**Vysoká škola ekonomická v Praze**

**2012**

**Fakulta informatiky a statistiky**

**Závěrečné práce**

**Požadavky na formální úpravu závěrečných prací**



**Katedra**

**informačních**

**technologií í**

Obsah

[Úvod 3](#_Toc383599929)

[1.1 Cíle 3](#_Toc383599930)

[1.2 Rešerše 3](#_Toc383599931)

[1.3 Řešená oblast 5](#_Toc383599932)

[2 Analýza potřeb 7](#_Toc383599933)

[2.1 Kritéria 7](#_Toc383599934)

[2.2 Předvýběr množiny DBMS 8](#_Toc383599935)

[3 Měření 9](#_Toc383599936)

[3.1 Vstupní data, severy, metodiky 9](#_Toc383599937)

[3.2 Postup 9](#_Toc383599938)

[3.3 Výsledky 9](#_Toc383599939)

[4 Výběr 10](#_Toc383599940)

[4.1 Váhy kritérií výběru 10](#_Toc383599941)

[4.2 Výsledek 10](#_Toc383599942)

[5 Závěr 11](#_Toc383599943)

Úvod

* 1. Cíle

Cílem mé práce je dodat podklady pro výběr a vybrat konkrétní DBMS pro katalog dílů včetně vhodného enginu a architektury. Předvýběr i výběr provedu na základě kritérií, které si v práci vytyčím, data pro výběr poskytnou vhodná měření na strojích shodných s produkčními. Cíl bude naplněn, budou-li zevrubně porovnány výsledky alespoň tří enginů relačních DBMS a firma, které bude sloužit výsledek mé práce, bude mít dostatek podkladů, na základě kterých bude moci nasadit mnou navržený systém, či se na základě dodaných fakt rozhodnout jinak.

Důvodem, proč jsem si vybral tuto práci je to, že se jedná o skutečný problém, který v současné době řešíme v naší firmě a je to problém v oblasti, která mě zajímá a chtěl bych se jí i nadále v mé profesní kariéře věnovat.

Největším očekávaným přínosem mé práce bude možnost firmy učinit rozhodnutí o výběru konkrétního DBMS na základě mnou dodaných podkladů a v ideálním případě zrychlení cílové webové aplikace. Dalším přínosem této práce ukázat provozovatelům obdobných aplikací, jakým způsobem vybírat engine relačních DBMS a také ukázat, který je v konkrétním využití nejrychlejší.

* 1. Rešerše

Na téma porovnávání nejen nekomerčních databází existuje velké množství prací. Jednotlivé práce se zabývají jednak metodikami testování či samotným testováním. Samotné testování pak je prováděno na základě různých kritérií, kterými jsou především různé výkonnostní ukazatele, množina podporovaných funkcí, cena, či dostupnost podpory.

Mnoho metodických poznatků ohledně testování relačních databázovách systémů předkládá Bc. Jakub Král ve své bakalářské práci Porovnání open source databázových systémů s využitím TPC-C testu (1). Zabývá se potřebou testování databází, postupů v testování a standardizovanými testy, jejich historií a možnostem použití. Největší část této sekce je věnována TPC testům, nejvíce pak testu typu TPC-C, který je dodnes standardem v této oblasti.

Práce Jakuba Krále navazuje na bakalářskou práci Bc. Martina Matějky s názvem Implementace testu k porovnání výkonnosti databázových systémů (2). Ten se ve své práci, kromě teoretického základu věnuje praktickému vytvoření programu pro testování databází dle scénáře TPC-C. Jak Jakub Král kriticky hodnotí, implementace to není úplná a je záměrně vynecháno, nebo přepracováo několik částí tak, aby vyhovovala použití s vybranými open source databázemi.

Bc. Marek Véle v práci Porovnání open source databázových systémů (3) řeší hlavně licenční a funkční faktory databázových systémů. Zaměřuje se tak na výhodnost nasazení open source řešení v podnicích na úkor řešení komerčních. Věnuje se také různým způsobům ukládání dat, a to sloupcově orientovanému, používaném v MonetDB, a řádkově orientovanému používáném v MySQL a PostgreSQL, a v neposlední řadě také dostupnosti zmíněných DBMS pro různé architektury a platformy. V části výkonnostního porovnávání neklade velkou váhu bulk loadu dat do databází, protože z jeho pohledu se nejedná o častou operaci, naopak velkou váhu klade na operace select.

V práci Porovnání open source databázových systémů (4) od Ondřeje Kubáta je probírána hlavně teorie architektury datových modelů síťového, hierarchického, objektového a následně je podrobně popsán model relační. Autor srovnává DBMS Firebird, PostgreSQL, a MySQL dle výkonu a mnoha nevýkonnostních kritérií.

Od předchozích prací se liší práce Ing. Jana Filipa s názvem Ladění relačních databázových systémů (4). Cílem práce je nejen teoretický úvod do optimalizace výkonu databázových systémů, skládající se z návrhu a datového modelování, principů ukládání dat, používání indexů a úrovní izolace, ale i vytvoření metodiky ladění. Zároveň předkládá jednotlivé techniky a příležitosti jejich použití v konkrétních případech.

Další lehce odlišnou prací je Porovnání prostředí pro ukládání dat v MySQL (5) Bc. Adama Vopičky. V ní autor neporovnává jednotlivé DBMS, ale pouze storage enginy použitelné v MySQL. Úvodem práce je představení storage enginů a technických faktorů ovlivňujících výkon databázového systému. Ve druhé části představuje charakteristiky a funkcionalitu jednotlivých enginů, a to jak zabudovaných, tak enginů třetích stran. Dále autor nejen, že zveřejňuje výsledky testů jednotlivých enginů, ale také dodává čtenáři návod pro výběr pro určité připady nasazení podle různých kritérií.

Tyto práce mi dávají silný teoretický základ, ze kterého mám možnost čerpat. Rozhodně využiji poznatků z testu TPC-C, ačkoli pro mnou řešený případ není úplně vhodný ve všech směrech. Taktéž rozhodně využiji poznatky o úložných mechanismech a storage enginech systému MySQL v doprovodu výsledků práce o ladění databázových systémů, protože mnohdy je správná konfigurace klíčovou záležitostí.

V katalogu prací VŠE jsem nenalezl žádnou práci týkající se problematiky dotazování hierarchických dat uložených v relačním systému řízení báze dat, proto budu muset využít jiných, neakademických, zdrojů.

Z mimoškolních zdrojů jsem nalezl disertační práci s názvem Comparing the Performance of Open Source and Proprietary Relational Database Management Systems autora Seana Stevena Coatse, která by mohla být cenným vodítkem při určování metodologických postupů a způsobů hodnocení měření.

* 1. Řešená oblast

Předmětem mé bakalářské práce je výběr vhodného ukládacího enginu relačního databázového systému pro katalog dílů. Jedná se o webovou aplikaci společnosti, která nabízí autodíly v několika státech nejen střední Evropy. Tato společnost má již vybudovanou infrastrukturu počítačového vybavení včetně databázových strojů, na kterém provozují podnikový informační systém pro řízení skladů. Na základě dat z tohoto systému provozuje firma webový katalog, který by měl zjednodušit zákazníkům výběr a nákup zboží této firmy.

Webová aplikace je jednou ze samostatných služeb, které firma provozuje. Katalog má exkluzivní přístup k vlastní databázi, je tedy do jisté míry nezávislou službou firmy. Technologicky je webová aplikace rozdělená na část frontendovou a část backendovou. Frontendem se rozumí ta část aplikace, která slouží k zobrazování dat uživateli a zpracovávání uživatelských požadavků, obsahuje veškerou logiku, jak aplikační, tak prezentační a operuje nad daty, které jsou jí dodávány backendovou částí aplikace. Její serverová část je naprogramovaná v jazyce Python na webovém frameworku Django, který je na uživatelské straně doplněn JavaScriptem. V tomto projektu databázový stroj spadá pod frontendovou část, ačkoli v běžné praxi to není zvykem. Pro účely tohoto dokumentu budu používat toto označení, které je již v projektu zavedené. Backendová část aplikace je datová pumpa, která má za úkol periodicky aktualizovat data webové aplikace. Datová pumpa je vlastní řešení naší firmy, která slouží k vysokorychlostnímu přesunu velkých objemů dat mezi jednotlivými databázemi a pro potřeby různých projektů je parametrizována, popřípadě rozšiřována pomocí modulů. Jedná se o projekt naprogramovaný v .NETu, který může běžet na libovolném serveru a to i samostatně a nezávisle na cílové i zdrojové databázi. Architektura části systému je znázorněna obrázkem Obrázek 2 Architektura



Obrázek 1 Schéma DB

Přesuny dat jsou realizovány tak, že po načtení konfigurace daného projektu a ověření nutných předpokladů k úspěšnému přesunu, jsou postupně načítány celé tabulky ze zdrojové databáze. Poté jsou na cílovém databázovém systému vytvořeny schémata a tabulky, do který jsou následně vkládána hromadná data technikou bulk load. Tato technika je standardním postupem při vkládání dat velkých objemů, při které jsou data vkládána ze souboru či proudového zdroje přímo do databáze bez účasti SQL procesoru, čímž jsou ušetřeny systémové prostředky serveru a tím pádem je tento proces mnohem rychlejší než vkládání jednotlivých řádků. Případná omezení databázového modelu (tzv constrainty) jsou aplikovány až po načtení veškerých dat.

Webová aplikace

Backend

ERP

Zdrojová DB

Datová pumpa

Mnou vybíraný DBMS

Django Framework, HTML, JavaScript

Frontend

Obrázek 2 Architektura

Katalog umožňuje vyhledávat díly podle modelu automobilu, dostupnosti a kategorie dílu. Kategorie jsou implementovány pomocí několika kořenových stromů uložených v relační databázi schématem Modified Preorder Tree Travesal, zkráceně MPTT. Jedná se o způsob ukládání a dotazování hierarchických dat pomocí relačního schématu. Tento způsob byl vybrán mimo jiné proto, že v Django frameworku je pro něj dobrá podpora, zajištěná přímo modulem django.mptt. Kategorie jsou uspořádány tak, že každý produkt spadá alespoň do jedné kategorie, přičemž zároveň může spadat do kategorie jiné, či do několika jejích podkategorií. U každé kategorie produktů dostupných pro konkrétně vybraný model vozidla je zobrazena číslice udávající počet produktů dostupných v této kategorii. Po rozbalení této kategorie je u každé další její podkategorie zobrazen počet produktů. Tím, že jeden produkt může spadat do více kategorií zároveň, může se stát, že součet čísel vyjadřující počet produktů v jednotlivých podkategoriích je větší než číslice sdělující počet produktů v nadřazené kategorii. Řazení produktů do kategorií je nejen pro zákazníka velice důležité, zároveň však je jejich získávání velice náročné z pohledu databázového stroje a jedná se o jeden z nejčastějších zdrojů výkonnostních problémů. Například jednotlivé dotazy, které zjišťují počet produktů v podkategoriích, mnohdy trvají i stovky milisekund, čímž zdržují uživatelské rozhraní. V případech, kde je to vhodné, jsou jednotlivé komponenty uživatelského rozhraní načítány asynchronně, tudíž komponenta je vykreslena až ve chvíli, kdy jsou data, která jsou potřeba, dostupná. Ovšem v mnoha případech této aplikace postupné načítání působí rušivě, bylo tedy rozhodnuto, že budou vždy načítány všechny komponenty najednou, až ve chvíli, kdy budou dostupná všechna data. Tím pádem se stalo úzkým hrdlem získávání dat z databáze.

V aplikaci vystupují entitní množiny Vozidlo, Produkt, Strom. Nejdůležitější z nich je entitní množina Produkt, která reprezentuje jednotlivý díl. V databázi existuje 690 tisíc záznamů typu Produkt. Jednotlivé Produkty jsou vhodné pro jeden či více typů motorových vozidel. Typ vozidla je určen entitní množinou Vozidlo. Vozidel je v databázi 67 tisíc. Vazebná tabulka mezi Vozidlem a Produktem, která reprezentuje vhodnost dílu pro dané motorové vozidlo, obsahuje 55 milionů řádků. Jednou z nejdůležitějších je entitní množina Strom. Jedná se o stromovou strukturu 41 tisíc kategorií, do kterých patří entita Produkt. Produkt může nejen patřit do jednoho či více Stromů, ale zároveň i do podstromů určitého Stromu. Vazebná tabulka reprezentující příslušnost dílu určité kategorii, tedy vazba mezi Produktem a Stromem má 1,8 milionu řádků.

Logické schéma databáze jest znázorněno na obrázku Obrázek 1 Logické schéma databáze

Databázový systém je v současnosti PostgreSQL 9.1 a po mnoha optimalizacích zdrojového kódu aplikace stále přetrvává problém s rychlostí celé aplikace.

* + 1. Problém nedostatečná rychlost přesunu dat

Jedním problémovým místem je přenos dat datovou pumpou ze zdrojové databáze do databáze webové aplikace. Jde o proces, který byl dlouhodobě laděn také z organizačního hlediska, protože je žádoucí, aby data ze zdrojové databáze byla načítána v době, kdy je nejméně vytížena jinými požadavky, a proto tedy jsou data stahována jednou denně v určený čas. Celkový čas přenosu je v současnosti v řádu vyšších desítek minut a my bychom rádi tento čas snížili.

* + 1. Vysoká doba odezvy uživatelského rozhraní

Dalším z problémů je přílišná odezva uživatelského rozhraní webové aplikace v několika konkrétních případech. Jedná se jedny z nejčastějších požadavků uživatele v rámci aplikace, je proto vhodné, aby právě tyto akce probíhaly co nejrychleji. Jde o výběr produktů pro konkrétní vozidlo, výběr produktů na základě parametrů a získání počtu produktů v určité kategorii. Na základě již proběhnuvšího testování aplikace bylo zjištěno, že odezva je vysoká z důvodu dlouhotrvajících dotazů na databázi.

Moje práce je dílčí součástí procesu výzkumu optimalizace databázového řešení pro katalog dílů, který probíhá již několik měsíců a je v něm zapojeno několik lidí. Největší úsilí je věnováno právě optimalizaci SQL dotazů a optimalizaci ORM, které tyto dotazy produkuje. Tyto dotazy jsou analyzovány a rozebírány do nejmenších detailů. Také je v procesu zkoumání využití paralelního přístupu k databázi a paralelní vykonávání SQL dotazů, které je jedním z nejdůležitějších faktorů v databázích, ke kterým má v jednom okamžiku přístup mnoho uživatelů. Další významnou činností je analýza využívání cachovacích prostředků a jiných předpočítávacích mechanismů, a to jak vestavěných v dbms, tak prostředků ve formě samostatně stojícího produktu. Vzhledem k množství a rozmanitosti dat v této aplikaci není využití cache tak efektivní jako u rozsahem menších aplikací, proto by bylo ideální, aby přístup k datům a práce se samotnými daty byla tak rychlá, aby se vyrovnala rychlosti načítání předpočítaných dat. Rychlost přístupu k datům na médiu má na starosti komponenta nazývaná storage engine. Na ní záleží, jakým způsobem jsou data fyzicky ukládána na disku či jiném médiu a jakým způsobem se k datům na médiu přistupuje. A právě měření výkonu storage enginů ve specifických scénářích je náplní mé bakalářské práce a tím i částí řešení tohoto komplexního problému.

1. Požadavky na DBMS

Vzhledem k povaze celého systému, ve kterém je katalog dílů začleněn, je třeba vybrat takové řešení, které bude nejen, co nejefektivnější, ale zároveň nijak nenaruší současnou architekturu po technické a organizační stránce a tudíž přechod na nový DBMS bude co nejhladší. Technickou stránkou je myšleno zařazení do architektury systému, kompatibilita s ostatními částmi řešení, a to jak s backendem, tak frontendem, zejména s použitým webovým serverem a frameworkem. Organizační stránkou je myšleno především zajištění přístupu k datům ze zdrojové databáze. Není totiž možné ten server vytěžovat, jak se nám zachce, máme proto s klientskou firmou dohodnuto, jakým způsobem a kdy je možné jejich server využít. Je třeba zachovat určité vlastnosti databázového stroje, to znamená, že musím vybírat z takové množiny relačních databází, které se co nejvíce podobají současnému řešení.

Mezi vlastnosti, které chceme zachovat, je způsob jakým DBMS na logické úrovni data ukládá a přistupuje k nim. Budu tedy vybírat mezi relačními databázovými systémy, které ukládají řádkovým, nikoli sloupcovým způsobem a přístup k datům zajišťují pomocí některého z dialektů jazyka SQL. Další z klíčových vlastností, které musím bezpodmínečně zachovat je podpora databázových ovladačů přímo ve frameworku Django. Dalším z požadavků je možnost řízení přístupu přímo databázovým serverem. Z důvodu co nejnižších pořizovacích a provozních nákladů bylo již v cílech této práce vytyčeno, že musí jít o open-source řešení, a to takové, které nepodléhá restrikcím komerčního využití, a je dostupné zdarma. Z tohoto důvodu z výběru předem vypadává MySQL, které je jedním z nejpoužívanějších open-source DBMS (zdroj?)

Dle zadání není vyžadována podpora transakcí a diskové perzistence, je tedy možné využít paměťových databází.

* Kritéria hodnotící
  + Kritérium rychlost importu
  + Kritérium doba odezvy SQL dotazů na výběr produktů
  + Kritérium doba odezvy SQL dotazu na získání počtu produktů v dané kategorii
  + Kritérium doba odezvy SQL dotazu na získání produktů pro dané vozidlo
* Kritéria předvýběrová
  + Licence
  + Cena
  + Architektura
  + Dostupnost vhodných ovladačů pro použití v Django Frameworku na Linuxu
  + Oficiální vývoj
  1. Kritéria hodnotící

Jde o kritéria, která budou aplikována pro porovnání jednotlivých databázových enginů, které budou podrobeny měření v mojí práci. Tato kritéria budou sloužit ke kvantitativnímu hodnocení jednotlivých řešení a na jejich základě bude vybrán vhodný engine.

* + 1. Kritérium rychlost importu

Jak již bylo zmíněno v kapitole 1.3, v současné době je jedním z největších problémů nízká rychlost operací. Mezi operace, které je třeba zrychlit, patří rychlost bulk importu dat vyjádřená v záznamech za sekundu. Jedná se o stěžejní operaci, která neprobíhá zřídkakdy, během jejího vykonávání je zatížena jak síť, tak i zdrojový a cílový server, je proto žádoucí tento čas snížit na minimum. Operace bulk import se zakládá na vkládání dat to cílového dbms ze strukturovaného souboru. Tento soubor bude získán pomocí datové pumpy, a pro zajištění rovných podmínek bude soubor uložen a pro porovnání jednotlivých enginů bude vždy použit shodný soubor. Pro zajištění objektivního měření, bude ve všech dbms zajištěna taková struktura tabulek, aby sloupce měly nejen kompatibilní typ, ale zároveň i typ s pokud možno stejným datovým rozsahem, aby bylo k jednomu záznamu zapisováno stejné množství dat.

* + 1. Kritérium doba odezvy SQL dotazů na výběr produktů

Dalšími kritérii jsou rychlost výběru operací select v jednotlivých scénářích. Jedním z klíčových scénářů je výběr produktu na základě jeho parametrů. Jde o výběr z jedné tabulky nad indexovaným sloupcem. Které sloupce budou vybrány a jaké přesné SQL dotazy budou vykonávány, bude popsáno v kapitole Měření. Půjde především o sloupce, které se týkají identifikačních atributů produktu a jeho ceny. Veškeré dotazy budou podrobně vypsány v tabulce, včetně doby odezvy jeho vykonávání.

* + 1. Kritérium doba odezvy SQL dotazu na získání počtu produktů v dané kategorii

Jedním z nejpomalejších dotazů je v současné době rychlost dotazu pro získání počtu produktů v dané kategorii. Jedná se o poměrně komplikovaný dotaz, zahrnující agregační funkci count. Z hlediska uživatelského rozhraní, jde o jeden z nejvíce zpomalujících faktorů, protože uživatel pohybující se skrz navigaci způsobuje, že každým průchodem se vykoná minimálně tolik dotazů, kolik kategorií je právě zobrazeno.

* + 1. Kritérium doba odezvy SQL dotazu na získání produktů pro dané vozidlo

Toto kritérium je specifickým případem kritéria 2.1.2. Jako samostatné kritérium bylo vytyčeno proto, že je mu přisuzována velká důležitost, a zároveň proto, že vyhledávání neprobíhá nad tabulkou Vozidlo, ale na nad vazebnou tabulkou mezi Vozidlem a Produktem. Je to jedno z nejzásadnějších kritérií. To vyplývá už jen z účelu webové aplikace a chování běžného uživatele v této aplikaci. Hlavním motivem, proč uživatel do aplikace katalogu autodílů přichází, je jeho potřeba najít vhodný díl pro konkrétní automobil.

* 1. Kritéria předvýběrová

Jedná se o takzvaná KO kritéria, nebo také binární kritéria. Jsou to kritéria filtrační, a budou použita ještě před samotným měřením. Na základě těchto kritérií budou vybrány jednotlivé databázové systémy a jejich storage enginy.

* + 1. Licence

Jednou z nejdůležitějších KO kritérií je licence jak samotného DBMS tak i storage enginu. Licence musí splňovat charakteristiku svobodné licence (1) a to z několika důvodů. Hlavním důvodem výběru open source licence je možnost přejít na kompatibilní odnož ve chvíli, kdy dodavatel původního softwaru začne například požadovat finanční prostředky, nebo zavede jiné restrikce použití. Další z výhod open source softwaru je možnost vlastní kompilace zdrojových kódů.

* + 1. Cena

Spolu s open source licencí je důležitým faktorem cena. Cena vybíraného řešení musí být nulová. Vzhledem k tomu, že samotná open source licence nezaručuje, že software je zdarma, je toto kritérium ustaveno samostatně.

* + 1. Architektura

Hlavním kritériem co se architektury týká, je relační způsob ukládání dat orientovaný řádkově. To z toho důvodu, že celá aplikace byla navržena pro použití tímto způsobem a data uložená ve zdrojové databázi jsou stejného charakteru. Zachováním tohoto kritéria zajistíme snadný přesun dat ze zdrojové databáze, protože nebude nutné data transformovat, a zároveň bude mít kompatibilní datové úložiště.

Další z požadavků je možnost řízení přístupu k datům na úrovni dbms. Tím je myšlena možnost vytvářet a spravovat uživatele a přiřazovat jim rozličná oprávnění k různým tabulkám.

Ohledně storage enginů, je požadavkem, aby daný engine dokázal data ukládat na médium databázového stroje, nemáme zájem o enginy, které pouze zajišťují přístup do jiného datového úložiště, nebo slouží k experimentálním účelům. Chci do měření zahrnout pouze enginy s podporou indexů.

* + 1. Dostupnost vhodných ovladačů pro použití v Django Frameworku na Linuxu

Důležitým kritériem je použitelnost dbms pro účely naší aplikace, musí proto existovat vhodný a komunitou ozkoušený ovladač pro komunikaci mezi Django frameworkem a databázovým strojem. Podmínka dobré zkušenosti s ovladačem je nutná, chci se totiž věnovat práci s databázovými stroji a nikoli s ovladači. Vhodným ovladačem je takový ovladač, který je v Dajngo frameworku oficiálně podporován.

* + 1. Oficiální vývoj

Podmínkou zařazení dbms a jeho storage enginů do testování je skutečnost, že je řešení oficiálně vyvíjeno. Nechceme testovat velké množství různých forků a experimentálních řešení, chceme testovat opravdu široce používaná řešení, u kterých se dá předpokládat budoucí vývoj. U storage enginů budu vybírat jen ty, které jsou vyvíjeny stejným subjektem jako dbms, pro který jsou určeny.

1. Předvýběr množiny DBMS

V první řadě bylo nutné najít nějaký vhodný seznam dbms. Už v tomto kroku jsem aplikoval podmnožinu kritéria 2.2.3, a to takovou, že jsem hledal seznam relačních dbms. Jediný vhodný a dostatečně rozsáhlý seznam jsem našel bohužel jen na serveru wikipedia Seznam Relačních DBMS a jejich vlastností. V tomto seznamu se nachází mnoho užitečných informací o vlastnostech jednotlivých dbms, ale vzhledem k povaze serveru wikipedia a nízké důvěryhodnosti informací zde publikovaných, jsem musel každý údaj ověřit na webu příslušného dbms. Snažil jsem se co nejvíce omezit čerpání z tohoto zdroje a tak jsem pouze ověřoval data ze sloupce licence, všechna ostatní data pro kritéria jsem vyhledal jinde. Tím pádem jediné, co jsem získal z webu wikipedia, je pouze ucelený seznam relačních dbms.

Table 1 Seznam Relačních DBMS a jejich vlastností

| Produkt | Licence | Zdarma pro komerční využití | Vhodná architektura | Dostupnost oficiálních ovladačů | Oficiální vývoj | Splněna všechna kritéria |
| --- | --- | --- | --- | --- | --- | --- |
| 4D (4th Dimension) | Proprietární | Ne | Ne, správa uživatelů | Ne |  |  |
| ADABAS | Proprietární |  | Ne | Ne |  |  |
| Adaptive Server Enterprise | Proprietární |  |  | Ne |  |  |
| Advantage Database Server (ADS) | Proprietární |  |  | Ne |  |  |
| Altibase | Proprietární |  |  | Ne |  |  |
| Apache Derby | Apache | Ano |  | Ne |  |  |
| Clustrix | Proprietární |  |  | Ne |  |  |
| CUBRID | GPL |  |  | Ne |  |  |
| Datacom | Proprietární |  |  | Ne |  |  |
| DB2 | Proprietární |  |  | Ne |  |  |
| Drizzle | GPL, BSD | Ano |  | Ne |  |  |
| Empress Embedded Database | Proprietární |  |  | Ne |  |  |
| EXASolution | Proprietární |  |  | Ne |  |  |
| Firebird | IDPL | Ano |  | Ne |  |  |
| H2 | EPL | Ano |  | Ne |  |  |
| HP NonStop SQL |  |  |  | Ne |  |  |
| HSQLDB | BSD | Ano |  | Ne |  |  |
| Informix Dynamic Server | Proprietární |  |  | Ne |  |  |
| Ingres | Proprietární, GPL |  |  | Ne |  |  |
| InterBase | Proprietární |  |  | Ne |  |  |
| Linter SQL RDBMS | Proprietární |  |  | Ne |  |  |
| LucidDB | GPL | Ano |  | Ne |  |  |
| MariaDB | GPL | Ano | Ano | Ano | Ano | Ano |
| MaxDB | Proprietární |  |  | Ne |  |  |
| MemSQL |  |  |  | Ne |  |  |
| Microsoft Access | Proprietární | Ne | Ne, správa uživatelů | Ne |  |  |
| Microsoft SQL Server | Proprietární | Ne |  | Ne |  |  |
| Microsoft SQL Server Compact (Embedded Database) | Proprietární |  | Ne, správa uživatelů | Ne |  |  |
| Microsoft Visual Foxpro | Proprietární |  |  | Ne |  |  |
| MonetDB | MDBL |  |  | Ne |  |  |
| mSQL | Proprietární |  |  | Ne |  |  |
| MySQL | Proprietární,  GPL | Ne | Ano | Ano | Ano | Ne |
| Nexusdb | Proprietární |  |  | Ne |  |  |
| Omnis Studio | Proprietární |  |  | Ne |  |  |
| OpenBase SQL | Proprietární |  |  | Ne |  |  |
| OpenEdge | Proprietární |  |  | Ne |  |  |
| OpenLink Virtuoso | Proprietární,  GPL |  |  | Ne |  |  |
| Oracle | Proprietární | Ne | Ano | Ano | Ano |  |
| Oracle Rdb | Proprietární |  |  | Ne |  |  |
| Paradox | Proprietární |  |  | Ne |  |  |
| Pervasive PSQL | Proprietární |  |  | Ne |  |  |
| Polyhedra DBMS | Proprietární |  |  | Ne |  |  |
| PostgreSQL | PostgreSQL | Ano | Ano | Ano | Ano | Ano |
| R:Base | Proprietární |  |  | Ne |  |  |
| RDM Server | Proprietární |  |  | Ne |  |  |
| SAP HANA | Proprietární |  |  | Ne |  |  |
| ScimoreDB | Proprietární |  |  | Ne |  |  |
| SmallSQL | LGPL |  |  | Ne |  |  |
| SQL Anywhere | Proprietární |  |  | Ne |  |  |
| SQLBase (2) | Proprietární |  | Ne | Ne |  |  |
| SQLite (2) | Public Domain | Ano | Ne, správa uživatelů | Ano | Ano | Ne |
| Superbase (2) | Proprietární | Ne | Ne | Ne | Ne |  |
| Teradata (3) | Proprietární | Ne | Ano | Ne | Ano | Ne |
| UniData (3) | Proprietární | Ne | Ano | Ne | Ano |  |
| Xeround Cloud Database | Proprietární | Ne | Ne, Saas | Ne |  |  |
| Libre Office Base | LGPL | Ano | Ne | Ne | Ano | Ne |
| OpenOffice Base | LGPL | Ano | Ne | Ne | Ano | Ne |

Během zjišťování licencování jednotlivých dbms jsem také ověřoval, zda je možné využívat dané řešení i pro komerční účely bez restrikcí. V tomto kroku vypadla z množiny uvažovaných řešení MySQL z důvodu politiky Oraclu (7), která zakazuje použití MySQL Community Edition distributorům softwaru, který nesplňuje podmínky GNU-GPL, což naše webová aplikace nesplňuje. Mohli bychom využít pouze některou z vyšších edic, ale ty zase nesplňují naše kritéria Cena a Licence.

Na webech jednotlivých vydavatelů či distributorů konkrétních dbms jsem také zjišťoval, zdali uvažovaný databázový systém vyhovuje našim požadavkům ohledně architektury. Zde byl překážkou sloupcový způsob ukládání dat pro MonetDB (3) a ukládání přímo do souborových struktur bez možnosti kontroly přístupu uživatelů k datům pro SQLite (4), který by se jinak do užšího výběru dostal, a dále pro LibreOffice a OpenOffice Base a pro Microsoft Access, které by se ani tak to užšího výběru nedostaly.

Jedno z nejdůležitějších kritérií pro použitelnost v našem případě je dostupnost oficiálně dostupných ovladačů pro Django Framework. Zde jsem nemusel provádět žádný složitý průzkum, protože seznam je přímo na webu Djanga (5). Zde bylo možné zjistit, že oficiálně podporované ovladače existují pro MySQL, PostgreSQL, SQLite a Oracle. Překvapivé bylo zjištění, že ODBC ovladače jsou zařazeny do kategorie ovladačů nepodporovaných oficiálně, stejně jako třeba ovladače pro Firebird. Kvůli tomu se mnoho dbms do užšího výběru dostat nemohlo. Jeden z databázových systémů se ale do užšího výběru dostal navzdory tomu, že v tomto seznamu nebyl výslovně uveden. Do výběru se dostal dbms MariaDB, který je odnoží MySQL a je s ní binárně kompatibilní (6).

Do užšího výběru se dostaly pouze dva databázové systémy, a to PostgreSQL a MariaDB. Nyní je na řadě výběr jejich storage enginů. Vzhledem k tomu, že PostgeSQL využívá jeden jediný engine a MariaDB jich využívá několik, je potřeba je také podrobit předvýběru.

V MariaDB je možné využít tyto enginy (11):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Engine | Vyvíjeny stejným vvojářem jako DBMS | Ukládají data | Úložiště kompatibilní se stávajícím řešením | Splňuje všechna kritéria |
| XtraDB | Ano | Ano | Ano | Ano |
| InnoDB | Ne | Ano | Ano | Ne |
| Aria | Ano | Ano | Ano | Ano |
| MyISAM | Ne | Ano | Ano | Ne |
| Archive | Ano | Ano | Ne | Ne |
| BLACKHOLE | Ano | Ne | Ne | Ne |
| Casandra | Ano | Ne | Ne | Ne |
| Connect | Ano | Ne | Ne | Ne |
| CSV | Ano | Ano | Ne | Ne |
| FederatedX | Ano | Ne | Ne | Ne |
| TokuDB | Ne | Ano | Ano | Ne |
| MEMORY | Ano | Ano | Ano | Ano |
| MERGE | Ano | Ne | Ne | Ne |
| Mroonga | Ne | Ano | Ne | Ne |
| OQGRAPH | Ano | Ano | Ne | Ne |
| ScaleDB | Ne | Ano | Ano | Ne |
| Sequence | Ano | Ne | Ne | Ne |
| ShinxSE | Ne | Ne | Ne | Ne |
| Spider | Ano | Ne | Ano | Ne |

Tabulka 1 Enginy MariaDB

Výběr MariaDB enginů sestával ze dvou částí. V první fázi jsem z webu zjistil, jaké enginy MariaDB podporuje, v druhé části jsem jednotlivé enginy podrobil výběru podle zadaných kritérií. Nejčastějším důvodem toho, že daný engine neprošel, bylo porušení kritéria Architektura, konkrétně požadavek ukládání na lokální médium. Velká část storage enginů slouží jako mezistupeň mezi jedním běžným storage enginem a nějakým externím úložištěm, je pouze relační abstrakcí nad externím úložištem, nebo zajišťuje, aby se s několika rozdělenými tabulkami dalo pracovat jako s jednou jedinou. Další překážky pro zařazení do užšího výběru také byly prohřešky týkající se architektury. U CSV enginu to byla vedle viditelnosti dat a řízení přístupů na úrovni dbms nemožnost vkládat prázdné (null) hodnoty, u Archive enginu, který vypadal, že by mohl být pro naše použití vhodný, to byla absence možnosti indexování záznamů. Dále jsem narazil na enginy, které se vůbec nedají považovat za ukládací, protože slouží jen k testovacím nebo výukovým účelům. Mezi ty by mohl být zařazen již nevyvíjený engine EXAPLE, který ale už ani není uveden v seznamu použitelných enginů v MariaDB (zdroj). Ten sloužil pouze jako vývojářský příklad, jakým způsobem vyvíjet nové enginy. Dalším podobným je BLACKHOLE, který slouží k testovacím účelům. Nedrží žádná data a cokoli je do něj uloženo, je zahozeno, takže se pro naše využití nehodí. Dalším důvodem, proč nebyly zařazeny některé enginy, bylo nevyhovění kritéria Oficiální vývoj. Některé enginy již nejsou vyvíjeny vůbec, některé jsou vyvíjeny třetí stranou. To byl například důvod zamítnutí zajímavého enginu TokuDB od společnosti Tokutek a také jeden z důvodů zamítnutí z MySQL dobře známých enginů MyIssam a InnoDB. Ačkoli se tyto dva enginy do výběru nedostaly, jsou důležitou součástí mojí práce. Do výběru se totiž dostaly enginy Aria a XtraDB, které staví na základech zmíněných enginů. Nejsou s nimi úplně shodné, ale z důvodu udržení kompatibility mezi MariaDB a MySQL se engine Aria identifikuje jako MyIssam a XtraDB se identifikuje jako InnoDB. To je důvod, proč se v některých příkazech vyskytují názvy enginů, které nejsou testovány.

|  |  |
| --- | --- |
| Engine | Důovody zamítnutí |
| CSV Engine | Špatné řízení přístupů; nemožnost vkládat null hodnoty |
| Archive | Absence podpory indexů |
| Blackhole | Neukládá data |
| Casandra | Pouze zprostředkovává přístup do jiného úložiště |
| Connect | Neukládá data lokálně |
| FederatedX | Neukládá data lokálně |
| TokuDB | Nevyvíjen stejným subjektem jako dbms |
| Merge | Pouze propojuje již existující tabulky |

Tabulka 2 Důvody zamítnutí enginů

Jedinými storage enginy, které budu porovnávat, jsou tedy Postgres storage engine na dbms PostgreSQL, a enginy XtraDB, MEMORY a Aria databázového systému MariaDB.

1. Měření
   1. Server

Veškeré měření probíhalo na fyzickém stroji naší firmy, který má shodnou konfiguraci jako server nasazený v ostrém provozu. Hardware je provozován na osmi procesorových jádrech na dvou čipech Intel Xeon E5507 o frekvenci 2.227 GHz, pro každé jádro je k dispozici 256 KB L2 cache a pro každý fyzický procesor 4MB L3 cache. Server je osazen 32 GB DDR3 paměti.

Veškeré prostředí je nakonfigurované přesně tak jako produkční databázový server v produkci i po softwarové stránce, jen je nainstalován jeden dbms navíc. Celý server je poháněn GNU\Linuxovou distibucí Debian. Vždy během mého měření bylo zajištěno, že běží vždy jen jeden dbms a druhý je nečinný, ukončený příkazem pro zastavení služby.

/etc/init.d/postgres stop

Projekt optimalizace výkonu webové aplikace katalogu dílů probíhá už několik měsíců a má práce je jednou z jeho částí. Mohl jsem už čerpat z výsledků, kterých dosáhli moji kolegové. Největším přínosem pro mne byly, kromě možnosti využít servery v produkčním nastavení, také odladěné databázové konfigurační parametry databázových systémů, na kterých jsem prováděl veškerá měření. Použil jsem shodné nastavení parametrů, které vyšlo jako nejvhodnější pro každý z testovaných dbms. Nejdůležitější bylo zajistit rovné podmínky všem testovaným storage enginům. U dbms MariaDB byla věnována pozornost tomu, aby při nastavování použití požadovaného storage enginu byl opravdu použit ten správný. Jak již bylo řečeno, z důvodů kompatibility s MySQL se ve výchozím nastavení systém chová tak, že při požadavku použít InnoDB engine se použije engine XtraDB. Toto chování je možné změnit, proto jsem musel zjistit, zda toto chování bylo změněno nebo nebylo. Tyto příkazy

ignore\_builtin\_innodb

plugin\_load=innodb=ha\_innodb.so

se v konfiguračním souboru neobjevily, z čehož plyne, že opravdu využíváme engine XtraDB, a veškeré další vlastnosti a parametry s předponou *innodb* se týkají nastavení enginu XtraDB.

* 1. Vstupní data

Vstupní data byla obstarána téměř stejným způsobem, jako to dělá datová pumpa, avšak v tomto procesu nastalo několik výjimek. První výjimkou bylo to, že nebyly kopírovány všechny tabulky dané databáze, ale pouze pět konkrétních tabulek, které jsou potřeba k mému měření. Celá databáze má kolem dvou desítek tabulek, které se používají k různým účelům, mezi které patří například správa uživatelů, objednávek a lokalizace. Tyto tabulky však nevstupují do výkonově kritických scénářů a tak bylo zbytečné tato data uvažovat. Proto nebyly tyto tabulky importovány do cílového testovacího prostředí, čímž jsem ušetřil nejen mnoho času, ale také jsem se mohl zblízka věnovat importu tabulek, které mě zajímají.

Druhou výjimkou ze standardního procesu importu dat ze zdrojové databáze bylo to, že data byla stažena pouze jednou a uložena v perzistentním souboru na lokálním disku. Tím bylo jednak zajištěno ušetření času oproti vytváření nového souboru a stahování dat při každém měření a za druhé byla zajištěna mnohem důležitější věc, a to, že data byla ve všech případech shodná, čímž byly zajištěny rovné podmínky při každém měření.

Zdrojová data byla uložena v pěti souborech pojmenovaných podle tabulky, do kterých měla být data importována. Jednalo se o čistě textové soubory v CSV formátu naprosto shodným s mezivýstupem, jaký by byl v případě použití standardního procesu přenosu dat datovou pumpou. Byly aplikovány transformace dat tak, aby cílový databázový systém korektně přijal veškerá data a to jak v případě MariaDB tak PostgreSQL databáze, to znamená, že byly zvoleny takové znakové sekvence, které oba systémy přijímají. Šlo hlavně o oddělovače hodnot ve sloupcích, znaky konce řádků a správné uvozovky a escape sekvence v případě, kdy bylo použito řetězců, které by mohly být interpretovány jako některý z řídích znaků.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabulka | Velikost souboru | Počet záznamů | Počet sloupců |
| Product | 378 MB |  | 96 |
| Product\_vehicle | 960 MB |  | 2 |
| Treenode\_product | 30 MB |  | 4 |
| Treenode | 4,9 MB |  | 20 |
| Vehicle | 5,9 MB |  | 19 |

Tabulka 3 Vstupní data

V tabulce Vstupní data jsou uvedeny charakteristiky souborů, v nichž jsou uložena vstupní data pro import do dbms. Charakter dat, včetně jednotlivých sloupců odpovídá obrázku Schéma DB, avšak nachází se zde neshoda mezi počtem sloupců v tabulce product . Je to zapříčiněno tím, že jsem do obrázku pro jednoduchost a přehlednost nezahrnul lokalizační sloupce. Jedná se o sloupce, které slouží ke stejnému účelu, jako sloupce s příponou *cs*, avšak pro jiný jazyk nebo jiný stát. Vzhledem k tomu, že se jedná o reprezentaci stejných vlastností, jako u sloupců v obrázku uvedených, není jejich zamlčením ohroženo pochopení modelu.

* 1. Metodika měření bulk loadu

Jak jsem již zmínil v předchozí kapitole, veškerá data jsem měl již předpřipravená v souborech, z kterých jsem je poté následujícím způsobem nahrával do dbms. Prvním krokem bylo spuštění příslušného databázového serveru a ověření, zdali druhý z testovaných dbms je korektně zastaven. Druhým krokem bylo přihlášení pod účtem administrátora databáze do databázové konzole a vytvoření databáze DDL příkazem

CREATE DATABASE test1 CHARACTER SET utf8 COLLATE utf8\_general\_ci;

Ve chvíli kdy byla vytvořená databáze, bylo potřeba vytvořit strukturu jednotlivých tabulek, do kterých posléze měla být nahrávána data. Protože databázové tabulky mají hodně sloupců, například tabulka produkt jich má přes devět desítek, a každou tabulku budu vytvářet minimálně jednou pro každý testovaný engine, rozhodl jsem se zapsat si příkazy pro vytváření tabulek do souboru, a tento cestu k tomuto souboru pak předávat jako argument databázovému řádkovému klientu. V tomto soubory se nacházejí pouze příkazy

CREATE TABLE …

a žádné jiné. Jediné, co je nutné před bulk importem mít, je odpovídající struktura tabulek, do kterých budou data nahrávána. Vzhledem k tomu, že data jsou nahrávána po jednotlivých tabulkách, nelze mít předem zapnuta žádná relačně orientovaná databázová omezení tzv. constrainty typu foreign key. Dále také není vhodné mít nad tabulkou indexy v době nahrávání dat. Toto by mohlo zbytečně zpomalovat probíhající proces nahrávání dat a nevedlo by to k ničemu užitečnému, protože vytvořené indexy by se neustále přepisovaly. Indexy proto vytvářím až ve chvíli, kdy jsou opravdu potřeba.

Vytvoření tabulkových struktur v MariaDB jsem zajistil spuštěním příkazu

mysql -u user -p test1 < /tmp/mirek-db/mysql/mysql-create-table.sql

Jde o příkaz spouštěný z příkazové řádky na úrovni operačního systému. Tento příkaz spouští MySQL/MariaDB řádkového klienta pod uživatelem user a předá k vykonání všechny příkazy uvedené v souboru mysql-create-table.sql. Parametr –p zajistí, že se program uživatele zeptá na heslo. Bez tohoto parametru by se předpokládalo, že uživatel nemusí být ověřen vůči databázovému systému.

Důvod toho, že spouštíme příkaz s názvem mysql je ten, že MySQL client umožňuje připojení jak do MySQL databázového systému tak do MariaDB systému, který je s první zmíněnou kompatibilní. Z důvodu zachování plné kompatibility se klient MariaDB v GNU\Linuxové distribuci Debian, a nejen v ní, spouší stejným příkazem jako by se jednalo o MySQL klienta.

Posledním krokem načítání dat do MariaDB databáze je opětovné přihlášení do řádkového klienta, a spuštění příkazů pro načtení dat do tatabáze.

load data infile '/tmp/mirek-db/mysql/product.txt' replace into table product;

load data infile '/tmp/mirek-db/mysql/product\_vehicle.txt' replace into table product\_vehicle;

load data infile '/tmp/mirek-db/mysql/treenode.txt' replace into table treenode;

load data infile '/tmp/mirek-db/mysql/treenode\_product.txt' replace into table treenode\_product;

load data infile '/tmp/mirek-db/mysql/vehicle.txt' replace into table vehicle;

Tyto příkazy spouštím jeden po druhém, vždy po skončení jednoho spouštím další, stejně tak jako to dělá datová pumpa. Paralelní spouštění těchto příkazů by na tomto stroji v aktuálních podmínkách neumožnilo otestovat jednotlivé enginy spravedlivě. Případ, kdy by bylo možné paralelním výkonem těchto příkazů zrychlit čas načítání dat, by předpokládal jiné nastavení dbms a možná i jinou konfiguraci hardwaru. To z důvodu toho, že stroj, na kterém pracuji, má jediný pevný disk a ne všechny testované enginy podporují tablespaces. Hlavním důvodem, proč jsou všechny příkazy vykonávány sériově, je mimo jiné to, že jsou porovnávány enginy odlišných dbms a každý ze systémů může paralelizaci dotazů řešit jiným způsobem a tudíž naměřené výsledky by nemusely plně vypovídat o výkonu storage enginu, ale vypovídali by o samotném dbms, což není předmětem této práce. Tento příkaz po skončení vypisuje důležité údaje o svém běhu. Pro mé měření je nejdůležitější hodnota udávající dobu běhu. Ta je vypsána s přesností na setiny vteřiny, což je v našem případě dostatečná přesnost, zvláště u velkých tabulek, kde se počítá v řádech desítek vteřin. Dále příkaz vypisuje další běhové údaje, jako je počet vložených řádků, počet řádků z cílových tabulek vymazaných, počet nevložených, ale v souboru obsažených řádků a případné chyby. Z těchto údajů si ověřuji, zda všechno proběhlo tak jak mělo.

Tímto by měření rychlosti nahrávání dat mohlo být u konce, ale z důvodu vyvarování se náhodných vlivů a zachování statistické přesnosti jsem každé nahrání dat ještě šestkrát zopakoval. Před každým opakováním jsem vyprázdnil tabulky, aby se v nich nacházela pouze relevantní data, a ačkoli výše zmíněné příkazy obsahují klauzuli *replace into*, která data v tabulce nahrazuje, rozhodl jsem se před každým opakování data mazat proto, že ono přepsání dat může zatěžovat výkon samotného procesu nahrávání a tím zkreslovat výsledek. V ostrém provozu se také data načítají do prázdných tabulek, takže mazáním dat se můj proces měření přibližuje reálnému provozu.

V průběhu testování načítání dat do tabulek, které jsou obsluhovány enginem XtraDB, jsem narazil na výkonnostní problém při provádění hromadných SQL dotazů. Obyčejný příkaz pro smazání všech dat z tabulky trval ještě delší dobu než nahrávání dat, proto jsem se rozhodl provést po každém měření příkaz

DROP DATABASE test1;

a začít s nastavováním celé databáze od znova, jak již bylo popsáno. Tento proces, ačkoli vypadá na první pohled zdlouhavější, ušetřil mnoho času.

Výše uvedený postup jsem použil pro každý z testovaných enginů využívaných v rámci databázového systému MariaDB. Jedinou úpravou vždy bylo pouze upravení příkzů *C*

Podobný proces jsem použil i při měření výkonu storage enginu postgres. Postup se liší především způsobem práce s dbms, na kterém engine běží. Výhodou oproti MariaDB byl konzistentnější způsob práce. Nebylo nutné přepínat se mezi systémovým a klientským příkazovým řádkem během načítání dat. Věřím, že podobný způsob jako v PostgreSQL by se dal použít i v MariaDB, avšak mnou použitý postup je shodný s tím, co je popsáno v dokumentaci zdroj?.

Prvním krokem bylo přihlášení do příkazového řádku PostgreSQL databáze pod systémovým uživatelem postgres příkazy

su postgres

psql

Dalším krokem bylo načtení struktury tabulek ze souboru příkazem

\i /tmp/mirek-db/postgresql/postgres-create-table.sql

Soubory se strukturami tabulek jsou shodné jak pro MariaDB tak pro PostgreSQL, až na jednu drobnou odchylku v syntaxi pro vytváření tabulek. Protože v MariaDB je možno použít několik různých storage enginů, tak v rámci příkazu *CREATE TABLE* je pro tento MariaDB přidán parametr *ENGINE*, který určuje, jaký storage engine se má použít.

Abych byl schopen zjistit dobu trvání načítání jednotlivých tabulek do PostgreSQL databáze, bylo zapotřebí ještě nakonfigurovat PostgreSQL tak, aby zaznamenával do logovacích souborů, všechny dotazy bez ohledu na to, jak dlouho trvaly. V původním nastavení bylo logování dotazů zakázáno, bylo tedy třeba logování nastavit. V souboru */etc/postgresql/9.1/main/postgresql.conf* se nachází direktiva *log\_min\_duration\_statement*, která se používá k logování dlouhotrvajících dotazů na databázi. Její argument určuje spodní práh v milisekundách, od kterého budou dotazy do logu zapisovány. Vypnutí logování dotazů se provádí předáním argumentu o hodnotě -1, logování všech dotazů, nezávisle na době trvání, se zajišťuje nastavením hodnoty minimálního trvání na hodnotu 0. Protože se jedná o změnu konfigurace načítané z ze souboru při startu databáze, bylo potřeba dbms restartovat. Toto jsem zajistil příkazem

/etc/init.d/postgresql restart

Od této chvíle jsou všechny provedené dotazy jsou zapisovány do souboru */var/log/postgresql/postgresql-9.1-main.log* včetně doby trvání.

Následovalo spuštění příkazů pro načtení dat do tabulek ze souborů. Opět název souboru odpovídá názvu tabulky, do které se mají data načíst.

copy treenode from '/tmp/mirek-db/mysql/treenode.txt';

copy treenode\_product from '/tmp/mirek-db/mysql/treenode\_product.txt';

copy vehicle from '/tmp/mirek-db/mysql/vehicle.txt';

copy product\_vehicle from '/tmp/mirek-db/mysql/product\_vehicle.txt';

copy product from '/tmp/mirek-db/mysql/product.txt';

Tímto byla data nahrána do tabulek a v souboru */var/log/postgresql/postgresql-9.1-main.log* se objevily příslušné záznamy vypovídající o rychlosti jednotlivých dotazů.

Stejně jako při měření MariaDB enginů, jsem celý postup ještě šestkrát opakoval. Mezi jednotlivými měřeními jsem mazal celé tabulky v databázi. Struktura tabulek byla do databáze nahrána způsobem, jak bylo popsáno výše.

* 1. Metodika měření jednotlivých SELECT dotazů

Z logovacích souborů na produkčním serveru byly vybrány nejčastěji vykonávané a nejdéle trvající SQL dotazy, které webová aplikace posílá na databázový server. Tyto dotazy byly upraveny tak, aby měření bylo co nejobjektivnější. Každý jeden dotaz byl spuštěn několikrát se stejnou hodnotou parametru a posléze byla tato hodnota změněna. Každý dotaz byl spuštěn s pěti různými hodnotami parametru, které byly zvoleny tak, aby rovnoměrně pokrývaly šíři dat. Toho bylo dosaženo tak, že u číselných atributů, jako je třeba hodnota umělého primárního klíče tabuky nebo cena, byla množina uvažovaných hodnot rozdělena na pět dílů, a z každého dílu byla vybrána jedna hodnota. U identifikačních atributů nad tabulkou Produkt tímto výběr hodnot skončil, avšak u vazebné tabulky mezi Produktem a Vozidlem a u atributu retailprice\_cs, která reprezentuje cenu produktu, byl brán v úvahu ještě jeden faktor. Ačkoli by se to dít nemělo a databázové systémy by měly reportovat skutečný čas vykonání dotazu a toto vykonání by mělo být nezávislé na terminálu, ve kterém se dotaz spouští, není vyloučeno, že jsou databázové systémy zpomalovány výpisem na terminál. Toto je zvláště viditelné u dlouhých výpisů překračujících několik stránek, rozhodl jsem se proto to výběru hodnot parametrů zařadit pouze takové hodnoty, při kterých výpis na konzoli nepřekročí jednu obrazovku.

U dotazů, které nevracejí jediný záznam, ale množinu záznamů, je v praxi obvyklé výsledky řadit. Toho se dosahuje použitím klauzule *order by*. Avšak na základě zkušenosti s dbms MySQL, z kterého vyhází testovaný dbms MariaDB, jsem zjistil, že nad velkými daty dochází k mnohem výraznějšímu zpomalení výkonu dotazů při použití řazení pomocí *order by*, než při vynechání indexů. Takové zpomalení způsobené řazením na některých dotazech oproti neseřazeným záznamům mnohdy dosahovalo více než dva desetinné řády. Toto chování nemám přesně změřené a nevím, od jaké velikosti dat, z nichž je vybíráno a velikosti dat vracených se toto zpomalení projevuje, proto jsem se rozhodl řazení vůbec nepoužít. Myslím si, že zdokumentování tohoto chování by mohlo být dobrým námětem na závěrečnou práci.

Další, a hodně výraznou úpravou SQL dotazů jsem se vzdálil od reálného použití aplikace tím, že jsem některé dotazy zjednodušil. Toto zjednodušení bylo nutné z důvodu zachování objektivity měření jednotlivých enginů, protože pokud bych tyto dotazy nezjednodušil, vstoupil by do hry faktor samotného dbms. Každý z mnou testovaných dbms může mít odlišnou práci s pamětí a odlišný způsob zpracování komplikovanějších dotazů. Nebyly proto zahrnuty dotazy, ve kterých jsou spojovány tabulky klauzulí *join*, u porovnávání řetězců nebylo použito klíčového slova *like* a na místo toho byly řetězce porovnávány operátorem *je rovno*. Asi nejvýraznějším a nejviditelnějším zjednodušením je použití dotazu, který vrací množinu produktů na základě ceny. V reálném použití takovýto dotaz nepřijímá konkrétní hodnotu ceny, ale její rozsah, tedy minimální a maximální hranici po klíčovém slově *between*. Dále takový dotaz asi nebývá pokládán, aniž je specifikováno, o jaký druh produktu se má jednat. To by předpokládalo ještě propojení s tabulkou Kategorie, což by takovýto dotaz zkomplikovalo a výsledek by nevypovídal pouze o rychlosti vyhledávání nad indexovaným sloupcem.

Aby i výše zamítnutý případ vykonávání komplikovanějších dotazů nad spojenými tabulkami s několika vstupními parametry byl změřen, protože se určitě jedná o nezanedbatelný údaj o rychlosti, je v této práci zahrnut dotaz na počet produktů v dané kategorii. Tento dotaz nebyl po vytažení z logů nijak upravován. Bylo vzato několik variant vstupních parametrů a tyto dotazy byly spuštěny a změřeny. Aby byla zachována určitá míra statistické přesnosti, které se zvláště v měření dotazů nad stromovými strukturami špatně dosahuje, byl tento dotaz spuštěn s 37 různými variantami hodnot vstupních parametrů.

Podobný způsob výběru hodnot vstupních parametrů jsem použil i pro vyhledávání záznamů podle atributu internal\_number, který, ač to z názvu není úplně patrné, je datového typu varchar. Vybíral jsem takové hodnoty, které se liší prvními několika znaky v řetězci, a snažil jsem se, aby se lišila i délka řetězce.

Samotné měření select dotazů pak probíhalo tak, že po posledním provedeném měření bulk loadu dat na daném enginu jsem ze souboru načetl do databáze indexy nad příslušnými sloupci tak, jak jsou v nastaveny v produkčním nasazení. V prvé řadě byly nahrány primární klíče nad tabulkami Produkt, Vozidlo a Kategorie. Indexování těchto sloupců je automatické, proto nebylo nutné index výslovně uvádět. Dále byly vytvořeny indexy nad vazebnou tabulkou mezi Produktem a Vozidlem, a to nad oběma sloupci. V tabulce Produkt byl vytvořen index nad sloupcem supplier\_id. Poslední indexy, které se v dávkovém souboru vyskytují, jsou indexy v tabulce Kategorie a vazebné tabulce mezi Karegorí a Produktem.

ALTER TABLE product ADD PRIMARY KEY product\_pkey (id);

ALTER TABLE vehicle ADD PRIMARY KEY vehicle\_pkey (id);

ALTER TABLE treenode ADD PRIMARY KEY treenode\_pkey (id);

CREATE INDEX product\_vehicle\_product\_id\_idx ON product\_vehicle (product\_id);

CREATE INDEX product\_vehicle\_vehicle\_id\_idx ON product\_vehicle (vehicle\_id);

CREATE INDEX product\_supplier\_id\_idx ON product (supplier\_id);

CREATE INDEX treenode\_treenodetype\_id\_idx ON treenode (treenodetype\_id);

CREATE INDEX treenode\_product\_treenode\_id\_idx ON treenode\_product (treenode\_id);

CREATE INDEX treenode\_product\_product\_id\_idx ON treenode\_product (product\_id);

CREATE INDEX treenode\_product\_treenodetype\_id\_idx ON treenode\_product (treenodetype\_id);

CREATE INDEX treenode\_parent\_id\_idx ON treenode (parent\_id);

CREATE INDEX treenode\_parent\_treenodetype\_id\_idx ON treenode (parent\_treenodetype\_id);

CREATE INDEX treenode\_mptt\_min\_idx ON treenode (mptt\_min);

CREATE INDEX treenode\_mptt\_max\_idx ON treenode (mptt\_max);

LOAD INDEX INTO CACHE product;

LOAD INDEX INTO CACHE product\_vehicle;

LOAD INDEX INTO CACHE vehicle;

LOAD INDEX INTO CACHE treenode;

LOAD INDEX INTO CACHE treenode\_product;

Soubor o výše uvedeném obsahu jsem spustil příkazem

mysql -u user -p test1 < /tmp/mirek-db/mysql/mysql-indexes.sql

Vedle vytvoření indexů jsou v dávkovém souboru s SQL dotazy ještě příkazy pro načtení indexů do databázové vyrovnávací paměti. Tento příkaz by měl urychlit práci s indexy, bohužel jeho využití s mnou uvažovanými enginy je možné pouze s enginem Aria. Jedná se o funkcionalitu, kterou dbms umožňuje využít na úrovni storage enginu, tudíž jsem se rozhodl tento příkaz vykonat, ačkoli ostatní enginy vykonání tohoto příkazu neovlivní. Mým požadavkem je vybrat nejrychlejší storage engine, a proto skutečnost, že některý engine podporuje některé funkcionality zvyšující výkon a jiné enginy je nepodporují, není důvodem, abych těchto funkcionalit nevyužil.

Vytvoření indexů nad sloupci v tabulkách je nezbytně nutným krokem k praktickému využití dat a celé databáze. Jelikož v kritériích vytyčených v této práci se žádné kritérium rychlosti vytvoření indexů nevyskytuje, nezahrnul jsem tento případ do měření.

Načítání indexů probíhalo také na dbms PostgreSQL. Příkazem

\i /tmp/mirek-db/postgresql/postgres-indexes.sql

vykonaným v řádkovém klientu systému PostgreSQL jsem vykonal dávku příkazů sloužících ke stejnému účelu, jako podobný kód vykonávaný v MariaDB. Z důvodu jiné syntaxe uvádím i obsah tohoto souboru.

ALTER TABLE product ADD CONSTRAINT product\_pkey PRIMARY KEY(id);

ALTER TABLE vehicle ADD CONSTRAINT vehicle\_pkey PRIMARY KEY(id);

ALTER TABLE treenode ADD CONSTRAINT treenode\_pkey PRIMARY KEY(id);

CREATE INDEX product\_vehicle\_product\_id\_idx ON product\_vehicle USING btree (product\_id);

CREATE INDEX product\_vehicle\_vehicle\_id\_idx ON product\_vehicle USING btree (vehicle\_id);

CREATE INDEX product\_supplier\_id\_idx ON product USING btree (supplier\_id);

CREATE INDEX treenode\_treenodetype\_id\_idx ON treenode USING btree (treenodetype\_id);

CREATE INDEX treenode\_product\_treenode\_id\_idx ON treenode\_product USING btree (treenode\_id);

CREATE INDEX treenode\_product\_product\_id\_idx ON treenode\_product USING btree (product\_id);

CREATE INDEX treenode\_product\_treenodetype\_id\_idx ON treenode\_product USING btree (treenodetype\_id);

CREATE INDEX treenode\_parent\_id\_idx ON treenode USING btree (parent\_id);

CREATE INDEX treenode\_parent\_treenodetype\_id\_idx ON treenode USING btree (parent\_treenodetype\_id);

CREATE INDEX treenode\_mptt\_min\_idx ON treenode USING btree (mptt\_min);

CREATE INDEX treenode\_mptt\_max\_idx ON treenode USING btree (mptt\_max);

Tímto byla data připravena pro měření rychlosti dotazů. Bylo však nutno zajistit přesné statistiky běhu dotazů v MariaDB. Po každém vykonaném příkazu je sice zobrazen čas jeho běhu, ale pouze s přesností na setiny vteřiny, což neumožňuje přesné měření, zvláště když první platné číslice se objevují většinou v řádech desetin milisekundy.

Rychlost načtení indexů, korelace s bulk loadem?

Server 8 jader Intel(R) Xeon(R) CPU E5507 @ 2.27GHz

4096 KB per core cache

32 GB RAM

Timing cached reads: 4052 MB in 2.00 seconds = 2026.11 MB/sec

Timing buffered disk reads: 282 MB in 3.10 seconds = 90.99 MB/sec

Data maria

CREATE TABLE product

(

id integer NOT NULL,

supplier\_id int NULL,

elitsupplier\_id int NULL,

oldpartproduct\_id int NULL,

internal\_number varchar(22) NOT NULL,

active\_number varchar(22) NOT NULL,

active\_number\_normalized varchar(22) NOT NULL,

tecdoc\_article varchar(22) NULL,

tecdoc\_article\_normalized varchar(22) NULL,

description\_cs varchar(100) NULL,

extendeddescriptionpretext\_cs varchar(200)NULL,

extendeddescription\_cs varchar(200)NULL,

description\_sk varchar(100) NULL,

extendeddescriptionpretext\_sk varchar(200)NULL,

extendeddescription\_sk varchar(200)NULL,

description\_ro varchar(100) NULL,

extendeddescriptionpretext\_ro varchar(200)NULL,

extendeddescription\_ro varchar(200)NULL,

description\_ru varchar(100) NULL,

extendeddescriptionpretext\_ru varchar(200)NULL,

extendeddescription\_ru varchar(200)NULL,

description\_pl varchar(100) NULL,

extendeddescriptionpretext\_pl varchar(200)NULL,

extendeddescription\_pl varchar(200)NULL,

description\_hu varchar(100) NULL,

extendeddescriptionpretext\_hu varchar(200)NULL,

extendeddescription\_hu varchar(200)NULL,

description\_en varchar(100) NULL,

extendeddescriptionpretext\_en varchar(200)NULL,

extendeddescription\_en varchar(200)NULL,

retailprice\_cs numeric(19,2) NULL,

vatrate\_cs numeric(19,2) NULL,

priority\_cs int NOT NULL,

internal\_order\_quantity\_cs int NOT NULL,

donotorder\_cs boolean NOT NULL,

hideprice\_cs boolean NOT NULL,

noreturn\_cs boolean NOT NULL,

nodiscount\_cs boolean NOT NULL,

new\_from\_cs date,

new\_to\_cs date,

retailprice\_sk numeric(19,2) NULL,

vatrate\_sk numeric(19,2) NULL,

priority\_sk int NOT NULL,

internal\_order\_quantity\_sk int NOT NULL,

donotorder\_sk boolean NOT NULL,

hideprice\_sk boolean NOT NULL,

noreturn\_sk boolean NOT NULL,

nodiscount\_sk boolean NOT NULL,

new\_from\_sk date,

new\_to\_sk date,

retailprice\_ro numeric(19,2) NULL,

vatrate\_ro numeric(19,2) NULL,

priority\_ro int NOT NULL,

internal\_order\_quantity\_ro int NOT NULL,

donotorder\_ro boolean NOT NULL,

hideprice\_ro boolean NOT NULL,

noreturn\_ro boolean NOT NULL,

nodiscount\_ro boolean NOT NULL,

new\_from\_ro date,

new\_to\_ro date,

retailprice\_ua numeric(19,2) NULL,

vatrate\_ua numeric(19,2) NULL,

priority\_ua int NOT NULL,

internal\_order\_quantity\_ua int NOT NULL,

donotorder\_ua boolean NOT NULL,

hideprice\_ua boolean NOT NULL,

noreturn\_ua boolean NOT NULL,

nodiscount\_ua boolean NOT NULL,

new\_from\_ua date,

new\_to\_ua date,

retailprice\_pl numeric(19,2) NULL,

vatrate\_pl numeric(19,2) NULL,

priority\_pl int NOT NULL,

internal\_order\_quantity\_pl int NOT NULL,

donotorder\_pl boolean NOT NULL,

hideprice\_pl boolean NOT NULL,

noreturn\_pl boolean NOT NULL,

nodiscount\_pl boolean NOT NULL,

new\_from\_pl date,

new\_to\_pl date,

retailprice\_hu numeric(19,2) NULL,

vatrate\_hu numeric(19,2) NULL,

priority\_hu int NOT NULL,

internal\_order\_quantity\_hu int NOT NULL,

donotorder\_hu boolean NOT NULL,

hideprice\_hu boolean NOT NULL,

noreturn\_hu boolean NOT NULL,

nodiscount\_hu boolean NOT NULL,

new\_from\_hu date,

new\_to\_hu date,

available\_cs boolean NOT NULL,

available\_sk boolean NOT NULL,

available\_ro boolean NOT NULL,

available\_ua boolean NOT NULL,

available\_pl boolean NOT NULL,

available\_hu boolean NOT NULL

)

CREATE TABLE product\_vehicle

(

product\_id integer NOT NULL,

vehicle\_id integer NOT NULL

)

ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE treenode\_product

(

treenode\_id integer NOT NULL,

treenodetype\_id integer NOT NULL,

product\_id integer NOT NULL,

is\_universal boolean NOT NULL

)

ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE treenode

(

id integer NOT NULL,

treenodetype\_id integer NOT NULL,

parent\_id integer NULL,

parent\_treenodetype\_id integer NULL,

mptt\_min int NOT NULL,

mptt\_max int NOT NULL,

name\_cs varchar(100) NOT NULL,

name\_cs\_normalized varchar(100) NOT NULL,

name\_sk varchar(100) NOT NULL,

name\_sk\_normalized varchar(100) NOT NULL,

name\_ro varchar(100) NOT NULL,

name\_ro\_normalized varchar(100) NOT NULL,

name\_ru varchar(100) NOT NULL,

name\_ru\_normalized varchar(100) NOT NULL,

name\_pl varchar(100) NOT NULL,

name\_pl\_normalized varchar(100) NOT NULL,

name\_hu varchar(100) NOT NULL,

name\_hu\_normalized varchar(100) NOT NULL,

name\_en varchar(100) NOT NULL,

name\_en\_normalized varchar(100) NOT NULL

)

ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE vehicle

(

id integer NOT NULL,

vehiclemodel\_id integer NOT NULL,

fueltype\_id integer,

axletype\_id integer,

bodytype\_id integer,

enginetype\_id integer,

drivetype\_id integer,

fuelmixturetype\_id integer,

vehicletype\_id integer NOT NULL,

year\_from date,

year\_to date,

name varchar(200),

kw integer,

hp integer,

cylinders smallint,

ccm integer,

gross\_weight\_tons integer,

nominal\_capacity smallint,

valves\_total smallint

)

ENGINE=MyISAM DEFAULT CHARSET=utf8;

Maria Indexy

<U+FEFF>ALTER TABLE product ADD PRIMARY KEY product\_pkey (id);

ALTER TABLE vehicle ADD PRIMARY KEY vehicle\_pkey (id);

ALTER TABLE treenode ADD PRIMARY KEY treenode\_pkey (id);

CREATE INDEX product\_vehicle\_product\_id\_idx ON product\_vehicle (product\_id);

CREATE INDEX product\_vehicle\_vehicle\_id\_idx ON product\_vehicle (vehicle\_id);

CREATE INDEX product\_supplier\_id\_idx ON product (supplier\_id);

CREATE INDEX product\_elitsupplier\_id\_idx ON product (elitsupplier\_id);

CREATE INDEX treenode\_treenodetype\_id\_idx ON treenode (treenodetype\_id);

CREATE INDEX treenode\_product\_treenode\_id\_idx ON treenode\_product (treenode\_id);

CREATE INDEX treenode\_product\_product\_id\_idx ON treenode\_product (product\_id);

CREATE INDEX treenode\_product\_treenodetype\_id\_idx ON treenode\_product (treenodetype\_id);

CREATE INDEX treenode\_parent\_id\_idx ON treenode (parent\_id);

CREATE INDEX treenode\_parent\_treenodetype\_id\_idx ON treenode (parent\_treenodetype\_id);

CREATE INDEX treenode\_mptt\_min\_idx ON treenode (mptt\_min);

CREATE INDEX treenode\_mptt\_max\_idx ON treenode (mptt\_max);

LOAD INDEX INTO CACHE product;

LOAD INDEX INTO CACHE product\_vehicle;

LOAD INDEX INTO CACHE vehicle;

LOAD INDEX INTO CACHE treenode;

LOAD INDEX INTO CACHE treenode\_product;

* 1. Postup

mysql> CREATE DATABASE test1 CHARACTER SET utf8 COLLATE utf8\_general\_ci;$ mysql -u root -p test1 </tmp/mirek-db/mysql/mysql-create-table.sqlmysql> use test1

# MyIssam

load data infile '/tmp/mirek-db/mysql/product.txt' replace into table product;

Query OK, 740928 rows affected, 65535 warnings (22.06 sec),22.31, 20.68, 23.59, 25.28, 23.6, 22.38

Records: 740928 Deleted: 0 Skipped: 0 Warnings: 13377958

load data infile '/tmp/mirek-db/mysql/product\_vehicle.txt' replace into table product\_vehicle;

Query OK, 55252244 rows affected (30.98 sec), 35.02, 31.58, 34.75, 33.83, 41.35, 37.58

Records: 55252244 Deleted: 0 Skipped: 0 Warnings: 0

load data infile '/tmp/mirek-db/mysql/treenode.txt' replace into table treenode;

Query OK, 41573 rows affected (0.18 sec) 0.16, 0.18, 0.16, 0.17, 0.17

Records: 41573 Deleted: 0 Skipped: 0 Warnings: 0

load data infile '/tmp/mirek-db/mysql/treenode\_product.txt' replace into table treenode\_product;

Query OK, 1955081 rows affected (1.52 sec) 1.39, 1.41, 2.36, 1.38, 1.38, 1.37

Records: 1955081 Deleted: 0 Skipped: 0 Warnings: 0

load data infile '/tmp/mirek-db/mysql/vehicle.txt' replace into table vehicle;

Query OK, 66707 rows affected (0.23 sec) 0.97, 0.75, 0.40, 0.21, 0.22

Records: 66707 Deleted: 0 Skipped: 0 Warnings: 0

Vytvorenie a load indexov

mysql -u root -p test1 </tmp/mirek-db/mysql/mysql-indexes.sql

jednotlive selecty v příloze. Po kazdem selectu flush query cache; reset query cache

loadnuti novych indexu;

memory

The storage engine for the table doesn't support preload\_keys

Key caches not supported. Key caches are not supported for partitioned tables. The CACHE INDEX and LOAD INDEX INTO CACHE statements, when you attempt to use them on tables having user-defined partitioning, fail with the errors The storage engine for the table doesn't support assign\_to\_keycache and The storage engine for the table doesn't support preload\_keys, respectively.

max\_heap\_table\_size

U InnoDB

The storage engine for the table doesn't support preload\_keys

Vzpnout auto\_commit, jinak to trva dlouho

innodb\_autoinc\_lock\_mode=2 aby se při bulk loadu nezamykala tabulka – to jsem neudělal

Postgres

V configuraku

Log\_min\_duration\_statement = 0 // puvodne bylo -1

Předpokládejme ze postgre cachuje jen provadeci plany

Z postgre se lip zjistujou statistiky

Vybirali jsme query, který mají kratkej vzpis, aby to nezpomalovalo. Nepouzivali jsme like a between, ackoli by to bylo v use casech vhodny, ale vyhledem k tomu, ze mezi porovnavanymi systémy jsou ruzne dbms, které mnohou jinak provádět nektere operace, byly tyto prikazy nahrayzenz jinými

Nejak jsem zanedbal vytvareni indexu…

Order by v mysql spomaluje

* 1. Výsledky

1. Výběr
   1. Váhy kritérií výběru
   2. Výsledek
2. Závěr

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Výběr open-source databázového systému pro katalog dílů

BAKALÁŘSKÁ práce

Student : Miroslav Čech

Vedoucí : Ing. Dušan Chlapek, Ph.D

Oponent : Jméno a příjmení s tituly

**2014**

# 

**Prohlášení**

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne den. měsíc 201x . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Jméno a příjmení studenta

# 

**Poděkování**

Zde je prostor pro vyjádření poděkování…………………………….

**Abstrakt**

Obsahuje zaměření a hlavní cíl práce, způsob dosažení cíle, přínos práce (vlastní příspěvek k řešenému tématu) a stručně popsanou strukturu práce.

**Klíčová slova**

Seznam nejvýznamnějších odborných výrazů charakterizujících téma závěrečné práce.

**Abstract**

Anglická verze abstraktu.

**Keywords**

seznam klíčových slov v anglickém jazyce.

Potraviny 7600

Doprava 8400

Vstupy 2010

Ubytko 6600

Autodoprava 1100