VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝKONNOVÝ TESTER DATABÁZÍ

DATABASE PERFORMANCE TESTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN ŠULAJ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN OČENÁŠ**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Šulaj Martin**
Program: Informační technologie
Název: **Výkonnový tester databází**
Database Performance Tester
Kategorie: Databáze

Zadání:

1. Nastudujte problematiku SQL databází a výkonnostního testování.
2. Navrhněte způsob testování výkonu SQL databází s různou mírou datové složitosti.
3. Implementujte nástroj pro realizaci těchto testů v různých SQL databázích.
4. Navrhněte několik typů scénářů pro různě složité databáze a různé typy úloh.
5. Demonstrujte funkčnost nástroje na vybraných běhových prostředích.
6. Dosažené řešení zhodnoťte a navrhněte další vylepšení

Literatura:

- Grant Fritchey: SQL Server Query Performance Tuning: 2014

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Očenáš Martin, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 8. listopadu 2018

Abstrakt

Hlavným cieľom tejto práce je vytvorenie nástroja na meranie výkonnosti relačných databáz a následné preukázanie jeho funkčnosti. Výsledný nástroj je logicky rozdelený do 4 častí, pričom každá časť môže fungovať nezávisle. Pomocou nástroja vieme vytvoriť, vykonať a vyhodnotiť testovaciu sadu. Súčasťou nástroja je aj grafické zobrazenie výsledkov. Nástroj pracuje s testovacími sadami, ktoré sú spúšťané na zariadení Raspberry Pi 3 B a na výkonnom laptope. Medzi databázy, ktoré sú týmto nástrojom testovateľné, patria MySQL, PostgreSQL a SQLite.

Abstract

Main goal goal of this project is to create tool for performance testing of relational databases and to prove his functionality. The tool is logically divided into 4 parts and every one of them can work independently. With the use of the tool we ca create, execute and evaluate test suites. Graphical visualisation of processed results is a part of the tool. The tool works with test suites that are executed on Raspberry Pi 3 B device and on well performing laptop. Databases that are tested with this tool are MySQL, PostgreSQL and SQLite.

Kľúčové slová

Databázy, SQL, MySQL, PostgreSQL, SQLite, Výkonnostné testovanie, Raspberry Pi

Keywords

Databases, SQL, MySQL, PostgreSQL, SQLite, Performance testing, Raspberry Pi

Citácia

ŠULAJ, Martin. *Výkonnový tester databází.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Očenáš

Výkonnový tester databází

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Očenáša. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Šulaj
12. mája 2019

Poďakovanie

Rád by som sa poďakoval vedúcemu mojej bakalárskej práce, Ing. Martinovi Očenášovi za odborné rady, dobrú spoluprácu, celkovú pomoc a ochotu pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Výkonnosť systému je zvyčajne posledná vec, na ktorú ľudia myslia pri vývoji systému. Problém nastáva až po dokončení vývoja systému a jeho následnom nasadení. Vzniká paradox, pri ktorom vývojový tím tvrdí, že daný systém funguje správne, ale je v skutočnosti s pohľadu koncového užívateľa nepoužiteľný. Nepoužiteľný, zvyčajne v tomto kontexte znamená, že užívateľ nie je schopný dosiahnuť požadovaný výsledok od systému v dostupnom alebo niekedy ani v reálnom čase. Toto tvrdenie obzvlášť platí pre databázy, kde je výkon kľúčovou vlastnosťou [13].

Táto práca sa zaoberá vývojom nástroja na testovanie výkonnosti SQL databáz a jeho praktickým využitím, ktoré pozostáva z testov a ich vyhodnotenia.

Kapitola 2 sa podrobne zaoberá pojmami spomenutými v názve tejto práce. Je to výkonnostné testovanie, teória SQL databáz a spojenie týchto dvoch častí dohromady. Taktiež sú v nej spomenuté typy databáz použitých v tejto práci.

Kapitola 3 sa zaoberá existujúcimi riešeniami. Sú tu predstavené tri riešenia a to mysqlslap, SysBench a SQLQueryStress.

Kapitola 4 sa zaoberá návrhom riešenia. Je tu predstavené logické rozdelenie nástroja na menšie časti. V každej časti sa nachádzajú požiadavky na danú časť a návrh riešenia. Súčasťou návrhu sú aj dátové a konfiguračné súbory, ktoré vytvárajú rozhranie medzi časťami nástroja.

Kapitola 5 sa zaoberá implementáciou nástroja. Pri každej časti nástroja je popísaná implementácia požadovanej funkcionality, ktorá bola navrhnutá v kapitole zaoberajúcej sa návrhom.

Kapitola 6 sa zaoberá výsledkami. V tejto kapitole sú ukázané reálne namerané hodnoty z vykonaných a spracovaných testov. Hodnotenie výsledkov je zamerané na správnu funkčnosť nástroja.

Kapitola 2

Teória výkonnostného testovania databáz

Táto kapitola popisuje teoretický základ k pochopeniu výkonnostného testovania databáz. V prvej časti je vo všeobecnosti opísané výkonnostné testovanie, jeho dôležitosť, typy, pozorované parametre a samotný proces. V ďalšej časti je opísaná databáza. Sú uvedené jej základné vlastnosti a funkcie, pričom boli vybrané tie, ktoré by mohli byť zaujímavé z pohľadu testovania, ako napríklad index alebo cache. Posledná časť sa venuje konkrétnym databázam použitým v tejto práci a zvolenému typu testovania databáz. Tieto databázy sú SQLite, MySQL a PostgreSQL. S testovaním rôznych databáz vznikajú ďalšie problémy, ktorými sa zaoberá posledná časť.

2.1 Výkonnostné testovanie

Výkonnostné testovanie je definované ako typ softvérového testovania, ktoré má za úlohu zabezpečiť, aby softvérové aplikácie bežali bez problémov a to pod očakávanou záťažou.

Funkcionalita, ktorú softvér poskytuje, väčšinou nie je jediný parameter testovania. Sú tu rovnako dôležité parametre ako je spoľahlivosť, využitie prostriedkov a rozšíriteľnosť. Cieľ výkonnostného testovanie nie je odhaliť chyby v softvéri, ale eliminovať tie časti, ktoré sú zodpovedné za zníženie výkonu a to na akejkoľvek vrstve [3].

Výkonnostné testovanie sa sústredí na nasledujúce vlastnosti softvéru:

- **Rýchlosť** - určuje, či aplikácia reaguje promptne.
- **Rozšíriteľnosť** - určuje najväčšiu možnú záťaž, akú dokáže aplikácia zniesť.
- **Stabilita** - určuje, či je aplikácia stabilná pod rôznou záťažou.

2.1.1 Dôležitosť výkonnostného testovania

Výkonnostné testovanie je súčasťou celého ekosystému testovania, ktoré zahŕňa napríklad dielčie, funkcionálne alebo systémové testovanie. Na rozdiel od ostatných spomenutých typov testovania, je práve výkonnostné testovanie najviac zanedbávané. Problémy začínajú vznikať až po nasadení softvéru do produkcie, kde ho používajú koncoví užívatelia. Tieto problémy sú nasledovné:

- Pomalý beh aplikácie kvôli prístupu viacerých užívateľov naraz.

- Nekonzistentnosť naprieč rôznymi prostrediami, napríklad rozdielne operačné systémy.
- Nepoužiteľnosť systému z hľadiska odozvy.

Ak si predstavíme aplikácie, ktoré sú nasadené v medicíne alebo v kozmickom priemysle, vieme, že aj tá najmenšia chyba môže spôsobiť obrovské škody. Takéto aplikácie by mali byť schopné fungovať po dlhšiu časovú periódu a práve tieto problémy pomáha odhaliť výkonnostné testovanie.

Pre konkrétnejšiu predstavu dôležitosti tohto typu testovania uvediem príklad, ktorý sa týka spoločnosti *Google*. Firma Google v roku 2013 v druhom štvrťroku zarobila 14,1 miliardy dolárov. Teda na jednu minútu to činí zisk 108 000 dolárov. V tom istom čase mali stránky *Googlu* výpadok na 5 minút. Tento krátky výpadok stál spoločnosť približne 545 000 dolárov, nehovoriac o znížení dobrej reputácie [15].

Výkonnostné testovanie by malo byť vždy súčasťou projektu a malo by sa s ním rátať od začiatku. V inakšom prípade sa môže dokonca stať, že výsledný produkt bude nepoužiteľný.

2.1.2 Typy výkonnostného testovania

- **Záťažové** - overuje schopnosť aplikácie fungovať pod očakávanou záťažou. Úlohou tohto testovanie je odhaliť miesta s najnižšou priepustnosťou pred tým ako pôjde aplikácia do produkcie.
- **Stresové** - aplikácia je vystavená extrémnej záťaži, aby sa zistilo ako zvláda náročné spracovanie dát. Cieľom tohto testu je nájsť bod, kedy začne byť aplikácia nepoužiteľná alebo menej použiteľná.
- **Vytrvalostné** - podobné záťažovému testovaniu s tou výnimkou, že aplikácia musí byť vystavená testom po dlhší čas.
- **Objemové** - pri tomto testovaní je do databázy nahrané veľké množstvo dát a testuje sa celková reakčná schopnosť systému. Cieľom je zistiť ako sa systém správa pri rôznych veľkostiach dát uložených napríklad v databáze.
- **Rozšíriteľnosť** - zisťuje efektivitu aplikácie pri jej rozširovaní. Pomáha plánovať zväčšenie kapacity systému pre budúce požiadavky.

2.1.3 Proces výkonnostného testovania

Proces výkonnostného testovania sa môže líšiť, ale jeho cieľ ostáva rovnaký. Môže pomôcť demonštrovať, že preddefinované kritéria pre aplikáciu boli splnené. Alebo môže porovnať výkonnosť dvoch softvérových systémov. Taktiež vie pomôcť s identifikovaním slabších alebo problémových častí v rámci aplikácie.

Generický proces výkonnostného testovania vyzerá nasledovne:

1. **Identifikovanie testovacieho prostredia** - Oboznámenia sa s produkčným systémom a s dostupnými testovacími nástrojmi. Pochopenie detailov hardvéru, softvéru a sieťovej konfigurácie pred začatím testovania.
2. **Identifikovanie akceptačných kritérií** - Toto zahŕňa ciele a podmienky pre priepustnosť, odozvu a alokovanie prostriedkov. Podstatné je, si zvoliť, okrem predchádzajúcich vlastností, aj kritérium úspechu, pretože veľakrát samotný projekt nezahŕňa

požiadavky na splnenie výkonnostného testovania. Niekedy neobsahuje žiadne a je potrebné nájsť podobnú aplikáciu, s ktorou je možné porovnávať výsledky.

3. **Návrh a dizajn testov** - Vytvorenie testovacích sád na základe identifikovania koncových používateľov a k nim príslušných testovacích scenárov. Je dôležité počítať s rôznorodosťou užívateľov, testovacích dát a pevne si stanoviť aké dáta budú zaznamenávané pre ďalšiu analýzu.

4. **Konfigurácia testovacieho prostredia** - Pripravenie testovacieho prostredia pred vykonaním testov. Tiež príprava prostriedkov potrebných pre testovaný systém.

5. **Implementácia testov** - Vytvorenie výkonnostných testov podľa návrhu.

6. **Spustenie testov** - Vykonanie a monitorovanie testov.

7. **Analýza a opakované testovanie** - Analýza testov a ich zdieľanie. Taktiež zmena konfigurácie systému a opakované vykonanie testov. Týmto spôsobom sa inkrementálne odstraňujú problémy spojené s výkonom.

2.1.4 Pozorované parametre

Parametrov, ktoré môžeme pozorovať počas testov, je viacero a tu sú tie najpodstatnejšie z nich:

- **Čas odozvy** - čas od odoslania prvého požiadavku po prijatie prvého znaku z odpovede.
- **Využitie procesoru** - čas, ktorý strávi procesor aktívnym vykonávaním inštrukcií.
- **Využitie pamäte** - veľkosť fyzickej pamäte, ktorá je dostupná pre daný proces.
- **Využitie disku** - čas, ktorý strávi disk vykonávaním operácií typu čítanie a zápis.
- **Priepustnosť** - tempo pri ktorom dostáva počítač alebo sieť požiadavky za jednotku času.
- **Databázové zámky** - zamykanie tabuliek.
- **Pomer zásahov** - pod týmto spojením sa myslí počet SQL príkazov, ktoré boli spracované v dočasnej rýchlej pamäti namiesto drahých I/O operácií na disku.
- **Maximum aktívnych relácií** - maximálny počet relácií, ktoré môžu byť súčasne aktívne.

2.2 SQL databázy

Relačná databáza je množina štruktúrovaných dát, uložených v počítačovom systéme takým spôsobom, že je možné pracovať s týmito dátami pomocou SQL jazyka. SQL znamená v preklade, štrukturovaný dotazovací jazyk, čo je počítačový jazyk určený pre ukladanie, manipulovanie a získavanie dát, ktoré sú uložené v relačnej databáze.

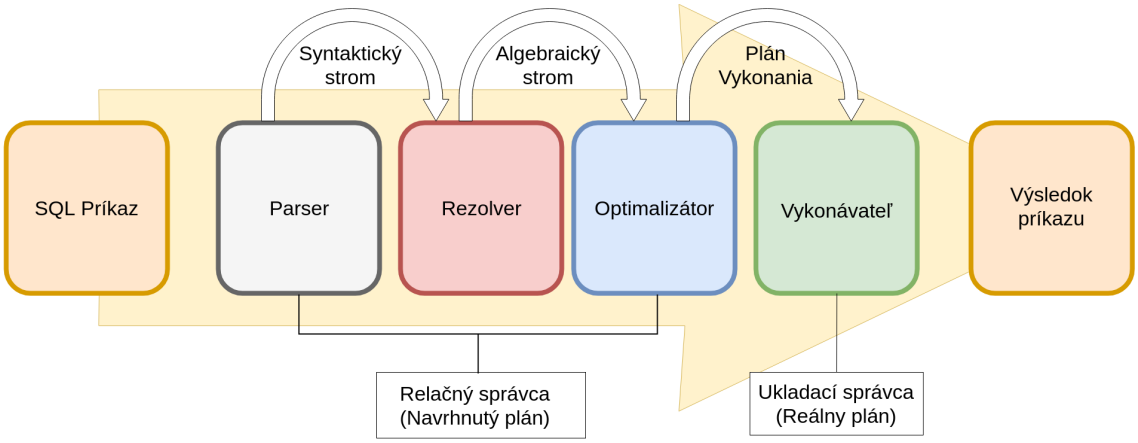
2.2.1 SQL jazyk

SQL je štandardný jazyk pre relačné databázové systémy. Všetky relačné systémy riadenia báze dát, tiež nazývané RDBMS (Relational Database Management Systems)[11], ako sú MySQL, Postgres, Oracle, SQLlite a ďalšie, používajú SQL ako ich štandardný jazyk. Taktiež môžu používať rôzne varianty a dialekty, napríklad MS SQL Server používa T-SQL, teda procedurálny jazyk v rámci SQL.

2.2.2 Proces vykonania príkazu

Keď vykonávate SQL príkaz pre RDBMS, systém rozhoduje o najlepšej možnej ceste ako vykonať požadovaný SQL príkaz a SQL procesor zisťuje spôsob ako najlepšie interpretovať danú úlohu.

Proces spracovania príkazu vyzerá nasledovne. Najprv sa prijme SQL príkaz, ktorý sa parsuje. Na tento vyparsovaný príkaz sa spustí optimalizačný nástroj, ktorý zistí najlepší spôsob ako vykonať príkaz. Po tomto kroku sa vytvorí exekučný plán, podľa ktorého sa nájdu odpovedajúce výsledky. Na týchto výsledkoch sa vykoná pôvodná akcia SQL príkazu, čo môže byť napríklad vybrať, vymazať alebo upraviť. Zápis a načítavanie z fyzickej databázy má na starosť správca súborov, ktorý je súčasťou RDBMS. Aby nedošlo k nekonzistencii dát, RDBMS používa správcu transakcií, čo znamená, že jednotlivé SQL príkazy sú navzájom nezávislé a ich vykonanie je atomické. Tento proces je zobrazený na obrázku 2.1.



Obr. 2.1: Popis celého procesu vykonania SQL príkazu [4]

2.2.3 Základné prvky databázy

Na to, aby sme pochopili ako funguje databáza, musíme si ujasniť základné prvky z ktorých je databáza poskladaná.

Tabuľka

Dáta v RDBMS sú uložené v databázových objektoch, ktoré sa volajú tabuľky. Tabuľka je najčastejšia a najjednoduchšia forma dátového úložiska v relačnej databáze. Takáto tabuľka je v podstate kolekcia podobných dátových vstupov a skladá sa zo stĺpcov a riadkov.

Riadok

Riadok alebo aj záznam je každý individuálny vstup ktorý v tabuľke existuje.

Stĺpec

Stĺpec je vertikálna entita v tabuľke, ktorá obsahuje všetky informácie spojené so špecifickou položkou v tabuľke.

Položka

Každá tabuľka je zložená z menších entít nazvaných tiež položky. Položka je stĺpec v tabuľke, ktorý uchováva špecifickú informáciu o každom zázname v tabuľke.

2.2.4 SQL obmedzenia

- **NULL hodnota** - zaručuje, že stĺpec nemôže obsahovať hodnotu NULL, teda prázdnu položku.
- **Unikátna hodnota** - zaručuje, že všetky hodnoty pre daný stĺpec sú rozdielne, teda jedinečné.
- **Primárny kľúč** - jednoznačne identifikuje každý riadok v tabuľke. Primárne kľúče musia obsahovať unikátne hodnoty a zároveň nesmú obsahovať hodnoty typu NULL. Tabuľka môže mať len jeden primárny kľúč, ktorý sa môže skladať z jednej alebo viacerých položiek.
- **Cudzí kľúč** - je kľúč, ktorý sa používa na spojenie dvoch tabuliek dohromady. Cudzí kľúč je položka v jednej tabuľke, teda ukazuje na primárny kľúč v inej tabuľke.

2.2.5 Dátová integrita

Každý RDBMS musí presadzovať nasledovné integritné obmedzenia:

- **Entitná integrita** - v tabuľke sa nemôžu nachádzať dva rovnaké riadky.
- **Doménová integrita** - do stĺpca môžu byť vložené iba validné vstupy, pričom validita vstupov sa určuje podľa typu, formátu alebo rozsahu hodnôt.
- **Referenčná integrita** - riadky, ktoré sú odkazované inými záznamami, nemôžu byť vymazané.
- **Užívateľsky definovaná integrita** - sú to špeciálne biznis pravidlá, ktoré nespadajú do žiadnej predchádzajúcej kategórie.

2.2.6 Typy SQL príkazov

Základné príkazy jazyka SQL [9] sú nasledovné:

SELECT

Používa sa pri výbere dát z tabuľky. Vracia takzvanú výslednú množinu alebo inak povedané, záznamy, ktoré odpovedajú vybranej tabuľke alebo podmienkovej klauzule ako je napríklad WHERE. Vrátené záznamy môžu obsahovať len niektoré položky. Táto operácia zvyčajne funguje paralelne s obdobnou operáciou, pretože dáta nepozmeňuje, ale iba číta.

Nasledujúce 3 príkazy pozmeňujú nejakým spôsobom vybrané záznamy. Treba si uvedomiť, že vykonávajú operáciu zápisu a väčšinou je paralelné použitie týchto príkazov a príkazu SELECT nemožné, pretože by došlo k porušeniu transakcií a nevalidným výsledkom. Tu je ich popis:

INSERT

Tento príkaz pridáva nové riadky.

DELETE

Tento príkaz maže riadky. Môže použiť klauzulu WHERE.

UPDATE

Tento príkaz aktualizuje existujúce záznamy s vybranými položkami. Môže použiť klauzulu WHERE.

Tieto príkazy podporujú dodatočné klauzule ako sú napr.:

JOIN

Táto klauzula je používaná na kombináciu záznamov z viacerých tabuliek. Na toto spojenie sa využívajú záznamy s podobnými hodnotami. Najčastejšie je spojenie cudzích kľúčov s primárnymi.

WHERE

Je to klauzula, ktorej podmienka určuje záznamy, ktoré budú vybrané.

2.2.7 Index

Jedným z najlepších možností ako zredukovať množstvo diskových operácií je použitie indexu. Index umožňuje nájsť dáta v tabuľke bez toho, aby sa musela skenovať celá tabuľka. Index je nič viac ako dátová štruktúra naplnená ukazateľmi na konkrétne záznamy. Tieto záznamy sú z pohľadu indexu zoradené a práve táto vlastnosť umožňuje rýchle vyhľadávanie záznamov.

Existujú dva základné typy indexov, ktoré sa používajú v SQL. Je to zhlukový a nezhlukový index [1]. Podľa zhlukového indexu sú dáta zoradené v rámci tabuľky. Väčšinou je takýto index priradený primárnemu kľúču. Nezhlukový index tvorí samostatnú dátovú štruktúru, teda nie je súčasťou tabuľky. Používa sa na indexovanie stĺpcov, ktoré neobsahujú primárny kľúč, ale napriek tomu by sme chceli mať rýchly prístup k týmto položkám.

Pri nezhlukovom indexe je potrebné si dobre rozmyslieť na ktoré stĺpce bude aplikovaný. Pri každom vytvorenom indexe vznikajú režijné náklady. Jednak je to pamäť, ktorú samotná

štruktúra indexu zaberá a jednak je to vytváranie alebo skôr reindexovanie položiek. Vždy keď sa pridá alebo odoberie záznam z tabuľky, je nutné prerobiť indexovú štruktúru tak, aby odpovedala novému stavu. Čím viac indexov je vytvorených, tým viac je tento proces náročný na výkon a taktiež na pamäť [12].

2.2.8 Cache

Cache je vyrovnávacia pamäť alebo inak povedané dočasná pamäť, ktorá umožňuje opakovaný prístup k dátam, ktoré sú v nej uložené. Pri spojení pojmov databáza a cache sa rozumejú dve rozdielne veci.

Po prvé, je to operačná pamäť, ktorú má daný RDBMS k dispozícii. RDBMS sa snaží nahrať všetky používané tabuľky a výsledky spracovania SQL príkazov práve do tejto pamäte, aby nemusel často pristupovať na disk. Nedostatok tejto pamäte vedie k samotnému spomaleniu celej databázy.

Po druhé, poznáme cache exekučného plánu. Keď sa vytvorí exekučný plán podľa optimalizátora, je tento plán uložený v špeciálnej pamäti nazvanej plánovacia alebo procedurálna cache. Vždy keď sa vykonáva SQL príkaz, zavolá sa najskôr optimalizátor. Ak sa však rovnaký príkaz nachádza v plánovacej cache pamäti, použije sa rovnaký plán ako pri prvom príkaze a optimalizátor sa vôbec nezavolá. Príkazy v plánovacej pamäti sú parametrizovateľné, čím dovoľujú lepšie znovu použitie exekučného plánu [12].

2.3 Výkonnostné testovanie SQL databáz

Zatiaľ boli spomínané všeobecné pojmy, čo sa týka testovania a databázy. Táto časť sa bude zaoberať teóriou a problémami konkrétne spojenými s touto prácou.

2.3.1 Použitý typ testovania

Sú dve možnosti testovania databázy [14], čo sa týka znalosti databázy. Pri testovaní pomocou techniky WhiteBox poznáme interné komponenty databázy. Toto testovanie sa vykonáva najlepšie v rámci samotnej databázy. V tomto projekte bude použitý druhý typ, ktorý sa nazýva BlackBox testovanie. BlackBox testovanie je definované ako testovacia technika, pri ktorej nepoznáme vnútornú implementáciu databázy a testujeme na základe softvérových požiadavok a špecifikácie. Konkrétne bude použité BlacBox testovanie s pomocou dizajnu, čo znamená, že budeme vedieť ako sa SQL databázy všeobecne správajú, ale nebudeme poznať implementáciu vybranej databázy.

Ďalším rozdelením BlackBox testovania dostaneme funkcionálne a nefunkcionálne testovanie. Funkcionálne testovanie sa zaoberá požiadavkami, ktoré testujú funkčnosť alebo korektnosť systému. Táto práca použije nefunkcionálne testovanie. Nebude sa zaoberať špecifickou funkcionalitou, ale merateľnými požiadavkami a to konkrétne výkonom.

2.3.2 Použité databázy

V tejto sekcii sú uvedené databázy, ktoré sú použité v tejto práci [7].

MySQL

Jedna z najpoužívanejších databáz. Má veľkú funkcionalitu, o ktorej existuje veľa zdrojov a samotný produkt je open-source. Ako komunikačný prostriedok používa MySQL démona,

cez ktorého komunikujú užívatelia s databázou. Medzi výhody patrí bezpečnosť, rýchlosť a rozšíriteľnosť. Aj táto databáza má však svoje limity. V tejto práci bude použitá s RDBMS MariaDB. MariaDB vznikla z databázy MySQL. Je s ňou kompatibilná a tieto dva systémy riadenia báze dát sú zameniteľné.. Znamená to, že definície dát, tabuliek a indexov sú rovnakej štruktúry pri obidvoch databázach. To isté platí aj pre protokoly, ktoré používajú databázové konektory pre komunikáciu s databázou. Rozdiel medzi týmito databázami je napríklad v spracovaní príkazov a to konkrétne v ich rýchlosti.

SQLlite

SQLlite je odľahčená databáza zameraná na vstavané systémy, testovanie a celkovo systémy s menšími hardvérovými požiadavkami. Pri komunikácii s databázou nepoužíva sockety, ale komunikuje priamo s databázovým súborom. Keďže sa celá databáza skladá z jedného súboru, je veľmi dobre prenositeľná. Jej nevýhodu je, že neposkytuje plnú funkcionalitu ako ostatné známe databázy. Taktiež nie je stavaná pre prístup viacerých užívateľov alebo zapisovanie väčšieho objemu dát.

PostgreSQL

Najviac pokročilý objektový RDBMS, ktorý podporuje jazyk SQL. Táto databáza podporuje objektové aj relačné operácie, pričom udržuje spoľahlivé transakčné spracovanie. Vďaka použitej technológii zvláda veľa operácií súčasne. Ak by sme ju však chceli použiť len pre rýchle operácie typu čítanie, MySQL je vhodnejším kandidátom.

2.3.3 Rozdiely v testovaní databáz

Všetky databázy spomínané v predchádzajúcom odstavci majú spoločné a rozdielne vlastnosti. Pre vytvorenie testovacieho nástroja je kľúčové poznať tieto rozdiely, aby sme vedeli danú testovaciu sadu navrhnúť tak, aby bolo možné použiť nástroj s každým typom databázy.

Databázový konektor

Na vytvorenie spojenia s databázou je potrebný databázový konektor. Ten zaisťuje rozhranie medzi užívateľom a databázovým systémom riadenia dát. Pomocou neho vieme vykonávať príkazy, získavať výsledky týchto príkazov alebo vykonávať iné obslužné funkcie podporované databázou ako sú napríklad debugovanie alebo získanie databázových metadát. Každá databáza má svoj vlastný konektor, ktorý má špecifické rozhranie pre danú databázu. Konektor zvyčajne obsahuje štandardné funkcie, ktoré sa očakávajú od databázového konektoru, ako je vykonanie SQL príkazu a neštandardné funkcie, špecifické pre danú databázu. Preto, aby sme mohli porovnať viacero databáz, je dôležité v časti vykonávania príkazu a spracovania výsledku, či už návratovej hodnoty alebo hodnôt vrátených databázou, použiť čo najviac totožné funkcie. Cieľom je zaistiť rovnakú funkcionalitu a teda rovnaké správanie, čo vyústi v rovnaké testovacie prostredie.

SQL jazyk

Každá z použitých databáz podporuje z väčšej časti štandardné funkcie jazyka SQL. Avšak každá databáza používa aj neštandardné SQL príkazy, funkcie alebo úplne odlišné vlastné

funkcie. Rozdiely v jednotlivých databázových jazykoch sú spôsobené nasledovnými faktormi: iné dátové typy, iný zápis špeciálnych znakov ako sú napríklad znakové reťazce, rozdielnou syntaxou pri sémanticky rovnakých funkciách alebo ich úplným vynechaním. Cieľom tejto práce nie je popísať všetky rozdiely v použitých jazykoch a ani ich nijako odstrániť. Je však dôležité si túto skutočnosť uvedomiť a počítať s ňou pri písaní testov. Tento problém je teda ponechaný na samotnom užívateľovi.

Kapitola 3

Existujúce riešenia

Pre návrh programu bolo potrebné sa oboznámiť s existujúcimi riešeniami. Za príklad som si zobral programy mysqlslap, SysBench a SQLQueryStress. Každý z nich je určený na výkonnostné testovanie databáz. Všetky spomínané programy majú spoločné a rozdielne vlastnosti. V tejto kapitole opisujem ich základné použitie a vlastnosti. Taktiež explicitne uvádzam vlastnosti, ktorými sa odlišujú. Ich funkcionalita bude porovnaná s vytvoreným riešením na konci kapitoly 5.

Spoločné zaujímavé parametre

Tu sú uvedené parametre, ktoré sa dajú nastaviť alebo použiť pri každom programe.

- –**user** - meno pre databázu
- –**password** - heslo pre databázu
- –**host** - adresa pre databázu
- –**port** - port pre databázu
- –**query** - použitie vlastného príkazu
- –**iterations** - počet opakovaní testu

3.1 mysqlslap

Program mysqlslap je súčasťou klientských programov od MySQL [2]. Je to diagnostický nástroj, ktorý je navrhnutý na emulovanie klientskej záťaže pre MySQL server a zaznamenávanie časov z každej fáze. Napodobňuje pripojenie viacerých klientov na server. Ako vstupný parameter je možné zadať špecifický reťazec obsahujúci SQL príkaz alebo súbor obsahujúci viacero príkazov. V prípade súboru sú jednotlivé príkazy oddelené oddeľovačom, ktorý je v základe nový riadok. Súbor nemôže obsahovať komentáre.

Implementácia

Mysqlslap pracuje v troch fázach:

1. Vytvorenie schémy, tabuliek alebo uloženého programu a dát pre test. Táto časť je tvorená len jedným klientskym pripojením.

2. Beh samotných testov. Táto časť môže použiť viacero klientov pre pripojenie.

3. Upratanie prostriedkov (odpojenie sa alebo vymazanie tabuliek, ak je to špecifikované). Táto časť používa len jedno spojenie.

Zaujímavé parametre

- –**auto-generate-sql** - vytvorenie vopred nakonfigurovanej testovacej sady

Príklad použitia

Použitie vlastného príkazu a vytvorenie vlastnej tabuľky, pripojenie 50 klientov s čoho každý použije 200 príkazov typu výber (select):

```
mysqlslap --delimiter=";"
--create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
--query="SELECT * FROM a" --concurrency=50 --iterations=200
```

3.2 SysBench

Program SysBench je modulárny, viac platformný a viac vláknový benchmarkový nástroj [10]. Slúži na vyhodnocovanie parametrov, ktoré sú dôležité pre systém, na ktorom beží databáza pod veľkou záťažou. Myšlienka tohto nástroja je dostať rýchlu predstavu o výkone systému bez vytvárania zložitých testov alebo dokonca úplne bez databázy.

Implementácia

SysBench vie vytvoriť a spustiť viacej vlákien paralelne. Veľkosť a typ produkovanej záťaže závisí od vybraného módu. Základný mód je oltp (Online Transaction Processing), teda spracovanie dát v reálnom čase. Ten sa delí na jednoduchý, kedy sa vyberajú jednoduché príkazy. Ďalej na pokročilý transakčný, kedy sa používajú transakcie a teda pri zmazaní dát z tabuľky sú tie isté dáta po skončení transakcie opätovne vložené do tabuľky. Posledný mód sa nazýva netransakčný a je rovnaký ako jednoduchý, pričom je možné si zvoliť príkazy manuálne. Na rozdiel od transakčného, mení dáta v tabuľkách a je potrebné ich znovu vytvoriť. Test je možné obmedziť buď časovo alebo počtom príkazov.

Zaujímavé parametre

- –**max-time** - maximálny čas strávený testovaním
- –**max-requests** - maximálny počet príkazov
- –**oltp-read-only** - žiadny príkaz, ktorý by pozmenil databázu, nebude vykonaný
- –**oltp-test-mode=simpe** - jednoduchý mód

Príklad použitia

Prvý príkaz vytvorí tabuľku v databáze 'sbtest' na MySQL serveri definovanom daným socketom a naplní tabuľku s miliónom záznamov:

```
sysbench --test=oltp --mysql-table-type=myisam --oltp-table-size=1000000
--mysql-socket=/tmp/mysql.sock prepare
```

Druhý príkaz vykoná testovaciu sadu s použitím 16 klientov ako vlákien a obmedzí počet príkazov na maximálne 100 000.

```
sysbench --num-threads=16 --max-requests=100000 --test=oltp
--oltp-table-size=1000000 --mysql-socket=/tmp/mysql.sock
--oltp-read-only run
```

3.3 SQLQueryStress

Program SQLQueryStress je na rozdiel od predchádzajúcich dvoch programov vytvorený ako veľmi zjednodušený nástroj a neobsahuje veľa prepínačov [8]. Jeho úlohou je testovať len jeden SQL príkaz. Používa prehľadné GUI, kde hneď vidieť všetky sledované parametre.

Implementácia

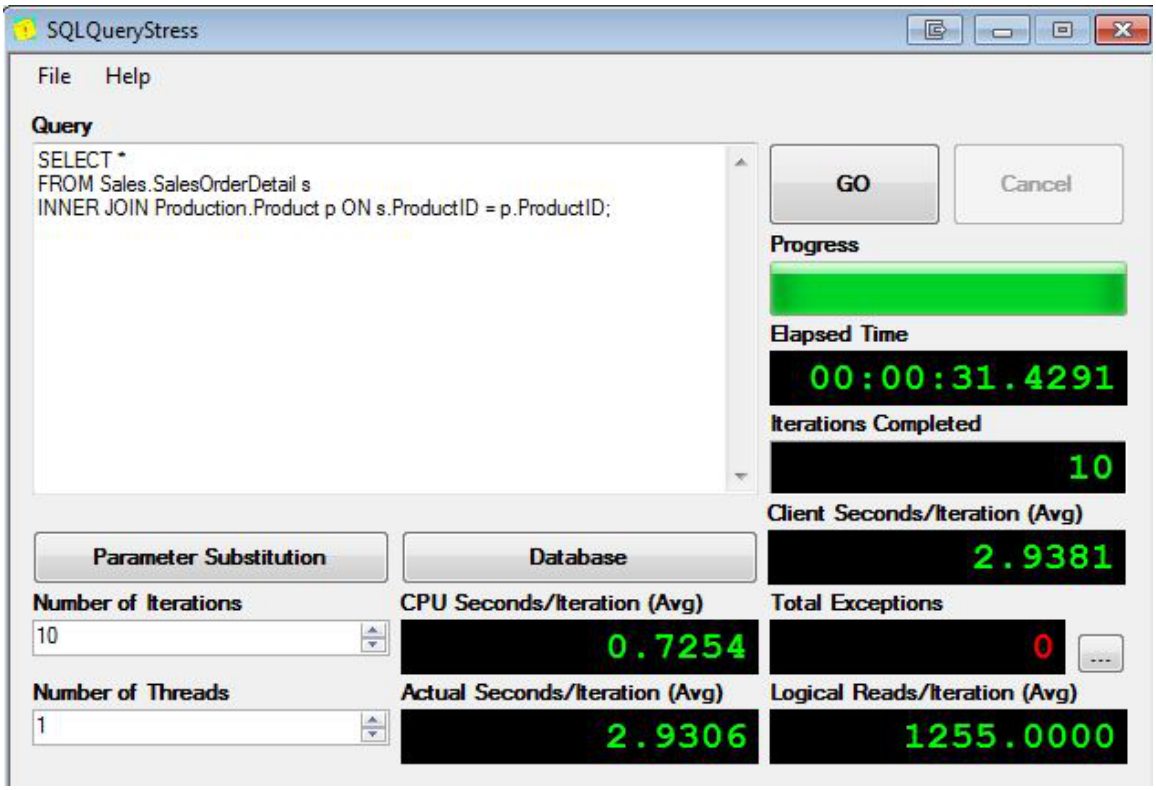
SQLQueryStress je odľahčený nástroj, ale obsahuje základné nastavenia. Nastaviť sa dá počet vlákien a aj počet opakovaní testu. Naraz sa môže testovať len jeden príkaz, ktorý je vpísaný priamo do textového poľa v GUI. Podporuje randomizáciu vstupných parametrov, aby sa predošlo vyťahovaniu výsledkov len z pamäte cache. Obsahuje jednoduché nástroje na meranie spotrebovaných prostriedkov, ktoré sa dajú vypnúť.

Zaujímavé parametre

- **Force Client Retrieval of Data** - zaistí, že nástroj prejde cez všetky dáta vrátené z databázy. Tým pádom sa všetky dáta dostanú k užívateľovi a nestane sa, že by ostali nabuffrované na serveri.
- **Connection Pooling** - povolenie vytvárania a rušenia spojenia pred a po každom teste

Príklad použitia

Príklad použitia je ilustrovaný na obrázku 3.1, pretože tento nástroj používa grafické rozhranie pre komunikáciu s užívateľom.



Obr. 3.1: Grafické rozhranie programu SQLQueryStress [8]

Kapitola 4

Návrh riešenia

V tejto kapitole nájdeme požiadavky, ktoré musí výsledný nástroj spĺňať a k nim vytvorený návrh riešenia. Na začiatku kapitoly sú rozpísané všeobecné požiadavky a základný návrh, teda rozdelenie nástroja na jednotlivé logické celky. V ďalších častiach sa hovorí konkrétne o každej časti. Znovu sú pri každej časti uvedené požiadavky a návrh na implementáciu riešenia. K týmto častiam sa viaže rozhranie v podobe súborov. Každý súbor je definovaný a je ukázané ako by reálne mohla jeho štruktúra vyzerať.

4.1 Požiadavky

Pred návrhom nástroja je dôležité si určiť požiadavky na jeho funkcionalitu. Výsledkom tohto projektu bude nástroj, ktorý minimálne spĺňa nasledovné všeobecné požiadavky, ktoré budú neskôr v tejto kapitole upresnené. Nástroj musí vedieť:

- vytvárať testovaciu sadu,
- opakovane vykonať testovaciu sadu,
- vedieť merať výkonnostné parametre počas behu testovacej sady,
- čo najmenej ovplyvniť vykonávanie príkazu kvôli presným výsledkom,
- zrozumiteľne vyhodnotiť výsledky testov,
- pracovať s viacerými databázami,
- meniť správanie podľa vstupných parametrov,
- spustiť program z príkazového riadku a teda nepoužívať grafické rozhranie.

4.2 Základná myšlienka nástroja

Úlohou tejto práce je vytvorenie nástroja pre výkonnostné testovanie databáz. Preto je potrebné si povedať, aké vlastnosti by mal taký nástroj spĺňať. Nástroj by mal byť jednoduchý na použitie a spĺňať požiadavky, ktoré sa od neho očakávajú. Jednoduchosť nástroja znamená, aby bolo jeho používanie jednoduché pre koncového užívateľa. Prostredie nástroja by malo byť čo najprehľadnejšie a samotné používanie nástroja by malo vyžadovať čo najmenší počet krokov. Prvoradá by však mala byť funkcionalita nástroja.

4.3 Formát vstupov a výstupov

Pre prácu s testami a s výsledkami testov je nutné tieto informácie niekde ukladať. Pre ukladanie dát sú použité súbory typu XML. XML je rozšíriteľný značkovací jazyk, ktorý ukladá dáta v hierarchickej štruktúre. Je prehľadný a má jednoduchý spôsob zápisu dát. Dáta sa nachádzajú medzi značkami, ktoré opisujú ich sémantický význam. Pomocou značiek sa tieto dáta môžu hierarchicky zanorovať. Užívateľ je schopný prečítať tieto informácie a porozumieť ich štruktúre. Vzhľadom na tento fakt môže súbory používať nástroj a zároveň aj užívateľ. Užívateľ si môže buď prečítať výstupné súbory bez použitia externého nástroja na spracovanie alebo vizualizovanie týchto dát. Užívateľ môže taktiež použiť tento typ súboru pre vytvorenie vstupov pre nástroj. Tieto vstupy predstavujú napríklad jednoduché konfiguračné súbory alebo vstupné dáta, ktoré budú pomocou nástroja transformované na výstupné dáta.

Výhodou XML formátu je možnosť použitia XSD. XSD je jazyk pre popis XML. Pomocou tohto jazyka môžeme napríklad popísať názvy, poradie, počet alebo dátové typy prvkov v XML súbore. So syntaxou určíme povolené elementy a atribúty, pričom vieme povedať o aké dátové typy sa jedná a koľko ich môže byť. Taktiež vieme zabezpečiť unikátnosť dát v rámci nejakého rodičovského elementu, čo nám môže pomôcť s jednoznačnými identifikátormi. Pomocou XSD jazyka vieme vytvoriť sadu pravidiel pre validáciu XML dokumentu. Tieto pravidlá uložíme napríklad do súboru a vybraný automatický nástroj ich sám aplikuje na zvolenom XML dokumente.

4.4 Logické rozdelenie nástroja

Nástroj je rozdelený do viacerých väčších logických celkov. Toto rozdelenie umožňuje väčšiu prehľadnosť nad vstupnými a výstupnými dátami. Na rozdiel od existujúcich riešení, ktoré fungujú ako jeden celok, je pri tomto nástroji možnosť nahradenia jednotlivých častí. Pretože nástroj má byť jednoduchý a ovládateľný pomocou príkazového riadku pri jeho spustení, všetko zložitejšie nastavovanie bude uložené v súboroch. Vzniká nám teda štruktúra, kde každá časť nástroja prijíma jeden alebo viacero vstupných dátových súborov a produkuje jeden alebo viacero výstupných dátových súborov. Pre získanie dodatočných informácií sú k dispozícii konfiguračné súbory, ktoré sa editujú užívateľom. Nástroj je rozdelený podľa zadaných požiadavkov na 4 samostatné časti:

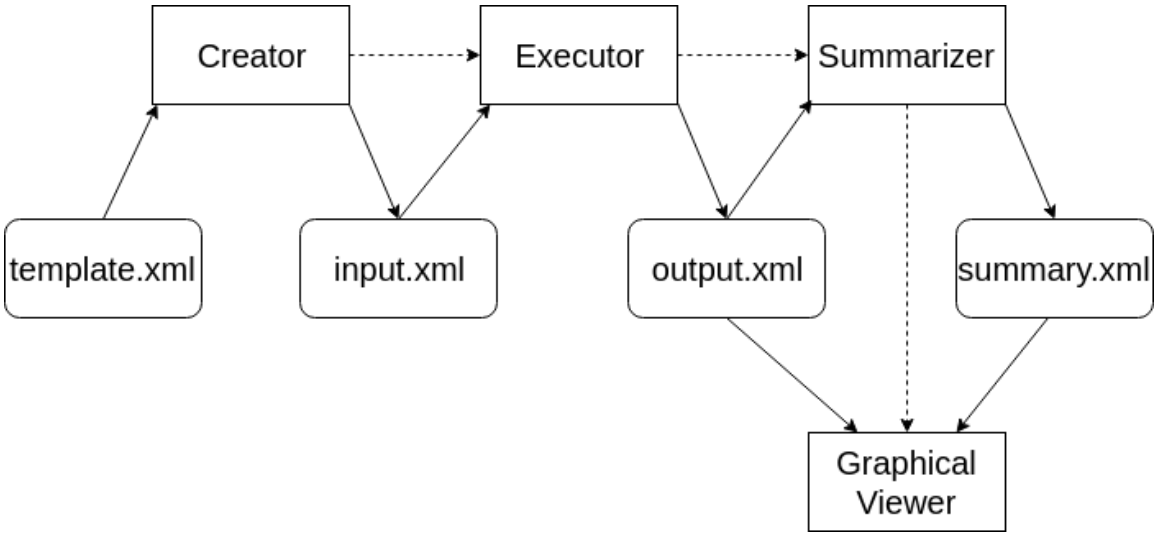
- **Creator** - vytváranie testovacej sady.
- **Executor** - vykonávanie testovacej sady a meranie parametrov.
- **Summarizer** - spracovanie výsledkov.
- **Graphical Viewer** - zobrazenie výsledkov v graficky prehľadnej forme.

Medzi týmito časťami sa nachádzajú dátové súbory, ktoré sú buď vstupné alebo výstupné. Spolu s logickými časťami vytvárajú jeden celok. Nasledujúce súbory ešte nie sú konkrétne súbory, ale je to návrh rozhrania medzi vyššie spomínanými časťami nástroja:

- **template.xml** - vzory pre vytvorenie testovacej sady.
- **input.xml** - vstupná testovacia sada.
- **output.xml** - výsledky testovacej sady spolu s nameranými hodnotami.
- **summary.xml** - dodatočné spracovanie výsledkov.

Rozpracovaný diagram celého nástroja

Na obrázku 4.1 môžeme vidieť grafické rozdelenie nástroja spolu s jeho rozhraním, ktoré tvorí komunikácia so súbormi.



Obr. 4.1: Rozdelenie nástroja na logické celky a príslušné dátové súbory. Rozpracovaný návrh.

4.5 Použitie príkazového riadku

Jednou s podmienok nástroja je, aby mohol bežať v serverovom prostredí, kde sa nenachádza grafické užívateľské rozhranie. Vstupom nástroja budú okrem súborov aj vstupné argumenty príkazového riadku. Táto podmienka neplatí pre časť nástroja *Graphical Viewer*, ktorou úlohou je grafické zobrazenie výsledkov. Je to preto, lebo spomínaná časť nemení výsledky, len ich interpretuje.

4.6 Časť nástroja Creator

Creator tvorí prvú časť nástroja. Jeho hlavnou úlohou je vytvorenie testovacej sady, na ktorú sa vzťahujú požiadavky popísané v nasledujúcom odstavci.

Požiadavky

Testovacia sada sa má skladať z testov zložených z SQL príkazov. Každý príkaz má byť jednoznačne identifikovateľný. Viacero príkazov by sa malo dať kombinovať do jedného testu. Príkazy toho istého typu sa môžu v jednom teste opakovať a ich počet musí byť nastaviteľný. Aby opakovanie príkazov neviedlo k úplne rovnakým príkazom, kedy by databáza načítavala výsledky iba z pamäte cache, musia byť príkazy parametrizovateľné a ich poradie sa musí dať nastaviť na náhodné.

Návrh

Pre jednoduché vytváranie testovacej sady z SQL príkazov je potrebné oddeliť definície príkazov od ich použitia. Tým pádom nám vzniknú dva typy súborov: definície príkazov a šablóny pre vytvorenie samotnej testovacej sady.

Najprv si navrhneme definície príkazov. Definícia príkazu bude obsahovať typ príkazu, respektíve šablónu príkazu pre vytváranie skutočných príkazov pre testovaciu sadu. Pre jednoznačné identifikovanie typu príkazu, teda skupiny príkazov, bude obsahovať skupinový identifikátor (*groupId*), ktorý dostane každý príkaz, ktorý bol vytvorený podľa tejto definície príkazu. Ďalej bude definícia príkazu obsahovať samotný znakový reťazec SQL príkazu (*string*). Pre vytvorenie parametrizovaného znakového reťazca sa musí tento reťazec deliť na ďalšie časti. Prvá časť bude jednoduchý, nepozmenený príkaz a element sa bude volať *raw*.

Ďalšou časťou musí byť parametrizovateľný prvok. Na to aby v SQL príkaze boli zmysluplné hodnoty, najlepším spôsobom je tieto hodnoty dostať priamo z databázy, na ktorú sa tieto príkazy vzťahujú. Veľmi dobrým príkladom je podmienka WHERE v SQL príkaze, kde sa pri vyhľadávaní výsledku používajú konkrétne hodnoty. Element sa bude volať *sql* a bude obsahovať sql príkaz, ktorý vytiahne konkrétne hodnoty. Keďže príkaz je len reťazec znakov, ktorákoľvek časť môže byť parametrizovaná.

Nie vždy sa však hodí použiť hodnoty z databázy. Celý nástroj je postavený tak, aby bol všestranne použiteľný a to platí aj pre formát dátových súborov. Preto je k dispozícii element tretieho typu, ktorý sa bude volať *values*. Tento typ sa môže naplniť hocijakými konštantami, ktoré sa po jednom ukladajú do elementov typu *const*. Vo výsledku sa časť *values* parametrizuje hodnotami z jej elementov typu *const*.

Pre lepší prehľad medzi šablónami príkazov je možné pre každý príkaz pridať element pre popis (*description*). Do popisu sa môže pridať napríklad príkaz v jeho výslednej podobe, kde parametrizované prvky budú abstraktne popísané. Všetky definície dotazov budú umiestnené v jednom súbore, aby bolo zaručené, že ich identifikátory budú jednoznačné a neobjaví sa iná definícia s rovnakým identifikátorom. Taktiež sa zlepší prehľadnosť definícií, ak sa budú nachádzať na jednom mieste.

Teraz si navrhneme šablóny pre vytvorenie testovacej sady. Šablóny budú obsahovať referencie na definície SQL príkazov. Tento element sa bude volať *queryRef*. Musí obsahovať identifikátor definície na ktorú sa vzťahuje. Tento element sa bude volať *groupId*.

Jedným z požiadavkov bolo, aby sa mohol dať nastaviť počet príkazov v testovacej sade. Vzniká tým ďalší element *count*, ktorý drží hodnotu počtu príkazov a je súčasťou elementu *queryRef*. Element typu queryRef sa môže opakovať viackrát, a to pomocou elementu *count*, čím vzniká komplexnejší test. Počet opakovaní celého testu sa dá taktiež nastaviť.

Aby sa príkazy neopakovali v tom istom poradí, je pridaný element *shuffle*. Ten zaručí premiešanie výsledných príkazov v rámci daného testu. Ďalším elementom je *databaseReset*. Pri jeho použití sa do výsledného testu zapíše rovnaký element, ktorý označuje, že sa má daná databáza vrátiť do pôvodného stavu ako bola pred testom. Nakoniec pribudne element *description* pre popis testu.

Zadefinované typy súborov

- **template.xml** - šablóny pre vytvorenie testovacej sady:

```
<queryTemplate>
    <description>shuffle</description>
```

```
    <id>1</id>
    <shuffle/>
    <databaseReset/>
    <queryRef>
        <groupId>3</groupId>
        <count>10</count>
    </queryRef>
    <queryRef>
        <groupId>4</groupId>
        <count>1</count>
    </queryRef>
</queryTemplate>
```

- **queryDefinitions.xml** - zadefinované príkazy:

```
<queryDefinitions>
    <query id="6">
        <description>UPDATE big_table SET enum =
            (sql) WHERE string = (3x const)
        </description>
        <string>
            <raw>UPDATE employees SET gender = </raw>
            <sql>SELECT gender FROM dept_manager LIMIT 10</sql>
            <raw>WHERE first_name = '</raw>
            <values>
                <const>Parto</const>
                <const>Zvonko</const>
                <const>Zorica</const>
            </values>
            <raw>'</raw>
        </string>
    </query>
</queryDefinitions>
```

Potrebné typy súborov

- **template.xml** - vstupný súbor
- **queryDefinitions.xml** - vstupný súbor
- **databases.xml** - vstupný súbor
- **input.xml** - výstupný súbor

4.7 Časť nástroja Executor

Executor tvorí druhú časť nástroja. Jeho hlavnou úlohou je vykonanie testovacej sady a monitorovanie kľúčových výkonnostných parametrov. Je dôležité, aby bol navrhnutý ako prvý, pretože je najhlavnejšou časťou nástroja.

Požiadavky

Nástroj musí vykonať testovaciu sadu pozostávajúcu z SQL príkazov. Táto testovacia sada sa bude ďalej označovať ako vstup. Vo výstupe Executora musia byť SQL príkazy, ktoré sa dajú jednoznačne priradiť príkazom zo vstupu, ktoré sú ďalej označované ako výstup. Pri vykonávaní príkazov sa musia ukladať dôležité výkonnostné parametre. Ak je pozorovaným parametrom čas, musí byť jednoznačne priradený konkrétnemu príkazu. Meranie výkonnostných parametrov musí byť čo najmenej ovplyvnené externými procesmi.

Návrh

Executor bude premieňať vstupné príkazy na výstupné príkazy. Vstupné príkazy budú obsahovať jednoznačnú identifikáciu a samotný SQL príkaz. Identifikácia bude pozostávať z identifikátora typu príkazu, teda skupinového identifikátora a identifikátora konkrétneho príkazu v rámci daného typu príkazu. Viac o skupinovom identifikátore v sekcii 4.6. Výstupné príkazy budú obsahovať identifikáciu vstupného príkazu a namerané hodnoty.

Každý test sa bude nachádzať v osobitnom súbore. Meno súboru bude zároveň meno testu a v rámci zložky, ktorá môže predstavovať testovaciu sadu, bude jedinečné. Jeden test sa bude vykonávať pre viacero databáz. Výstupný súbor bude teda zložený z mena testu a identifikátora databázy. V praxi vznikne takýmto činom pre jeden test a tri testované databázy trojo výstupných súborov, ktoré budú zdieľať jedno meno testu, ale budú obsahovať aj identifikátor databázy.

V dobe vykonávania testov, by malo na cieľovom systéme bežať čo nejmenej procesov, ktoré by ovlivnili výsledky meraní. Meranie parametrov musí byť preto pred vykonaním príkazu, respektíve po ňom. Sledovaným parametrom bude čas vykonania príkazu. Do výstupu sa zaznamená počiatočný čas a koncový čas vykonania príkazu. Je to najjednoduchší spôsob ako zaznamenať čas a obsahuje najviac informácií. Upresnenie merania času a jeho formát sa nachádza v kapitole 5.

Ak sa v testovacom súbore nachádza element *databaseReset*, tak sa databáza musí po teste vrátiť do pôvodného stavu.

Zadefinované typy súborov

- **input.xml** - vstupná testovacia sada zložená z príkazov:

```
<inputQuerys>
    <databaseReset/>
    <query groupId="6" id="1" str="DELETE * FROM salaries"/>
</inputQuerys>
```

- **output.xml** - výsledky vykonania vstupnej testovacej sady:

```
<outputQuerys>
    <databaseId>1</databaseId>
    <query>
        <groupId>6</groupId>
        <id>1</id>
        <startTime>1555333304100192908</startTime>
        <endTime>1555333304381298854</endTime>
    </query>
</outputQuerys>
```

Potrebné typy súborov

- **input.xml** - vstupný súbor
- **databases.xml** - vstupný súbor
- **output.xml** - výstupný súbor

4.8 Časť nástroja Summarizer

Summarizer tvorí tretiu časť nástroja. Jeho hlavnou úlohou je spracovanie výsledkov z časti Executor. Pod spracovaním sa rozumie použitie štatistických funkcií, ktoré nám prezradia viac informácií o tom, čo sa dialo počas vykonávania testovacej sady.

Požiadavky

Od tejto časti nástroja sa očakáva vyhodnotenie sledovaných parametrov z časti Executor. Pre toto vyhodnotenie sa majú použiť najpoužívanejšie štatistické funkcie.

Návrh

Nad výstupnými príkazmi sa budú vykonávať štatistické funkcie. Tieto príkazy tvoria množinu všetkých príkazov. Treba si však uvedomiť, že v tejto množine sa nachádzajú príkazy vytvorené podľa šablón príkazov, z ktorej je každý príkaz priradený k nejakému typu príkazu. Množinu, nad ktorou sa budú vykonávať štatistické funkcie, treba rozdeliť na množinu všetkých príkazov a množiny jednotlivých typov príkazov podľa identifikátora *groupId*.

Na radu prichádza voľba štatistických funkcií. Najzákladnejšími štatistickými funkciami s veľkou výpovednou hodnotou sú určite funkcie minima, maxima, sumy a priemeru. Ďalšími funkciami môžu byť napríklad odchýlka a priemer odchýlky. Tieto funkcie nám povedia v akom rozsahu sa pohybovali výsledné hodnoty.

Zadefinované typy súborov

- **summary.xml** - štatistické zhrnutie výsledkov zo súboru *output.xml*:

```
<querySummary>
    <allQuerys>
        <Min>15073</Min>
        <Max>265430</Max>
        <Average>29433</Average>
        <Sum>2943310</Sum>
        <Dispersion>1042684604</Dispersion>
        <StandardDeviation>32290</StandardDeviation>
    </allQuerys>
    <query id="1">
        <Min>15073</Min>
        <Max>265430</Max>
        <Average>29433</Average>
        <Sum>2943310</Sum>
        <Dispersion>1042684604</Dispersion>
```

```
            <StandardDeviation>32290</StandardDeviation>
        </query>
    </querySummary>
```

Potrebné typy súborov

- **output.xml** - vstupný súbor
- **summary.xml** - výstupný súbor

4.9 Časť nástroja Graphical Viewer

Graphical Viewer tvorí štvrtú časť nástroja. Táto časť nástroja má prehľadne graficky zobraziť výsledky.

Požiadavky

Od tejto časti nástroja sa očakáva prehľadné grafické spracovanie výsledkov. Pod grafickým spracovaním sa rozumie použitie grafov, ktoré budú najlepšie zobrazovať daný typ dát. Testy sa musia dať porovnávať. Táto časť nemá meniť žiadne dáta, iba ich zobrazovať.

Návrh

Výsledky testov sa nachádzajú vo viacerých súboroch a každý môže obsahovať rôzne typy príkazov. Súbor typu *summary.xml* obsahuje výsledky pre rôzne funkcie. Aby bolo možné prepínať medzi toľkými parametrami a na základe toho vykresľovať grafy, je potrebné vytvoriť grafické rozhranie, ktoré bude obsahovať jednoduché prepínače. Ako vstupné sa budú prijímať buď surové dáta (*output.xml*) alebo súhrnné dáta (*summary.xml*). Testy sa budú porovnávať podľa mena testu a databázy. Na jednom grafe bude teda vidno jeden test pre viacero databáz, inak povedané, porovnávať sa bude rovnaká testovacia sada pre viacero databáz, pretože účelom nástroja má byť porovnávanie databáz. Na to, aby sme mohli porovnať testy medzi viacerými testovacími sadami, ktoré obsahujú testy s rovnakými názvami, potrebujeme ich oddeliť nejakým identifikátorom od seba. Ak by sme chceli porovnať testy s rôznymi menami, môžeme ich zobraziť, ale nebudú sa dať priamo porovnať. Najlepší graf na porovnanie hodnôt zo súboru *output.xml* je obyčajný čiarový graf, ktorý vie dobre zobraziť priebeh. Pre porovnanie hodnôt zo súboru *summary.xml* použijeme stĺpcový graf, kde jasne vidieť rozdiely medzi malým počtom hodnôt.

Potrebné typy súborov

- **output.xml** - vstupný súbor
- **summary.xml** - vstupný súbor

4.10 Rozhranie databáz

Táto sekcia popisuje spojenie celého nástroja s databázovým rozhraním.

Požiadavky

Testovacia sada musí byť vykonateľná nad viacerými databázami. Tieto databázy sú: PostgreSQL, MySQL a SQLite. Testovacia sada musí byť vykonateľná viackrát a to na rovnakej databáze.

Návrh

Použitie rôznych databáz je zabezpečené príslušnými databázovými konektormi. Pre pripojenie k databáze s použitím konektoru je potrebné poznať prihlasovacie údaje na databázu. Tieto údaje sa budú pravidelne používať a je rozumné si ich niekde uložiť. Vzniká teda nový konfiguračný súbor *databases.xml*, kde má každá databáza jednoznačný identifikátor. Ďalej obsahuje prihlasovacie údaje pre daný typ databázy a samotnú databázu, na ktorú sa pripojenie vzťahuje. Niektoré SQL príkazy ovplyvňujú stav databázy a preto je nutné celý jej stav pred vykonaním testu uložiť a potom zase nahrať späť. Toto správanie nemusí byť vždy žiaduce, napríklad pri SQL príkazoch, ktoré nemenia stav databázy a preto musí byť voliteľné.

Zadefinované typy súborov

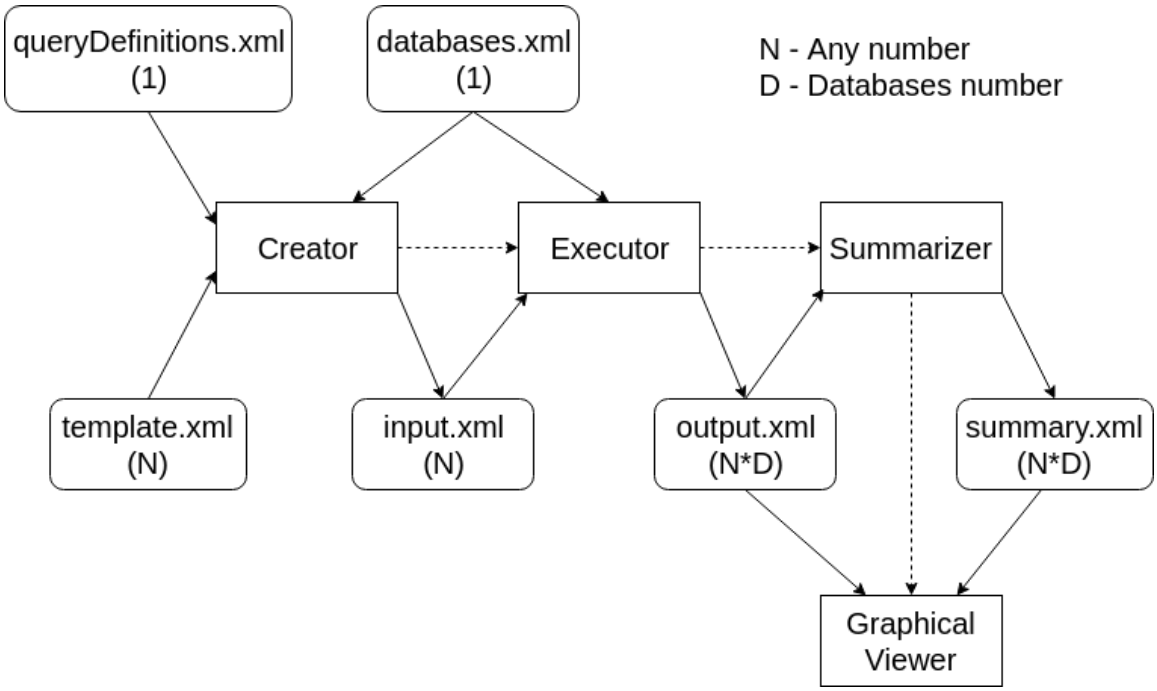
- **databases.xml** - konfiguračný súbor naplnený údajmi o databázach:

```
<databases>
    <database>
        <id>1</id>
        <type>mysql</type>
        <schema>employees</schema>
        <address>127.0.0.1</address>
        <user>jozko</user>
        <password>heslo</password>
    </database>
</databases>
```

4.11 Konečný návrh nástroja

Po vykonaných zmenách na formátoch súborov sa trochu zmenil aj celkový diagram nástroja. Výsledný diagram je popísaný na obrázku 4.2. Diagram tentokrát obsahuje všetky potrebné dátové a konfiguračné súbory. Pridaný je počet súborov, ktoré sú buď potrebné pre vstup alebo vzniknú ako výstup. Konštanta N predstavuje ľubovoľné číslo. Konštanta D predstavuje počet použitých databáz. Napríklad pri výstupe časti nástroja *Executor* vznikne D-krát väčší počet súborov, pretože jeden test je vykonaný na viacerých databázach. Konkrétne súbory sú popísané v jednotlivých návrhoch častí nástroja.

Výsledný diagram celého nástroja



Obr. 4.2: Rozdelenie nástroja na logické celky a príslušné súbory spolu s pridanou kardinalitou pre každý typ súboru.

Kapitola 5

Implementácia nástroja

Táto kapitola popisuje implementáciu jednotlivých častí nástroja. V prvej časti sú spomenuté použité technológie. V ďalších častiach je vysvetlený princíp fungovania jednotlivých častí nástroja. Ak je to potrebné, je pri implementácii pridané odôvodnenie použitého riešenia.

5.1 Použité technológie

V tejto časti sú popísané použité jazyky a knižnice. Časti nástroja *Creator*, *Executor* a *Summarizer* boli naprogramované v jazyku C++ a používajú knižnicu Xerces na spracovanie XML súborov. Časť Graphical Viewer je naprogramovaná v jazyku Python a pre grafické užívateľské rozhranie používa *GTK*. Grafy sú v tejto časti vykresľované pomocou knižnice *matplotlib*. Nižšie sú rozpísané všetky spomenuté technológie a ich dôvody, kvôli ktorým boli vybrané.

Jazyk C++

Jazyk C++ bol špecifickou požiadavkou pre použitie v prvých troch častiach. Dôležitým faktorom pri výbere jazyka bola rýchlosť, kde je jazyk C++ veľmi dobrým kandidátom. Oproti iným jazykom nie je interpretovaný a so správou pamäti sa dá pracovať veľmi efektívne. Nevýhodou tohto jazyka je pomalší vývoj. Použitý štandard je c++14.

Xerces

Pre spracovanie XML súborov som sa rozhodol použiť existujúcu knižnicu. Je ňou knižnica *Xerces*, ktorá má pestrú funkcionalitu a jej rozhranie je v jazyku C++. Knižnica vie parsovať XML súbory a taktiež ich zapisovať do súborov. Podporuje jazyk XSD, ktorý je spomínaný v sekcii 4.3.

Jazyk Python

Jazyk Python bol vybraný pre poslednú časť nástroja. Jeho výhodou je rýchle prototypovanie, automatická správa pamäti a veľké množstvo kvalitných knižníc. V poslednej časti nástroja nie je dôležitý výkon, ale grafické zobrazenie výsledkov a to je dôvod pre použitia jazyka Python namiesto jazyka C++. Použitý verzia Pythonu je 3.7.

GTK

GTK je sada nástrojov pre vytváranie grafického užívateľského rozhrania. Programy vytvorené týmto nástrojom sú spustiteľné na viacerých platformách. Tento nástroj má otvorenú licenciu. Nástroj je písaný pre jazyk C, ale dá sa použiť v spolupráci s jazykom Python, pretože obsahuje nadstavbu rozhrania pre tento jazyk. Jeho grafické widgety sú jednoduché a pre tento nástroj úplne postačujúce. Použitá verzia je *GTK+*.

matplotlib

Pre zobrazovanie grafov je použitá knižnica *matplotlib*. Táto knižnica je napísaná pre jazyk Python a je postavená nad knižnicou *DS3*, ktorá obsahuje veľké množstvo grafov a veľkú slobodu v nastavovaní parametrov. Zaujímavými možnosťami je približovanie na grafe a ukladanie zobrazených grafov. Ďalšou výhodou je kompatibilita s *GTK+*.

5.2 Návrh implementácie

Pred zahájením implementácie si musíme projekt rozdeliť na menšie celky, ktoré budeme neskôr implementovať. Nástroj je už logicky rozdelený na 4 základné časti. Medzi týmito časťami sa nachádzajú súbory, ktoré vytvárajú rozhranie a spájajú časti dohromady. Popis týchto súborov sa nachádza v kapitole 4. Pred implementáciou jednotlivých častí nástroja je dôležité si identifikovať časti kódu, ktoré by mohli byť podobné alebo spoločné pre časti nástroja. Najprv sa pozrieme na prvé tri časti nástroja, pretože majú podobný princíp rozhraní a budú písané v tom istom jazyku a to v C++. Štvrtá časť nástroja sa pre svoje odlišnosti bude navrhovať zvlášť.

5.3 Návrh prvých troch častí nástroja

Pod prvými tromi časťami nástroja rozumieme časť *Creator*, *Executor* a *Summarizer*. Všetky časti pracujú so súbormi typu XML. Každá časť načítava, respektíve parsuje nejaký súbor a každá časť zapisuje, respektíve serializuje jej výstup do súboru. Pre vytvorenie testovacej sady, ktorá sa skladá z viacerých testov, teda súborov, je taktiež nutné pracovať zo zložkami, ktoré obsahujú tieto súbory. Okrem súborov, každá časť tiež používa vstupné argumenty príkazovej riadky na nastavovanie svojich parametrov. Podľa predchádzajúcich tvrdení musíme naprogramovať nasledujúce časti: spracovanie vstupných argumentov, spracovanie súborov a repozitáre pre XML súbory. Osobitnú časť bude tvoriť spojenie nástroja s databázami.

5.4 Spracovanie súborov

Testy predstavujú súbory a tie sa musia načítavať a zapisovať. Pre rýchlejšie spracovanie testov je užitočné testy spracovávať po celých testovacích sadách. Jedna testovacia sada predstavuje jednu zložku. Testy sa načítavajú buď zo zložky alebo sa prijme len jeden test, teda jeden súbor. V prípade zložky sa celá zložka prebehne a hľadajú sa XML súbory. Pri zápise to funguje opačným spôsobom.

Ďalšou vecou, ktorú treba vyriešiť, je práca s menami súborov. Mená súborov sa menia v jednom prípade, ktorý nastáva pri výstupe z *Executora*. Vznikajú nové mená súborov,

ktoré obsahujú identifikátor databázy. Pri vzniku mena sa zoberie staré meno, pridá sa k nemu meno databázy pre lepší prehľad a identifikačné číslo.

5.5 Spracovanie vstupných argumentov

Každá časť nástroja je ovládaná pomocou iných vstupných argumentov. To znamená, že každá časť bude mať vlastnú dátovú štruktúru, ktorá bude naplňovaná vstupnými argumentmi. Avšak parsovanie týchto argumentov zaistí jedna knižnica. Táto knižnica sa volá *getopt*. Argumenty treba samozrejme previesť na patričný formát a pri neúspešnom prevedení vyhodiť chybu.

5.6 Spracovanie XML

Každá časť nástroja načítava a zapisuje do XML súborov pomocou nástroja *Xerces*.

Pre parsovanie súborov sa používa SAX parser, ktorý je pomerne rýchly. Vždy keď nájde element alebo atribút, zavolá callback funkciu. Práve v tejto callback funkcii sa porovná nájdený element alebo atribút s očakávanými a vykoná sa príslušná akcia. Takáto akcia môže predstavovať uloženie hodnoty elementu, respektíve atribútu alebo rekurzívne zavolanie callback funkcie s názvom ďalšej funkcie. Táto funkcia sa určí podľa toho, ktorá časť sa parsuje. S názvom funkcie sa tiež predáva dátová štruktúra, ktorá sa postupne napĺňa. Pre sekvenčné dáta sa používa kontajner typu vektor, kvôli jeho rýchlosti a pre štruktúru sa používa *struct*.

Pre serializovanie sa používa DOM serializátor, pretože SAX rozhranie neposkytuje možnosť zápisu. Najprv sa daná dátová štruktúra prevedie na DOM objekt a potom sa serializuje a zapíše do súboru.

Rozhranie pre všetky funkcie týkajúce sa zapisovania a načítavania sú spoločné pre všetky časti nástroja, ale ich použitie je špecifické pre každú časť.

5.7 Spojenie s databázami

Pri práci s databázami si treba uvedomiť, že pracujeme s troma rôznymi relačnými databázami. Pri vykonávaní príkazov by sme nemali rozlišovať, ktorú databázu voláme. Vytvoríme si preto spoločné rozhranie pre všetky databázy. Toto rozhranie je implementované ako trieda s čisto virtuálnymi funkciami. Každá databázová trieda, ktorá dedí s tejto triedy, respektíve rozhrania, musí povinne implementovať všetky jej funkcie. Od databáz očakávame nasledujúcu funkcionalitu: vykonávanie príkazov a ukladanie stavu databázy spolu s jeho opätovným obnovením. Pre vykonávanie príkazu použijeme databázové konektory. Tie sa potrebujú pripojiť na konkrétnu databázu a na to potrebujú prihlasovacie údaje. Tieto údaje sa nachádzajú v konfiguračnom súbore databáz. Implementácia týchto častí sa nachádza nižšie v tejto sekcii.

Konektory

Každá databáza má vlastný konektor. Konektor realizuje pripojenie a odpojenie k databáze, na čo mu stačí odovzdať správne prihlasovacie údaje. Pri vykonávaní príkazu potrebujeme použiť dva typy funkcií, pričom každý z nich vykonáva SQL príkaz. Prvým typom potrebujeme vrátiť výsledok z databázy. Tento výsledok sa použije pri napĺňaní parametrizovaných

hodnôt z databázy pomocou časti nástroja Creator. Druhým typom funkcie bude funkcia, ktorá vykoná SQL príkaz bez návratovej hodnoty. Táto funkcia síce nevracia žiadne dáta, ale je dôležité, aby ich žiadala od každej databázy. Pri testovaní sa snažíme, aby všetky databázy vykonávali tú istú funkcionalitu a len ich vnútorná implementácia by vytvárala rozdiely vo výsledkoch testov. Mohla by nastať situácia, kedy by databáza optimalizovala daný SQL príkaz. Ak by sme napríklad nežiadali o výsledné dáta, tak by ich databáza mohla úplne preskočiť. Meranie parametrov začína od doby odoslania SQL príkazu až po obdržanie posledných vyžiadaných dát.

Prihlasovacie parametre

Pre pohodlnejšiu prácu s databázami sme si prihlasovacie údaje uložili do súboru. Každá databáza potrebuje iný typ údajov a dokonca rôzny počet parametrov pre prihlásenie. Vždy keď sa vyberajú databázy, ktoré sa použijú počas vykonávania testovacej sady, načítajú sa všetky databázy z konfiguračného súboru. Tento prístup nemusí byť vždy vyhovujúci. Preto je pri každej databáze voliteľný parameter, ktorý určuje, či bude daná databáza testovaná. Parameter predstavuje XML element *active*. Pri parametrizovaní dát v časti nástroja *Creator* pomocou databázy chceme použiť len jednu databázu a preto je pridaný ďalší parameter, ktorý vyberá len jednu databázu podľa identifikačného čísla. Jeho meno je *activeDatabaseNumber*. Konfiguračný súbor **databases.xml** vyzerá po úpravách nasledovne:

```
<databases xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="databases.xsd">
<activeDatabaseNumber>2</activeDatabaseNumber>
    <database>
        <id>1</id>
        <type>mysql</type>
        <schema>employees</schema>
        <address>127.0.0.1</address>
        <user>jozko</user>
        <password>heslo</password>
    </database>
    <database>
        <id>4</id>
        <type>postgresql</type>
        <schema>mydb</schema>
        <name>mydb</name>
        <address>127.0.0.1</address>
        <user>jozko</user>
        <password>heslo</password>
        <active/>
    </database>
        <database>
        <id>5</id>
        <type>sqlite</type>
        <name>/home/martin/identifier.sqlite</name>
        <!--<active/>-->
    </database>
</databases>
```

Obnovovanie stavu databázy

Implementovať zálohovanie pre databázy nie je triviálna úloha. Najlepším riešením je použiť existujúce nástroje. Každá z použitých databáz má takýto nástroj, ktorý vie databázu zálohovať a nahrať zo zálohy. MySQl má *mysqldump* pre vytvorenie zálohy a *mysql* pre nahranie zálohy. PostgreSQL má zase *pg dump* pre obidve operácie. Pri MySQL sa ukladajú SQL príkazy, čo má za následok pomalé zálohovanie a pomalé načítavanie. PostgreSQL vie ukladať databázu binárne, takže predchádzajú operácie sú rýchlejšie. Nie je však prenositeľná medzi rôznymi systémami. Najjednoduchšie a najrýchlejšie je to pri SQLite, kedy stačí skopírovať celý súbor. Všetky nástroje sa volajú cez príkazovú riadku. V kóde sa volá funkcia *system* s menom zálohovacieho nástroja a jeho parametrami.

Obnovenie databázy sa dá nastaviť v každom teste. Obnovenie databázy môže byť zdĺhavá procedúra a preto sa dá nastaviť obnovenie databázy až po vykonaní celej testovacej sady. Za použitia nahrávania databázy si vieme nahrať novú databázu bez toho aby sme museli niečo testovať.

5.8 Hlavné časti

Doteraz boli opisované časti nástroja, ktoré boli spoločné pre všetky prvé tri časti. V nasledujúcich odstavcoch je ukázané ako fungujú špecifické časti pre každú časť nástroja.

Creator

Creator si najprv načíta šablóny testov spolu s definíciami príkazov, ktoré neskôr spojí dokopy na základe skupinového identifikačného čísla. Pre každú šablónu testu sa vytvára odpovedajúci test. Pre zadaný počet v šablóne sa vytvorí odpovedajúci počet príkazov. Jednotlivé typy príkazov sa vytvárajú v poradí v akom sú napísané v šablóne. Ich poradie sa môže premiešať podľa funkcie náhodného generátoru. Premiešať príkazy stačí jednoduchou funkciou zo štandardnej knižnice. Pri vytváraní príkazov sa niektoré hodnoty môžu parametrizovať. Ak sa parametrizujú z hodnôt z databázy, tak sú hodnoty uložené v bufferi, ešte pred tým ako sa začnú príkazy vytvárať. Konštanty sú taktiež uložené v bufferi. Z tohto bufferu sa hodnoty vyťahujú sekvenčne, pričom sa jedná o cyklický buffer, a teda ak by malo nastať jeho pretečenie, tak sa hodnoty začnú vyberať znovu od prvej hodnoty. Takto sa naplnia všetky hodnoty až kým sa nenaplní odpovedajúci počet príkazov pre vytvorenie.

Executor

Executor si najprv vytiahne všetky príkazy do kontajneru typu vektor, aby ich neskoršie použitie bolo čo najrýchlejšie. Hlavnou úlohou vykonania testu je čo najmenej ovplyvniť vykonanie SQL príkazu a preto sa robia len najnutnejšie úkony medzi jednotlivými testami. Vykonanie testu pozostáva z nasledujúcich krokov.

- Zaznamenanie času (počiatočný čas).
- Vykonanie dotazu.
- Zaznamenanie času (koncový čas).
- Uloženie zaznamenaných parametrov.

Aby bolo vykonávanie testu rýchle, zaznamenané parametre sa uložia najprv do kontajneru typu vektor a až po ukončení testu sa uložia do súboru. Medzi jednotlivými testami sa môže podľa zvolených parametrov obnovovať stav databázy do pôvodného. Pre každú databázu vykoná *Executor* rovnakú procedúru.

Meranie času

Jednou z činností *Executora* je zmeranie doby, po ktorú sa vykonáva SQL príkaz. Táto doba sa vypočíta ako rozdiel času dokončenia príkazu a času zahájenia príkazu. Pod časom sa rozumie aktuálny čas, teda čas v danom okamihu. Meranie času musí byť čo najpresnejšie a najrýchlejšie. Čas je meraný v nano sekundách a je to najvyššia presnosť. Niektoré systémy však takúto presnosť nemusia ponúknuť a v tom prípade sa len zmení presnosť merania, ale samotná technika merania času ostane rovnaká. Na meranie aktuálneho času boli vyhodnotené dve možnosti.

Prvá možnosť zahŕňa použitie assemblerovej inštrukcie `RDTSC`. Táto inštrukcia vyčíta hodnotu registra, ktorý obsahuje aktuálny čas. Keďže je to na úrovni assembleru a je to len jedna inštrukcia, zaznamenanie času je veľmi rýchle.

Druhá možnosť obsahuje volanie funkcie `chrono::system_clock::now()` zo štandardnej knižnice. Toto volanie je pomalšie ako predchádzajúca možnosť, ale má výhody, ktoré ho robia preferovaným riešením. Je to volanie zo štandardnej knižnice a preto môžeme túto funkciu použiť na rôznych zariadeniach a rôznych systémoch. Ďalšou dôležitou vlastnosťou je presnosť merania času. Podľa definície inštrukcie `RDTSC` [6], táto inštrukcia nezaručuje poradie vykonania, čo vedie k odlišným odchýlkam v meraní. Naproti tomu volanie funkcie zo štandardnej knižnice zaručuje poradie vykonania.

Treba si uvedomiť, že hlavnou úlohou testov je porovnávanie. Pre porovnávanie je lepšie, ak sú výsledky konzistentné a teda majú rovnaké odchýlky. Vzniknuté oneskorenie je aj tak zanedbateľné oproti meraným hodnotám.

Summarizer

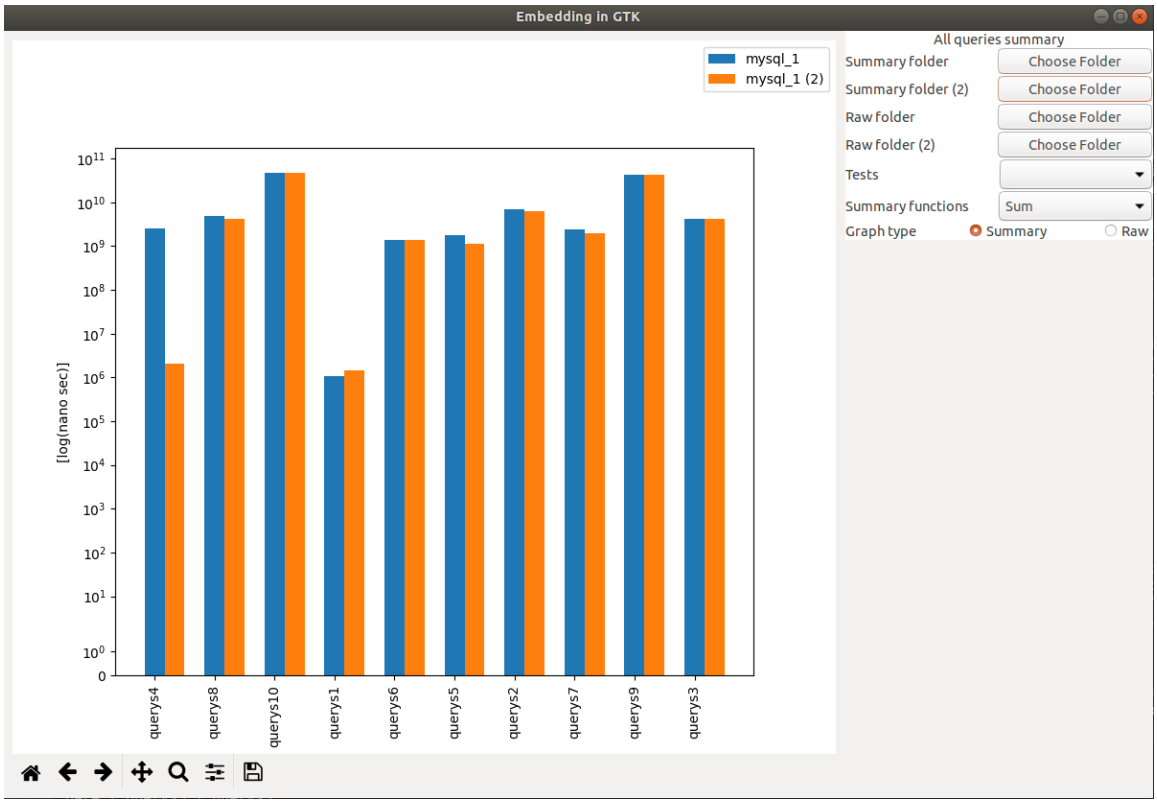
Summarizer si najprv vyparsuje výsledky testovania príkazov. Nad každým testom vykoná všetky štatistické funkcie, ktoré má definované a výsledky uloží do príslušného súboru. Tieto funkcie vie vykonať aj len nad určitým typom príkazov, ak je to definované vo vstupných argumentoch. Ich popis môžeme nájsť v sekcii 4.8.

5.9 Graphical Viewer

Táto časť je písaná v jazyku Python a používa *GTK+* spolu s MVC modelom. Pre vysvetlenie jej implementácie si prejdeme jej grafickými prvkami a ich funkcionalitou. Hlavnú časť okna tvorí graf. Tento graf je vytvorený pomocou knižnice *matplotlib* a jej doplnkom, knižnicou *mplcursors*. Prvá knižnica sa stará o vykreslenie grafov, stĺpcového pre súhrnné dáta a čiarového pre surové dáta bez spracovania. Druhá knižnica nám umožňuje vidieť hodnotu bodu, respektíve stĺpca, pri kliknutí na daný bod, respektíve stĺpec. Typ grafu si môžeme vybrať pomocou widgetu *RadioButton*. Pre vybranie testu alebo skupinového identifikátoru príkazu sa používa *ComboBox*. Ten je použitý tiež pri zvolení funkcie pri súhrnných dátach. Dáta sa načítavajú cez widget *Button*. Je možné načítať súhrnné a surové dáta. Ak by sme chceli porovnať testy s rovnakými menami, teda rovnakú testovaciu sadu pre inú databázu, môžeme druhú sadu načítať nezávisle od prvej. V takom prípade sa k databáze pridá značka

(2), ktorá ukazuje, že ide o druhú testovaciu sadu, respektíve databázu. Dáta sa parsujú zo zvoleného súboru. Všetky dáta sú iba na čítanie.

Vzhľad výslednej aplikácie



Obr. 5.1: Graphical Viewer

5.10 Porovnanie s existujúcimi riešeniami

Po skončení implementácie je čas na porovnanie vytvoreného nástroja s existujúcimi riešeniami s kapitoly 3. Všetky spoločné parametre, ako sú prihlasovacie údaje na databázu, vytvorenie príkazov a menenie ich počtu tento nástroj obsahuje.

Neobsahuje automatické generovanie testovacej sady a to z dôvodu, že sa musí vzťahovať už na nejakú existujúcu databázu. Ak je však definovaná testovacia sada, tak samotné vstupné testy sú vytvorené už automaticky.

Netestuje prípady, keď na databázu prichádzajú požiadavky od viacerých klientov. Prvý dôvod je, že táto schopnosť testovania nebola v cieľových požiadavkách. Druhým dôvodom je presnosť testovania. Pri vykonávaní viacerých príkazov by mohlo dôjsť k skresleniu výsledkov.

Ďalšou vlastnosťou, ktorú neobsahuje je napríklad maximálny čas testovania. To by mohlo byť užitočné spolu s parametrom pre maximálny počet príkazov a dalo by sa to použiť pri automatickom generovaní testov s vopred nekonečným počtom. Avšak pre porovnanie rôznych testov alebo celých testovacích sád je dôležité mať rovnaké podmienky pri testoch a to znamená mať rovnaké testy s rovnakým počtom.

Výhodami oproti existujúcim riešeniam je použitie alebo vymenenie každej časti vzhľadom na použitý formát testov. Ďalšou výhodou je grafické spracovanie výsledkov a to v podobe grafov.

Kapitola 6

Testovanie

Táto kapitola sa venuje testovaniu funkčnosti nástroja. Nástroj je testovaný z pohľadu viacerých databáz a z pohľadu rôznych behových prostredí, ktoré sú popísané hneď na začiatku tejto kapitoly. Nasleduje popis spôsobu testovania s príkladmi konkrétnych testov. Na konci kapitoly sa nachádza zhodnotenie výsledkov testov.

6.1 Behové prostredia

V nasledujúcej kapitole sú popísané rôzne behové prostredia, ktoré sú v tejto práci použité. Behové prostredie je rozdelené na systémové ako je operačný systém a hardvérové. Hardvérovým prostredím sa myslia rôzne typy zariadení. Táto kapitola taktiež popisuje dôvod výberu zariadení a ich špecifikácie.

6.1.1 Operačný systém

Jednou z požiadaviek na výslednú aplikáciu bolo, aby bola spustiteľná v linuxovom systéme. Pod linuxovým systémom sa rozumie operačný systém s linuxovým jadrom. Pre túto prácu som si zvolil linuxovú, zdrojovo otvorenú distribúciu Ubuntu, ktorá je postavená na architektúre Debian. Zvolil som si ho, pretože je to jeden z najpoužívanejších linuxových systémov s veľkou podporou pre rôzne nástroje a knižnice. Ďalším dôvodom bola existencia verzie Ubuntu, ktorá podporuje IoT zariadenia. Viac informácií sa nachádza v nasledujúcej časti 6.1.2. Ďalšou požiadavkou je, aby na systéme bežali len najnutnejšie služby a aby boli výsledky testov čo najmenej ovplyvnené operačným systémom. Taktiež operačný systém a ani aplikácia nesmú bežať vo virtuálnom prostredí, čo by mohlo mať za následok skreslenie výsledkov kvôli penalizácii za virtualizáciu.

6.1.2 Zariadenia

Ďalšou požiadavkou pre výslednú aplikáciu a jej testovanie je beh na viacerých zariadeniach. Kvôli presnosti testovania musí aplikácia bežať na jednom fyzickom zariadení a nie napríklad na cloude.

6.1.3 Výber zariadení

Rozhodujúcim faktorom pri výbere zariadení je ich rôznorodosť. Každé vybrané zariadenia má svoj špecifický účel a poznateľne rozdielny výkon. Medzi tieto vybrané zariadenia

patrí IoT zariadenie Raspberry Pi 3 a laptop. Raspberry Pi 3 bolo vybrané ako zariadenie zastupujúce IoT zariadenia. Spomedzi obidvoch zariadení má najmenší výkon a najmenšiu operačnú pamäť. Ďalším zariadením je spomínaný laptop. Reprezentuje vysoký výkon. Taktiež sa odlišuje architektúra procesorov.

Zariadenie Raspberry Pi 3 Model B

Raspberry Pi je malý jedno doskový počítač [5]. Model spomenutý v tejto práci je model tretej generácie s dodatočným označením model B. Ako operačný systém je nainštalovaný Ubuntu Core 18 s priamou podporou pre zariadenia Raspberry Pi. Tento operačný systém je odľahčená verzia desktopového operačného systému Ubuntu 18 a je vo všeobecnosti určená pre IoT zariadenia. Operačný systém je nahraný na Micro SD karte, ktorá slúži ako interná pamäť. Samotný operačný systém je typu server a preto je potrebné pre používanie zariadenie vzdialené pripojenie. Pre komunikáciu so zariadením je potrebné naviazať SSH spojenie. Zariadenie je vyobrazené na obrázku 6.1.

Dôležité technické údaje

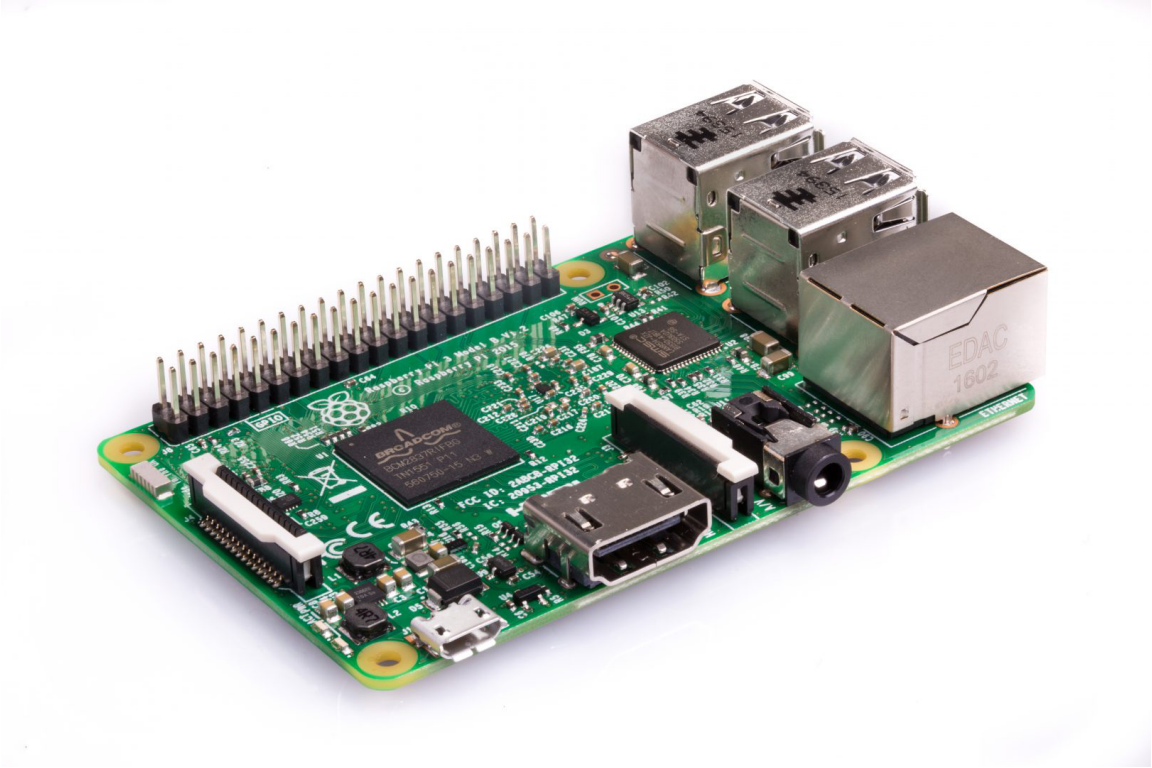
- Štvor jadrový procesor ARM Cortex-A53 CPU (0.60-1.20GHz) Broadcom BCM2837 64bit
- 1GB SDRAM 500 MHz
- Micro SD port pre SD kartu pre nahratie operačného systému a uloženie dát
- Kingston Micro SD HC karta triedy 10 s veľkosťou 16 GB, ktorá je použitá ako interná pamäť v Micro SD slote, sekvenčné čítanie 19MB/s
- Napájanie cez Micro USB port a to až 2.5 A

Laptop

Laptop má modelové označenie Acer Aspire 7 (A715-72G-72Z5). Operačný systém je Ubuntu 18.04.

Dôležité technické údaje

- Šesť jadrový procesor Intel Core i7-8750H (2,20-4,10GHz)
- 8GB RAM DDR4 2133MHz
- primárny pevný disk - Samsung SSD EVO 970 250GB, sekvenčné čítanie 3,4GB/s



Obr. 6.1: Zariadenie Raspberry Pi 3 Model B [5]

6.2 Proces testovania

Najlepším spôsobom testovania nástroja je jeho reálne použitie, teda akceptačné testovanie. Testovanie častí nástroja *Creator*, *Summarizer* a *Graphical Viewer* nie je až tak kľúčové ako testovanie časti *Executor*, pri ktorej musia byť výsledky presné. V tejto časti totiž prebieha meranie parametrov, čo je najdôležitejšia časť nástroja. Samozrejme, ostatné časti musia byť tiež správne. Pre tento účel sa vytvorí testovacia sada, ktorá bude hodnotiť výkon databázy a zároveň preverí funkčnosť nástroja. Testy budú vytvorené, vykonané, spracované a nakoniec zobrazené, čo preverí každú časť nástroja. V testoch budú použité rôzne nastavenia parametrov testovania, ktoré nám preveria rôznu funkčnosť. Táto testovacia sada môže reálne otestovať výkonnosť databázy, ale stále je jej hlavný účel predviesť vlastnosti a funkčnosť celého nástroja.

Testovať sa budú tri databázy 2.3.2: MySQL, PostgreSQL a SQLite. Testy budú spustené na dvoch zariadeniach a to zariadenie Raspberry Pi 3 6.1.3 a spomínaný laptop 6.1.3.

6.2.1 Vytvorenie testovacej sady

Testy pracujú obecne s dvoma typmi tabuliek, ktoré sú rozdelené podľa veľkosti. Malá tabuľka je tabuľka, ktorá obsahuje niekoľko desiatok riadkov a veľká tabuľka obsahuje niekoľko sto tisíc riadkov. Získať nejaké hodnoty pre vytvorenie parametrizovaných testov môžeme viacerými spôsobmi. Získať hodnoty pomocou SQL príkazu znamená, že sa hodnoty pre vytvorenie testov dopĺňajú z hodnôt získaných pomocou SQL príkazu. Získať hodnoty pomocou konštánt znamená, že sa hodnoty pre vytvorenie testov dopĺňajú z vopred stanovených konštánt v šablóne testu. Resetovať databázu znamená, že sa pred vykonaním testu uloží jej stav a po vykonaní tento stav znovu nahrá do databázy. Poprehadzovať riadky znamená, že sa poprehadzujú SQL príkazy vo vytvorenom teste. Počet príkazov je počet vytvorených príkazov pre daný typ príkazu. Viac informácií k parametrom sa nachádza v sekcii 4.6.

Testy sú len príkladom funkčnosti nástroja a ukazujú reálne možnosti ako nástroj použiť. Testovanie sa bude odvíjať od nasledujúcej testovacej sady:

- **1. test** - vyberá riadky z veľmi malej tabuľky podľa podmienky.
- **2. test** - vyberá položku typu dátum z veľkej tabuľky.
- **3. test** - vyberá riadky z veľkej tabuľky a to za podmienky. Podmienka je typu porovnanie čísel, pričom sa tieto čísla získavajú pomocou SQL príkazu.
- **4. test** - vyberá položky z dvoch veľkých tabuliek, ktoré spája pomocou primárnych kľúčov z prvej tabuľky. Test obsahuje 2 podmienky na porovnanie a zoradenie podľa znakového reťazca. Riadky sú poprehadzované.
- **5. test** - maže riadky z veľkej tabuľky za podmienky. Podmienka porovnáva dátumy, ktorých hodnoty získava z SQL príkazu. Databáza sa resetuje.
- **6. test** - aktualizuje riadky veľkej tabuľky za podmienky, ktorej hodnoty sú získané z konštánt. Databáza sa resetuje.
- **7. test** - vyberá riadky na základe rovnosti rovnakej hodnoty a hodnota je typu znakový reťazec.

- **8. test** - rovnaký test ako test číslo 3, ale riadky sú poprehadzované.
- **9. test** - obsahuje dva typy príkazov. Prvý príkaz je rovnaký ako v teste číslo 3, ale jeho počet je nastavený na 10. Druhý príkaz je rovnaký ako v teste číslo 4 a jeho počet je nastavený na 1. Riadky sú poprehadzované.
- **10. test** - obsahuje 4 rôzne typy príkazov, ktorých počty sú nastavené na rôzne hodnoty. Riadky sú poprehadzované a databáza sa resetuje.

6.2.2 Zhodnotenie testov

Testy sa podarilo úspešne vytvoriť, otestovať ich s vybranými databázovými systémami a na rôznych behových prostrediach. Následne boli úspešne spracované a ich výsledky sa dali zobraziť na grafe. Databázy skončili po testoch v rovnakom stave v akom boli pred vykonaním testov. Všetky časti nástroja fungovali správne.

Ďalej by som sa chcel vyjadriť k samotným výsledkom, ktoré sme dostali z testovania. Predmetom tejto práce nie je porovnávanie výkonu databáz, ale vytvorenie nástroja, ktorý to umožňuje. Taktiež nie je predmetom tejto práce vedieť ako presne sú databázy implementované a na základe toho posúdiť ich výsledky. Preto budú nasledujúce výsledky zhodnotené len čiastočne a to hlavne voči samotnému nástroju.

Zhodnotenie výsledkov sa bude vždy týkať rozdielu medzi databázami. Ďalším parameterom budú rôzne zariadenia, rozdielny počet testov a jednotlivé testy z pohľadu nastavených parametrov.

Základná testovacie sada je vždy rovnaká a je to sada, ktorá je opísaná v podsekcii 6.2.1.

Rôzny počet testov

Veľký vplyv na výsledné hodnoty testov môže mať ich kvantita. Pre overenie tohto tvrdenia boli vytvorené dve testovacie sady, kde prvá sada je bez zmeny počtu príkazov a druhá sada má v každom teste 20-krát viac príkazov. Ďalej budeme vychádzať z grafu 6.2. Použité zariadenie je laptop a graf zobrazuje priemerné časy vykonania príkazov.

Ako môžeme vidieť na grafe, priemerný čas príkazov jednotlivých databáz je veľmi podobný až na zanedbateľné hodnoty. Zaujímavý je test *querys4*, kde je priemerný čas príkazu databázy PostgreSQL znateľne pomalší oproti ostatným databázam, ale počet pokusov nevytvára veľkú odchýlku. V teste *querys1* je naopak pri rovankej databáze veľká odchylka v počte pokusov. V tomto teste sa výsledok hľadá vo veľmi malej tabuľke a väčšina času vykonávania príkazu je tak pravdepodobne strávená v samotnom spracovaní a optimalizovaní príkazu, ktorý sa môže líšiť.

Dva pokusy za sebou

Ďalším faktorom, ktorý môže ovplyvniť testy je ich opakované spúšťanie. Pre tento prípad je vybraný graf, ktorý zobrazuje jednotlivé časy príkazov pre jeden test a je to graf 6.3.

Ak sa pozrieme na druhú polovicu grafu, prvý aj druhý beh testu vyzerajú skoro rovnako, čo je aj očakávaný výsledok. Taktiež si môžeme všimnúť, že príkazy rovnakého typu, teda skupinového id, majú veľmi podobné časy. Zaujímavý je štvrtý príkaz pre MySQL databázu. Tento príkaz je najzložitejší a trvá najviac času, čo dokazuje bod s najvyššou hodnotou. Pri druhom behu vidno, že bol príkaz nejakým spôsobom optimalizovaný a tým pádom trvá jeho vykonanie asymetricky veľmi málo času.

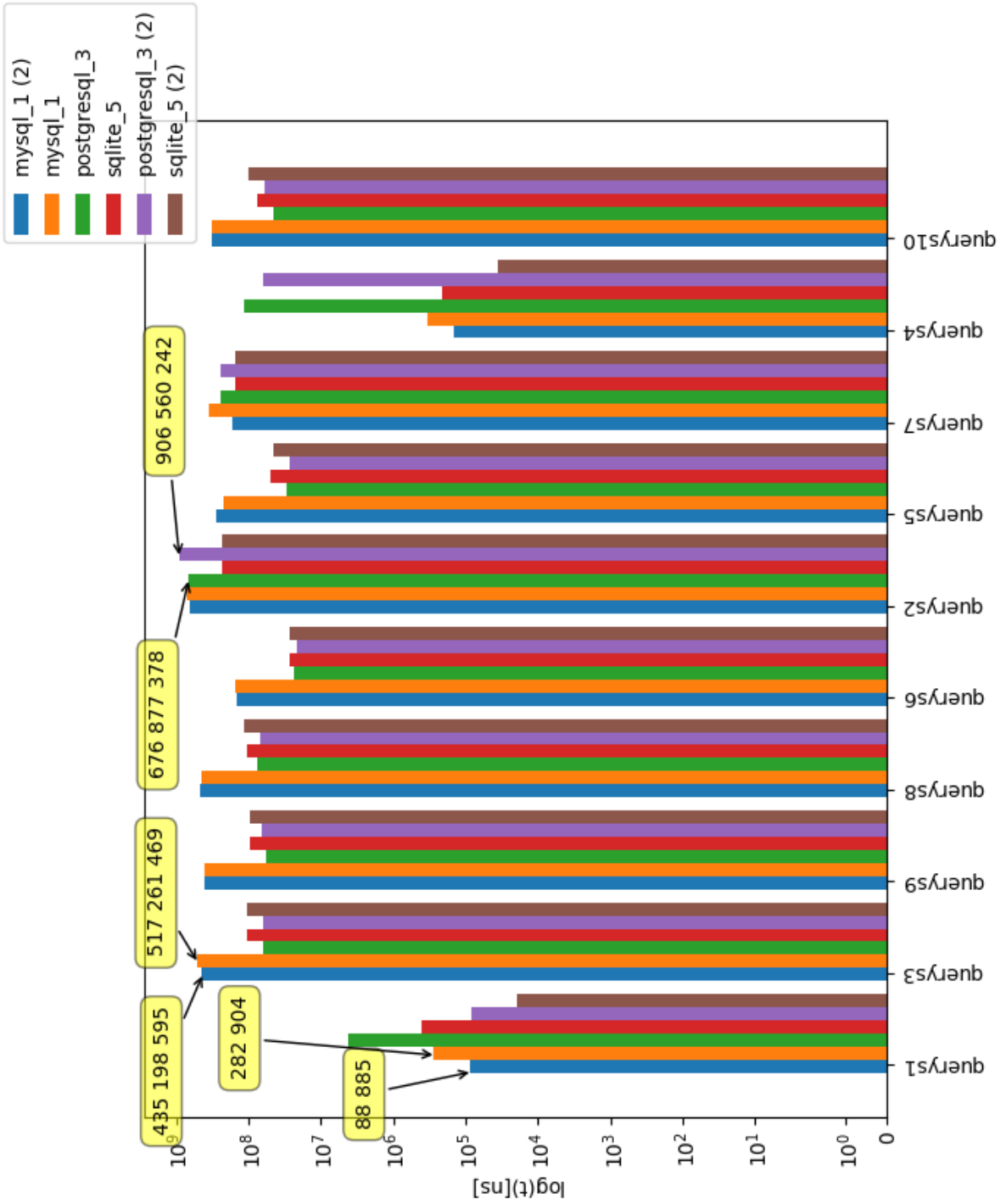
Rôzne zariadenia

Ďalej budeme pracovať s grafom 6.4. Na tomto grafe jasne vidieť, že zariadenie Raspberry je rádovo pomalšie ako laptop. Jediná odchylka je v teste *querys8*, ktorý sa podobá na test *querys3*. Výsledok pre databázu MySQL je podstatne lepší ako pre zariadenie Raspberry. Treba si však uvedomiť, že tieto testy nemenia hodnoty v databáze a preto neobnovujú stav databázy. Test *querys8* je teda optimalizovaný a poprehdzovanie príkazov na neho nemalo žiadny vplyv, lebo boli rovnakého typu. Keď sa tieto dva testy spustia s resetovaním databázy, majú skoro rovnaké hodnoty.

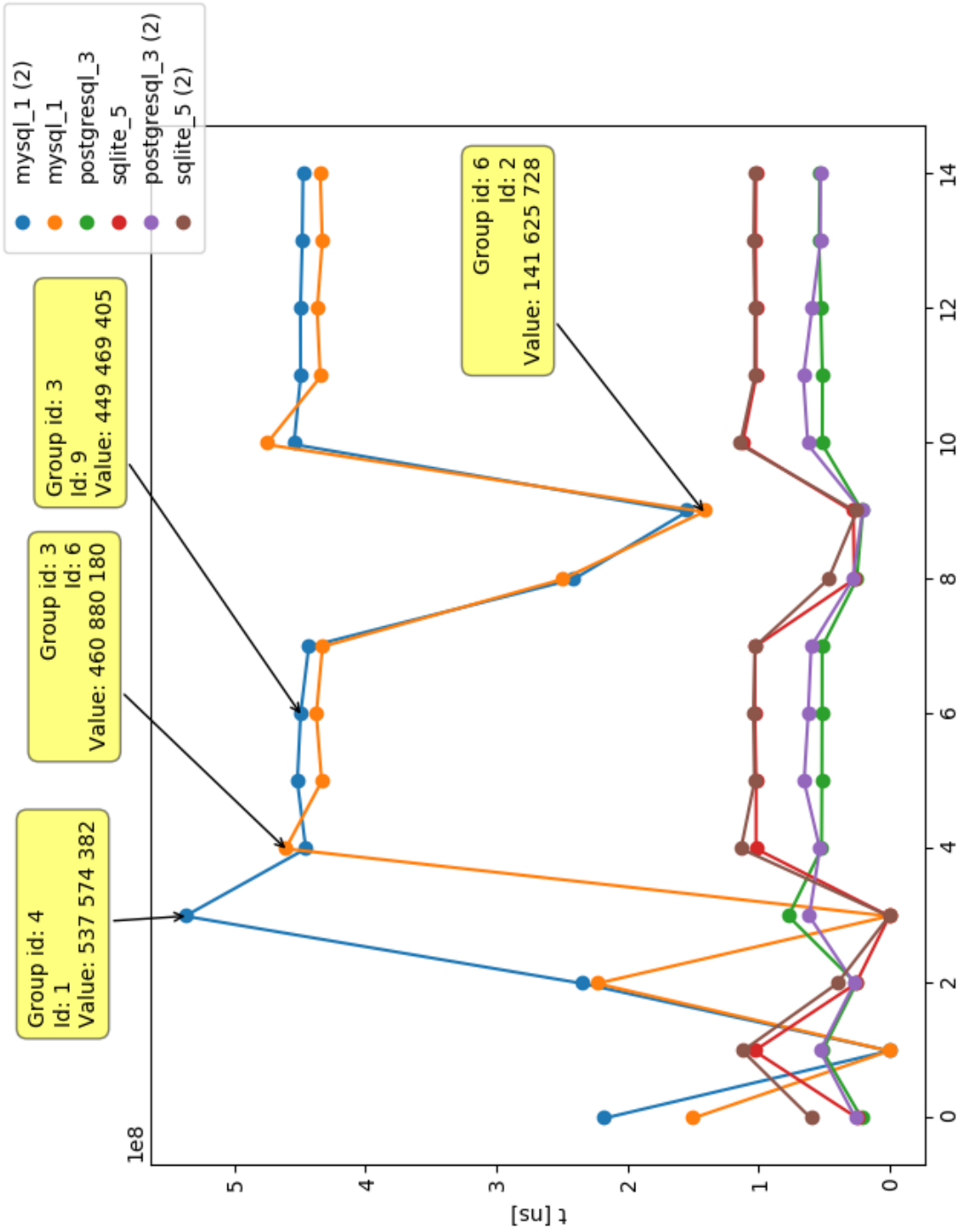
6.2.3 Záver testovania

Všekty časti nástroja pracovali očakávaným spôsobom a vytvorili prezentovateľné a validné výstupy. Pri opakovaných testoch nastávali buď malé odchylky, ktoré mohli byť spôsobené operačným sytémom, hardvérom alebo databázov, napríklad ich mohol spôsobiť výpadok pamäte cache. Väčšie odchylky boli spôsobené databázov a jej optimalizovaním.

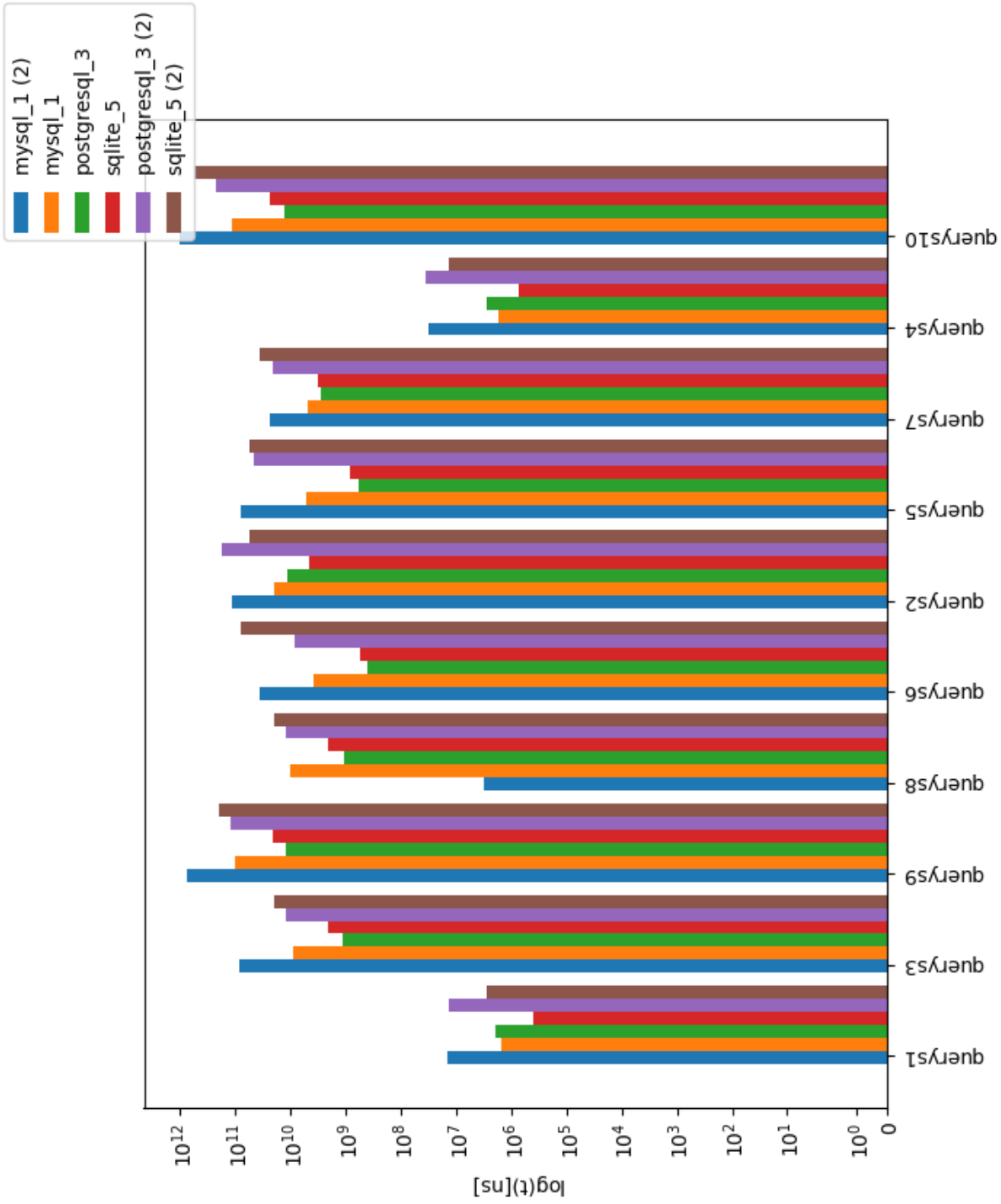
Parameter resetovania databázy a parameter pre poprehadzovanie príkazov fungovali správne. Je len na uživateľovi ako správne ich nastaví. Počet a rôzne typy testov sa dali kombinovať v jednej testovacej sade.



Obr. 6.2: Graf zobrazujúci testy s počtom 1 a testy s počtom 20. Testy s počtom 20 majú pri databázach označenie *(2)*. Na grafe je zobrazený priemerný čas na príkaz a na test. Použité zariadenie je laptop. *querys* je označenie pre test.



Obr. 6.3: Graf zobrazujúci opakované spustenie toho istého testu a to testu *querys10*, čiže testu číslo 10. Testy v prvom behu majú pri databázach označenie *(2)*. Na grafe je zobrazený čas pre každý príkaz pre jeden test. Použité zariadenie je laptop. Počet opakovaní v rámci testu je 1.



Obr. 6.4: Graf zobrazujúci spustenie testovacej sady na rôznych zariadeniach. Testy pre zariadenie Raspberry Pi 3 majú pri databázach označenie *(2)*. Druhá množina testov bežala na zariadení laptop. Na grafe je zobrazený súhrnný čas pre každý test. Počet opakovaní v rámci testu je 20.

Kapitola 7

Záver

Cieľom práce bolo vytvorenie nástroja na testovanie výkonnosti rôznych typov databáz. Medzi testované databázy patrí MySQL, PostgreSQL a SQLite. Pri vývoji sa kládol dôraz na použiteľnosť aplikácie a presnosť pri vytváraní výsledkov. Pri implementácii nástroja sa počítalo s tým, že ho bude používať skúsenejší uživateľ.

Výslednou aplikáciou je nástroj, ktorý je zložený zo štyroch častí. Prvá časť *Creator* sa stará o vytvorenie testovacej sady podľa uživateľom vytovrených šablón. Testovacia sada sa skladá z viacerých testov, ktoré obsahujú jednotlivé príkazy. Tieto príkazy môžu byť parametrizovateľné. Druhá časť *Executor* vykonáva testy a zaznamenáva pri tom dobu vykonania príkazu. Tretia časť *Summarizer* vyhodnocuje výsledky testov. Používa na to štatistické funkcie, ktoré majú veľkú výpovednú hondotu. Štvrtá časť *Creator* používa grafy na vizualizáciu nameraných hodnôt.

Samostatným celkom bola práca s databázami. Nástroj pristupuje k rôznym databázam rovnako pomocou jednotného rozhrania. Vie sa prihlasovať na databázu pomocou konektorov, vykonávať SQL príkazy alebo vrátiť stav databázy do pôvodného stavu, teda stavu pred vykonaním testu.

Nástroj bol otestovaný a to s použitím reálnych testov. Výsledky testov zobrazovali predpokladané hodnoty a počas testovania sa nevyskytol žiadny problém. Tieto výsledky sú zobrazené v kapitole 6.

Nástroj splnil všetky vopred zadané požiadavky na funkcionalitu. Oproti existujúcim riešeniam mu chýbajú určité vlastnosti, ktoré by sa dali doimplementovať. Cieľom tejto práce však nebolo vytvorenie jednej dokonalej časti, ale vytvorenie celku v ktorom bude každá časť fungovať podľa vopred zadanej špecifikácie.

Literatúra

[1] *Clustered and Nonclustered Indexes Described*. [Online].
URL https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-2017

[2] *mysqlslap — Load Emulation Client*. [Online].
URL https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html

[3] *Performance Testing Tutorial: What is, Types, Metrics & Example*. [Online].
URL https://www.guru99.com/performance-testing.html

[4] *Query Execution Flow Architecture (SQL Server)*. [Online].
URL
https://www.sqlrelease.com/query-execution-flow-architecture-sql-server

[5] *Raspberry Pi 3 Model B*. [Online].
URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[6] *Read Time-Stamp Counter*. [Online].
URL https://c9x.me/x86/html/file_module_x86_id_278.html

[7] *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. [Online].
URL https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[8] *SQLQueryStress: The Source Code*. [Online].
URL http://dataeducation.com/sqlquerystress-the-source-code/

[9] *SQL Tutorial*. [Online].
URL https://www.tutorialspoint.com/sql/

[10] *sysbench manpage*. [Online].
URL http://manpages.ubuntu.com/manpages/trusty/man1/sysbench.1.html

[11] *What is RDBMS?* [Online].
URL https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm

[12] Fritchey, G.: *SQL Server 2012 Query Performance Tuning*. Apress, tretie vydanie, 2012, ISBN 978-1-4302-4203-1.

[13] Molyneaux, I.: *The Art of Application Performance Testing*. O'Reilly Media, Inc., druhé vydanie, 2014, ISBN 9781491900536.

[14] Tezer, O.: *Black Box Testing*. [Online].
URL http://softwaretestingfundamentals.com/black-box-testing/

[15] Tweney, D.: *5-minute outage costs Google $545,000 in revenue*. [Online].
URL https://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue/