousi souhrnou informaci. K tomuto nejlépe poslouží již zmiňované titulky. Jde tedy o jakési zobrazení krátkých výňatků, které ilustrují jak hledaný výraz dokument používá, respektive v jakém kontextu jej používá. Titulky jsou většinou generovány z celého textu dokumentu. Ne tedy pouze z místa, které určí pozice uložená ve vektoru tsvector, protože zde by byla prázdná místa po stopwords, po interpunkčních znaménkách, předložkách a tím pádem by byl výstup nesrozumitelný. Pokud máme uložený celý text v databázi lze pro vygenerování titulku velice jednoduše použít funkci TSearch2. Pokud máme uložený celý text dokumentu někde mimo databázi, můžeme buďto celý text získat ze zdroje a na něm použít funkci TSearch2 pro generování titulku (je to obdoba situace kdy máme uložen text v databázi jen v tomto případě dojde k jistému zdržení při dotazu na celý text a jeho následné získání) nebo použít vlastní kód titulku mimo databázi.

I přestože může výsledek hledání obsahovat stovky až tisíce dokumentů je rozumnější prezentovat v jeden okamžik uživateli pouze deset až dvacet

výsledků. Tohoto můžeme poměrně snadno dosáhnout pomocí limitování dotazu klauzulí LIMIT a OFFSET. Použití této klauzule sebou ale nese dva problémy. Prvním je značné namáhání běžícího dotazu pokaždé při vytváření uživatelského pohledu pro každou stránku (zobrazíme první stránku LIMIT=10 a OFFSET=0, další stránka bude mít opět deset položek ale OFFSET=10, atd.). Pro malé soubory dokumentů nebo při málo vytíženém běhu serveru to zřejmě nebude problém, ale dopad to může mít mnohem vyšší pokud hledání musí stále opakovaně řadit a třídit desetitisíce výsledků na stále zaneprázdněném serveru. Namísto vybírání jen jedné stránky s výsledky, je vhodnější použít LIMIT a OFFSET k získání několika desítek či stovek výsledků, které můžeme umístit do cache serveru. Další otázkou k vyřešení je konzistence dočasně ukládaných dat (cache). Pokud se databáze změní v momentě kdy si uživatel prohlíží výsledky pak by mohl dokument zobrazovat a skrývat stránky, které mají ve výsledku být či naopak nemají. Toto je poměrně závažný problém, protože zatímco spousta databází je statická a aktualizována velice sporadicky, uživatel vyhledává v souboru dynamických dokumentů. Potom ho samozřejmě nemohou uspokojit takovéto výsledky. Toto ovšem není problém pouze databázového světa, ale jakékoliv technologie kde se využívají informace. Ty totiž povětšinou v momentě jejich prezentace jsou morálně zastaralé. Je zde ovšem určitý prostor pro správce datových úložišť pro optimalizaci aktualizačních algoritmů aplikovaných na datových strukturách.

Co se týká řazení používá TSearch2 dvě funkce. Obě využívají pozici slova uvedenou v tsvector, proto nelze použít setřídění za pomoci vektoru kde jsme odstranili pozice pomocí funkce strip(). Jedná se funkce rank() a rank_cd(). Funkce rank() (jedná se starší verzi) je charakteristická tím, že přiřazuje váhu slovům z různých částí dokumentu, které máme k dispozici. rank_cd() používá současně techniku vážení výsledku a také povolení rozdílných váh z různých částí dokumentu. Obě řadící funkce dovolují specifikovat svým posledním volitelným argumentem, jestli chceme jejich výsledek normalizovat (tj. zda by mělo být řazení přizpůsobeno délce dokumentu):

0 – nepřizpůsobovat

1 – rozdělení dokumentu logaritmicky dle délky dokumentu

2 – rozdělení dokumentu v rovných dílech

Funkce rank_cd() provádí pokusné měření nazývané cover density ranking, které ohodnocuje dokument na frekvence výskytu hledaných termínů blízko sebe. V současné době podporuje TSearch2 čtyři rozdílná písmenná označení váhy. Jde o D, což je základní váha a pak A, B, C. Všechny vektory vytvořené za pomoci to_tsvector() přiřazují váhu D každé pozici. (ta ale jako základní hodnota není při prohlížení vektoru zobrazována).

TSearch2 operuje také nástrojem pro vlastní obsazení vektoru pokud by nám nevyhovovalo rozdělení přímo od modulu. Jak bylo řečeno výše použijeme-li funkci strip() odstraníme z vektoru mezery, interpunkční znaménka a změníme přípony slov tak aby ve vektoru figurovalo slovo ve své základní formě. Zatímco je tato funkce často žádána lze využít funkcionalit TSearch2 pro vlastní zpracování vektoru respektive lexémů. Pokud si uvědomíme co funkce strip() odstraňuje, velice záhy nás napadne kdy se vlastní vytváření vektoru z lexémů může hodit. Pokud chceme, aby v lexémech figurovaly apostrofy, zpětná lomítka, potřebujeme aby byla v některém lexému záměrně ponechána mezera nebo prostě nějaký znak či nezvyklé písmeno, lze toho docílit. Na největší problém zde narazíme u apostrofu a zpětného lomítka. Je tomu tak, protože se jedná o jakési systémové značky PostgreSQL. Jednoduchou uvozovkou jsou označovány řetězcové konstanty a zpětné lomítko slouží s písmenem jako systémové konstanty.

Funkce to_tsvector() redukuje dokument do lexémů ve dvou stupních. V prvním stupni parser separuje vstupní dokument do krátkých sekvencí text, které se nazývají tokeny. Každý z těchto vytvořených tokenů je zpravidla slovo, mezera nebo kousek interpunkce (většinou ne pouze interpunkční znaménko, ale většinou ve vazbě s mezerou za tímto interpunkčním znaménkem). Můžeme se ale také setkat s parsery, které vracejí větší části. Jedná se většinou o speciálně upravené parsery, které se používají například na zpracování HTML tagů. Tyto tagy potom parser vrací jako celé značky a ne jako značky ostré nerovnosti plus to co je mezi těmito značkami (např. '
' bude uloženo ve vektoru v této formě a ne jako '<', 'BR', '>'). Každý token vrácený parserem je buďto vyřazen (např. mezera pokud ji nepotřebujeme) nebo je „protažen“ slovníkem, který token zkonvertuje do lexému. Výsledné lexémy jsou posbírány do vektoru. Volba parseru a slovníku, který se použije je řízena volbou v konfiguraci. TSearch2 modul přichází s několika konfiguracemi a my můžeme dodefinovat naše vlastní. [6]

12. GiST a GIN indexy

Pokud mluvíme o TSearch2 projektu je třeba zároveň s tímto projektem a dvěmi hlavními osobami, které za ním stojí (Teodor Sigaev a Oleg Bartunov – členové výzkumného týmu Moskevské univerzity-Moscow State University-Sternberg Astronomical Institute), se zmínit o projektu GiST a GIN.

GiST znamená „zobecněný vyhledávací strom" a jedná se o zobecněnou formu indexování. Navíc ke GIS indexování, GiST je používán pro hledání na všech druzích nepravidelných datových struktur (číselná data apod.), která nejsou přístupná pro normální B-Tree (B-Stromy viz výše) indexování. Tabulka s GIS daty obvykle přesahuje několík tisíc řádek a my budeme chtít vyhledávání zrychlit použitím indexů. Vytvoření prostorových indexů je celkem náročná početní záležitost a jak uvádějí některé zdroje, tabulku s jedním milionem řádek a 300 MHz Sunovským počítačem trvalo zindexovat kolem jedné hodiny.

GiST indexy mají oproti R-Tree (R-Trees - R stromy - rozděluje data do obdélníků a ty dále rozděluje na menší a menší části. R-Trees jsou používány některými databázemi pro indexaci GIS dat – geografický informační systém, naplněn prostorovými informacemi - avšak PostgreSQL R-Trees implementace není tak robustní jako implementace GiST) indexům dvě výhody. Za prvé, GiST indexy jsou tzv. "null safe", můžeme indexovat sloupce, které zahrnují nulové hodnoty. Za druhé, umožňují indexovat jen důležité části objektu, který je větší než 8k, což dělá R-Tree indexům velké problémy. [7]

GIN znamená „zobecněný převrácený-invertovaný- index“. Jedná se o strukturu indexu ukládající sadu (klíč, „posting list“) párů, kde „posting list“ je souhrn řádků kde se vyskytuje klíč. Každá indexovaná hodnota může obsahovat spoustu klíčů, stejně tak ID řádku může ukazovat na násobné „posting lists“. To je zobecňováno v tom smyslu, že GIN index si nepotřebuje být vědom operace, která ho urychluje. Místo toho se používají uživatelské strategie definované pro jednotlivé datové typy. Jednou z výhod GIN je, že dovoluje vyvíjet uživatelské datové typy s vhodnými přístupovými metodami na základě znalosti okruhu datových typů než na základě znalosti databáze (stejně tak u GiST).

Vnitřně obsahuje GIN index B-Tree index vytvořený pro klíče, kde každý klíč je elementem indexované hodnoty (např. členem nějakého pole) a kde každá

n-tice na listu stránky je buď ukazatel do B-Tree přes hromadu ukazatelů nebo seznam hromady ukazatelů pokud je seznam dostatečně malý. [8]

13. Měření výkonnosti

V dnešní době se mnoho výrobců předhání v závodě zvaném výkon. Nejdominantnější je to asi na poli grafiky. Výkonný počítač je dnes posuzován podle toho jak obstojí v nějakém testu grafického výkonu. To je také důvod proč tvůrci her (největší hráč na poli grafiky) a výrobci grafických karet jsou velkou částí určující segment v posuzování toho co je výkon. Jedním z dalších určujících faktorů v dnešním trendu zvyšování výkonu je výpočetní výkon. Dalo by se říci, že týden co týden oznámí některá firma prolomení další x-Herzové hranice. Výrobci hardwaru ale nejsou jediní, kteří se snaží profitovat z chtíče uživatelů vlastnit či obsluhovat nejvýkonnější „stroj“. Dalšími, kteří chtějí ukrojit z koláče tohoto odvětví jsou nejrozličnější úpravci již hotového hardwaru. A je jedno jestli se jedná o například přetaktování (paměti, procesory, grafické čipy) či o dodavatele extrémních chladících systémů.

Proč o výkonu mluvím? Protože se v následujících odstavcích budu věnovat výkonnosti a měření výkonu databází. Již sem zmiňoval, že dnešní komerční svět se neobejde bez databází. A je jedno jestli se jedná o stavebnictví či o automobilový průmysl. Stejně tak je jedno jestli se jedná o databázi určenou pro webový obchod firmy sídlící na okraji města v garáži nebo o nadnárodní koncert čítající desetitisíce skladových položek, které potřebuje udržovat v nějakém datovém skladu. Pro oba subjekty uvedené v příkladu, stejně tak jako pro každého jiného uživatele je důležité jestli jeho využívaní databázového systému je optimální a databáze mu poskytuje maximální možný výkon. Proto vzniklo několik systému (a zároveň s nimi firem), které se zabývají testováním (benchmarkingem) databází.

Testování databází je velice obsáhlá a složitá problematika, skýtající mnoho pohledů na tuto problematiku. Jako jednu z prvních důležitostí musím uvést, že obecné povědomí o faktu, že rychlá databáze se automaticky rovná výkonná databáze je dosti mylné. Testovat totiž pouze rychlost vykonání něčeho bez ohledu na ostatní součásti celého souboru této problematiky je dosti absurdní. Při posuzování výkonnosti databáze je nutné zohledňovat spoustu jiných faktorů. Můžeme jimi být například administrace databázového serveru. K čemu bude zákazníkovi, že jeho dotaz je vykonán rychleji na jeho zvolené

platformě než na konkurenční, pokud administrací serveru stráví technik této firmy každý den dvě hodiny? Některé firmy mohou potřebovat poněkud sofistikovanější řešení. Schopnost práce na datech, která jsou velice různorodá co se datových typů týká. Bezpečností algoritmy při práci s daty. Schopnost zamykání dat při práci s nimi jedním uživatelem pro ostatní. Ale zároveň neznemožnění třeba jen pasivní práce s daty se kterými se právě pracuje někde jinde. Schopnost zabránit deadlockům (jedna úloha čeká na dokončení druhé, přičemž ta zase čeká na ukončení té první).

Jen okrajově zmíním jakým způsoben lze tento jev eliminovat. V momentě kdy databáze začne alokovat zámky, které pro danou transakci použije se provádí analýza zacyklenosti grafu, a při objevení cyklu se iniciuje deadlock. Rollbackem (rolování zpět – veškeré operace se zapisují do log souboru a dle toho postupu se při zjištění nějakého problému provedou ty samé operace pouze však v obráceném pořadí) se ta méně důležitá transakce vrátí do původního stavu.

Dále je nutné si uvědomit rychlost čeho vlastně měříme? Pokud by nás totiž zajímala rychlost samotné databáze tak bychom museli měřit přímo z konzole této databáze. Pokud totiž budeme měřit například pomocí PHP skriptu neměříme pouze výkonnost samotné databáze, ale povětšinou také rychlost nejrůznějších ovladačů. Do konfigurace PHP je totiž nutné zaimplementovat jakési konektory, které se starají a samotnou obsluhu databáze. V prostředí Microsoft Windows (MSW) se jedná o dynamické knihovny s příponou .dll. Jejich linuxovým ekvivalentem jsou sobory s příponou .so. To kterým konektorem budeme obsluhovat tu kterou databázi určujeme v konfiguračním souboru php.ini a to v sekci extensions, kde jednotlivé knihovny povoluje nebo zakazujeme. Pokud tedy bude knihovna napsána neoptimalizovaným kódem může tím pádem být čas měření značně zkreslený.

Schopnost škálovatelnosti, možnost psaní vlastních obslužných rutin, rychlost a volby replikace databáze, možnosti jemného vyladění dle přesných a konkrétních potřeb uživatele, to vše jsou vlastnosti databází na které není radno zapomínat při výběru a implementaci některé z databázových platforem a které ovlivňují jakým způsobem a jak výkonně nám tyto budou sloužit.

V neposlední řadě je zapotřebí zmínit fakt, že samotné testování na databázích se neprovádí pouze pro zjištění výkonnosti databáze, ale také pro

pohled z druhé strany. Pro zjištění výkonnosti serverů při provádění stejných sad testů na různých konfiguracích serverů nebo na různých serverech co se týká výrobce.

Firem, které se zabývají benchmarkingem databází je poměrně dost. Bohužel většina se zabývá spíše enterprise řešeními. Z čehož vyplívá, že nahlédnout do jejich kuchařek je nemožné. Mezi firmy, které testují výkonnost (a nejen databází) je vhodné zmínit americkou firmu Quest Software. Zároveň je nutné uvést českého zástupce, firmu Per4mance, protože využívá softwaru z portfolia firmy Quest a to Benchmark Factory for Databases (BMF). Tento software je určen pro benchmarking těchto platforem - Oracle, SQL Server, DB2, Sybase, MySQL a všech dalších zástupců, kteří jsou schopni pracovat přes ODBC (Zkratka ODBC vznikla složením počátečních písmen z anglického názvu Open DataBase Connectivity. Jedná se o standardní aplikační rozhraní (tzv. API) pro přístup k datům). [9] BMF k testování buď uživatelem vytvořené testy nebo standardní testy, jako např.: AS3AP, Scaleable Hardware, Set Query, TPC-B, TPC-C, TPC-D, WebStone a Wisconsin. Bohužel BMF se používá pouze na MSW. Firma pracuje na poli linuxu, ale pouze při srovnávání např. výkonnosti serveru.

Protože dnes na trh vstupuje stále více softwarových výrobků na benchmarking databází, bylo nutné vytvořit jakési standardy, aby bylo možné výsledky z různých softwarů mezi sebou porovnávat.

Proč hovořím o standardech. Standardy potažmo standardizace? Až v dnešní době lidé, respektive podniky, tento termín zažínají chápat. Chápat jeho důležitost. Na tomto má největší zásluhu automobilový průmysl, konkrétně asijské firmy. Celá filozofie standardizace spočívá v jejím dopadu na dva aspekty. Bezpečnost a kvalitu. Tyto dvě záležitosti mají zásadní vliv ve vývoji technologií a to nejen v automobilovém průmyslu. Lze si jejich význam ukázat na zcela obyčejném případě ze života. Kde v domácnosti používáme standard? Můžeme uvést recept na přípravu nějakého pokrmu. Na ukázku bude ale lepší svíčková. Jde o výrobní technologii, která se předává z pokolení na pokolení. Proč? Protože ten daný recept je kvalitní což je dokázáno tím, že se předává z generace na generaci. A pokud by chtěl někdo namítnou, že mu uniká význam bezpečnosti? To že nám měl recept kdo předat je jasným důkazem toho že se naši předkové recepturou neotrávili a tím pádem je bezpečná.

O to aby se jakési standardy zavedli a byly také vzaté za jakési etalony v oblasti benchmarkingu se postarala americká rada The Transaction Processing Council (TPC). Ta se skládá z osmi společností. Tato rada byla zformována v roce 1988. Rada byla zformována, aby dovolila prodejcům hardwaru porovnat jejich produkty užitím standardních sad testů známých jako benchmarky. Tyto sady testů sou provedeny na produktech výrobců a jejich výsledky jsou publikovány, aby mohly společnosti optimalizovat svoje systémy pro jejich zrychlení a zvýšení celkového výkonu. Je třeba si uvědomit, že tyto testy neodrážejí reálnou situaci použití testovaného produktu. Benchmarking je jakýsi experimentální výzkum, který se snažíme co nejvíce přiblížit realitě. TPC stojí za již zmíněnými benchmarky AS3AP, TPC-A, TPC-B, TPC-C, Wisconsin. Je třeba také zdůraznit, že prioritou TPC není benchmarking databází, ale za pomoci databází a sad testů na nich testovat hardware. To ale nemění nic na tom, že tyto testy pracují s databázemi a pokud se použijí na stejném hardware lze jimi otestovat právě výkonnost databáze.

Všechny zmiňované benchmarky pracují na základě dvou proměnných, které jsou používány ve spoustě experimentálních nastavení. První je sada nezávislých proměnných nazývaná experimentální faktor. Ta ovlivňuje výkon databáze. Experimentální faktor v sobě zahrnuje velikost databáze na které se test provádí a zároveň složitost dotazů běžících na databázi. Druhá je sada závislých proměnných nazývaná výkonnostní matice, kde jsou sbírána data během testu. Výkonnostní matice se soustřeďuje na zahrnutí času zpracování transakce stráveného při vykonávání dotazu na databázi pevné velikosti.

Standardní seznam benchmarků může být rozdělen do dvou základních kategorií. Jedná se o On-Line Transaction Processig (OLTP) benchmark a benchmark relačních dotazů. Do kategorie OLTP spadají z výše jmenovaných benchmarky TPC-A, TPC-B a TPC-C. Do skupiny benchmarků relačních dotazů patří Wisconsin a ANSI SQL Standard Scalable and Portable (AS3AP).

13.1. On-Line Transaction Processing benchmark

TPC-A

Jedná se o benchmark simulující hypotetickou bankovní transakci v síťovém prostředí. Důvod zrovna pro bankovní transakci jen ten, že TPC-A byl skutečně vyvíjen kolem dřívějšího benchmarku známého jako DebitCredit. A ten byl vyvíjen pro návrh a měření OLTP v bankovním sektoru.

TPC-B

Jde o dávkou verzi DebitCredit benchmarku bez síťové a uživatelské interakce při vykonávání testu (tj. terminálově).

TPC-C

Tento benchmark se objevil v roce 1992 (v roce 1993 byl revidován). TPC-C je souhrnný benchmark emulující hypotetické třídění záznamů a seznam řídících transakcí ve výrobním průmyslu. Tento průmyslový standard, na rozdíl od TPC-A a TPC-B, se stal široce používaným benchmarkem pro měření rychlostí transakcí pro výkonné aplikace v procesu výroby.

13.2. Benchmark relačních dotazů

Wisconsin

Byl vyvinuta začátku let osmdesátých a snažil se zavést sérii testů na systémech s relačními databázemi. Hlavním argumentem proti používání tohoto benchmarku byla jeho jednoduchost a proto poskytoval nerealistické měření výkonu.

AS3AP

Testy tohoto benchmarku jsou rozděleny do následujících dvou sekcí:

Single-user testy obsahují:

Nástroje pro nahrání dat a vytvoření struktury databáze.

Dotazy navrhnuté k testování přístupových metod a k optimalizaci základních dotazů (selekce, jednoduchá spojení, agregace, rozsáhlé updaty).

Multi-user testy modelují rozdílné vytížení databáze:

OLTP vytížení.

Vytížení získáváním informací.

Kombinované vytížení zahrnující rovnováhu krátkých transakcí, oznamovacích dotazů, prohledávání relací a dlouhých transakcí.

AS3AP benchmark má několik výhod. Poskytuje komplexní, ale poddajné sady testů pro zpracování síly databáze. Náročnost pro člověka při implementaci a běhu testu je minimalizována. AS3AP poskytuje jednotnou metriku směřující k jednoznačné interpretaci výsledku benchmarku.

Současné AS3AP databáze na kterých se provádějí SQL dotazy obsahují následujících pět relací:

A_TINY: relace používaná k měření režie na databázi
A_UNIQUES: relace kde všechny atributy mají unikátní hodnotu

A_HUNDRED: relace kde většina z atributů má přesně 100 unikátních hodnot, které spolu souvisejí.

A_TENPCT:relace kde většina atributů má 10% unikátních hodnot.

A_UPDATES: relace uzpůsobená k updatům. Rozdílné distribuce používají tři typy indexů, které vytvářejí na těchto relacích.

Všechny relace mají (vyjma A_TINA) stejných deset sloupců se stejnými názvy a datovými typy.

Jen okrajově se zmíním o další možnosti fulltextového vyhledávání, které se používá.

14. Fulltextové vyhledávání-jinak než v databázích

Fulltextový vyhledávací nástroj Lucene, je dnes již vyvíjen pouze pro komerční použití. Jako poslední open source bylo vydání kolem verze 1.3. nejedná se o vyhledávací nástroj nad databázemi ale o univerzálně využitelnou utilitu pro prohledávání dokumentů.

Lucene je opensource nástroj pro generování a používání fulltextových indexů, vyvinutý The Apache Foundation. [10] Jeho původní verze byla napsána v Javě. Dnes je již předělána do .NET. Nejedná se samostatný vyhledávácí stroj a o knihovnu, která poskytuje své funkce dalšímu zpracování. Proto ji lze využít pro např. poštovní klient (indexování a prohledávání mailů), webové aplikace nebo aplikace se sadami dokomuntů např. na CD-ROMu.

Hlavní rozdíly mezi jednotlivými mechanizmy se mezi vyhledávacími stroji liší především v tom, jak ukládají indexy do invertovaného indexového souboru (GIN). Většina ukládá ve tvaru B-Tree. Místo toho, aby vytvářel jeden index, vytváří při indexování mnoho indexových segmentů, které pravidelně slučuje. Pro každý nový dokument Lucene vytváří nový indexový segment, hned ovšem dochází k reorganizaci segmentů spočívající ve slučování malých segmentů do velkých. Takto zůstává na relativně malém počtu indexových segmetů takže je zachována rychlost vyhledávání. Pro indexy, které se jen málokdy použijí je Lucene schopen vložit tyto do jednoho segmentu. Lucene při jakékoliv reorganizaci indexových segmentů neprovádí přepisování souborů. [11] Pro zabránění kolizím při čtení a zapisování, Lucene vždy vytváří nové soubory segmentu a maže ty staré.

15. Provedení testu

Na počátku popisování samotného, mnou provedeného testu, je třeba se zamyslet nad samotným významem testování. Testy popisované výše lze do určité míry považovat za dosti syntetické a málo vypovídající. Je to především z důvodu používání stále stejných mechanismů se stejnými daty. Proto se také používají spíše pro testovaní výkonnosti hardwaru (jak již bylo zmíněno), než pro měření výkonnosti samotné databáze. Už samotný pojem výkonnost databáze je zavádějící. Výkon databáze je totiž ovlivněn několika aspekty.

Na ukázku lze uvést ODBC konektory. Protože pokud do databáze přistupujeme pomocí konektoru, je výsledný výkon dán nejen pouze výkonností databáze, ale to jak „čistě“ je konektor napsán a jak je jeho kód optimalizovaný. Nemalou měrou se na výkonu podpisuje samotný hardware. Výkon celého datového skladu ovlivňuje nejenom výkon (jak rychlí máme procesor, paměti, atd.), ale také to jak máme server nastavený. Jak moc stabilní je (používaný termín up time).

Pokud bychom chtěli měřit opravdu výkon databáze, museli bychom měřit přímo z konzole (odstranění vlivu konektoru). Já jsem ale rozhodl pro jiný přístup měření výkonnosti.

15.1. Měření pomocí PHP skriptu

Rozhodl jsem se pro měření času vykonání dotazu do databáze. Pro toto změření používám PHP skript. Protože se domnívám, že tento přístup (přes PHP) je dnes nejrozšířenější. Mou snahou bylo naměřit taková data, která by nejvíce připomínala reálný provoz podniku. Poskytnou tak uživateli možnost změřit si rychlost databáze na svých vlastních datech (už jenom jejich struktura a rozložení v jednotlivých tabulkách ovlivňují výkon databáze) a svém vlastním hardwaru. A na základě naměřených dat se rozhodnout pro tu nejvhodnější variantu právě pro daný podnik. Co bude totiž společnosti platné, že někdo někde na nějakém úplně jiném hardwaru s jinými daty prohlásí, že nejvýkonnější je ta a ta databáze, když to v reálu pro daný podnik nebude pravda. [12]

Měřím nejenom vykonání dotazu, ale celou transakci – otevření spojení, vykonání dotazu, uzavření dotazu. Dotaz provádím v cyklech 20, 50 a 100,. a každou z těchto sad provádím desetkrát. Z těchto hodnot počítám průměr jak dlouho trvá jedna transakce.

Skript slouží pouze pro změření času. Lze ho ale upravit například pro uživatelem zadávání dotazu nebo hledaného slova. Fantazii se meze nekladou.

15.2. Data pro testování

Jako data pro testování jsem použil datový sklad liberecké firmy GEOS A.G.T.. Jde o jednu tabulku o 13 885 řádcích a struktuře sloupců dle Přílohy 1 - Názvy sloupců v tabulce a jejich datový typ. Jsou zde zaznamenány informace o produktech, kterými se společnost zabývá. V tabulce je použita celá škála

datových typů. Pro mé testování jsem použil sloupec, kde je uložen textový popis daného artiklu. Tuto tabulku jsem naimportoval do databáze MySQL a PostgreSQL na kterých jsem provedl měření.

15.3. Použitý hardware a software pro testování

Notebook Toshiba Satellite SA10-S103

CPU-Intel Celeron 2.00GHz, 256MB RAM, 30GB HDD

Debian kernel_image_2.6.8-2-686

Apache 1.3.33-6 sarge 3

PHP4 4:4.3.10-19

MySQL 4.1.11 Debian 4 sarge 7

PostgreSQL 7.5.22

15.4. Jednotlivé SQL dotazy

SELECT COUNT(*) FROM test_tab WHERE Nazev LIKE('hled. slovo‘);

SELECT COUNT(*) FROM test_tab WHERE MATCH(Nazev) AGAINST('+hled. slovo‘);

SELECT COUNT(*) FROM test_tab1 WHERE Nazev LIKE('hled. slovo');

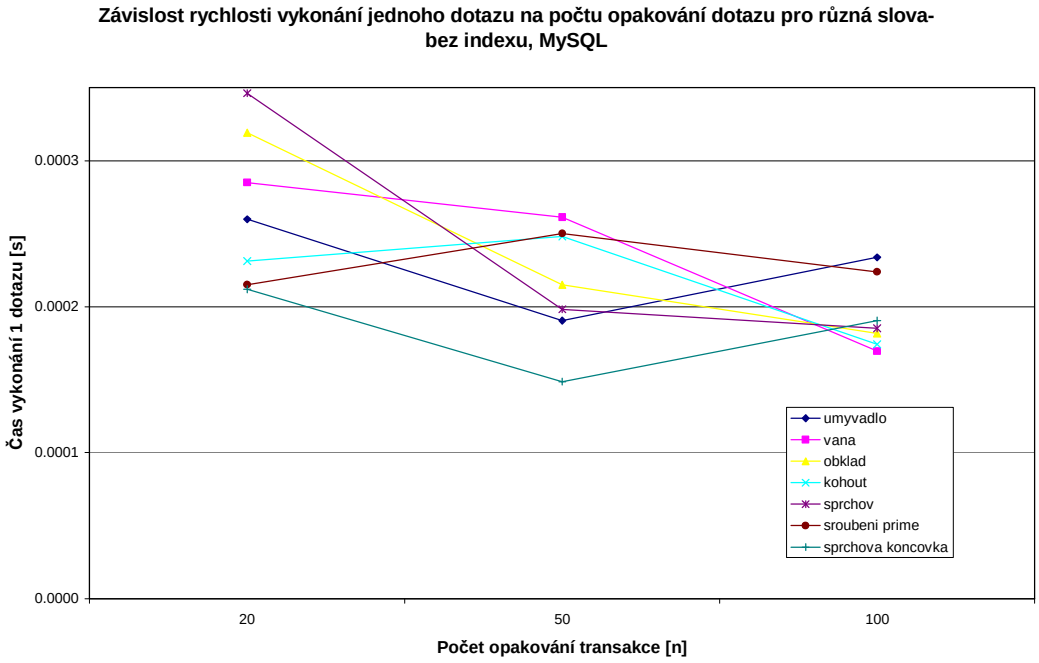
SELECT COUNT(*) FROM test_tab1 WHERE v @@ to_tsquery('hled. slovo');

15.5. Hledaná slova

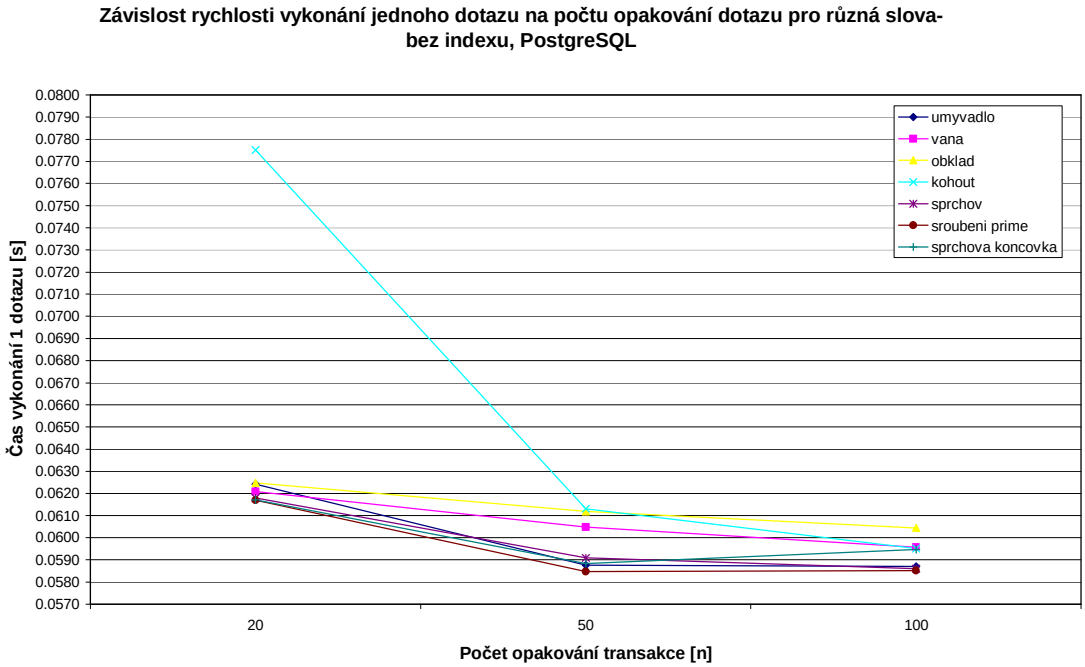
Hledané slovo	Výskyt [%]
umyvadlo	1.169
vana	0.734
obklad	1.242
kohout	0.565
sprchov	1.368
sroubeni prime	0.056
sprchova koncovka	0.084

Tab. 1 – Hledaná slova a jejich výskyt
(celkový počet slov ve sloupci 60890)

Tabulky s naměřenými hodnotami lze nalézt v přílohách, protože zde by neposkytnuly adekvátní výstup. Na tomto místě se spíše zaměřím na popis grafů, které vycházejí z těchto hodnot.

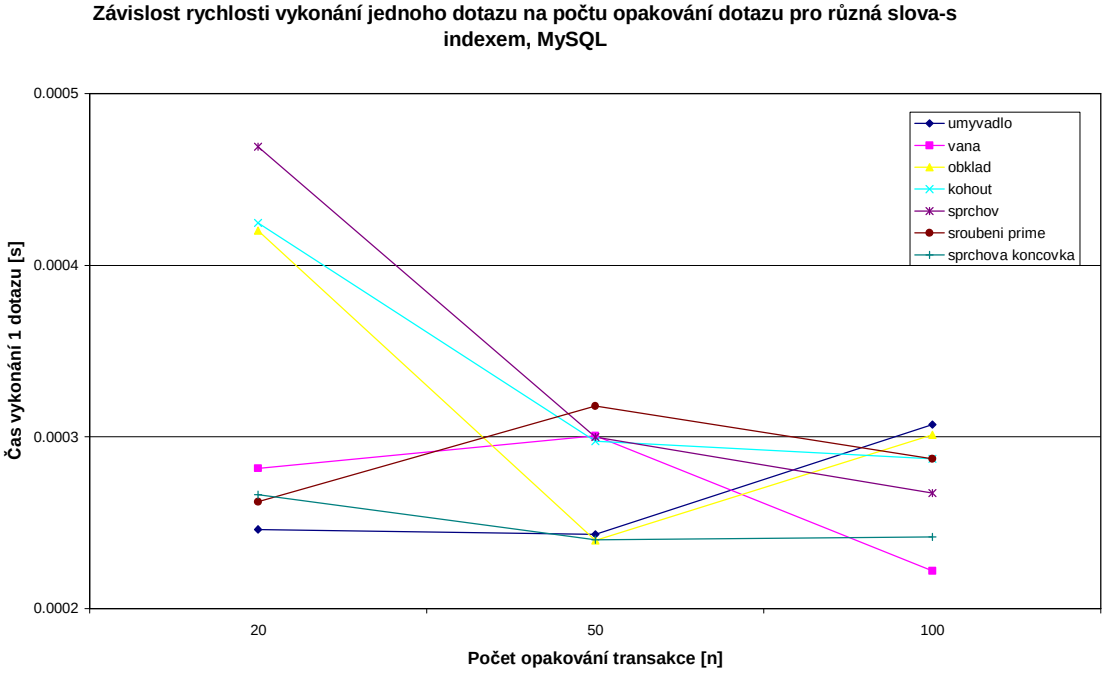


Graf č. 1 – Výsledky bez použití indexu

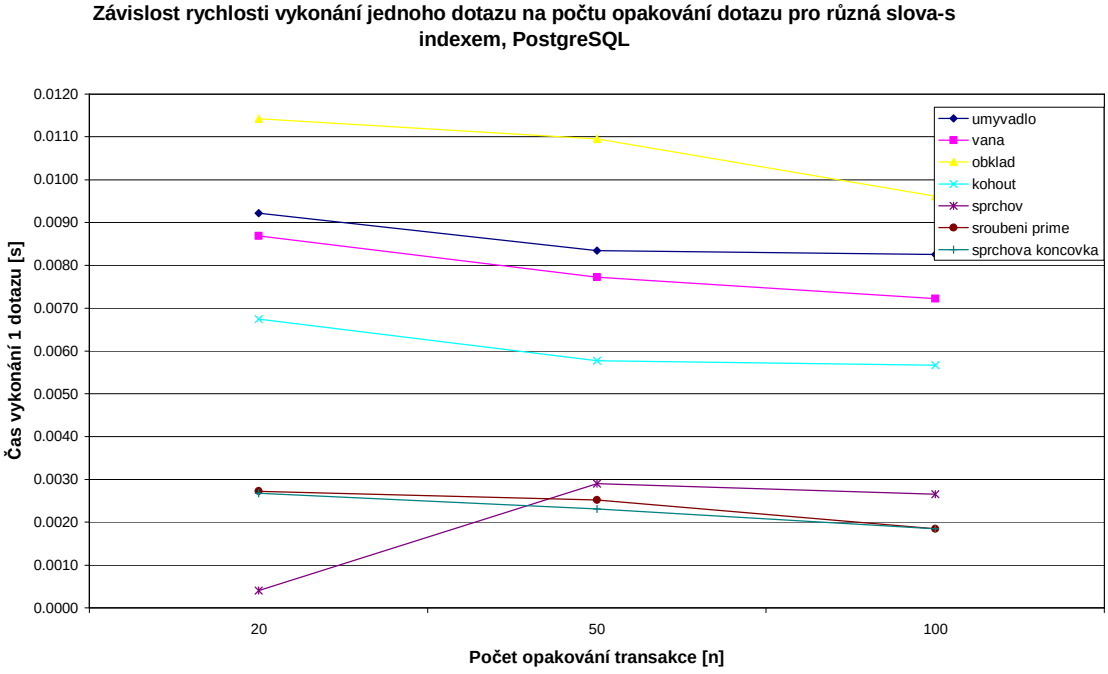


Graf č. 2 – Výsledky bez použití indexu

U MySQL je při prvním vykonání patrný vliv výskytu hledaného slova. Čím více slov je k nalezení, tím delšího času vykonání dotazu dostaneme. Stejného jevu bychom jistě viděli u PostgreSQL, ale zde je bohužel tento jev smazán celkově vyššími časy vykonání. Naopak u PostgreSQL je patrný vliv využívání cache dotazu (je po prvním vykonání uložen výsledek právě pro urychlení následného dotazu).

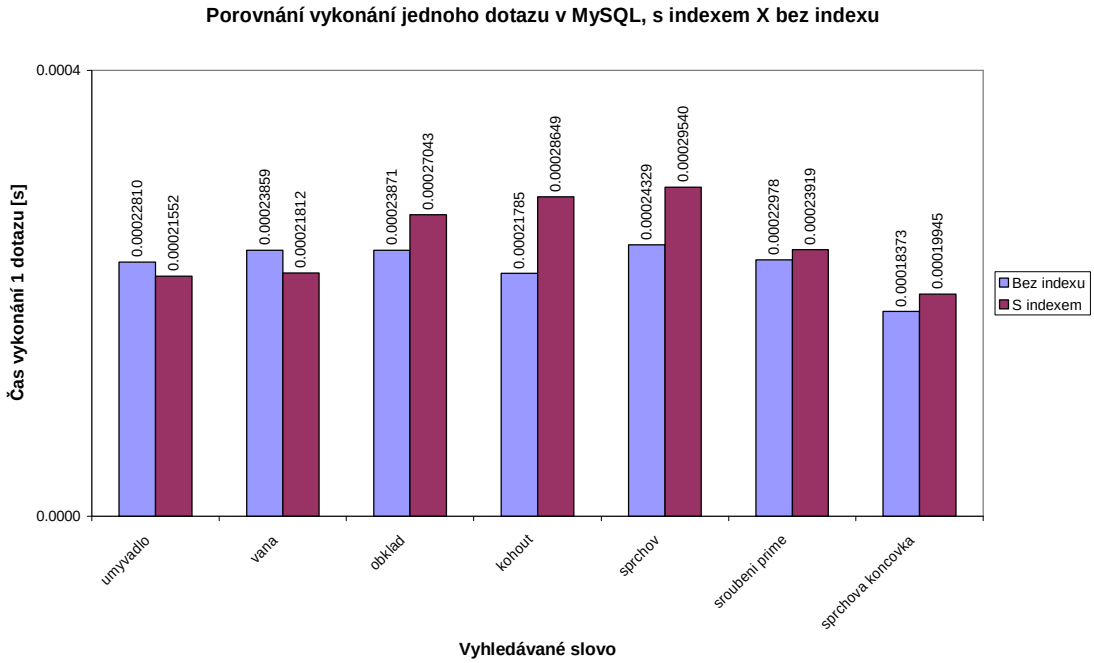


Graf č. 3 – Výsledky s použitím indexu

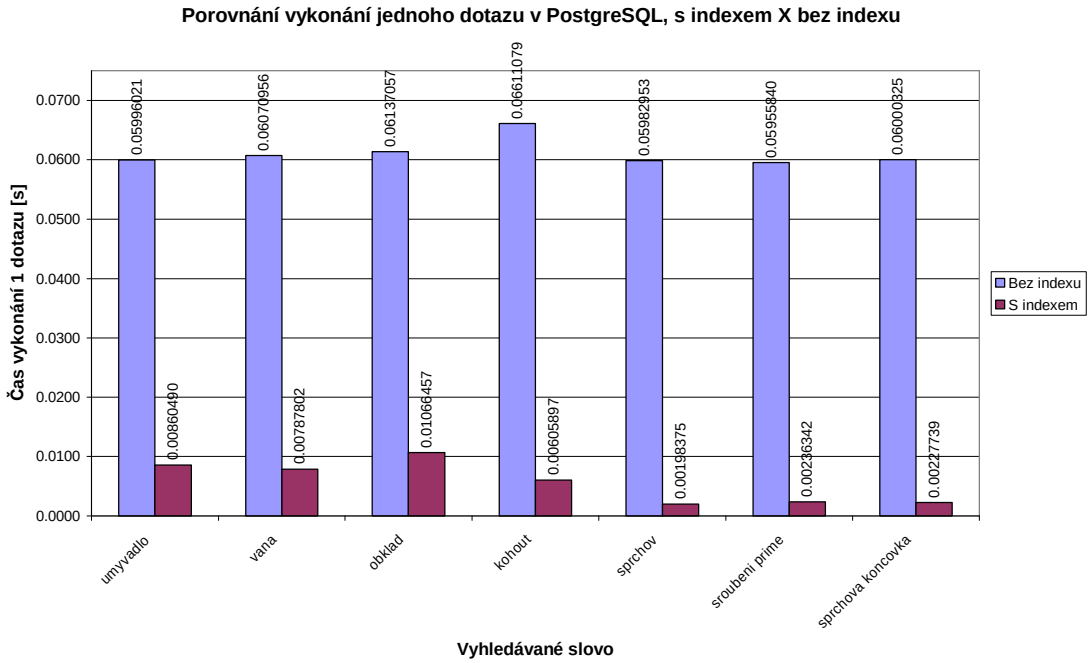


Graf č. 4 – Výsledky s použitím indexu

Na předešlých dvou grafech jsou vidět obdobné tendence jako u minulých (bez indexu). Jen je zde patrný vliv na rychlost databáze PostgreSQL.

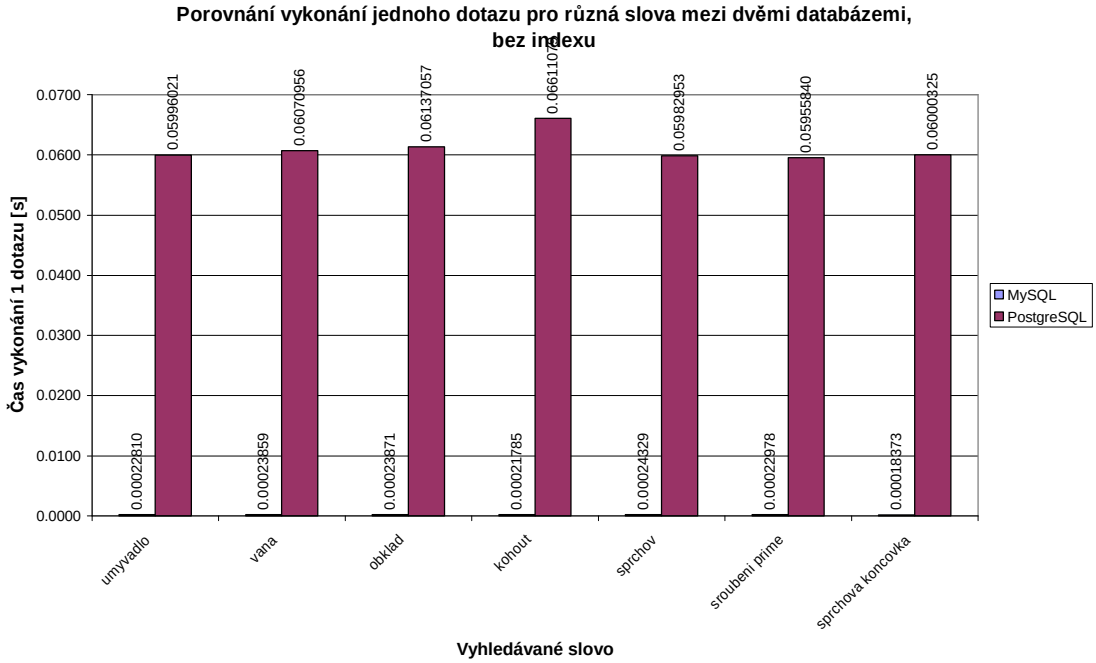


Graf č. 5 – Porovnání využití indexů

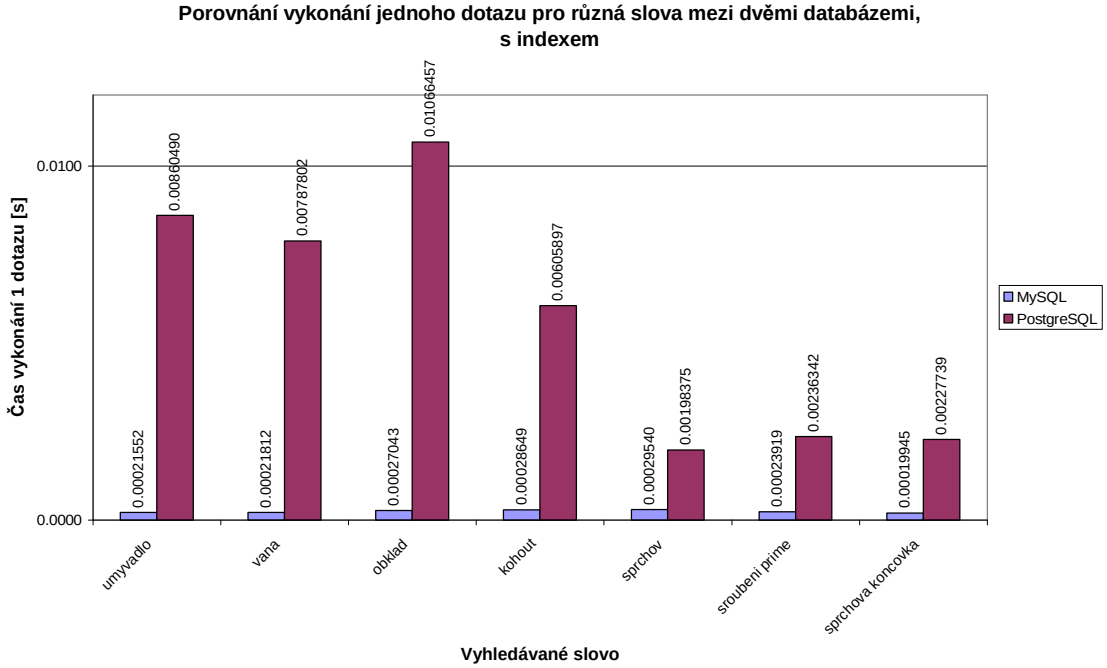


Graf č. 6 – Porovnání využití indexů

Na předposledních dvou grafem je opět vidět (jen z jiného úhlu) vliv indexu na zrychlení u PostgreSQL. Co je zarážející je prodloužení času u MySQL. Jde sice jen o nepatrné časy, ale jsou viditelné. Jsou zřejmě způsobené nevhodností textu (jeho struktury) pro danou databázi. Tím, že PostgreSQL je vcelku jedno o jaký text, protože si ho stejně rozdělí do tokenů, je u PostgreSQL jasný pozitivní vliv na rychlost.



Graf č. 7 – Porovnání databází



Graf č. 8 – Porovnání databází

Na posledních dvou grafech je už pouze finální shrnutí, že ať v dané firmě použijeme indexy či nepoužijeme jasným vítězem je MySQL.

Slovo závěrem

Jak znělo v zadání, jedním z úkolů bylo provést rešerši používaných metodik a systémů. Proto je možné že tato práce působí strohým až nezajímavým. Bohužel tato teoretická základna byla nezbytné zlo pro pochopení dané problematiky. Bez tohoto teoretického základu by nebylo možné nastínit potřeby podniků při implementaci nového datového skladu a hlavně již avizovanou nemožnost porovnávat různě prováděné testy na databázích. Doufám, že práce poskytnula čtenáři srozumitelnou příručku pro rozhodnutí se kterou ze zmiňovaných databázových platforem chce použít.

V následujících přílohách (naměřená data a grafické závislosti) se můžete přesvědčit o různých závislostech, kterých je odvislá rychlost databáze. V tomto se stala vítězem databáze MySQL. Je ovšem opět nutné upozornit, že pro jiný podnik s jinými daty a hardwarem může být výsledek zcela opačný. Nehledě na to, že rychlost vykonání dotazu není jediným ovlivňujícím faktorem. Uživatel může dát přednost možnosti psaní vlastních triggerů v dosti širokém spektru programovacích jazyků (PostgreSQL). Nebo dá na výsledky a dá za vděk rychlosti a neuvěřitelně jednoduché administraci v podobě MySQL.

Reference / zdroje

[1] Arnošt, Pavel. Článek. URL: www.root.cz/clanky/co-je-to-open-source-software/. 2001

[2] Kolektiv autorů. Linux-dokumentační projekt. 3. aktualizované vydání: Computer Press, 2003.

[3] Článek. URL: cs.wikipedia.org/wiki/MyISAM

[4] Referenční manuál. URL: www.mysql.com

[5] Výukový materiál. VŠB TU Ostrava. Fakulta informatiky. URL: www.cs.vsb.cz/benes/vyuka/upr/texty/adt/ch01s07s04.html

[6] Webové stránky autorů projektu TSearch2 (fulltextové indexy v PostgreSQL). URL: www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

[7] Výukový materiál. URL: klobouk.fsv.cvut.cz/%7Ebeny/skola/pj10/postgis/node23.html

[8] Výukový materiál. URL: klobouk.fsv.cvut.cz/%7Ebeny/skola/pj10/postgis/node1.html

[9] Článek. URL: reboot.cz/info/databaze/odbc-v-kostce/articles.html?id=152

[10] Valášek, Michal. Článek. URL: archive.aspnetwork.cz/art/clanek.asp?id=260

[11] Novotný, Tomáš. Esej. URL: nb.vse.cz/~zelenyj/it442/eseje/xnovt16/final.htm

[12] Šuňavec, Ján. Článek. URL:

www.root.cz/clanky/mysql-vs-postgresql-vs-firebird/

Články. URL: www.dbsvet.cz

Články. URL: www.root.cz/clanky/maly-pruvodce-konfiguraci-apache/

Články. URL: http://www.root.cz/clanky/vyuziti-databazovych-indexu/

Články. URL: www.fuzzy.cz/databaze/indexy-zakladni-principy.php

Články PHP. URL: www.builder.cz

Články. URL: www.databasejournal.com

Články. URL: www.debian-administration.org

Články projekt PosgreSQL a indexy. URL: people.planetpostgresql.org/mha/index.php?/archives/112-GIN-performance-postgresql.org-websearch-part-2.htm

Článkek import dat. URL:

pc203b.fzu.cz/cgi-bin/man/man2html?1+mysqlimport

Články. URL: www.abclinuxu.cz

Články, odborné texty. URL: nb.vse.cz/~zelenyj/

Přílohy

Příloha 1 – Názvy sloupců v tabulce a jejich datové typy

Název sloupce	Typ
KartaSort	int(11)
Priznak	char(1)
V	varchar(2)
TypSort	char(1)
Cislo	varchar(20)
Nazev	varchar(150)
HMJ	int(2)
MJ	varchar(3)
DPH	int(2)
CSazebniku	int(8)
Skupina	varchar(5)
Popis	varchar(150)
UcetD	int(1)
UCVyvoz	int(1)
MaxSl	float
ZSleva	int(1)
SpDan	float
DodCislo	varchar(7)
TandemPol	char(1)
PoslPohyb	varchar(10)
JeObal	int(1)
SObalem	int(1)
Aktivni	int(1)
Pomcismat3	varchar(17)
Minimum	int(1)
Maximum	int(1)
CR	int(1)
KlicPart	int(5)
Export	int(1)
CasZapsani	varchar(10)
Stanice	int(4)
ZSC	int(1)
RozmNorma	char(1)
JakosNorma	int(3)
DovObal	int(1)
Znovu	int(1)
VahaObalu	int(1)
MJVahyObalu	char(1)
TypObalu	int(1)
Specialky	int(1)
PKlicMa1	int(6)
OldKlicMa	int(6)
Pomcismat1	varchar(14)
Pomcismat2	double
VahaObalu2	double
NazevSaz	varchar(150)

Příloha 2 – Testovací skript

```
<HTML>
<HEAD>
<TITLE>Testovani</TITLE>
</HEAD>
<BODY>
<?php
        set_time_limit(0);
        function getmicrotime()
        {
                list($usec, $sec) = explode(" ",microtime());
                return ((float)$usec + (float)$sec);
        }
 $time_sum = 0;
 $zaznam = 0;
 $vysledek = 0;
 $time_start = 0;
 $ms = 0;
 for ($i=0;$i<=$_GET["a"];$i++):
  if ($_GET["db"]=='mysql'):
        time_start = getmicrotime();
        $req = "SELECT COUNT(*) FROM test_tab WHERE Nazev LIKE('hled.
        slovo');";
         $ms = mysql_connect('localhost','root','');

    if (!$ms):
      echo "Nepodařilo se připojit.<BR>\n";
      break;
    endif;

    mysql_select_db('testovaci_db',$ms);
    $vysledek = mysql_query($req,$ms);
    $time_end = getmicrotime();
    $time = $time_end - $time_start;
    $time_sum = $time_sum + $time;
   endif;
        if ($_GET["db"]=='pgsq'):
          $time_start = getmicrotime();
          $req = "SELECT COUNT(*) FROM test_tab1 WHERE Nazev LIKE('hled.
          slovo¨)
          $pg = pg_connect("host='localhost' port='5432'user='stepan'
          password='stepan' dbname='test'");

          if (!$pg):
                cho "Nepodařilo se připojit.<BR>\n";
        break;
         endif;

    $vysledek = pg_query($pg,$req);
    $time_end = getmicrotime();
    $time = $time_end - $time_start;
```

```
    $time_sum = $time_sum + $time;
  endif;
endfor;
        echo "  Celkovy cas:".round($time_sum,8)."<br>";
        echo "  Cas vykonani jednoho
        dotazu:".round($time_sum/$_GET["a"],8)."<br><BR>";

        if ($_GET["db"]=='mysql'):
while($zaznam = MySQL_Fetch_Row($vysledek)):
      echo $zaznam[0]." ";
      echo $zaznam[1]." ";
      echo $zaznam[2]." ";
      echo $zaznam[3]." ";
      echo $zaznam[4]." ";
      echo $zaznam[5]." ";
      echo $zaznam[6]."<br>";
    endwhile;
endif;
if ($_GET["db"]=='pgsq'):
while($zaznam = pg_Fetch_Row($vysledek)):
      echo $zaznam[0]." ";
      echo $zaznam[1]." ";
      echo $zaznam[2]." ";
      echo $zaznam[3]." ";
      echo $zaznam[4]." ";
      echo $zaznam[5]." ";
      echo $zaznam[6]."<br>";
    endwhile;
endif;
?>

</BODY>
</HTML>
```

Příloha 3 – Tabulky naměřených hodnot

Databáze hledané slovo index

MySQL umyvadlo Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00235565	0.00924216	0.02107851
	0.00517527	0.00986693	0.02469920
	0.00690864	0.00954465	0.02853933
	0.00127699	0.00984052	0.02507032
	0.00478670	0.00933913	0.01937509
	0.00580321	0.00969036	0.01952379
	0.00519603	0.00969347	0.02604959
	0.00374127	0.00913494	0.02707816
	0.00711473	0.00911125	0.02131256
	0.00962193	0.00979396	0.02117139
Čas 1 dotazu [s]	**0.00025990**	**0.00019051**	**0.00023390**

MySQL vana Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00664141	0.01546713	0.01922198
	0.00475046	0.01492734	0.01973988
	0.00514180	0.00891837	0.01612929
	0.00696427	0.01462697	0.02010865
	0.00372626	0.01773638	0.01857535
	0.00497366	0.01046370	0.01166693
	0.00443517	0.01401425	0.01589119
	0.00751722	0.00968380	0.01877287
	0.00523524	0.01238767	0.01321520
	0.00761384	0.01241353	0.01616203
Čas 1 dotazu [s]	**0.00028500**	**0.00026128**	**0.00016948**

MySQL obklad Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00644966	0.00906098	0.01896887
	0.00604195	0.01221733	0.01872715
	0.00840035	0.01267195	0.01758603
	0.00805286	0.00934706	0.01810424
	0.00645530	0.00997199	0.01831606
	0.00620626	0.00959872	0.01762016
	0.00649559	0.01285363	0.01789895
	0.00784234	0.01275720	0.01779366
	0.00474172	0.00985646	0.01808460
	0.00316848	0.00923264	0.01862603
Čas 1 dotazu [s]	**0.00031927**	**0.00021514**	**0.00018173**

MySQL kohout Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00452537	0.01259048	0.01721152
	0.00405040	0.01213004	0.01786747
	0.00482009	0.01202738	0.01764208
	0.00433514	0.01285904	0.01701643
	0.00458356	0.01279528	0.01729616
	0.00475515	0.01200131	0.01768057
	0.00488651	0.01282406	0.01709951
	0.00469716	0.01231756	0.01796556
	0.00474632	0.01219933	0.01749633
	0.00482342	0.01230559	0.01705422
Čas 1 dotazu [s]	**0.00023112**	**0.00024810**	**0.00017433**

MySQL sprchov Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00603569	0.00913629	0.01853538
	0.00699562	0.00992615	0.01853882
	0.00689410	0.00920165	0.01859387
	0.00725598	0.00927306	0.01881417
	0.00691468	0.00954878	0.01853813
	0.00747174	0.00923954	0.01840367
	0.00652604	0.01225276	0.01840670
	0.00697435	0.00938078	0.01894328
	0.00699130	0.01200619	0.01848680
	0.00718826	0.00919265	0.01804383
Čas 1 dotazu [s]	**0.00034624**	**0.00019832**	**0.00018530**

MySQL sroubeni prime Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00420203	0.01252033	0.02138792
	0.00426420	0.01286512	0.02318750
	0.00424493	0.01244622	0.02091432
	0.00420563	0.01289350	0.02200333
	0.00428840	0.01147549	0.02170366
	0.00420861	0.01285521	0.02331291
	0.00426142	0.01201960	0.02303855
	0.00424026	0.01261712	0.01977824
	0.00487826	0.01271419	0.02456574
	0.00423462	0.01272341	0.02405842
Čas 1 dotazu [s]	**0.00021514**	**0.00025026**	**0.00022395**

MySQL sprchova koncovka Bez indexu

	Opakování		
	20	50	100
Čas [s]	0.00451577	0.00724414	0.01957339
	0.00420978	0.00679811	0.01829257
	0.00393064	0.00786832	0.01917687
	0.00433547	0.00748522	0.01923495
	0.00429304	0.00767882	0.01890305
	0.00442092	0.00739031	0.01945792
	0.00380284	0.00762520	0.01944220
	0.00433758	0.00714384	0.01870158
	0.00422875	0.00739702	0.01859903
	0.00432576	0.00773945	0.01906536
Čas 1 dotazu [s]	**0.00021200**	**0.00014874**	**0.00019045**

PostgreSQL umyvadlo Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.23007586	2.91531608	5.83587710
	1.22802603	2.94850468	5.87018827
	1.22379298	2.90561193	5.88367708
	1.25703871	2.92683810	5.87083590
	1.27642975	2.92479199	5.88773834
	1.26866160	2.93207779	5.88103108
	1.25964894	2.94073946	5.86205996
	1.22697741	2.90092751	5.88059819
	1.26056243	2.91757488	5.87874728
	1.25288927	3.06167808	5.86125346
Čas 1 dotazu [s]	**0.06242051**	**0.05874812**	**0.05871201**

PostgreSQL vana Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.24716164	3.02745387	5.91667641
	1.24968836	3.03539715	5.94072343
	1.20519933	2.99685442	5.98339241
	1.24821857	3.09435300	5.95600973
	1.23622484	2.97277184	5.99109768
	1.23782680	3.08505321	5.94128919
	1.24560452	2.99358635	5.94603112
	1.25304611	2.98499724	5.90446972
	1.24868587	3.05527824	5.99418173
	1.24493499	2.99451330	5.99133928
Čas 1 dotazu [s]	**0.06208296**	**0.06048052**	**0.05956521**

PostgreSQL obklad Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.28646826	3.02934153	6.01065441
	1.22268092	3.06084006	6.03988320
	1.23010060	3.02212150	6.08698440
	1.22368411	3.07021347	6.06605258
	1.23044085	3.09526856	6.08958321
	1.23892967	3.08368099	6.04564160
	1.26909448	3.06629770	6.01937937
	1.26074260	3.02045239	6.02244843
	1.26256727	3.06084921	6.04310096
	1.27158716	3.08621207	6.01594916
Čas 1 dotazu [s]	**0.06248148**	**0.06119055**	**0.06043968**

PostgreSQL kohout Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.53293161	3.01717530	5.91814225
	1.54767018	3.03714690	5.91578028
	1.58750883	3.09802651	5.97997638
	1.55085070	3.07772559	5.96231872
	1.55335967	3.06947764	5.97316360
	1.56529300	3.06495004	5.94195958
	1.53229370	3.08377854	5.97165164
	1.51030715	3.05272270	5.93984608
	1.54778793	3.07887046	5.98282414
	1.57487503	3.07722025	5.91813767
Čas 1 dotazu [s]	**0.07751439**	**0.06131419**	**0.05950380**

PostgreSQL sprchov Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.23814104	2.94883079	5.88170551
	1.23533625	2.97619266	5.81611914
	1.23228380	2.92530230	5.88700769
	1.23722256	2.93197947	5.87940892
	1.23573446	2.93891909	5.85299612
	1.23886821	2.97508285	5.83910580
	1.23088639	2.98295791	5.88508512
	1.23875239	2.95921419	5.83915583
	1.23704860	2.96040709	5.89346309
	1.23652851	2.94498426	5.82278614
Čas 1 dotazu [s]	**0.06180401**	**0.05908774**	**0.05859683**

PostgreSQL sroubeni prime Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.23947452	2.94017124	5.86090302
	1.23143540	2.94008205	5.87642909
	1.23175870	2.91045121	5.88628795
	1.23417307	2.89490888	5.80082776
	1.23592858	2.85129851	5.82006519
	1.23164427	2.91500511	5.86425653
	1.23839801	2.97503512	5.84989147
	1.23468443	2.93257556	5.87071408
	1.22961062	2.92237859	5.82084503
	1.22996263	2.95192710	5.87196712
Čas 1 dotazu [s]	**0.06168535**	**0.05846767**	**0.05852219**

PostgreSQL | sprchova koncovka | Bez indexu

	Opakování		
	20	50	100
Čas [s]	1.23897645	2.92338612	5.97217752
	1.23094567	2.90745192	5.97928568
	1.23522801	2.95872461	5.98078480
	1.23158682	2.95086839	5.91249701
	1.23964482	2.95740085	5.92322149
	1.23626428	2.92630753	5.98275592
	1.23143666	2.94889792	5.92683989
	1.23045588	2.98044529	5.93941371
	1.23657133	2.93584828	5.94006595
	1.23104259	2.92655169	5.91016886
Čas 1 dotazu [s]	**0.06171076**	**0.05883177**	**0.05946721**

MySQL umyvadlo S indexem

	Opakování		
	20	50	100
Čas [s]	0.00324908	0.00990400	0.02609383
	0.00377573	0.00944866	0.02404582
	0.00331399	0.00993402	0.02415701
	0.00352237	0.00936765	0.02740383
	0.00357292	0.00971205	0.02543834
	0.00357397	0.00944400	0.02772124
	0.00314428	0.00932241	0.02192307
	0.00353504	0.00988252	0.02876875
	0.00316453	0.00984404	0.02643254
	0.00836330	0.00971077	0.02534928
Čas 1 dotazu [s]	**0.00019608**	**0.00019314**	**0.00025733**

MySQL vana S indexem

	Opakování		
	20	50	100
Čas [s]	0.00483540	0.01266298	0.01729598
	0.00416584	0.01247906	0.01665605
	0.00497653	0.01168816	0.01794062
	0.00462125	0.01274805	0.01626878
	0.00481174	0.01241946	0.01738700
	0.00426740	0.01292660	0.01723447
	0.00422547	0.01278073	0.01766397
	0.00473389	0.01249911	0.01720757
	0.00496792	0.01270104	0.01703589
	0.00472520	0.01239439	0.01742145
Čas 1 dotazu [s]	**0.00023165**	**0.00025060**	**0.00017211**

MySQL obklad S indexem

	Opakování		
	20	50	100
Čas [s]	0.00710231	0.00916312	0.02323159
	0.00765789	0.00936364	0.02119787
	0.00760662	0.00924819	0.02386926
	0.00725874	0.00946743	0.02987927
	0.00722051	0.00959663	0.02617773
	0.00720798	0.00969093	0.02858376
	0.00722399	0.00949162	0.02348241
	0.00737253	0.00931656	0.02591194
	0.00796275	0.00965222	0.02321204
	0.00743973	0.00986100	0.02577129
Čas 1 dotazu [s]	**0.00037027**	**0.00018970**	**0.00025132**

MySQL kohout S indexem

	Opakování		
	20	50	100
Čas [s]	0.00799449	0.01269676	0.02731086
	0.00799779	0.01186710	0.02361714
	0.00738407	0.01208263	0.02452476
	0.00727063	0.01238066	0.02916635
	0.00727838	0.01214052	0.02104315
	0.00716934	0.01223378	0.02093603
	0.00767325	0.01218102	0.02303374
	0.00721700	0.01281717	0.02669416
	0.00733852	0.01287489	0.02066331
	0.00762278	0.01249672	0.02021457
Čas 1 dotazu [s]	**0.00037473**	**0.00024754**	**0.00023720**

MySQL sprchov S indexem

	Opakování		
	20	50	100
Čas [s]	0.00872074	0.01260104	0.01952283
	0.00815244	0.01228344	0.02054337
	0.00858917	0.01238836	0.02261780
	0.00864705	0.01243891	0.02711682
	0.00832081	0.01296589	0.01922295
	0.00810771	0.01273916	0.02186866
	0.00831529	0.01290103	0.02185110
	0.00866067	0.01214613	0.01950456
	0.00827623	0.01249069	0.02145266
	0.00801277	0.01194869	0.02367449
Čas 1 dotazu [s]	**0.00041901**	**0.00024981**	**0.00021738**

MySQL sroubeni prime S indexem

	Opakování		
	20	50	100
Čas [s]	0.00428558	0.01307686	0.02296542
	0.00428925	0.01318819	0.02136936
	0.00419928	0.01354794	0.02463204
	0.00425413	0.01304311	0.02654624
	0.00423369	0.01355301	0.02165428
	0.00425166	0.01316422	0.02199922
	0.00430170	0.01347364	0.02568855
	0.00419393	0.01391846	0.02256605
	0.00419781	0.01308678	0.02322322
	0.00424181	0.01398193	0.02660836
Čas 1 dotazu [s]	**0.00021224**	**0.00026807**	**0.00023725**

MySQL sprchova koncovka S indexem

	Opakování		
	20	50	100
Čas [s]	0.00448350	0.00968383	0.01907233
	0.00422243	0.00916558	0.01927734
	0.00441406	0.00917568	0.01958869
	0.00442245	0.00967703	0.01849104
	0.00424390	0.00985741	0.01872482
	0.00423841	0.00927233	0.01943626
	0.00426725	0.00905858	0.01963631
	0.00445067	0.00981939	0.01937102
	0.00425248	0.00965198	0.01900911
	0.00425058	0.00976114	0.01926107
Čas 1 dotazu [s]	**0.00021623**	**0.00019025**	**0.00019187**

PostgreSQL umyvadlo S indexem

	Opakování		
	20	50	100
Čas [s]	0.18547342	0.41865871	0.81575404
	0.18431844	0.41908819	0.81586143
	0.18121712	0.41330386	0.84876554
	0.18949116	0.41607301	0.81749546
	0.18603705	0.41847502	0.80877111
	0.18208899	0.41542807	0.85733348
	0.18110487	0.41812254	0.80486423
	0.18615898	0.42393780	0.81889851
	0.18763864	0.41270337	0.84662677
	0.18001945	0.41749611	0.81602241
Čas 1 dotazu [s]	**0.00921774**	**0.00834657**	**0.00825039**

PostgreSQL vana S indexem

	Opakování		
	20	50	100
Čas [s]	0.17454588	0.38812573	0.72829944
	0.17488472	0.38202828	0.72684470
	0.17930185	0.38994861	0.72481596
	0.16109393	0.38723277	0.72566054
	0.17201596	0.38930286	0.72370475
	0.17410491	0.38974386	0.72477132
	0.17820492	0.38589617	0.72513247
	0.17124638	0.38759327	0.72069933
	0.17597324	0.38339653	0.72678827
	0.17567066	0.38151759	0.69256552
Čas 1 dotazu [s]	**0.00868521**	**0.00772957**	**0.00721928**

PostgreSQL obklad S indexem

	Opakování		
	20	50	100
Čas [s]	0.21229706	0.58292907	0.96061592
	0.22594601	0.58213673	0.95438425
	0.24464750	0.51045158	0.96387143
	0.22448109	0.51519596	0.96030320
	0.22617552	0.52537092	0.96116935
	0.22572884	0.52676281	0.96353261
	0.22506901	0.57758614	0.95290697
	0.23730778	0.52092273	0.96374867
	0.23712843	0.53674853	0.98308238
	0.22666736	0.59805103	0.95055375
Čas 1 dotazu [s]	**0.01142724**	**0.01095231**	**0.00961417**

PostgreSQL kohout S indexem

	Opakování		
	20	50	100
Čas [s]	0.13067413	0.29330269	0.56895552
	0.13971403	0.29586595	0.56970208
	0.13987834	0.29012778	0.56167217
	0.13445309	0.28901595	0.58012276
	0.13862546	0.28100679	0.56222056
	0.13330083	0.29099412	0.56614700
	0.13958511	0.29333627	0.56295310
	0.13412514	0.27671202	0.54958198
	0.12922103	0.29786204	0.57358964
	0.12900886	0.27648165	0.56962233
Čas 1 dotazu [s]	**0.00674293**	**0.00576941**	**0.00566457**

PostgreSQL sprchov S indexem

	Opakování		
	20	50	100
Čas [s]	0.00790247	0.14472213	0.26595523
	0.00796221	0.14521405	0.26942421
	0.00827917	0.14952247	0.26197398
	0.00819579	0.14334765	0.26510789
	0.00793943	0.14914466	0.26581888
	0.00812343	0.14380790	0.26180057
	0.00794131	0.14050499	0.26787397
	0.00794758	0.14591662	0.26589930
	0.00792175	0.14051940	0.26543402
	0.00817106	0.14600252	0.26264885
Čas 1 dotazu [s]	**0.00040192**	**0.00289740**	**0.00265194**

PostgreSQL sroubeni prime S indexem

	Opakování		
	20	50	100
Čas [s]	0.05412397	0.12651479	0.18447794
	0.05465916	0.12897750	0.18423100
	0.05499325	0.12883179	0.18590183
	0.05450014	0.12582405	0.18724803
	0.05473449	0.12678289	0.18448182
	0.05433524	0.12727403	0.18207443
	0.05426449	0.12792055	0.18047864
	0.05445849	0.12352417	0.18520794
	0.05428623	0.12420205	0.18295597
	0.05490699	0.12018295	0.18682104
Čas 1 dotazu [s]	**0.00272631**	**0.00252007**	**0.00184388**

PostgreSQL sprchova koncovka S indexem

	Opakování		
	20	50	100
Čas [s]	0.05310667	0.11518053	0.18401096
	0.05387207	0.11799505	0.18212645
	0.05346176	0.11849079	0.18341897
	0.05380521	0.11760876	0.18309775
	0.05327672	0.11099094	0.18970691
	0.05388567	0.11818801	0.18610858
	0.05318373	0.11590852	0.18795749
	0.05310152	0.11761097	0.18207553
	0.05369196	0.11458451	0.18012217
	0.05389807	0.11065105	0.18269941
Čas 1 dotazu [s]	**0.00267642**	**0.00231442**	**0.00184132**