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 : RNDr. Helena Palovská, Ph.D.

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 1.května 2014

.....

Miroslav Čech

Poděkování

Děkuji vedoucímu mé práce za cenné rady a připomínky a mým kolegům, kteří mi umožnili na této práci pracovat v dobrých podmínkách.

Abstrakt

Cílem této práce je vybrat vhodný relační systém řízení báze dat pro konkrétní aplikaci. Výběr bude učiněn na základě kritérií, která budou v práci stanovena. Práce je rozdělena na sedm kapitol počínaje Úvodem, v němž budou rozepsány cíle práce, řešená oblast a problémy současného stavu. Ve druhé kapitole budou stanoveny požadavky na vybíraný dbms (systém řízení báze dat) a přesný popis kritérií pro výběr a to jak kritérií předvýběrových, tak hodnotících. Na tuto kapitolu navazují kapitoly Předvýběr a Měření. V nich bude vybráno, jaká řešení budou měřena a popsáno, jakým konkrétním způsobem. Dalšími kapitolami budou Výsledky a Výběr, ve kterých budou prezentovány výstupy z měření, a bude proveden výběr na základě stanovených kritérií. Celá práce bude završena Závěrem.

Klíčová slova

Dbms, storage engine, katalog dílů, MariaDB, PostgreSQL, výběr.

Abstract

The target of this thesis is choice of appropriate relation database management system for the specific application. Choice will be based on criteria formulated in this thesis. Thesis is divided to seven chapters beginning with Preamble, where the will be specified the domain, targets and problems of contemporary state. In the second chapter the demands to the selected dbms (databse management system) and precise description of pre-selection and evaluation criteria will be set. This chapter is followed up with chapters Pre-selection and Measurement. There will be selected, which solutions will be measured, and will be descripted the methods. Next chapters will be Results and Selection, where the outputs of measurement will be presented and the selection will be done on basis of set criteria. Whole thesis will peak in Conclusion

Keywords

Dbms, storage engine, katalogue of parts, MariaDB, PostgreSQL, selection.

Obsah

1 Úvod

1.1 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.

1.2 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 databázových 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 přepracováno 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 používaném 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 (5). 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 (6) 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 případy 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.

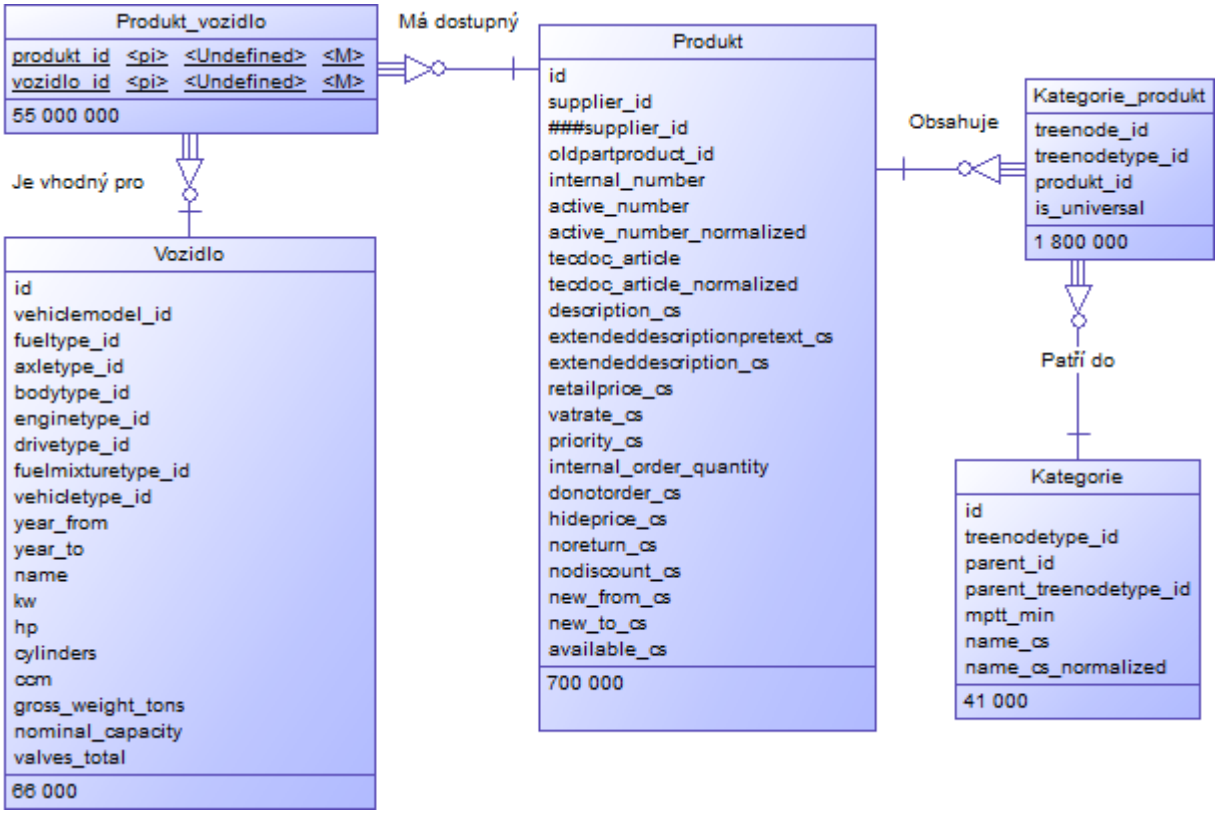
Z mimoškolních zdrojů jsem nalezl disertační práci s názvem Comparing the Performance of Open Source and Proprietary Relational Database Management Systems autora Seana Stevena Coatse,

která by mohla být cenným vodítkem při určování metodologických postupů a způsobů hodnocení měření.

1.3 Řešená oblast

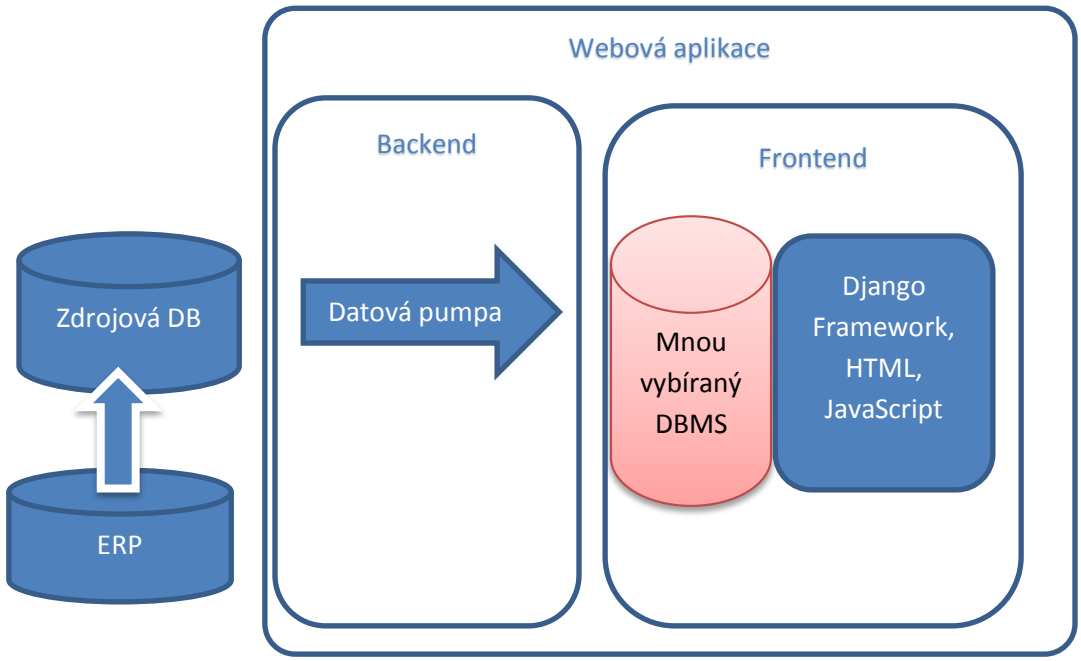
Předmětem mé bakalářské práce je výběr vhodného ukládacího enginu relačního databázového systému pro katalog dílů. Jedná se o webovou aplikaci společnosti, která nabízí autodíly v několika státech nejen střední Evropy. Tato společnost má již vybudovanou infrastrukturu počítačového vybavení včetně databázových 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.



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 přes 700 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 66 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 Schéma DB

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

1.3.1 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.

1.3.2 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. ORM je zkratkou pro objektově relační mapování. 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.

2 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, tedy systém řízení báze dat, 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 (7). Proč MySQL nebylo zahrnuto do předvýběru je více informací v kapitole Předvýběr množiny DBMS.

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

2.1 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.

2.1.1 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 jako doba v sekundách, za kterou jsou načtena všechna data do databáze. 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.

2.1.2 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.

2.1.3 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.

2.1.4 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.

2.2 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.

2.2.1 Licence

Jednou z nejdůležitějších KO kritérií je licence jak samotného DBMS tak i storage enginu. Licence musí splňovat charakteristiku svobodné licence (8) 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.

2.2.2 Nulová 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.

2.2.3 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.

2.2.4 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.

2.2.5 Oficiální vývoj

Podmínkou zařazení dbms a jeho storage enginů do testování je skutečnost, že je řešení oficiálně vyvíjeno. Nechceme testovat velké množství různých odnoží 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.

3 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.

Tabulka 1 Seznam relačních dbms a jejich vlastností

Produkt	Licence	Nulová cena	Architektura	Dostupnost oficiálních ovladačů	Oficiální vývoj	Splněna všechna kritéria
4D (4th Dimension) (9)	Proprietární	Ne	Ne	Ne	Ano	Ne
ADABAS (10)	Proprietární	Ne	Ne	Ne	Ano	Ne
Adaptive Server Enterprise (11)	Proprietární	Ne	Ano	Ne	Ano	Ne
Advantage Database Server (ADS) (11)	Proprietární	Ne	Ano	Ne	Ano	Ne
Altibase (12)	Proprietární	Ne	Ano	Ne	Ano	Ne
Apache Derby (13)	Apache	Ano	Ano	Ne	Ano	Ne
Clustrix (14)	Proprietární	Ne	Ne	Ne	Ano	Ne
CUBRID (15)	GPL	Ano	Ano	Ne	Ano	Ne
Datacom (16)	Proprietární	Ne	Ne	Ne	Ano	Ne
DB2 (17)	Proprietární	Ano	Ano	Ne	Ano	Ne
Drizzle (18)	GPL, BSD	Ano	Ano	Ne	Ano	Ne
Empress Embedded Database (19)	Proprietární	Ne	Ano	Ne	Ano	Ne
EXASolution (20)	Proprietární	Ne	Ano	Ne	Ano	Ne
Firebird (21)	IDPL	Ano	Ano	Ne	Ano	Ne
H2 (22)	EPL	Ano	Ano	Ne	Ano	Ne
HP NonStop SQL (23)	Proprietární	Ne	Ano	Ne	Ano	Ne
HSQLDB (24)	BSD	Ano	Ano	Ne	Ano	Ne
Informix Dynamic Server (17)	Proprietární	Ne	Ano	Ne	Ano	Ne
MariaDB (25)	GPL	Ano	Ano	Ano	Ano	Ano
MaxDB (11)	Proprietární	Ano	Ano	Ne	Ano	Ne
MemSQL (26)	Ne	Ne	Ne	Ne	Ano	Ne
Microsoft Access (27)	Proprietární	Ne	Ne	Ne	Ano	Ne
Microsoft SQL Server (27)	Proprietární	Ne	Ano	Ne	Ano	Ne
Microsoft SQL Server Compact (Embedded Database) (27)	Proprietární	Ne	Ne	Ne	Ano	Ne

Produkt	Licence	Nulová cena	Architektura	Dostupnost oficiálních ovladačů	Oficiální vývoj	Splněna všechna kritéria
Microsoft Visual Foxpro (27)	Proprietární	Ne	Ano	Ne	Ano	Ne
MonetDB (4)	MDBL	Ano	Ne	Ne	Ano	Ne
MySQL (28)	Proprietární, GPL	Ne	Ano	Ano	Ano	Ne
Oracle (29)	Proprietární	Ano	Ano	Ano	Ano	Ne
PostgreSQL (29)	PostgreSQL	Ano	Ano	Ano	Ano	Ano
SQLBase (30)	Proprietární		Ne	Ne		
SQLite (31)	Public Domain	Ano	Ne	Ano	Ano	Ne
Superbase (32)	Proprietární	Ne	Ne	Ne	Ne	
Teradata (33)	Proprietární	Ne	Ano	Ne	Ano	Ne
UniData (34)	Proprietární	Ne	Ano	Ne	Ano	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 (30), 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 Nulová 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 (31) a ukládání přímo do souborových struktur bez možnosti kontroly přístupu uživatelů k datům pro SQLite (26), 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 (32). 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 (33).

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 (38):

Tabulka 2 Enginy MariaDB

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

Výběr MariaDB enginů sestával ze dvou částí. V první fázi jsem z webu (34) 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 úložištěm, 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 (34). 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 kritériu 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 například engine XtraDB identifikuje jako InnoDB. To je důvod, proč se v některých příkazech vyskytují názvy enginů, které nejsou testovány.

Tabulka 3 Důvody zamítnutí enginů

Engine	Důvody 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

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

4 Měření

4.1 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 2.27 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 distribucí 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 mí 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.

4.2 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 4 Vstupní data

Tabulka	Velikost souboru	Počet záznamů	Počet sloupců
Product	378 MB	740928	96
Product_vehicle	960 MB	55252244	2
Treenode_product	30 MB	1955081	4
Treenode	4,9 MB	41573	20
Vehicle	5,9 MB	66707	19

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.

4.3 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 product 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 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 cizí klíč. 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, spouští 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 vykonáváním 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 by naměřené výsledky nemusely plně vypovídat o výkonu storage enginu, ale vypovídaly 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.

Měření načítání dat na enginu XtraDB jsem prováděl dvakrát, protože výsledky, které jsem naměřil během prvního měření, se mi zdály nereálné, protože doba načítání byla oproti předchozím dvěma enginům několikanásobně pomalejší, ačkoli byl transakční mód vypnut. Znovu jsem tedy zkontroloval toto nastavení, a ujistil jsem se, že parametr *auto_commit* je nastaven na hodnotu 0. To znamená, že je vypnuto automatické potvrzování příkazů. Změna nastavení v druhém měření spočívala ve vypnutí zamykání tabulky během importu dat. Toho bylo docíleno přidáním řádku

```
innodb_autoinc_lock_mode = 2
```

do konfiguračního souboru dbms MariaDB a restartováním systému příkazem

```
/etc/init.d/mysql restart
```

Ve druhém měření se hodnoty nepatrně zlepšily, avšak nebylo dosaženo takového zlepšení, jaké jsem si představoval. Výsledky enginu XtraDB jsou stále výrazně horší než ostatní enginy dbms MariDB.

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

TABLE o parametr *ENGINE*, který jako argument přijímá název storage enginu, který má být pro tabulku použit.

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 (39).

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 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.

4.4 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 tabulky 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.

Indexy nad ostatními sloupci, které nejsou uvedeny v tomto dávkovém souboru, jsem si vytvářel až ve chvíli, kdy byly nutné k provedení měření.

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. Pro měření přesnějších časů jsem použil tzv. profiling. Nastavit profiling je potřeba při každém spuštění řádkového klienta příkazem

```
Set profiling = 1;
```

Protože ve výchozím nastavení příkaz

```
Show profiles;
```

zobrazuje pouze deset posledních dotazů, bylo třeba ještě zvýšit počet zobrazovaných dotazů. Toho jsem dosáhl příkazem

```
set profiling_history_size = 100;
```

Jednotlivá měření jsem dělal tím způsobem, že jsem vybral jeden dotaz s konkrétním parametrem a ten jsem spustil. Po dokončení běhu příkazu jsem v MariaDB vymazal vyrovnávací paměť a tento dotaz jsem zopakoval. Vymazání vyrovnávací paměti jsem provedl příkazy

```
flush query cache;
reset query cache
```

Jde o paměť, která si dočasně pamatuje znění SQL dotazu a jeho výsledky, nazývá se query cache. Není podobná vyrovnávací paměti známé jako buffer cache z PostgtreSQL, její účel jejiný. Query cache si pamatuje dotaz (40), tak jak byl napsán a pokud je znovu použit tentýž dotaz, který je identický s již uloženým dotazem, tento dotaz se nevykoná standardním způsobem, ale výsledky se přečtou z této vyrovnávací paměti. PostgreSQL takovouto vyrovnávací paměť nemá. Pracuje však s buffer cache, a to tím způsobem, že často používané datové bloky zkopíruje z disku do paměti, aby se tím snížila rychlost dotazů (30). Toto chování není v rozporu s mým měřením, a proto není potřeba tuto paměť mazat. Ve výsledcích mého měření budou zohledněny výsledky prvního vykonání dotazu.

5 Výsledky

5.1 Načítání dat

Způsobem popsaným v předchozí kapitole, jsem dospěl k následujícím výsledkům. Nejprve ukáži výsledky jednotlivých měřených enginů, a nakonec udělám porovnání všech enginů mezi sebou.

V tabulce Doba trvání načítání dat - engine Aria jsou hodnoty naměřené na enginu Aria. Vždy je uvedeno, do jaké tabulky bylo vkládáno, a jakých časů bylo dosaženo. V posledním sloupci je zprůměrovaná hodnota všech sedmi měření. Jednotky měření jsou vteřiny.

Tabulka 5 Doba trvání načítání dat - engine Aria

Tabulka s enginem Aria	Měření 1 [s]	Měření 2 [s]	Měření 3 [s]	Měření 4 [s]	Měření 5 [s]	Měření 6 [s]	Měření 7 [s]	Průměr [s]
Product	26,06	22,06	22,31	20,68	23,59	25,28	26,60	23,80
Product_vehicle	33,87	30,98	35,02	31,58	34,75	33,83	41,35	34,48
Treenode	0,17	0,18	0,16	0,18	0,16	0,17	0,17	0,17
Treenode_product	1,40	1,52	1,39	1,41	2,36	1,38	1,37	1,55
Vehicle	0,22	0,23	0,97	0,75	0,40	0,21	0,22	0,43

V následující tabulce Doba trvání načítání dat - engine MEMORY jsou uvedena data naměřená na databázových tabulkách s MEMORY enginem.

Tabulka 6 Doba trvání načítání dat - engine MEMORY

Tabulka s enginem MEMORY	Měření 1 [s]	Měření 2 [s]	Měření 3 [s]	Měření 4 [s]	Měření 5 [s]	Měření 6 [s]	Měření 7 [s]	Průměr [s]
Product	25,92	26,85	28,3	28,91	24,56	28,76	26,12	27,06
Product_vehicle	32,41	29,19	34,1	30	30,84	29,82	26,31	30,38143
Treenode	0,28	0,26	0,18	0,18	0,18	0,18	0,18	0,205714
Treenode_product	1,59	1,46	1,28	2,15	1,26	1,26	1,28	1,468571
Vehicle	0,23	0,21	0,18	0,56	0,18	0,19	0,18	0,247143

Následující tabulka Doba trvání načítání dat - engine XtraDB ukazuje údaje naměřené na tabulkách s enginem XtraDB. U tohoto enginu je na první pohled vidět značný rozdíl v rychlosti načítání dat oproti ostatním enginům. Zvláště hodnoty pro načítání do tabulky product_vehicle, které se pohybují okolo čtyř minut.

Tabulka 7 Doba trvání načítání dat - engine XtraDB

Tabulka s enginem XtraDB	Měření 1 [s]	Měření 2 [s]	Měření 3 [s]	Měření 4 [s]	Měření 5 [s]	Měření 6 [s]	Měření 7 [s]	Měření 8 [s]	Průměr [s]
Product	47,62	50,45	44,76	45,67	54,37	54,10	51,90	49,84	49,84
Product_vehicle	303,00	288,00	281,00	296,00	288,00	316,00	289,00	294,43	294,43
Treenode	3,22	1,94	1,58	0,98	1,00	0,55	0,83	1,44	1,44
Treenode_product	25,13	27,64	25,57	23,15	24,90	25,41	28,58	25,77	25,77
Vehicle	1,83	0,82	1,19	0,77	1,06	0,95	2,31	1,28	1,28

Posledním měřeným storage enginem byl engine postgres. Naměřené hodnoty se nacházejí v tabulce Doba trvání načítání dat - engine postgres.

Tabulka 8 Doba trvání načítání dat - engine postgres

Tabulka s enginem postgres	Měření 1 [s]	Měření 2 [s]	Měření 3 [s]	Měření 4 [s]	Měření 5 [s]	Měření 6 [s]	Měření 7 [s]	Průměr [s]
Product	18,34	17,62	14,46	15,50	16,21	15,07	20,77	16,85
Product_vehicle	84,35	94,90	84,51	84,04	79,61	76,79	91,27	85,07
Treenode	0,23	0,26	0,20	0,41	0,21	0,20	0,20	0,25
Treenode_product	3,38	3,64	3,21	3,39	3,72	3,35	2,95	3,38
Vehicle	0,33	0,30	0,32	0,33	0,35	0,33	0,29	0,32

Z dat, které jsou zobrazeny v předchozích tabulkách byly vzaty průměrné hodnoty doby trvání načítání dat do jednotlivých tabulek a tyto hodnoty byly sečteny. Výsledné hodnoty jsou uvedeny ve sloupci Průměrná doba načtení dat do všech tabulek v tabulce Porovnání jednotlivých enginů v rychlosti načítání dat do tabulek

Tabulka 9 Porovnání jednotlivých enginů v rychlosti načítání dat do tabulek

Pořadí	Engine	Průměrná doba načtení dat do všech tabulek
1.	MEMORY	59,36 vteřin
2.	Aria	60,43 vteřin
3.	postgre	105,86 vteřin
4.	XtraDB	372,75 vteřin

Z těchto výsledků je zřejmé, že použití enginu XtraDB se nejeví jako vhodné v případě, že chceme často načítat do databáze velká množství dat. Oproti zbylým dvěma testovaným MariaDB enginům je výkon více než dvakrát horší. Může to být dáno tím, že se jedná o engine s podporou transakcí, avšak engine postrge tak výrazně špatných výsledků nedosáhl. Překvapením pro mne bylo, že enginy MEMORY a Aria mají téměř identické výsledky, ačkoli principiálně fungují každý jinak. MEMORY engine si udržuje veškerá data v paměti, Aria data zapisuje na disk. Původně jsem očekával, že rozdíl, ať ve prospěch toho, či onoho enginu, bude výraznější.

5.2 Jednoduché dotazy

V následujících tabulkách uvedu výsledky naměřené v testu rychlosti různých operací select. Tyto tabulky obsahují jen nejdůležitější data, a to dobu, za kterou byl dotaz vykonán ve chvíli, kdy byl dotaz položen poprvé, a průměr ze všech naměřených hodnot pro konkrétní hodnotu parametru, a konečně průměrnou hodnotu doby odezvy daného dotazu nezávisle na hodnotě parametru. Surová naměřená data se nacházejí v příloze.

Prvním a nejdůležitějším dotazem je vyhledávání produktu podle hodnoty primárního klíče. Ona hodnota je uvedena v prvním sloupci tabulky Doba odezvy dotazu na vyhledání produktu podle primárního klíče. Hodnoty jsou v milisekundách.

Tabulka 10 Doba odezvy dotazu na vyhledání produktu podle primárního klíče

	Aria		MEMORY		XtraDB		postgres	
parametr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr
1	0,49	0,53	0,48	0,75	2,47	0,90	1,75	0,63
5161000	0,51	0,53	0,52	0,49	8,90	1,83	1,18	0,71
1000007	0,47	0,46	0,56	0,75	0,68	0,66	0,60	0,68
500000	0,51	0,81	0,48	0,46	28,87	4,74	0,58	0,59
2000	2,68	0,81	0,47	0,47	10,56	3,17	0,61	0,84
Průměr		0,63		0,59		2,26		0,69

V tabulce Doba odezvy dotazu na vyhledání produktu podle ceny, jsou uvedeny doby odezvy v milisekundách, pro jednotlivé hodnoty parametru *retailprice_cs*, které jsou uvedeny v prvním sloupci.

Tabulka 11 Doba odezvy dotazu na vyhledání produktu podle ceny

	Aria		MEMORY		XtraDB		postgres	
parametr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr
12,5	0,69	0,59	2,69	1,03	0,52	0,57	0,45	0,44
110000	9,68	1,78	0,41	0,42	0,53	0,86	0,21	0,20
2932650	0,51	1,42	0,48	0,47	0,56	0,62	0,15	0,20
1855875	0,50	0,52	0,37	0,43	0,56	0,56	0,20	0,18
954375	0,63	0,63	0,41	0,40	0,54	0,54	0,15	0,19
Průměr		0,99		0,55		0,63		0,24

V tabulce Doba odezvy dotazu na vyhledání produktu podle interního identifikátoru jsou uvedeny hodnoty doby odezvy pro vyhledávání nad indexovaným polem tabulky *product* s textovou hodnotou *internal_number*.

Tabulka 12 Doba odezvy dotazu na vyhledání produktu podle interního identifikátoru

	Aria		MEMORY		XtraDB		Postgres	
parametr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr	První dotaz	Průměr
0 001 108 211	0,68	0,88	0,76	0,47	0,83	0,92	8,31	1,35
QB WS 0344 A	0,55	0,51	0,55	0,44	0,58	0,51	0,41	0,39
SD 0 986 474 332	0,78	0,62	0,43	0,44	0,55	0,56	0,19	0,21
0 132 801 002	1,02	0,86	0,50	0,43	0,64	0,74	0,17	0,17
9XX 340 369-011	0,58	0,87	0,42	0,43	0,65	0,60	0,18	0,19
Průměr		0,75		0,44		0,67		0,46

Posledním jednoduchým testovaným dotazem bylo vyhledávání ve vazebné tabulce *product_vehicle* na základě hodnoty ve sloupci *vehicle_id*. Výsledky jsou uvedeny v tabulce Doba odezvy dotazu na vyhledání produktů pro dané vozidlo.

Tabulka 13 Doba odezvy dotazu na vyhledání produktů pro dané vozidlo

	Aria		MEMORY		XtraDB		Postgres	
parametr	První dotaz	průměr	První dotaz	průměr	První dotaz	průměr	První dotaz	Průměr
13706	0,71	0,72	3,78	1,04	2854,46	409,24	0,37	0,44
66707	0,51	0,48	2,63	0,74	100,01	14,76	0,43	0,34
63014	0,47	0,47	6,93	1,78	47,34	7,23	0,15	0,14
5630	0,48	0,47	0,50	0,64	67,30	10,09	0,72	0,22
28971	0,55	0,65	0,39	0,39	73,70	11,00	0,40	0,21
Průměr		0,56		0,92		90,46		0,27

V tabulce Porovnání doby odezvy jednotlivých dotazů jsou shrnuty výsledky z výše uvedených tabulek, a na základě těchto hodnot jsou spočítána pořadí, na kterých se v každém testovaném scénáři daný engine umístil. V posledním řádku je součet průměrných časů dosažených daným enginem. Toto číslo reprezentuje, za jak dlouho by konkrétní engine stihl vykonat všechny měřené dotazy. Pokud by důležitost všech dotazů uvedených v tabulce byla shodná, tak v tomto měření se jako nejvhodnější engine ukazje postgre, následovaný MEMORY enginem a Ariou. Nejnevhodnějším se ukazuje XtraDB, který dosáhl hodnoty o řád vyšší. Hodnoty v tabulce jsou udávány v milisekundách.

Tabulka 14 Porovnání doby odezvy jednotlivých dotazů

	Aria		MEMORY		XtraDB		postgres	
Vyhledávání	Průměr	Pořadí	Průměr	Pořadí	Průměr	Pořadí	Průměr	Pořadí
Dle primárního klíče	0,63	2	0,59	1	2,26	4	0,69	3
Dle ceny	0,99	4	0,55	2	0,63	3	0,24	1
Dle textového pole	0,75	4	0,44	1	0,67	3	0,46	2
Ve vazebné tabulce	0,56	2	0,92	3	90,46	4	0,27	1
Součet	2,93		2,5		94,02		1,66	

5.3 Získání počtu produktů v dané kategorii pro dané vozidlo

Stejně jako pro předchozí dotazy, přesné znění jednotlivých SQL dotazů a všechna naměřená data se nacházejí v příloze. V tabulce Výsledky měření dotazu pro získání počtu produktů v dané kategorii jsou zobrazeny průměrné dosažené hodnoty. Na základě průměru je určeno pořadí, v jakém se daný engine umístil. Pro představu, jak naměřená data vypadají, jsou v tabulce ještě uvedeny upřesňující statistické vlastnosti.

Tabulka 15 Výsledky měření dotazu pro získání počtu produktů v dané kategorii

	Aria	MEMORY	XtraDB	postgres
průměr	17,79589216	7,443514324	207,7151814	21,01543243
maximum	36,41251	8,6562	6104,5128	90,619
minimum	16,39174	6,6486	21,21476	16,264
medián	17,17638	7,075	33,34028	17,349
Pořadí dle průměru	2	1	4	3

Je opět zřejmé, že engine XtraDB není nejvhodnějším pro tento typ úlohy, dosáhl nejhorších výsledků a na postgres, umístivší se na třetím místě, ani zdaleka nedosahuje. Je také zajímavé, jakého rozsahu hodnoty XtraDB nabývají. Nedá se říci, že každý položený dotaz je o řád pomalejší než u ostatních enginů. U tohoto enginu se často stává, že v některých případech stejný dotaz trvá neúměrně delší dobu než při ostatních měřeních.

6 Výběr

Na základě dat změřených a publikovaných v předchozí kapitole je nyní třeba vybrat nejvhodnější engine pro databázový systém pro webovou aplikaci katalogu dílů. Bylo by možné vzít pořadí v jednotlivých scénářích, a engine s nejnižším součtem pořadí by se stal vítězem. Tento přístup je ovšem problémový z toho důvodu, že nereflektuje, o kolik je jeden engine v daném scénáři rychlejší než jiný.

Druhým možným přístupem by bylo sečíst hodnoty, kterých každý engine dosáhl, a jako nejlepší by se ukázal ten s nejnižší výslednou hodnotou. Tento přístup už zohledňuje rozdíly mezi enginy, ale vzhledem k tomu, že bychom přičítali k času načítání dat, který je ve vteřinách, časy výběru produktu na základě parametrů, které jsou v milisekundách, výsledek by byl hodně zkreslený prvně zmíněným scénářem. Ovšem tento přístup hodlám použít s jednou významnou modifikací. Pro každý scénář stanovím váhy, které budou zohledňovat četnost použití scénáře a také jednotky, v kterých jsou měřeny.

6.1 Váhy kritérií výběru

V první fázi stanovování vah jsem vynásobil dosažené výsledky takovým koeficientem, abych se dostal na porovnávání stejných jednotek. Tímto jsem dosáhl toho, že výsledky budou porovnatelné, a každé z kritérií bude mít dostatečnou váhu na ovlivnění výsledku. Protože většina výsledků je v milisekundách, a výsledky načítání dat jsou ve vteřinách, byly nejprve tyto výsledky převedeny do shodných jednotek, tedy milisekund.

Bylo také třeba zohlednit, jak důležitá jsou jednotlivá kritéria, a jak často konkrétní procesy probíhají. Lze říci, že select dotazy jsou vykonávány více než sto tisíckrát častěji než načítání dat do databáze. Dále také je důležitější, jak rychlé je vyhledávání podle parametrů včetně primárního klíče, než získání počtu produktů v kategorii, ale ne tolikrát. Získávání počtu produktů je důležité kritérium, na druhou stranu kritérium výběru produktů v sobě zahrnuje několik dotazů. Rozhodl jsem se tedy, vyhledávání produktů na základě parametrů bude mít pouze dvakrát vyšší váhu než získání počtu produktů v kategorii.

Váhy jsou tedy stanoveny následovně. Všechna naměřená data byla převedena na stejné jednotky, milisekundy. Kritérium načítání dat dostalo váhu 1. Součet všech průměrných hodnot výsledků vyhledávání dostal váhu 100 000, a získání počtu produktů v kategorii o polovinu nižší, tedy 50 000.

Tabulka 16 Naměřené hodnoty ve všech scénářích

Kritérium	Aria	MEMORY	XtraDB	postgres
načítání dat	60430	59360	372750	105860
vyhledávání produktů, součet	2,93	2,5	94,02	1,66
získání počtu produktů v kategorii	17,79589	7,443514	207,7152	21,01543

Naměřené výsledky v milisekundách jsou v tabulce Naměřené hodnoty ve všech scénářích, výsledná data včetně vah a výsledků je v kapitole 6.2 v tabulce Výsledky porovnání po aplikaci vah.

6.2 Výsledek

Po aplikaci vah na naměřené hodnoty bylo dosaženo výsledků, jak je uvedeno v tabulce Výsledky porovnání po aplikaci vah. Nejlépe se umístil engine MEMORY, který stabilně dosahoval dobrých výsledků. Bylo pro mne ovšem velkým překvapením, že jeho výsledky se nijak dramaticky nelišily od výsledků enginu Aria. S přihlédnutím k faktu, že využití MEMORY enginu spotřebovává nezměrně více paměti oproti použití řešení s perzistentním úložištěm, a také že data nejsou spolehlivě uložena například po pádu serveru, se použití tohoto enginu v kontextu webové aplikace pro katalog dílů nevyplatí. Ovšem je nutno říci, že podle toho, jak byla stanovena kritéria, se výkonnostně ukazuje jako nejvhodnější, což je pravda.

Tabulka 17 Výsledky porovnání po aplikaci vah

Kritérium	Váha	Aria	MEMORY	XtraDB	postgres
načítání dat	1	60430	59360	372750	105860
vyhledávání produktů, součet	100000	293000	250000	9402000	166000
získání počtu produktů v kategorii	50000	889795	372176	10385759	1050772
Výsledek		1 243 224	681 535	20 160 509	1 322 631

Dále se ukázalo, že výkon enginu postgres není, oproti použití dvou výše zmíněných MariaDB, tak špatný jak se na webové aplikaci ukazovalo. Pokud má dbms PostgreSQL nějaké výkonnostní problémy, tak se v mém měření neukázaly, a můžeme předpokládat, že tyto problémy nejsou zapříčiněny výkonem využívaného enginu, ale samotného dbms. Z měření je ale zřejmé, že načítání dat technikou bulk load, rozhodně není z nejrychlejších.

Co překvapením nebylo, jsou hodnoty dosažené enginem XtraDB, který je odnoží původního InnoDB enginu. Jedná se o transakční engine, a jeho architektura je tomu přizpůsobená, a proto využití v oblastech, kde se nevyžaduje transakčního přístupu, není z výkonnostního hlediska vhodné. Dále také u tohoto enginu byly naměřeny obrovské odchylky některých měření. Většinou se jednalo o první měření s konkrétní hodnotou parametru, ale nebylo to pravidlem. U ostatních

enginů také byly naměřeny různé odchylky od průměru, ale u žádného z enginů nebyly tak výrazné, jako právě u tohoto. Po zjištění, že první vykonání dotazu zabírá takovýto čas, jsem se rozhodl změnit způsob měření a to tak, aby nebyly spouštěny dotazy se stejnou hodnotou parametru po sobě, ale aby se vždy spouštěly všechny varianty dotazu po dávkách. Naměřená data se však nijak významně nelišila od předchozích naměřených.

Mnou vybraným storage enginem pro webovou aplikaci katalogu dílu se stává engine MEMORY pro dbms MariaDB. Dosáhl nejlepších výsledků a předčil tak enginy Aria, XtraDB a postgres.

7 Závěr

V mé práci bylo hlavním cílem dodat podklady pro výběr vhodného systému řízení báze dat pro webovou aplikaci katalogu díl a učinit výběr databázového systému a jeho storage enginu. Součástí práce bylo i stanovení kritérií výběru a předvýběru množiny dbms a také popis postupu měření. Tyto podklady byly dodány a posloužily k výběru, konkrétního řešení, takže tento cíl byl naplněn. Zároveň také způsob, kterým jsem postupoval a kritéria, která jsem vytyčil, v dohledné době poslouží po úpravě předvýběrových kritérií k měření ostatních enginů, a zároveň je umožněno měření zopakovat a ověřit ho, popřípadě po vydání nových verzí enginů zjistit, zdali došlo ke zlepšení.

Cíle vytyčené v zadání mé práce byly naplněny, v prvé řadě jsem charakterizoval řešenou oblast a požadavky na dbms. Myslím se, že velice komplexně jsem naplnil především druhý vytyčený cíl, a to návrh výběrových a předvýběrových kritérií včetně vah, a výběr a způsob testování předvybraných řešení. Praktickou část, tedy testování, hodnocení a výběr jsem v rámci této práce provedl, a na základě mnou naměřených dat a publikovaných výsledků, bylo vybráno řešení pro nasazení. Všechny cíle tedy byly v celé šíři naplněny. V úvodu této práce jsem si určil, že cíle budou naplněny, budou-li porovnány alespoň tři enginy, a to jsem nejen naplnil, ale i předčil.

Nyní shrnu obsah mé práce. V první části mé práce jsem rozepsal oblast, které se má práce týká. Jde o zaběhnutou webovou aplikaci, kterou trápí některé výkonnostní problémy. Také jsem uvedl, že cílem této práce není porovnávat jednotlivé dbms s hlediska všech myslitelných aspektů, ale pouze porovnání storage enginů, protože tato práce je součástí mnohem širšího výzkumu a testování databází prováděného v naší firmě.

Ve druhé části byly rozvedeny požadavky na vybíraný systém řízení báze dat. Byla vytyčena kritéria předvýběru a kritéria hodnotící. Obě tyto skupiny kritérií jsou odůvodněny a jednotlivá kritéria reflektují reálně vytyčená kritéria tak, jak jsou v projektu naší firmy zavedena.

Následující kapitoly se zabývají předvýběrem a postupem měření, ve kterém je popsáno, jakým způsobem jsem postupoval v měření, jaká jsou vstupní data a servery. Způsobem popsaným v této kapitole jsem dospěl k výsledkům publikovaných v páté kapitole, na které navazuje kapitola 6. Výběr.

Největším mým přínosem je, že na základě mnou poskytnutých údajů bylo opravdu vybráno konkrétní řešení pro nasazení. Má práce nebyla jediným vodítkem, ale zato jedním z nejvýznamnějších.

Jako pravděpodobné pokračování v této práci vidím měření a testování ostatních enginů, zvláště pak těch, které se nedostaly do výběru testovaných řešení z důvodu neukládání dat do lokálního úložiště, či z důvodu vývoje jiným subjektem než je vývojář daného dbms. Už nyní se v naší firmě uvažuje o otestování enginu TokuDB.

Dále si myslím, že by mohla být užitečná práce, která by zdokumentovala mnou zjištěné chování XtraDB enginu, kdy některé výsledky jsou značně horší než ostatní, ačkoli byl položený dotaz shodný.

Terminologický slovník	
backend	část aplikace neviditelná uživateli
frontend	část aplikace, se kterou uživatel interaguje
bulk load, také bulk import	načítání dat do databáze ze souboru bez účasti SQL procesoru
cache	Vyrovnávací paměť sloužící ke zrychlení operací mezi dvěma různě rychlými systémy
dbms	systém řízení báze dat. softwarový produkt
escape sekvence	zvláštní posloupnost znaků, která má za úkol zamezit interpretování následujících znaků jakožto příkazů
KO kritéria	filtrační kritéria, mohou být splněna či nesplněna. Při nesplnění alespoň jednoho KO kritéria se testovaná entita hodnotí jako neuspěla
kořenový strom	v teorii grafů jde o takový graf, ve kterém neexistuje cyklus a existuje právě jeden uzel, který nemá přímého předka
Modified Preorder Tree Traversal	technika dotazování a ukládání hiearchických dat v relačním modelu
ORM	objektově relační mapování - proces, kdy se data objektového charakteru převádějí do relačního modelu a naopak
storage engine	komponenta dbms, která zajišťuje ukládání dat do určitého úložiště
transakce	množina operací, které se navenek tváří jako jeden celek

[Vlastní definice autora]

Seznam Literatury

1. **Král, Jakub.** *Porovnání open source databázových systémů s využitím TPC-C testu.* místo neznámé : VŠE, 2013.

2. **Matějka, Martin.** *Implementace testu k porovnání výkonnosti databázových systémů.* místo neznámé : VŠE, 2012.

3. **Véle, Marek.** *Porovnání open source databázových systémů.* místo neznámé : VŠE, 2008.

4. **Kubát, Ondřej.** *Porovnání open source databázových systémů.* místo neznámé : VŠE, 2007.

5. **Filip, Jan.** *Ladění relačních databázových systémů.* místo neznámé : VŠE, 2006.

6. **Vopička, Adam.** *Porovnání prostředí pro ukládání dat v MySQL.* místo neznámé : VŠE, 2008.

7. **Oracle Corporation.** Market Share. *MySQL.* [Online] [Citace: 26. 4 2014.] https://www.mysql.com/why-mysql/marketshare/.

8. **Open Source Initiative.** Open source licences. *Open Source Initiative.* [Online] Open Source Initiative. [Citace: 1. 04 2014.] http://opensource.org/licenses/alphabetical.

9. **4D SAS.** 4D Server. *4D.* [Online] [Citace: 26. 4 2014.] http://www.4d.com/products/4dv13/4dserver.html.

10. **Software AG.** Products. *Software AG.* [Online] [Citace: 26. 4 2014.] http://www.softwareag.com/corporate/products/transactions/adabas/capabilities/default.asp.

11. **Sybase.** Sybase Docs. *Sybase.* [Online] [Citace: 26. 4 2014.] http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.help.ase.15.7.121/doc/html/title.html&docSetID=2036.

12. **Altibase Corp.** In-Memory Database Products. *Altibase.* [Online] [Citace: 26. 4 2014.] http://altibase.com/in-memory-database-hybrid-products/.

13. **Apache.** Apache Derby. *The Apache DB Project.* [Online] 17. 4 2014. [Citace: 26. 4 2014.] http://db.apache.org/derby/.

14. **Clustrix.** Clustrix. *Clustrix.* [Online] [Citace: 26. 4 2014.] http://www.clustrix.com.

15. **CUBRID.** About. *CUBRID.* [Online] [Citace: 26. 4 2014.] http://www.cubrid.org/about.

16. **CA technologies.** Product Sheet CA Datacom/DB. *CA.* [Online] [Citace: 26. 4 2014.] http://www.ca.com/us/~/media/Files/ProductBriefs/cs2277-ca-datacom-v14-ps.PDF.

17. **Chong, Raul F.** *Začínáme s DB-2 Expres - C.* Praha : DNS, a.s., 2011.

18. **Drizzle.** About the Drizzle Project. *Drizzle.* [Online] [Citace: 26. 4 2014.] http://docs.drizzle.org/.

19. **Empress Software Inc.** Embedded database. *Empress.* [Online] [Citace: 26. 4 2014.] http://www.empress.com/about.

20. **EXASOL AG.** Products. *EXASOL.* [Online] [Citace: 26. 4 2014.] http://www.exasol.com/en/products/exasolution-42/.

21. **Firebird Project members.** Firebird 2.5 Quick Start Guide. *Firebird.* [Online] 26. 9 2011. [Citace: 26. 4 2014.] http://www.firebirdsql.org/file/documentation/reference_manuals/user_manuals/Firebird-2.5-QuickStart.pdf.

22. **H2 Group.** QuickStart. *H2.* [Online] [Citace: 26. 4 2014.] http://www.h2database.com/html/quickstart.html.

23. **Hewlett-Packard Development Company, L.P.** HP NonStop SQL/MX database software. *HP.* [Online] [Citace: 26. 4 2014.] http://h20195.www2.hp.com/V2/GetDocument.aspx?docname=4AA4-2651ENW&cc=us&lc=en.

24. **The hsql Development Group.** HyperSQL. *HyperSQL.* [Online] [Citace: 26. 4 2014.] http://hsqldb.org/.

25. **SkySQLcorporation AB.** MariaDB versus MySQL - Compatibility. *MariaDB.* [Online] [Citace: 3. 4 2014.] https://mariadb.com/kb/en/mariadb-versus-mysql-compatibility/.

26. **MEMSQL INC.** One Database for Real-Time & Historical Data. *memsql.* [Online] [Citace: 26. 2 2014.] http://www.memsql.com/product/.

27. **Microsoft.** Support. *Microsoft.* [Online] [Citace: 26. 4 2014.] http://support.microsoft.com/find-solutions/more/?ln=en-us.

28. **Oracle.** Commercial License for OEMs, ISVs and VARs. *MySQL.com.* [Online] [Citace: 3. 4 2014.] http://www.mysql.com/about/legal/licensing/oem/.

29. **Oracle Corporation.** Oracle Products. *Oracle.* [Online] [Citace: 26. 4 2014.] http://www.oracle.com/cz/products/database/standard-edition-one/overview/index.html.

30. **Douglas, Korry and Susan.** *PostgreSQL.* Seatle : Sams Publishing, 2006. 0-672-32756-2.

31. **SQLBase Key Features. *Gupdata.* [Online] [Citace: 3. 4 2014.] http://www.guptatechnologies.com/Products/Data_Management/SQLBase/features.aspx.**

32. Distinctive features. *SQLite.* [Online] [Citace: 3. 4 2014.] https://sqlite.org/different.html.

33. Simpol. Simpol. *Superbase Documentation.* [Online] [Citace: 3. 4 2014.] http://www.simpol.com/docs.html.

34. Teradata Corporation. Technical owerview. *Teradata Database 15.0.* [Online] [Citace: 6. 4 2014.] http://www.teradata.com/WorkArea/linkit.aspx?LinkIdentifier=id&ItemID=12884906114.

35. Rocket Software. UniData v7.3 Technical Documentation. *UniData.* [Online] [Citace: 3. 4 2014.] http://www.rocketsoftware.com/.

36. MonetDB BV. The column-store pioneer. *monetdb.* [Online] [Citace: 3. 4 2014.] http://www.monetdb.com/Home.

37. Django Software Foundation. Databases. *Django Documentation.* [Online] [Citace: 2. 4 2014.] https://docs.djangoproject.com/en/dev/ref/databases/.

38. SkySQLcorporation AB. Storage Engines. *MariaDB.* [Online] [Citace: 3. 4 2014.] https://mariadb.com/kb/en/mariadb-storage-engines/.

39. Oracle Corporation. MySQL 5.7 Reference Manual. *MySQL.* [Online] [Citace: 26. 4 2014.] http://dev.mysql.com/doc/refman/5.7/en/.

40. —. How the Query Cache Operates. *MySQL.* [Online] [Citace: 22. 4 2014.] http://dev.mysql.com/doc/refman/5.7/en/query-cache-operation.html.

41. Hipp, D. Richard. Distinctive Features Of SQLite. *SQLite.* [Online] [Citace: 3. 4 2014.] https://sqlite.org/different.html.

Seznam tabulek

Seznam obrázků

Rejstřík