UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rene Škarabot

Performančna analiza podatkovnih zbirk v pomnilniku

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2014

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00474 / 2013
Datum: 12.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **RENE ŠKARABOT**

Naslov: **PERFORMANČNA ANALIZA PODATKOVNIH ZBIRK V POMNILNIKU**
PERFORMANCE ANALYSIS OF IN MEMORY DATABASES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Hitrost izvajanja operacij je pogosto dejavnik, ki vpliva na izbiro podatkovne baze. Podatkovnim bazam na disku že dlje časa konkurirajo podatkovne baze v pomnilniku, ki podatke hranijo v primarnem pomnilniku računalnika. V primerjavi z diskom so mnogo hitrejše, s pocenitvijo glavnega pomnilnika pa so postale cenovno dostopnejše. V okviru diplomskega dela najprej predstavite podatkovne baze na disku in v pomnilniku ter prednosti in slabosti enih in drugih. Nato definirajte in izvedite testne scenarije na MySQL podatkovni bazi na disku in na ostalih podatkovnih bazah v pomnilniku, s poudarkom na merjenju hitrosti. Izvedite analizo rezultatov in jih primerjajte med seboj.

Mentor: Dekan:

viš. pred. dr. Aljaž Zrnec prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Rene Škarabot,

z vpisno številko 63100125,

sem avtor diplomskega dela z naslovom:

Performančna analiza podatkovnih zbirk v pomnilniku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aljaža Zrneca;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 10. 3. 2014 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju viš. pred. dr. Aljažu Zrnecu za pomoč in koristne nasvete pri izdelavi diplomskega dela.

Posebna zahvala gre moji družini, ki mi je stala ob strani vsa leta študija in me spodbujala ter punci Galli za spodbudne besede.

Zahvaljujem se tudi razvijalcem odprtokodnih projektov, da sem lahko uporabil odprtokodne programe, ki jih sicer ne bi mogel.

Kazalo

Seznam uporabljenih kratic in simbolov

PB	– podatkovna baza
IMDB	– podatkovna baza v pomnilniku (ang. in-memory database)
SUPB	– sistem za upravljanje s podatkovno bazo
API	– programski vmesnik (ang. application programming interface)
Checkpoint	– kontrolne točke
ACID	– atomarnost, konsistentnost, izolacija, trajnost
DDL	– data definition language, omogoča definiranje podatkovne baze
RSUPB	– relacijski sistem za upravljanje s podatkovno bazo
NVDIMM	– Non-Volatile Dual In-line Memory Module
CPE	– centralno procesna enota
SSD	– solid-state disk

Povzetek

Izbira ustreznega tipa podatkovne baze je zelo pomembna odločitev preden začnemo delo na kakršnemkoli projektu. Na izbiro imamo standardne diskovne podatkovne baze, vse bolj pa se uveljavljajo podatkovne baze v pomnilniku, predvsem zaradi njihove hitrosti dostopa do podatkov. Večina resnejših podatkovnih baz v pomnilniku podpira ACID tako, da so podatki varni pred izgubo tudi v primeru izpada električne energije, kar je te podatkovne baze naredilo še bolj priljubljene.

Namen diplomskega dela je preizkusiti in izmeriti kako se podatkovne baze v pomnilniku performančno razlikujejo med seboj in kako se performančno razlikujejo v primerjavi z zelo razširjeno podatkovno bazo MySQL na disku z indeksi in brez njih. Ker imajo podatkovne baze v pomnilniku različen namen uporabe kot diskovne, sem izbral tudi več testnih scenarijev, da bi bila razlika čim bolj vidna. Na začetku so predstavljeni mediji za hranjenje podatkovnih baz, relacijske podatkovne baze in podatkovne baze, ki sem jih uporabil pri diplomski nalogi. Sledi predstavitev testnega okolja, strojne in programske opreme, ki sem jo uporabil pri testih. Na koncu pa predstavitev testnih scenarijev in analiza meritev ter možne izboljšave.

Ključne besede:

podatkovne baze v pomnilniku, TimesTen, MySQL, IBM SolidDB, SQLite, relacijske podatkovne baze.

Abstract

The choice of the appropriate database type presents an important factor in any project development. On account of their data access speed, memory databases have gradually gained ground against the formerly predominant standard disk databases. Since the majority of significant memory databases support the ACID properties, data loss is prevented even in case of power failure; hence their growing popularity.

The thesis will examine and measure the extent to which, in performance, memory databases differ from one another, as well as from the extensively widespread, be it indexed or non-indexed, MySQL disk database. As the purposes of memory and disk databases diverge, several test scenarios will be presented in order to further illustrate the differences between the two. The thesis will first present the data storage media, relational databases and indexes, along with the databases examined in the thesis research. Then, the focus will shift to the presentation of the test environment, namely the hardware and software used in the research. Finally, the thesis will provide an overview of test scenarios, followed by the analysis of the obtained measurements, as well as the potential improvements.

Key words:

in-memory database, TimesTen, MySQL, IBM SolidDB, SQLite, relational databases.

1. UVOD

Zgodovina raziskav podatkovnih baz v zadnjih 30 letih je izjemno produktivna in je privedla do tega, da so SUPB postali eni najpomembnejših razvojnih procesov na področju programske opreme. Podatkovna baza je sedaj temeljni okvir informacijskega sistema, ki je temeljito spremenil način delovanja mnogih organizacij.

Podatkovne baze v pomnilniku so se pojavile kot odgovor na nove cilje aplikacij, sistemskih zahtev in operacijskih okolij. V aplikacijah, kjer je odzivni čas kritičnega pomena, kot so telekomunikacijska in mobilno oglaševalna omrežja, se baze v glavnem pomnilniku veliko uporabljajo. Podatkovne baze v pomnilniku (ang. in-memory database - IMDB) so v zadnjih letih zelo v porastu, še posebno pri analizi podatkov (ang. data analytics). Velika rast se je pričela po letu 2000, ko so se cene glavnih pomnilnikov začele spuščati. [6]

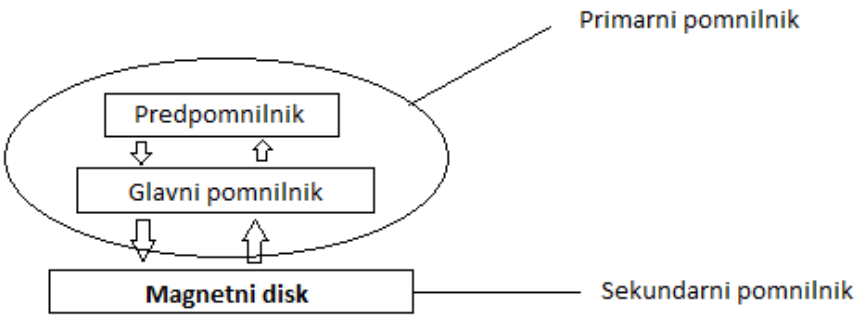
Namen diplomskega dela je predstavitev, analiza in performančna primerjava podatkovnih baz v pomnilniku s klasično - relacijsko MySQL podatkovno bazo na disku. Zmogljivost sem meril z različnimi scenariji uporabe podatkovnih baz. Prikazal bom tudi, kako sem produkte namestil, ter kako in kašne teste sem nad njimi izvajal.

Diplomsko delo vsebuje devet poglavij. V prvem poglavju je predstavljena tematika diplomskega dela in namen izdelave le tega. V drugem sem opisal medije za hranjenje podatkovnih baz. V tretjem poglavju so predstavljene relacijske podatkovne baze, primerjava med diskovnimi in pomnilniškimi bazami ter definicija indeksov. V četrtem poglavju so opisane relacijske podatkovne baze, ki sem jih uporabil pri diplomski nalogi in njihove lastnosti. V petem poglavju sledi predstavitev testnega okolja, strojne in programske opreme, nakar so v šestem poglavju prikazani testni scenariji in uporabljeni model podatkovne baze Northwind. Sledi sedmo poglavje z analizo rezultatov, osmo poglavje sem posvetil možnim izboljšavam testiranja, deveto pa sklepnim ugotovitvam.

2. MEDIJI ZA HRANJENJE RELACIJSKIH PODATKOVNIH BAZ

Obstajajo različni mediji za hranjenje relacijskih podatkovnih baz. Danes sta največ v uporabi obstojni medij – trdi disk in neobstojni – glavni pomnilnik. Slednji primer uporablja za zagotavljanje trajnosti posebne mehanizme, ki sodelujejo z diskom. Za pravilno izbiro hranjenja podatkov, na disku ali v pomnilniku, moramo vedeti, katerim pogojem bomo dali prednost. Če hitrost streženja podatkov za nas ni toliko pomembna in želimo imeti cenovno ugodno podatkovno bazo, je primernejša diskovna podatkovna baza. Če pa je pomembna hitrost in cena glede na enoto shranjene informacije ne predstavlja problema, potem so bolj priporočljive podatkovne baze v pomnilniku.

Glavni kriteriji za izbiro vrste pomnilnika za hranjenje fizične podatkovne baze so:
- sposobnost hranjenja velikih količin podatkov;
- trajnost pomnjenja;
- hitrost vpisovanja, spreminjanja in dostopa do podatkov;
- cena na enoto shranjenih podatkov. [9]



Slika 1: Hierarhija pomnilnikov.

3. RELACIJSKE PODATKOVNE BAZE

3.1. Kaj je relacijska podatkovna baza

Podatkovna baza je enotna, po možnosti velika zbirka podatkov, ki se lahko sočasno uporablja z veliko oddelki in uporabniki. Namesto nepovezanih datotek z veliko podvojenimi podatki, so podatki v relacijski bazi povezani z minimalnim podvajanjem. Podatkovna baza ni le organizacija podatkov, ampak vsebuje tudi opise le teh (meta podatki), ki se nahajajo v sistemskem katalogu. Podatkovna baza je namenjena zagotavljanju podatkovne neodvisnosti od aplikacije.

Relacijska podatkovna baza vsebuje tabele z zapisi. Podatki iz različnih tabel so lahko medsebojno povezani [2].

Relacijske baze podatkov so najbolj pogosta oblika hranjenja podatkov, zato se bom osredotočil le nanje. Ostali modeli baz podatkov (hierarhičen model, mrežni model, objektni model, itd.) so zelo redki in se trenutno uporabljajo le v posebnih primerih. [8]

3.1.1. SUPB in RSUPB

SUPB je sistem za upravljanje s podatkovno bazo, ki komunicira z uporabniškimi programi in upravlja s podatki, ki se nahajajo na disku. Uporabnikom omogoča kreiranje podatkovne baze prek DDL jezika, izvajanje poizvedb, posodobitev, vstavljanj, pridobivanje podatkov in zagotavlja nadzorovan dostop do podatkov v podatkovni bazi.

RSUPB predstavlja relacijski SUPB, ki temelji na relacijskem podatkovnem modelu, katerega je predlagal E.F. Cood (1970). V relacijskem modelu so vsi podatki logično predstavljeni z relacijami. Vsaka relacija ima ime in je sestavljena iz atributov (stolpcev). Vsaka n-terica (vrstica) vsebuje po eno vrednost za atribut. Prav ta preprostost je moč relacijskega modela. [2]

3.2. Podatkovne baze na disku

Navadno so podatkovne baze shranjene na disku. Večina podjetij uporablja standardne diskovne podatkovne baze za realizacijo svojega beleženja podatkov, saj je disk v primerjavi z glavnim pomnilnikom veliko cenejši. Relacijske podatkovne baze na disku so v uporabi že od leta 1970 in so, v primerjavi s pomnilniškimi, nekoliko bolj priljubljene. [9]

3.2.1. Učinkovitost dostopa do podatkov

Učinkovitost dostopa do podatkov ni odvisna le od tehničnih značilnosti izbrane strojne opreme, na primer od hitrosti trdega diska. Veliko vlogo igra tudi programska realizacija, pri čemer se upoštevajo naslednja pravila:

- maksimalno izrabiti glavni pomnilnik - če program potrebuje veliko podatkov, jih naenkrat čim več prepišemo z diska v glavni pomnilnik;
- minimizirati število dostopov do diska - bolje je hkrati prebrati več podatkov;
- pazljivo organizirati podatke na disku - poiskati način, ki omogoča čim bolj neposredno dostopanje do želenih podatkov in se s tem izogniti zaporednemu iskanju po velikih datotekah. [9]

3.2.2. Kriteriji za izbiro diska

Pri izbiri trdega diska se upoštevajo naslednji parametri:
- dostopni čas,
- hitrost prenosa podatkov,
- pričakovana življenjska doba,
- cena.

Dostopni čas je vsota iskalnega časa, ki ga bralno pisalna glava potrebuje, da se premakne na želeno sled in rotacijske zakasnitve. Rotacijska zakasnitev je čas, ki ga plošča potrebuje, da se postavi na začetek želenega sektorja. Povprečne dostopne čase merimo v milisekundah. Pri današnjih diskih te znašajo približno 10 milisekund. [9]

3.3. Podatkovne baze v pomnilniku

Podatkovne baze v pomnilniku so se pojavile kot odgovor na nove zahteve aplikacij, sistemskih zahtev in operacijskih okolij.

Podatkovna baza v pomnilniku je sistem za upravljanje s podatkovno bazo, ki se pri shranjevanju podatkov primarno opira na glavni pomnilnik, kar predstavlja nasprotje podatkovnim bazam, ki za shranjevanje podatkov uporabljajo mehanizem shranjevanja podatkov na trdi disk.

Ker je pomnilnik hitrejši od diska, se podatkovne baze v pomnilniku pogosto uporabljajo v aplikacijah, kjer so potrebni zelo majhni in predvidljivi odzivni časi, kot so telekomunikacije, aplikacije poslovanja v realnem času, omrežni opremi in velikih spletnih aplikacijah. Prav tako se razlikujejo od drugih sistemov za shranjevanje podatkov v pomnilniku, ki za

shranjevanje uporabljajo pare "ključ - vrednost", kot so Memcached, Hazelcast ali Coherence, ki podatke le pripravijo v pomnilnik, realno pa za njimi stoji diskovna baza podatkov.

Z uvedbo NVDIMM tehnologije lahko sedaj baze v glavnem pomnilniku tečejo s polno hitrostjo, saj se podatki v pomnilniku ohranijo tudi v primeru izpada električne energije. [6]

3.3.1. NVDIMM (Non-Volatile Dual In-line Memory Module)

Je DRAM, ki uporablja super kondenzatorje kot vir energije, ohranja podatke tudi v primeru nepričakovanega izpada električne energije, sistemske nesreče ali v primeru običajne zaustavitve sistema. NVDIMM lahko uporabimo za izboljšanje delovanja aplikacij in varnost podatkov. Je mešan podsistem, ki združuje hitrost in vzdržljivost DRAM pomnilnika, skupaj z obstojnostjo podatkov NAND flash spominskih modulov. NVDIMM, ki uporabljajo DRAM in NAND tehnologijo, lahko prinesejo visoke hitrosti in nizko latenco. [11]

3.3.2. ACID (Atomarnost, konistentnost, izolacija, trajnost) pri podatkovnih bazah v pomnilniku

Podatkovne baze v glavnem pomnilniku shranjujejo podatke na pomnilnik, ki za svoje delovanje in obstoj podatkov potrebuje stalno napajanje. Če se napetost izgubi, se s tem izgubijo tudi vsi podatki, ki smo jih hranili v glavnem pomnilniku. V tem primeru, IMDB ne ustreza standardom, ki jih določa ACID, torej ne podpira vseh zahtev. Neobstojni pomnilnik lahko podpira ostale tri ACID zahteve po atomarnosti, konsistentnosti in izolaciji, ne podpira pa trajnosti, ker mora imeti za ohranjanje podatkov neprekinjeno električno energijo.

Od kar je postal NVDIMM pomnilnik dostopen, je s tem bila odpravljena tudi zadnja pomanjkljivost o trajnosti podatkov.

Številne podatkovne baze v glavnem pomnilniku uporabljajo za zagotavljanje trajnosti datoteke s stanjem sistema, ki shranijo stanje podatkovne baze v določenem trenutku. Ti določeni trenutki, v katerih se delajo varnostne kopije podatkov, so navadno periodično generirani. Ta način pa ne ponuja polne trajnosti podatkov, saj se v primeru nepričakovane zaustavitve računalnika vseeno izgubijo podatki, ki so bili vneseni po zadnji varnostni kopiji. Za popolno zagotavljanje trajnosti podatkov mora biti sistem dopolnjen še z algoritmom beleženja transakcij.

Beleženje transakcij je zgodovina izvršenih akcij SUPB-ja, za zagotavljanje ACID lastnosti pri nesrečah ali okvarah strojne opreme. Fizično je to dnevnik, datoteka vseh sprememb, ki so nastale do sedaj na podatkovni bazi, shranjena na stabilnem pomnilniku. Če podatkovna baza po zagonu naleti na nekonsistentno stanje ali je bila podatkovna baza izpostavljena nesreči, SUPB pregleduje dnevnike, v katerih so zapisane vse dosedanje transakcije. S pomočjo teh dnevnikov povrne podatkovno bazo nazaj v staro stanje, preden so se te transakcije izvedle.

Vse transakcije, ki so bile že potrjene (commit), vendar njihove spremembe niso bile realizirane v podatkovni bazi, se izvedejo še enkrat.

Beleženje transakcij se uporablja za zagotavljanje atomarnosti in trajnosti podatkov. [6]

3.3.3. Primerjava podatkovne baze na disku in v pomnilniku

Ko so se pojavili prvi sistemi za upravljanje podatkov, so bili vsi podatki shranjeni in dostopni iz trdega diska v primernem času. DBMS oblikovalci so se osredotočili na optimizacijo I/O in poskušali izboljšati dostop do podatkov z uvedbo strukture blokov. Osnovna strategija je pogosto osredotočena na skupni vmesni pomnilnik, kjer so podatkovni bloki pripravljeni na ponovno uporabo. [1]

Ko se je glavni pomnilnik pocenil, je veliko uporabnikov povečalo svoj vmesni pomnilnik, dokler ni bil ta dovolj velik, da je v njem lahko bila celotna baza podatkov.

Ena največjih razlik med podatkovnimi bazami na disku in podatkovnimi bazami v pomnilniku, je ta, da pri slednjih nimamo velikih blokov podatkov, ki se prenašajo z diska v pomnilnik. Tabele so shranjene v glavnem pomnilniku, tako da lahko podatke dodajamo brez reorganizacije velikih blokovnih struktur, kot je to potrebno pri podatkovnih bazah na disku, ki imajo podatke prednaložene v glavnem pomnilniku. Ko procesor zahteva podatke z diska, se morajo ti najprej prenesti v glavni pomnilnik in nato so na voljo za obdelavo. Pri spremembi podatkovne baze na disku, se morajo tudi v pomnilniku ažurirati podatki.

Podatkovne baze v glavnem pomnilniku so hitrejše od baz realiziranih na trdem disku, saj so notranji algoritmi optimizacije, ki so shranjeni v glavnem pomnilniku, preprostejši in potrebujejo manj navodil CPE-ja od tistih z diska. Pri slednjih se namreč morajo podatki, do katerih želi dostopati procesor, najprej iz diska prenesti v glavni pomnilnik. Če teh podatkov ni na razpolago, mora procesor dati ukaz za branje podatkov iz diska, kar pa postane zelo zamudno. Dostopanje do podatkov v pomnilniku odpravlja t.i. "seek time" (čas iskanja), ki je potreben, da se bralno – pisalna enota pomakne na pravo sled, znotraj te pa do pravega sektorja, iz katerega prebere podatke. S tem dosežemo hitrejše in bolj predvidljivo delovanje kot z diskom.

Cilji optimizacije podatkovne baze na disku so v popolnem nasprotju s tistimi iz baze v pomnilniku. Primarno breme pri podatkovnih bazah na disku predstavljajo I/O operacije. Zato podatkovne baze na disku skušajo zmanjšati I/O operacije in s tem obremenijo pomnilnik in CPE. To vključuje uporabo dodatnega pomnilnika za predpomnilnik in CPE ciklov za vzdrževanje predpomnilnika ter optimizacijo poizvedb. Z uporabo predpomnilnika klasične podatkovne baze na disku tudi prenašajo veliko odvečnih podatkov naokoli. V popolnem nasprotju pa podatkovne baze v pomnilniku ne potrebujejo I/O operacij. Že od začetka je

struktura podatkov bolj racionalizirana z vidika optimizacije porabe pomnilnika in CPE. Čeprav se cene pomnilnika v zadnjih letih nižajo, ga razvijalci še vedno obravnavajo kot dragocenega, zato mora biti IMDB zasnovan tako, da je poraba pomnilnika čim bolj izkoriščena. Podatkovna baza v pomnilniku je boljša zaradi svojih prednosti v zmogljivostih, drugi cilj izboljšave pa je vedno čim boljši izkoristek CPE moči. [7]

3.4. INDEKSI

V diplomski nalogi sem pri vseh testnih scenarijih uporabil tudi indekse, saj sem opazoval njihov vpliv na hitrost izvajanja testov.

Indeks je struktura, ki omogoča pospešen dostop do vrstic, na podlagi vrednosti enega ali več atributov. Uporaba indeksov lahko bistveno izboljša dostopni čas do podatkov. Indeksi se uporabljajo za pospešitev iskanja podatkov po tabelah, ki vsebujejo relativno veliko število podatkov. Kreacija indeksa ni standardni SQL,vendar pa jih večina SUPB-jev podpira. [2]

KREACIJA INDEKSA:

CREATE [UNIQUE] INDEX ImeIndeksa ON ImeTabele (ImeStolpca [ASC, DESC]);

Podani stolpci predstavljajo ključ indeksa, ki bi moral biti naveden od največjega do najmanjšega. Indeksi so lahko kreirani samo na osnovnih tabelah in ne na pogledih. Če je uporabljena predpona "unique", bo za unikatnost indeksa poskrbel SUPB. To je potrebno za primarne ključe in tudi za druge stolpce, kot npr. nadomestne ključe. Čeprav je mogoče indekse ustvariti kadarkoli, imamo lahko s tem težave. Če ustvarimo enoličen indeks na določenem stolpcu, lahko ta že vsebuje dvojnike. Zato je priporočljivo, da enolične indekse ustvarimo takoj ob kreaciji tabele. [2]

4. PREDSTAVITEV TESTNIH PODATKOVNIH BAZ

4.1. MySQL

Kot izhodiščno podatkovno bazo, ki je bila običajna diskovna podatkovna baza, sem uporabil MySQL 5.5.32 relacijsko podatkovno bazo.

MySQL je razvilo švedsko podjetje MySQL AB. Prva verzija je izšla leta 1995, sedaj je v lasti Oracla. MySQL podatkovna baza je popularna izbira podatkovne baze za spletne aplikacije in je osrednji sestavni del široko uporabljene LAMP (Linux, Apache, MySQL, Perl/PHP/Python) namestitve. MySQL se uporablja tudi pri velikih, odmevnih spletnih straneh kot so: Wikipedija, Google, Facebook, Twiter, Flickr in YouTube.

MySQL je sistem za upravljanje s podatkovnimi bazami. Napisan je v jeziku C in C++, deluje na več operacijskih sistemih, uporablja zelo hitre diskovne tabele MyISAM s stiskanjem indeksov, če izberemo "MyISAM storage engine". MySQL je odprtokodna implementacija relacijske podatkovne baze, ki za delo s podatki uporablja jezik SQL. MySQL deluje na principu "odjemalec - strežnik". [10]

Omejitve

MySQL je tako kot večina drugih relacijskih transakcijskih baz zelo omejena zaradi počasnosti trdega diska. Še posebej to velja pri pisanju v bazo. Tudi z uporabo SSD diskov, ki so zelo dragi in odpravijo zakasnitev mehanike pri navadnih SATA diskih, baze ne pohitrimo dovolj. Poleg tega ne izpolnjuje vseh SQL standardov, bazni prožilci pa so omejeni na enega na akcijo. [10]

Zagotavljanje visoke razpoložljivosti

Zagotavljanje visoke razpoložljivosti zahteva določeno količino redundance (podvajanje, potrojevanje) v sistemu. Pri podatkovnih bazah redundanca tradicionalno poteka tako, da imamo primarni strežnik (gospodar), z uporabo replikacije pa imamo na voljo tudi sekundarne strežnike z zadnjo verzijo podatkovne baze v primeru, če glavni odpove. To pomeni, da strežnik, na katerega se povezuje aplikacija, predstavlja zbirko strežnikov in ne en sam strežnik. [10]

4.2. Oracle TimesTen

Pri nalogi sem uporabil Oracle TimesTen podatkovno bazo v pomnilniku verzije 11.2.2.

TimesTen je relacijska podatkovna baza, shranjena v pomnilniku z obstojnostjo in obnovljivostjo. Razvilo jo je podjetje HP leta 1995, leta 2005 pa jo je prevzel Oracle.

Izdelek je postal hitro priljubljen predvsem v telekomunikacijah, kjer je pomemben odzivni čas mikrosekund, predvsem za aplikacije, npr. paketno preklapljanje. Vsi podatki v TimesTen podatkovni bazi se nahajajo v glavnem pomnilniku, kar pomeni, da nimamo opravka z I/O operacijami, ki so potrebne pri navadnih diskovnih bazah in s tem zelo upočasnjujejo delovanje. Zagotavlja kratke odzivne čase in ima zagotovljeno zelo veliko prepustnost podatkov tudi za najintenzivnejše delovne obremenitve. Ima vse funkcije relacijske baze podatkov, do katerih je mogoče dostopati s standardnimi programskimi vmesniki (ang. Application Programming Interface – API) ter zagotavlja polno funkcionalnost SQL poizvedovalnega jezika. [15]

4.2.1. Pregled arhitekture in komponent

Deljene knjižnice

Funkcionalnost TimesTen je vsebovana v nizu skupnih knjižnic, ki razvijalcem aplikacij omogoča, da se TimesTen izvede kot del procesa aplikacije. Pristop skupnih knjižnic je drugačen od običajnih RDBMS sistemov, kjer je podatkovna baza niz različnih procesov, do katerih se aplikacije povezujejo preko medprocesne komunikacije. Ta način komunikacije je lahko v obliki povezave "odjemalec - strežnik", ki poteka preko omrežja. Lahko je neka oblika notranjega komuniciranja npr. Unix domene ali pomnilnik v skupni rabi. [15]

Baza podatkov naseljena v pomnilniku (ang. Memory-resident Database)

Podatki za vsako aktivno podatkovno bazo TimesTen so shranjeni v skupnem segmentu pomnilnika, ki omogoča, da je lahko več TimesTen podatkovnih baz aktivnih istočasno in omogoča aplikacijam hkratni dostop do več TimesTen podatkovnih baz na istem sistemu. Na 64-bitnem sistemu je velikost podatkovne baze odvisna samo od količine glavnega pomnilnika. [15]

Procesi

Zagon TimesTen se začne z zagonom TimesTen glavnega procesa, ki se izvaja v ozadju. Ta požene več TimesTen pod-procesov za upravljanje z vsemi podatkovnimi bazami v sistemu. Ti pod-procesi izvajajo operacije kot so:

- nalaganje podatkovne baze v glavni pomnilnik in zapisovanje baze na disk pri nadzorovani zaustavitvi strežnika,
- periodično beleženje kontrolnih točk podatkovne baze na disk,
- prepisovanje dnevnika transakcij iz pomnilnika na disk,
- ravnanje s smrtnimi objemi. [15]

Načini priključitve

Odjemalčeve aplikacije, ki se povezujejo na tradicionalne diskovne podatkovne baze, običajno uporabljajo TCP/IP mehanizem za komunikacijo s strežnikom. Pri TimesTen se aplikacije, ki se nahajajo na istem strežniku kot podatkovna baza lahko direktno povežejo na »pomnilniško sliko« podatkovne baze z uporabo TimesTen direktnega gonilnika, kar odpravlja potrebo po kakršni koli medprocesni komunikaciji in zagotavlja izredno hitro delovanje. Če se aplikacija nahaja na oddaljenem strežniku, se lahko le ta z bazo podatkov poveže tudi preko tradicionalnega "odjemalec – strežnik" dostopa do podatkov. [15]

4.2.2. Razpoložljivost in celovitost podatkov

Kontrolne točke in dnevnik transakcij

Vsi podatki v TimesTen so v glavnem pomnilniku, vendar pa TimesTen uporablja obstojni pomnilnik, npr. trdi disk, za obstojnost in obnavljanje podatkovne baze. Vse transakcijske spremembe podatkov so shranjene na trdem disku v obliki dnevnikov transakcij. Poleg tega TimesTen opravlja tudi posnetke baze podatkov v pomnilniku (checkpoint – kontrolne točke) in jih shranjuje na disk. Kombinacija kontrolnih točk in dnevniških datotek transakcij omogoča obnovljivost v primeru nesreče.

Privzeto TimesTen deluje v netrajnem potrditvenem načinu (ang. non-durable commit mode). V tem načinu se operacija "commit" pojavlja izključno v glavnem pomnilniku, beleženje dnevnika transakcij na disku pa se izvaja asinhrono z operacijo "commit". Ta zagotavlja zelo hitre odzivne čase in zelo visoko prepustnost podatkov. Zaradi tega lahko v primeru okvare sistema pride do izgube manjše količine podatkov. Trajen "commit" način (sinhroni commit) je tudi zagotovljen, preprečuje kakršnokoli možnost izgube podatkov na račun zmanjšane zmogljivosti. Druga možnost za varovanje podatkov in visoko dostopnost le teh, je uporaba TimesTen replikacije. [15]

Replikacija

TimesTen mehanizem za replikacijo omogča izdelavo zmogljivih sistemov s pošiljanjem posodobitev podatkovne baze med dvema ali več gostitelji. Z replikacijo glavni gostitelj pošlje posodobitve enemu ali več naročniškim gostiteljem. TimesTen priporoča konfiguracijo "active-stanbay" za najvišjo zmogljivost. "Active-standby" je sestavljen iz dveh glavnih

podatkovnih baz, aktivne in čakajoče. Poleg aktivnih in čakajočih lahko imamo tudi več naročniških (ang. subscriber) podatkovnih baz, ki imajo vlogo kopij podatkov v primeru nesreče. [15]

4.3. IBM SolidDB

Uporabil sem IBM SolidDB podatkovno bazo v pomnilniku verzije 6.5.

SolidDB je bila prvič omenjena konec leta 2007 pod okriljem IBM. Je relacijska podatkovna baza v pomnilniku. IBM SolidDB se uporablja po vsem svetu, zaradi njene hitrosti in dostopnosti. Uporablja se v telekomunikacijskih omrežjih, poslovnih aplikacijah ter vgrajenih sistemih. Vodilna podjetja kot so Cisco, HP, Alcatel, Nokia in Siemens za svoje aplikacije uporabljajo SolidDB. [5]

Celotna podatkovna baza se hrani v glavnem pomnilniku namesto na disku, zaradi česar do podatkov dostopamo veliko hitreje. Ima podatkovne strukture in metode dostopa do podatkov posebej zasnovane za shranjevanje, iskanje in obdelavo podatkov v glavnem pomnilniku. Kot rezultat tega z lahkoto prekaša baze na disku, četudi imajo te vse podatke prednaložene v glavnem pomnilniku (ang. cached in memory).

Nekatere podatkovne baze imajo nizko latenco, vendar ne zmorejo velikega števila transakcij ali sočasnih sej. IBM SolidDB omogoča pretok več deset do sto tisoč transakcij na sekundo, odzivni časi (latenca) pa se merijo v mikrosekundah. [17]

4.3.1. Nadzorne točke in trajnost: poti do hitrosti

IBM SolidDB uporablja številne dodatne mehanizme za pospešitev obdelave podatkovne baze. Ena izmed njih je metoda nadzornih točk, ki proizvede varnostni posnetek (skladni zapis) brez ustavitve izvajanja normalnih transakcij. Skladni zapis omogoča ponovni zagon podatkovne baze samo iz kontrolnih točk. Druge podatkovne baze običajno tega ne omogočajo, ampak morajo biti za preračun skladnega zapisa (ang. consistent state) uporabljene datoteke dnevnika transakcij.

SolidDB omogoča dodeljevanje slik vrstic in slik senc vrstic (različne verzije istih vrstic). Samo tiste slike, ki se ujemajo z doslednimi (konsistentnimi) zapisi, se zapišejo v datoteko kontrolnih točk. Sence vrstic omogočajo, da se trenutno izvajajoče transakcije izvajajo nemoteno v času.

Druga uporabljena tehnika je uporaba asinhronega potrjevanja transakcij. Ta zagotavlja hitrejše delovanje, vendar pa lahko pride do izgube podatkov v primeru okvare sistema. [17]

Replikacija

a) Izmenjava informacij celotnega sistema

S SolidDB napredno replikacijo ima lahko vsak strežnik v sistemu svojo lokalno kopijo podatkov. Uporabnikom ni potrebno zagotoviti spletnega dostopa do centralnih virov za upravljanje s podatki. Poleg tega lahko vsaka kopija podatkovne baze z uporabo napredne replikacije služi svojemu namenu. Na primer, ena kopija je lahko namenjena za podporo poročanja ali za aplikacije poročanja, medtem ko je lahko druga namenjena aplikacijam za obdelavo spletnih transakcij.

b) Celovitost podatkov

V sistemu z več podatkovnimi bazami, kjer lahko pride do posodobitve na več podatkovnih bazah, je ohranjanje celovitosti podatkov velik izziv. SolidDB napredna replikacija omogoča razvijalcu aplikacij, da sam upravlja z zmogljivostmi potrjevanja transakcij in s tem učinkovitostjo replikacije.

c) Visoka zmogljivost in prilagodljivost

Sistem napredne replikacije uporabnikom omogoča, da prilagodijo proces sinhronizacije, za povečanje zmogljivosti. Na primer, prenos velikih količin podatkov preko omrežja le kadar je pasovna širina optimalna. [4]

4.4. SQLite

Pri diplomski nalogi sem uporabil SQLite verzijo 3.7.17.

SQLite je relacijska podatkovna baza in je odprtokodni SUPB, realiziran v programskem jeziku C. V nasprotju z drugimi SUPB-ji, SQLite ni ločen proces, do katerega odjemalec dostopa preko aplikacije, ampak je del le te. SQLite je priljubljena kot vgrajena podatkovna baza, za lokalno shranjevanje aplikacij, kot so spletni brskalniki. SQLite je prav tako prilagodljiva veliko programskim jezikom.

SQLite je leta 2000 zasnoval D. Richard Hipp. Hipp je podatkovno bazo razvil z namenom, da se program izvaja brez potrebe po namestitvi SUPB-ja ali zahteve po skrbniku podatkovne baze.

SQLite je prioritetno podatkovna baza na disku, v določenih okoliščinah pa jo lahko shranimo tudi v pomnilnik. Z ukazom "jdbc:sqlite::memory:", ko kreiramo povezavo, povzročimo, da se podatkovna baza shrani v pomnilnik in se na disku ne kreira nobena datoteka. Namesto

tega je nova podatkovna baza kreirana zgolj v RAM-u. Ob vsaki uporabi besede ":memory:" se ustvari nova neodvisna podatkovna baza. Zaradi svoje majhnosti, je SQLite zelo primerna za uporabo v vgrajenih sistemih. [14]

Šibkost te podatkovne baze je, da se ob prekinitvi povezave cela baza izbriše iz pomnilnika. Torej ne potrebuje mehanizmov, dnevnika sprememb, dnevnika transakcij in tudi ne posnetkov podatkovne baze. Prav zaradi tega je ta baza zelo hitra ob polnjenju podatkov vanjo.

Uporaba

a) Spletni brskalniki:

- SQLite uporabljata Mozilla Firefox in Mozilla Tunderbird za shranjevanje vrste konfiguracijskih podatkov (zaznamki, piškotki, stiki), ponujata tudi dodatek za upravljanje SQLite podatkovne baze;
- Google Chrome brskalnik;
- Opera brskalnik prav tako za upravljanje WebSQL baz uporablja SQLite 3.7.9.

b) Aplikacije:

- Skype je aplikacija, ki uporablja SQLite;
- Adobe sistemi uporabljajo SQLite kot svoj format datoteke v Photoshop Lightroom, kot standardno bazo v Adobe AIR in znotraj Adobe Reader-ja.

c) Operacijski sistemi:

- Blackberry-jev OS 10;
- Microsoft Windows Phone 8;
- Applovi OS;
- Simbian OS;
- Googlov Android. [14]

5. TESTNO OKOLJE

5.1. Strojna oprema

Podatkovne baze v pomnilniku zasedejo veliko glavnega pomnilnika, če jih želimo v celoti hraniti v njem. Vendar pa le tako lahko dosežemo želeno hitrost delovanja. Pri običajnih diskovnih podatkovnih bazah navadno ni teh težav, saj s prostorom na disku nismo toliko omejeni.

Vse teste sem izvedel na svojem računalniku, saj je za izbrane podatkovne baze zadostovala lastna strojna oprema. Testni računalnik je imel sledeče karakteristike:

- Procesor i7 - 2670QM, 2.20 GHz,
- 8 GB RAM,
- 500 GB SATA 7200 rpm,
- Windows 7, 64 bit.

5.2. Programska oprema

Za testiranje sem uporabil programsko orodje Eclipse, v katerem sem v programskem jeziku Java izdelal skripte za testiranje vseh izbranih podatkovnih baz. Testi so zajemali branje, pisanje, posodobitve ter branje in posodobitve v razmerju 1:1. Uporabljal sem tudi konzolo (CMD), za dostop do podatkovnih baz in preverjanje stanja le teh.

Programska oprema je zajemala namestitev MySQL podatkovne baze verzije 5.5.32, Apache strežnika verzije 2.4, Oraclovega TimesTen 11.2.2 strežnika, IBM-ovega SolidDB 6.5 strežnika ter SQLite verzije 3.7.17.

5.2.1. Namestitev podatkovnih baz

Namestitev MySQL podatkovne baze, verzije 5.5.32, je bila zelo enostavna, prav tako tudi namestitev Apache strežnika. Na bazo sem se povezal preko javanskega razreda DriverManager v Eclipsu.

Po namestitvi Oraclove TimesTen podatkovne baze v pomnilniku sem moral kreirati DSN (Data Source Names), preko katerega sem dostopal do baze podatkov. DSN vsebuje atribute, ki določajo lastnosti kreiranja ali dostopa do PB. Pri operacijskem sistemu Windows je to zelo enostavno: Nadzorna plošča -> skrbniška orodja -> viri podatkov (ODBC) za zagon ODBC Administratorja. Izberemo zavihek za sistemski DSN, iz seznama izberemo pravkar nameščeni TimesTen DataManager 11.2.1. Nato izberemo ime povezave, podatkovno bazo, pot in njeno ime, mapo na disku, kamor se bodo zapisovali transakcijski dnevniki, kodni

nabor znakov ter "permanent" in "temporary data size". Velikost podatkovne baze je vsota obeh "permanent" in "temporary data size".

a) PermSize
PermSize označuje velikost podatkovne baze v trajnem pomnilniku. Ob prvi povezavi lahko povečamo velikost PermSize, ne moremo je pa zmanjšati. Privzeta velikost je 32 MB.

b) TempSize
TempSize predstavlja celotno količino pomnilnika v MB, ki predstavlja začasno podatkovno bazo v pomnilniku. PermSize nima vnaprej določene vrednosti. Nastavitev ostane nedoločena, njena vrednost se določi iz PermSize po naslednjem pravilu:

Če je PermSize < 64MB, TempSize= 32 MB + (PermSize / 4 MB)

v nasprotnem primeru pa:

Tempsize = 40 MB + (PermSize / 8 MB)

TimesTen zaokroži vrednost navzgor najbližjemu MB. Ko je začasna (TempSize) podatkovna particija ustvarjena, se vsakič, ko je naložena nova PB TempSize, poveča ali zmanjša. Minimalni TempSize je 32 MB. [13]

Pri IBM SolidDB je bila namestitev zelo enostavna, saj je namestitveni program sam poskrbel za vse, tako da sem po namestitvi lahko takoj pričel z uporabo.

SQLite je samo začasna podatkovna baza v pomnilniku, ki se s klicem metode"connect.close();", torej z zaprtjem povezave, izbriše. SQLite ne hrani na disk podatkov, tako kot ostali dve bazi. Zato sem se nanjo povezal z ukazom "DriverManager.getConnection ("jdbc:sqlite::memory:");". Z besedo "::memory" sem povzročil kreacijo podatkovne baze v pomnilniku. Baza se kreira ob vsaki povezavi ponovno, zato ne potrebujemo imena baze, na katero se povezujemo, prav tako ne uporabniškega imena in gesla.

5.2.2. Testno orodje (Eclipse)

Testne programe sem pisal v programskem jeziku Java, v razvojnem okolju Eclipse. Eclipse ima osnovno delovno površino, v kateri pišemo programsko kodo ter razširljive vmesnike za prilagajanje okolju. Večinoma je realiziran v Javi in namenjen predvsem za razvoj Javanskih aplikacij. [3]

Izvajanje testov v Eclipsu

S pomočjo uvoza paketa sql (import java.sql.*;), sem lahko izvajal jezik SQL znotraj Jave. Za vsako posamezno podatkovno bazo sem v Eclipsovo knjižnico uvozil JDBC gonilnik, ki je omogočal dostop do podatkovne baze iz Eclipsa.

Kreiranaje tabel, polnjenje, poizvedbe ter posodobitve sem realiziral v programskem jeziku Java. Za izvajanje SQL stavkov sem uporabil vmesnik Statement, ki generira objekt ResultSet. To je tabela s podatki, ki predstavlja množico baze. Za merjenje časa posameznih scenarijev sem uporabil Javansko funkcijo "System.nanoTime();". Ta vrne trenutno vrednost najbolj natančne sistemske ure v nanosekundah. Vse scenarije sem pognal s pomočjo "for" zanke tisoč krat in izmeril čas.

6. PODATKOVNA BAZA NORTHWIND IN TESTNI SCENARIJI

6.1. Podatkovna baza Northwind

Za izvajanje testov sem uporabil bazo Northwnd, ki ima 13 tabel. Vse tabele imajo skupaj 975788 vrstic. Tabele sem izdelal na osnovi konceptualnega modela podatkovne baze, z uporabo jezika SQL, za vsako bazo posebej. Vsaka baza ima namreč različne podatkovne tipe in SQL notacijo. Testni podatki so bili generirani avtomatično z uporabo generatorja podatkov, kjer je bilo potrebno poskrbeti tudi za njihovo smiselnost in pravilnost. Skripto, ki podatke popravi glede na format, ki ga podatkovna baza podpira, sem napisal sam. Razlika je bila predvsem v formatu datuma, saj ima vsaka baza svoj format zapisa za datum.

Slika konceptualnega modela Northwind podatkovne baze je v prilogi [2].

Za vse štiri podatkovne baze sem opredelil štiri testne scenarije, da bi ugotovil, katera izmed njih se v posameznem scenariju najbolje odreže. Ker so si podatkovne baze različne, tako po arhitekturi, kot tudi po vrsti pomnilnika, na katerem delujejo, sem pričakoval zanimive rezultate. Za vsak testni scenarij sem teste izvedel pet krat. Nato sem rezultate seštel in jih delil s pet ter dobil povprečni čas, ki so ga baze potrebovale za določen scenarij. Teste sem izvedel najprej brez indeksov, nato pa sem iste testne scenarije izvedel še z indeksi.

6.2. Scenarij A

Pri scenariju A gre za polnjenje podatkovnih baz s podatki. V tabele vseh podatkovnih baz sem vstavil 975788 vrstic in meril čas, ki so ga baze potrebovale za vstavljanje te količine podatkov.

6.3. Scenarij B

Pri scenariju B sem iz baz bral podatke s "Select" stavki. Pri scenariju B prevladuje operacija iskanja in hitrost streženja podatkov podatkovne baze. Testni scenarij sem realiziral tako, da sem najprej izdelal osem poizvedb, ki sem jih nato s pomočjo funkcije "random();" naključno izvajal v zanki s tisoč ponovitvami. Funkcijo "random();" sem uporabil zato, da bi se poizvedbe čim manj zaporedno ponavljale in da bi bilo tako testiranje čim bolj verodostojno. Najprej sem teste izvedel brez indeksov, nato pa z njimi.

6.4. Scenarij C

Scenarij C je namenjen posodabljanju podatkov. Pri tem scenariju baza izvaja operaciji iskanja podatkov in ažuriranja le teh. Tudi pri tem scenariju sem pet posodobitev izvajal naključno v zanki s tisoč ponovitvami najprej brez, nato pa z indeksi. Tu sem beležil tudi stanje "update" stavkov tako, da ko je že uporabljen stavek ponovno prišel na vrsto, se je izvedel njemu nasprotni stavek. Tako sem dosegel, da se je baza res vedno posodabljala.

6.5. Scenarij D

Scenarij D posodablja podatkovne baze. Izmenično se izvaja tisoč "update" in tisoč "select" stavkov. Pri tem scenariju imamo 50% "iskanja – pisanja" in 50% "iskanja – branja". Pri scenariju D sem hotel ugotoviti ali kombinacija poizvedb in posodobitev izmenično prinese enake, boljše ali slabše rezultate, kot če bi scenarij B in C sešteli skupaj in dobili skupen čas. Prav tako sem tudi tukaj najprej izvedel scenarij brez, nato pa z indeksi ter primerjal dobljene rezultate.

7. ANALIZA REZULTATOV

7.1. Scenarij A

Vnos podatkov v podatkovno bazo nam prikazuje, kako hitro lahko izbrane podatkovne baze shranjujejo podatke. V tabeli so zbrani podatki o hitrosti vnosa podatkov v podatkovne baze. Izračunal sem povprečen čas izvajanja enega vnosa v podatkovno bazo. Vse podatkovne baze so test opravile uspešno. Najhitreje je podatke zapisala SQLite podatkovna baza v pomnilniku. Najpočasnejša je pričakovano bila MySQL diskovna podatkovna baza. Pri pomnilniških podatkovnih bazah je bila IBM-ova podatkovna baza daleč najpočasnejša. Bilo je jasno videti, da se podatkovna baza hrani v pomnilniku, saj je bil disk pri vseh treh bazah neuporabljen. Pri MySQL podatkovni bazi je disk ves čas izvajanja testov bil obremenjen 100%.

SQLite podatkovna baza je baza, ki se ob prekinitvi povezave izbriše iz pomnilnika. Podatki se ne zapišejo na disk, kot se to zgodi pri ostalih dveh pomnilniških bazah. To je tudi razlog, da ta podatkovna baza potrebuje tako malo časa za shranjevanje podatkov. Ostali dve bazi morata poleg shranjevanja podatkov hraniti še veliko podatkov o stanju baze in dnevnike transakcij. Vse to se shranjuje na disk, da v primeru izpada električne energije ne pride do popolne izgube podatkov, kot se to zgodi pri SQLite podatkovni bazi.

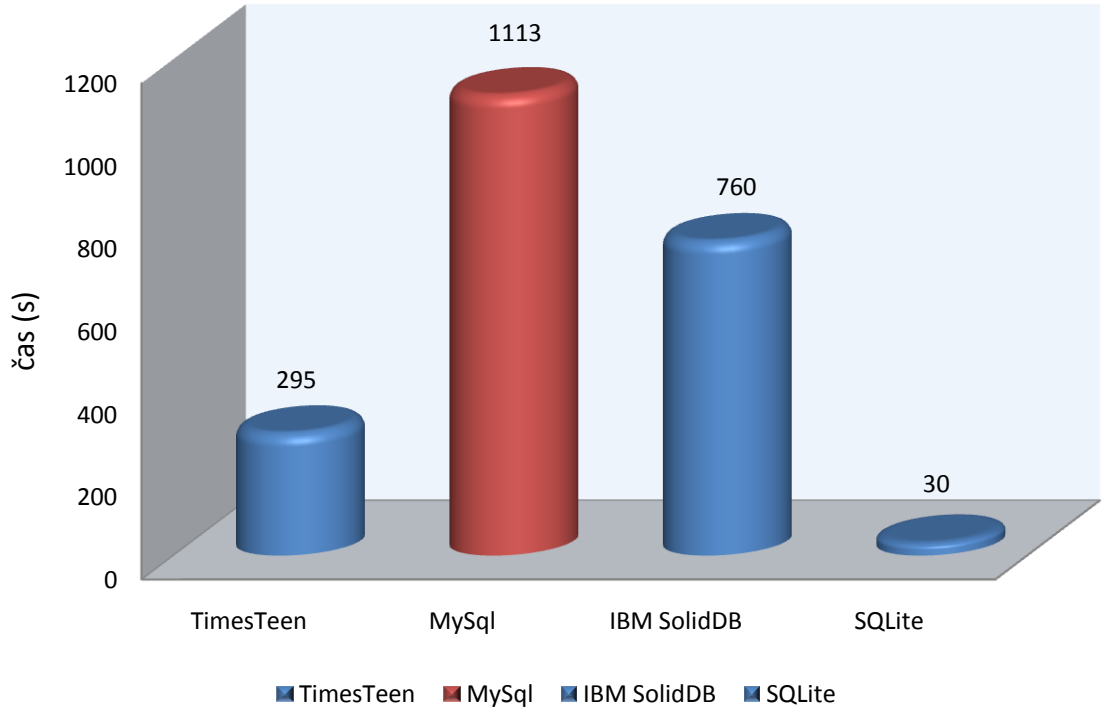
Posamezne podatkovne baze so ob kreiranju indeksov nad vsemi primarnimi ključi v vseh tabelah dosegle nekoliko boljše čase. Najbolj je izboljšava vidna pri IBM SolidDB podatkovni bazi, kjer se je čas polnjenja baze izboljšal kar za 250 sekund. Tudi MySQL podatkovna baza se je napolnila hitreje za 93 sekund. SQLite PB je po kreaciji indeksov potrebovala v povprečju 2 sekundi več.

Tabela 1: Statistika scenarija A brez indeksov.

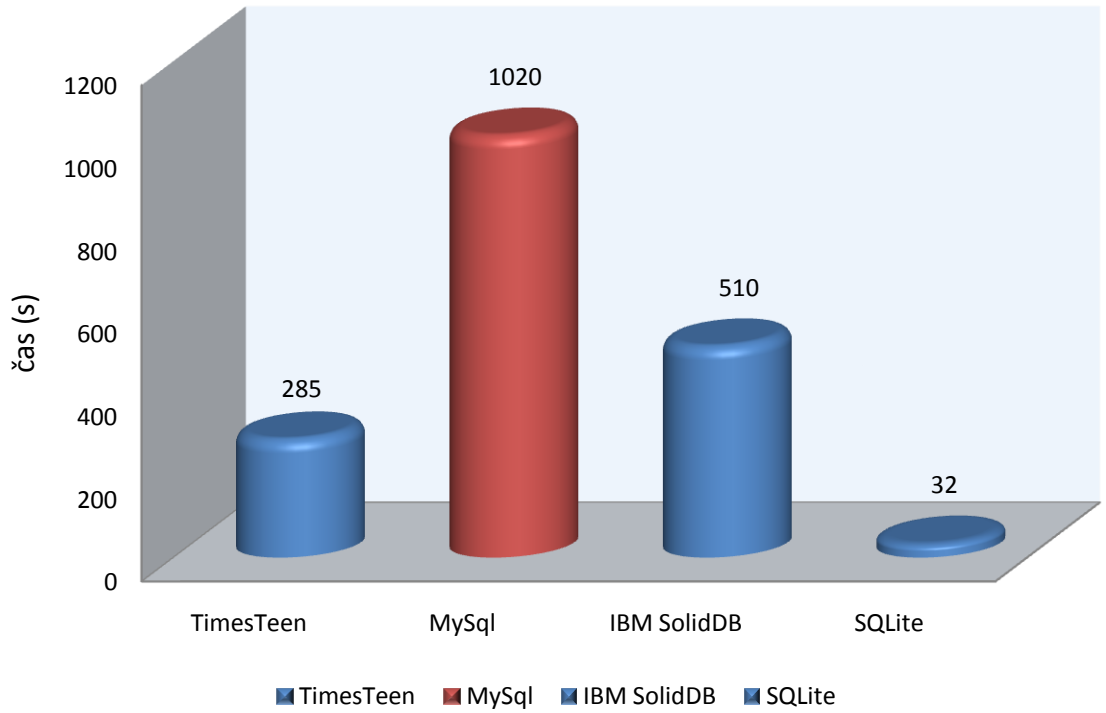
	čas izvajanja (s)	**povprečen čas vstavljanja ene vrstice (ms)**
MySQL	1113	1,141
TimesTen	295	0,302
IBM SolidDB	760	0,778
SQLite	30	0,030

Tabela 2: Statistika scenarija A z indeksi.

	čas izvajanja (s)	**povprečen čas vstavljanja ene vrstice (ms)**
MySQL	1020	1,045
TimesTen	285	0,292
IBM SolidDB	510	0,522
SQLite	32	0,033



Slika 2: Čas polnjenja brez indeksov.



Slika 3: Čas polnjenja z indeksi.

7.2. Scenarij B

Najprej sem vse podatkovne baze obremenil tako, da sem izvedel zanko s 1000 ponovitvami in osmimi različnimi poizvedbami, ki so se v vsakem krogu zanke izbirale naključno.

Največ težav je imela MySQL podatkovna baza, saj je pri taki količini poizvedb Eclipsu zmanjkoval prostor v kopici. Spremenljivka "connect" je zadrževala podatke v kopici. Podatki se niso praznili iz kopice, zato je program javil napako (Java heap space). Napako sem odpravil tako, da sem ustavil čas merjenja, prekinil povezavo na podatkovno bazo "connect.colse();", da se je kopica izpraznila. Nato sem ponovno vzpostavil povezavo z bazo, nadaljeval merjenje časa in naprej izvajal poizvedbe. Zaradi prekinitve merjenja časa nisem vplival na rezultate meritev. Vse podatkovne baze so ta scenarij izvedle uspešno.

Pri tem scenariju je bila uporaba indeksov zelo vidna. To je pričakovano, saj so indeksi namenjeni prav pohitritvi iskanja podatkov v podatkovni bazi.

TimesTen podatkovna baza je pri tem scenariju dosegala daleč najboljše čase med vsemi. Spet je bila pričakovano najpočasnejša MySQL podatkovna baza, in sicer kar za 10 krat v primerjavi z TimesTen podatkovno bazo. S kreacijo indeksov nad posameznimi atributi v tabelah, ki so bile pri tem scenariju uporabljene, je prineslo vidno izboljšanje rezultatov. Pri TimesTen bazi je bila pohitritev skoraj 6 krat, pri MySQL pa skoraj 2 krat. Pri MySQL bazi je bilo izboljšanje 592 sekund pri TimesTen pa 115 sekund. Pri ostalih dveh ni bilo tako velikih sprememb v času. Pri SolidDB je znašala sprememba 44 sekund, pri SQLite pa 91 sekund, vendar glede na razmerje obeh časov razlika ni bila tako očitna kot pri prvih dveh.

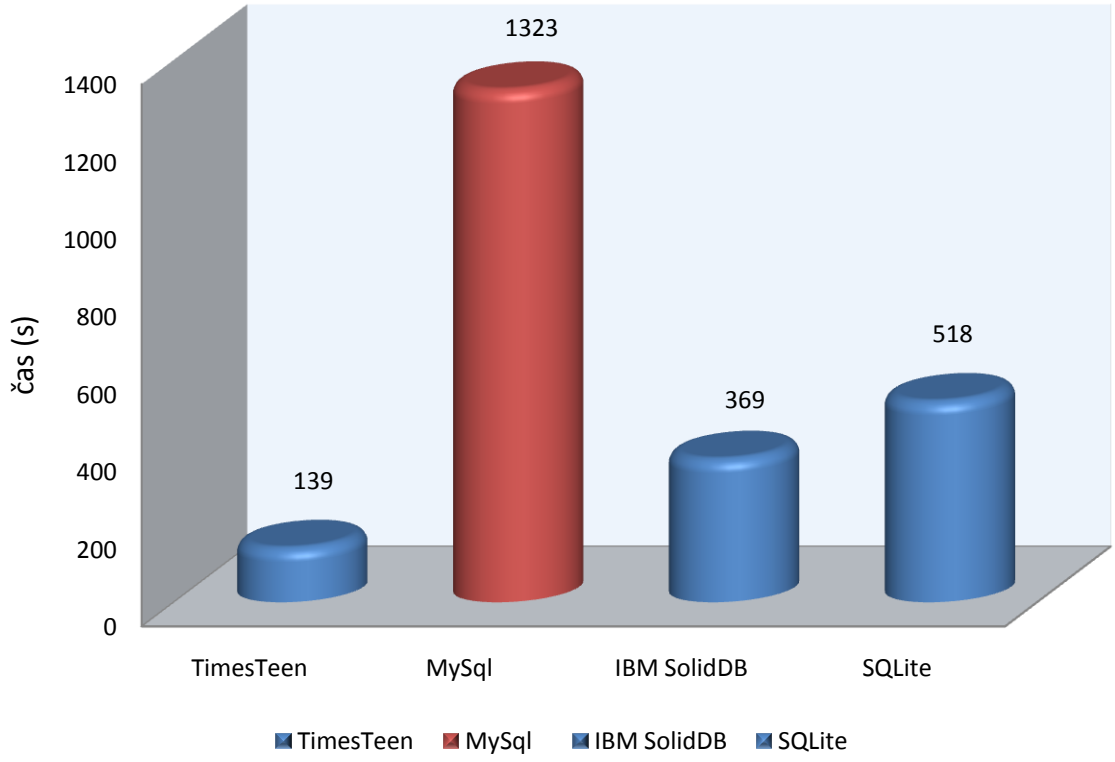
Indeksi, ki sem jih definiral v okviru tega scenarija, so navedeni v prilogi [3].

Tabela 3: Statistika scenarija B brez indeksov.

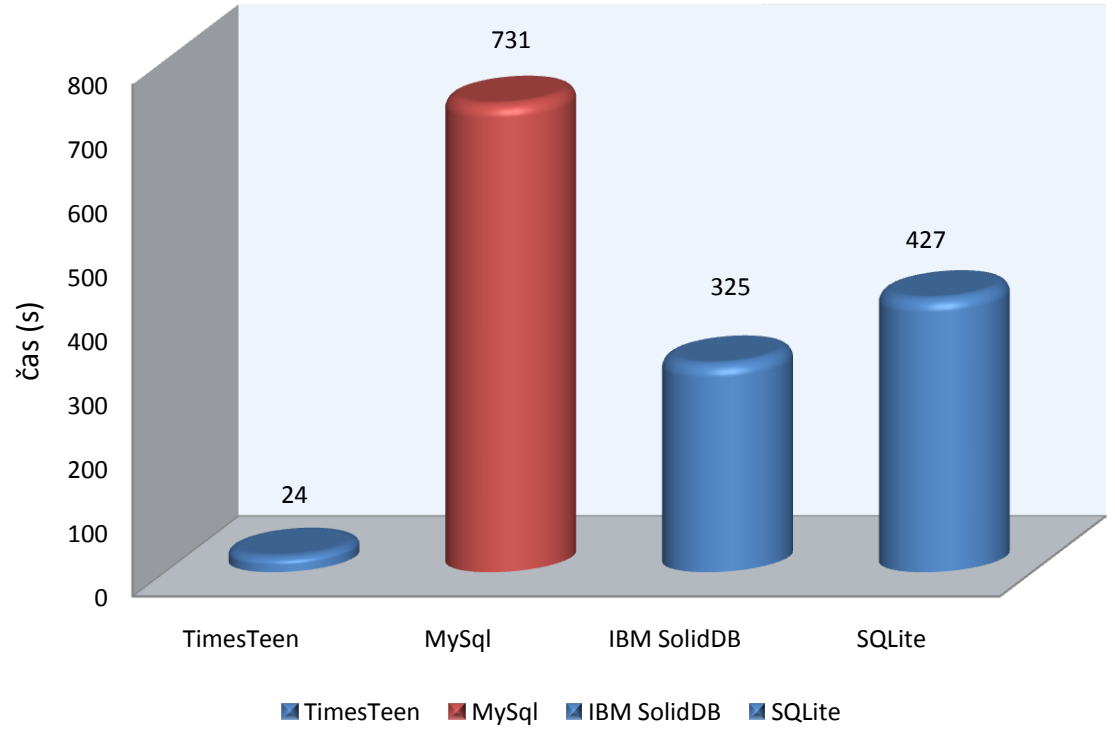
	čas izvajanja (s)	**povprečen čas izvajanja ene poizvedbe (s)**
MySQL	1323	1,323
TimesTen	139	0,139
IBM SolidDB	369	0,369
SQLite	518	0,518

Tabela 4: Statistika scenarija B z indeksi.

	čas izvajanja (s)	**povprečen čas izvajanja ene poizvedbe(s)**
MySQL	731	0,731
TimesTen	24	0,024
IBM SolidDB	325	0,325
SQLite	427	0,427



Slika 4: Čas branja brez indeksov.



Slika 5: Čas branja z indeksi.

7.3. Scenarij C

Tudi tu sem z zanko s 1000 ponovitvami in petimi različnimi posodobitvami, ki so se naključno izvajale, poskušal podatkovne baze kar najbolje testirati, z namenom ažuriranja podatkov. Vse baze so scenarij uspešno izvedle.

Najprej sem teste izvedel brez indeksov, nato pa z njimi. Tudi pri tem scenariju sem pričakoval očitno izboljšanje rezultatov z uporabo indeksov, saj imamo pri "update" stavkih tudi operacijo iskanja, za kar bi morali indeksi veliko doprinesti. Za razliko od iskanja je tu bila opaznejša razlika samo pri diskovni MySQL podatkovni bazi, ki je z indeksi pohitrila izvajanje za skoraj 3 krat. Tudi pri ostalih bazah so indeksi prinesli izboljšanje rezultatov, vendar ne tako očitno.

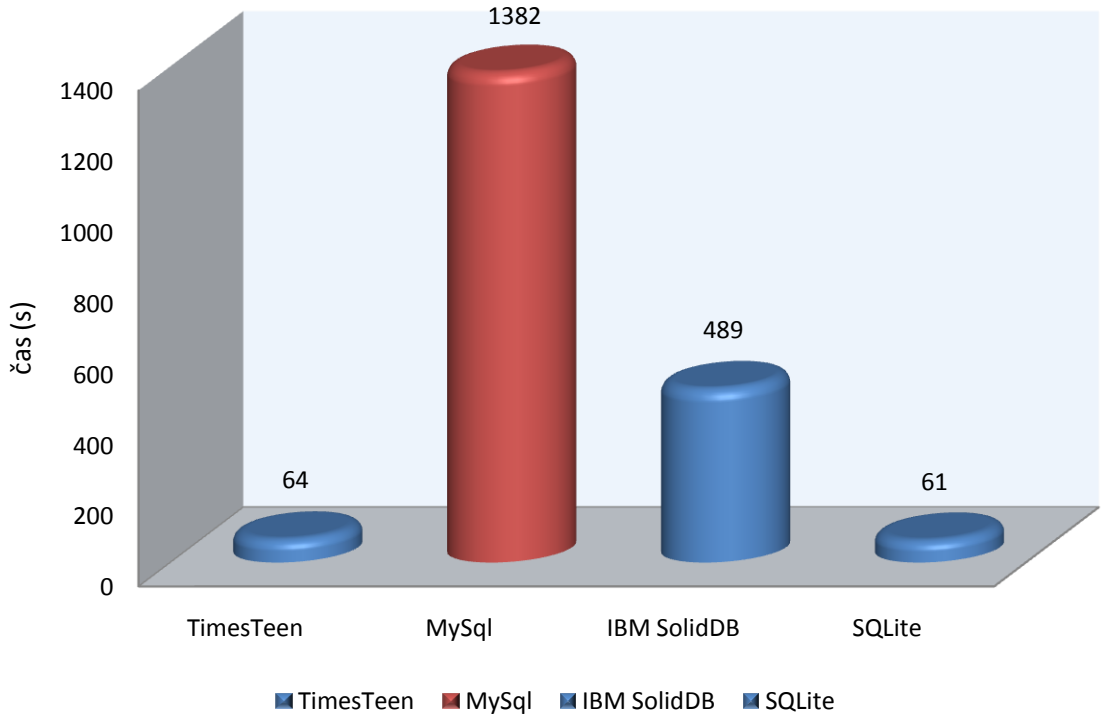
Indeksi, ki sem jih definiral v okviru tega scenarija, so navedeni v prilogi [4].

Tabela 5: Statistika scenarija C brez indeksov.

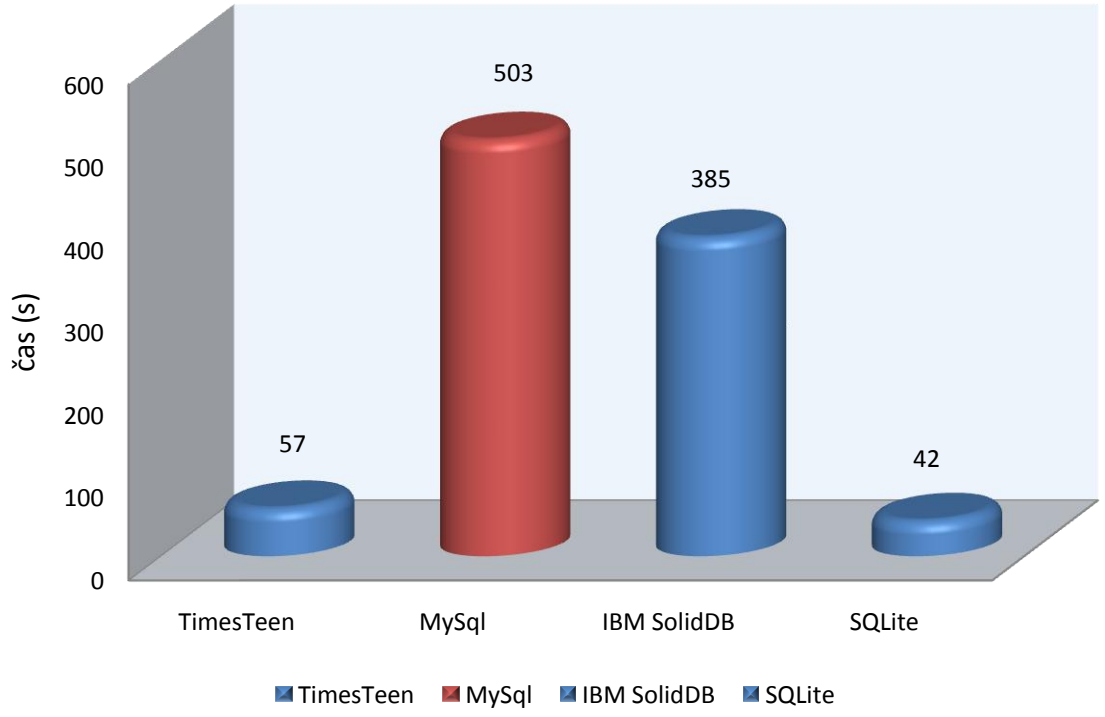
	čas izvajanja (s)	**povprečen čas izvajanja enega "update" stavka (s)**
MySQL	1382	1,382
TimesTen	64	0,064
IBM SolidDB	489	0,489
SQLite	61	0,061

Tabela 6: Statistika scenarija C z indeksi.

	čas izvajanja (s)	**povprečen čas izvajanja enega "update" stavka (s)**
MySQL	503	0,503
TimesTen	57	0,057
IBM SolidDB	385	0,385
SQLite	42	0,042



Slika 6: Čas ažuriranja brez indeksov.



Slika 7: Čas ažuriranja z indeksi.

7.4. Scenarij D

Pri scenariju D sem se odločil, da bom poskusil z izvajanjem poizvedb in posodobitev mešano. Poizvedbe in posodobitve so se izvajale izmenično, torej najprej ena poizvedba, nato ena posodobitev. Tako posodobitve kot poizvedbe so se izvajale v zanki do 1000, torej je bilo število operacij 2000. Zato sem pri izračunu časa izvedbe ene operacije čas izvajanja delil z 2000 in dobil povprečni čas izvedbe ene operacije.

Ker sta bili pri tem scenariju ponovno operaciji "iskanja – branja" in "iskanja – ažuriranja" podatkov, sem pričakoval, da se bo ponovno najslabše odrezala MySQL podatkovna baza, saj je bila že v prejšnjih scenarijih najslabša. Ponovno so rezultati zelo različni. TimesTen potrebuje skoraj 4 krat manj časa od svojih dveh sorodnic ter skoraj 11 krat manj kot MySQL diskovna baza. Ostali dve podatkovni bazi v pomnilniku, sta z rezultati dokaj izenačeni. Ob kreaciji indeksov nad atributi, na katerih se izvajajo poizvedbe in nad tistimi, na katerih se izvajajo posodobitve, je bilo največjo razliko opaziti pri TimesTen podatkovni bazi, kar za 3 krat, saj je le ta že pri izvajanju posodobitev bila najbolj pohitrena s strani indeksov. Časovno je bila ponovno najbolj pohitrena MySQL. Čas se ji je izboljšal za 531 sekund. Ostali dve bazi v pomnilniku sta bili z indeksi pohitreni za dobrih 100 sekund.

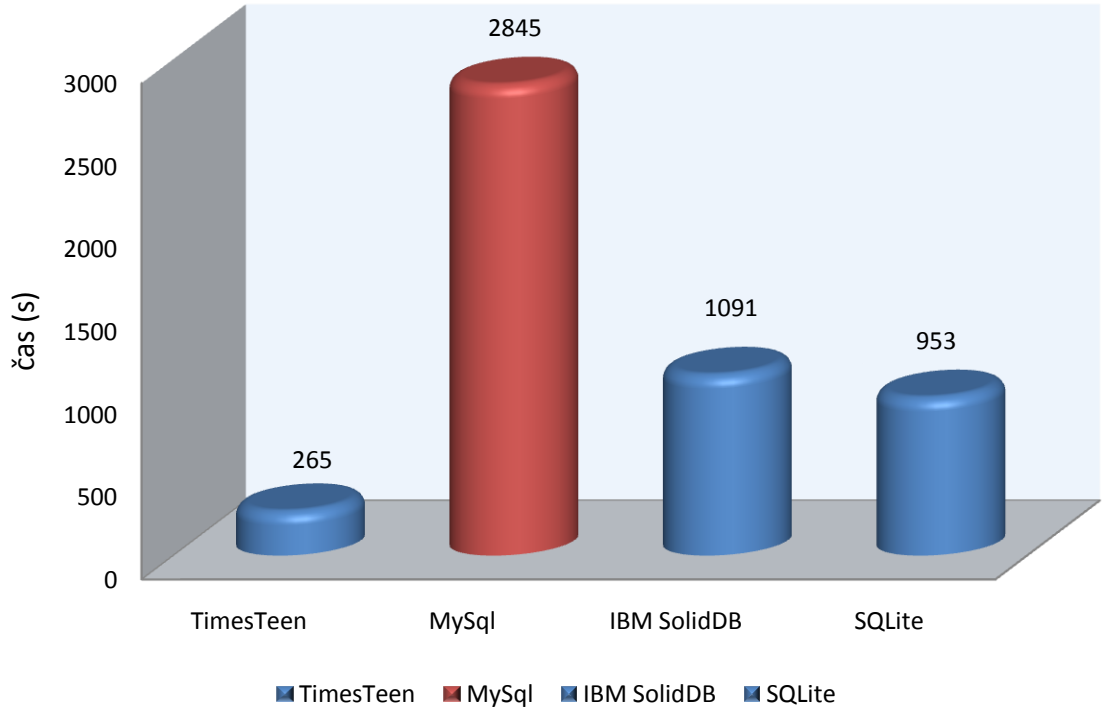
Indeksi, ki sem jih definiral v okviru tega scenarija, so navedeni v prilogah [3, 4].

Tabela 7: Statistika scenarija D brez indeksov.

	čas izvajanja (s)	**povprečen čas izvedbe ene operacije (s)**
MySQL	2845	1,423
TimesTen	265	0,133
IBM SolidDB	1091	0,546
SQLite	953	0,477

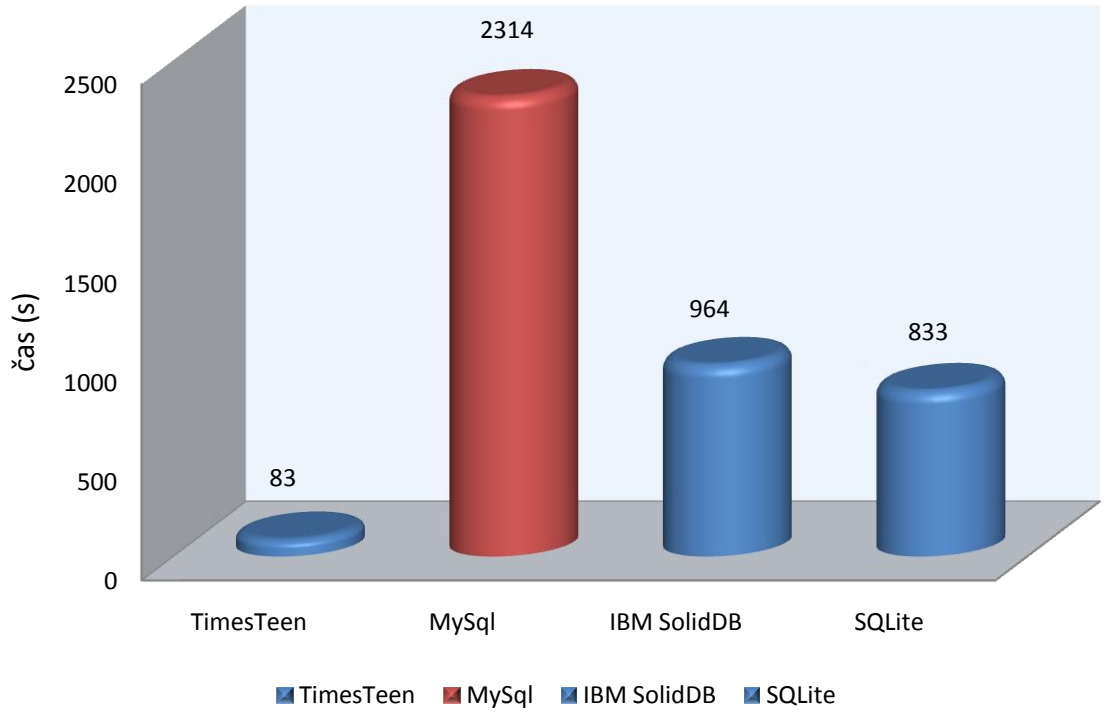
Tabela 8: Statistika scenarija D z indeksi.

	čas izvajanja (s)	**povprečen čas izvedbe ene operacije (s)**
MySQL	2314	1,157
TimesTen	83	0,042
IBM SolidDB	964	0,482
SQLite	833	0,417



Slika 8: Čas scenarija D brez indeksov.

.



Slika 9: Čas scenarija D z indeksi.

8. MOŽNE IZBOLJŠAVE TESTIRANJA

Pri pregledu končnih rezultatov, ki sem jih pridobil pri izvajanju testov, sem ugotovil več stvari, ki bi jih v prihodnje lahko izboljšal.

V bazo sem vstavil 975788 različnih vrstic. Če bi hotel, da bi se razlika med podatkovnimi bazami v pomnilniku in na disku še bolj nazorno pokazala, bi moral vstaviti še več podatkov ter nad njimi izvesti še več operacij, ki bi se izvajale tudi sočasno. Torej bi na podatkovne baze pošiljal več kompleksnejših poizvedovalnih stavkov hkrati.

Vse teste sem izvajal iz orodja Eclipse, ki je nekatere podatkovne baze lahko tudi upočasnil. Mogoče bi bilo bolje, če bi naredil aplikacijo, ki bi se zagnala posebej in bi sama dostopala do podatkovne baze ter ne bi imela vpliva Eclipsa. Eclipse sicer ni imel tako velikega vpliva, saj je nazorno videti, da so rezultati podatkovnih baz v pomnilniku boljši od tistih iz diskovne baze. Programi so se razlikovali le v povezavah na bazo, ostala koda je bila enaka. Boljše rezultate bi dosegel z uporabo SSD diska, predvsem pri podatkovni bazi MySQL, ki je služila kot edina diskovna baza. Boljši pa bi bili tudi rezultati podatkovnih baz v pomnilniku, predvsem zato, ker le te shranjujejo transakcijske dnevnike in posnetke podatkovnih baz na disk za obnovo v primeru nesreče.

Ti štirje scenariji, ki sem jih izbral, predstavljajo le del najosnovnejših operacij, ki se izvajajo na bazah podatkov v produkcijskih sistemih. V okviru teh se izvaja večje število kompleksnejših operacij nad podatki, ki se nahajajo v različnih tabelah.

9. SKLEPNE UGOTOVITVE

Primerjava relacijskih podatkovnih baz na disku in v pomnilniku se je izkazala kot zelo zanimiva. Naloga mi je bila všeč predvsem zato, ker je bilo zame to nekaj novega in še neraziskanega. Ker imajo obe skupini podatkovnih baz, diskovne in pomnilniške, različne namene uporabe ter se močno razlikujejo po ceni, je dobro vedeti kje katere uporabiti. S pomočjo orodja Eclipse in JDBC gonilnikov za podatkovne baze sem z lahkoto dostopal do njih in nad njimi izvajal različne testne scenarije.

Rezultati testov kažejo, da so podatkovne baze v pomnilniku v vseh testnih scenarijih prekašale podatkovno bazo na disku, kar je logično in pričakovano, saj je glavni pomnilnik mnogo hitrejši od diska. Pokazala se je velika razlika med bazami v pomnilniku. Vsaka izmed baz se je za različne scenarije odrezala drugače. Pri nekaterih scenarijih so se izkazale za odlične pri drugih pa slabše. V povprečju izvajanja vseh testov je bila najboljša Oraclova TimesTen podatkovna baza v pomnilniku, najslabša pa MySQL na disku. Zanimivo bi bilo testirati, kako se baze odzovejo v realnih situacijah, kjer je obremenitev še večja. Predvidevam, da bi se v tem primeru pokazala še mnogo večja razlika med pomnilniškimi in diskovnimi bazami.

Podatkovne baze v pomnilniku so prisotne že več kot 10 let. V množični uporabi pa so šele zadnjih nekaj let, ker se je glavni pomnilnik pocenil. Kljub temu podatkovne baze v pomnilniku ne morejo delovati brez obstojnega medija. Na disk shranjujejo posnetke baz v pomnilniku in beležke transakcij tako, da brez diska, oziroma obstojnega pomnilnika, tudi zanesljivih podatkovnih baz v pomnilniku ne moremo realizirati. Prav tako se v produkcijskih okoljih podatki nikoli ne zapisujejo samo na en strežnik, ampak uporabljalo način replikacije; to je podvajanje podatkov še na drugem strežniku v primeru izpada prvega. Menim, da bi bilo baze zanimivo testirati tudi z načinom replikacije in ugotoviti kako le ta vpliva na testne scenarije.

LITERATURA

[1] Ballard C. et al., *IBM solidDB: Delivering Data with Extreme Speed.* IBM RedBook, 2011, pp. 2-14.

[2] Connolly T. and C. Begg, *Database systems: A practical approach to design, implementation, and management.* Četrta izdaja. Essex: Pearson Education Limited, 2005, pp. 15-175.

SPLETNI VIRI

[3] Eclipse. Dostopno 25.01.2014 na: http://en.wikipedia.org/wiki/Eclipse_%28software%29.

[4] IBM SolidDB advanced replication features (2013). Dostopno 12.01.2014 na: http://publib.boulder.ibm.com/infocenter/soliddb/v6r3/index.jsp?topic=/com.ibm.swg.im.soliddb.replication.doc/doc/smartflow.features.html.

[5] IBM SolidDB-fastest data delivery: In-memory, relatonal database software for extreme speed. Dostopno 12.01.2014 na: http://www-01.ibm.com/software/data/soliddb/.

[6] In-memory database. Dostopno 15.10.2013 na: http://en.wikipedia.org/wiki/In-memory_database.

[7] In-memory database systems - questions and answers. Dostopno 15.10.2013 na: http://www.mcobject.com/in_memory_database.

[8] Mateja K. (2010), Relacijske baze podatkov. Dostopno 25.10.2013 na: http://varovanjepodatkov.blogspot.com/2010/04/relacijske-baze-podatkov.html.

[9] Mediji za hranjenje podatkov: Hierarhija pomnilnikov. Dostopno 15.10.2013 na: http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/PODATKOVNE_BAZE/mediji_za_hranjenje_podatkov.html.

[10] MySQL. Dostopno 11.1.2014 na: http://en.wikipedia.org/wiki/MySQL.

[11] NVDIMM. Dostopno 15.10.2013 na: http://en.wikipedia.org/wiki/NVDIMM.

[12] Oracle Times Ten In-Memory database overview.. Dostopno 11.11.2014 na: http://www.oracle.com/technetwork/products/timesten/overview/timesten-imdb-086887.html.

[13] Oracle TimesTen In-Memory database reference (2012). Dostopno 05.01.2014 na: http://docs.oracle.com/cd/E13085_01/doc/timesten.1121/e13069/attribute.htm#TTREF154.

[14] SQLite. Dostopno 12.01.2014 na: http://en.wikipedia.org/wiki/SQLite.

[15] TimesTen. Dostopno 11.11.2013 na: http://en.wikipedia.org/wiki/TimesTen.

[16] Transaction log. Dostopno 15.10.2013 na: http://en.wikipedia.org/wiki/Transaction_log.

[17] Wolski A. in Hartnell S. (2010), SolidDB and the secret of speed: How the IBM in-memory database redefines high preformance. Dostopno 12.01.2014 na: http://www.ibm.com/developerworks/data/library/dmmag/DBMag_2010_Issue1/DBMag_Issue109_solidDB/.

Kazalo tabel in slik

PRILOGE

Priloga 1

JAVANSKI RAZRED UPORABLJEN ZA TESTE BAZ

Dodatek vsebuje Javanski razred za kreiranje, polnjenje, poizvedovanje in ažuriranje podatkovne baze SQLite. Za vse ostale baze je popolnoma enak program, samo povezave na baze se razlikujejo.

```
package SQLite;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.*;
import javax.swing.JOptionPane;
import org.sqlite.SQLiteDataSource;
import org.sqlite.SQLiteJDBCLoader;
 public class Test {
 public static void connection(){
            try {
                  Class.forName("org.sqlite.JDBC");
            } catch (ClassNotFoundException e) {
                                    e.printStackTrace();
            }
}
//---------Metoda, ki nam vrača poizvedbe glede na random številko, ki je poslana  vanjo.------
public static String VrniPoizvedbo(int a){
String str="";
switch (a) {
              case 1:  str = "SELECT o.O_OrderId, o.O_OrderDate " +
                                 "FROM Orders o, Employees e " +
                                 "WHERE E_LastName='Sizemore' AND E_FirstName='Richard' " +
                                 "AND e.E_EmployeeID=o.E_EmployeeId";
                                 break;

              case 2:  str = "SELECT count(*) " +
                                 "FROM Orders " +
                                 "WHERE O_OrderDate BETWEEN '2005-02-01' AND '2005-02-20'";
                                 break;
```

```
case 3:  str = "SELECT p_productid, sum(OD_quantity) AS vsota " +
                "FROM orderDetails od, orders o " +
                "WHERE o.o_orderid=od.o_orderid " +
                "GROUP BY p_productid " +
                "ORDER BY vsota desc";
                break;


case 4:  str = "SELECT distinct p_productid " +
               "FROM orderdetails";
                break;

case 5:  str = "SELECT count(*) " +
               "FROM products " +
               "WHERE p_productname='Sayer'";
                break;

case 6:  str = "SELECT su_companyname, su_contactname, su_address " +
                "FROM suppliers s, products p " +
                "WHERE p.p_productname='Sayer'  "+
                "AND p.su_supplierid=s.su_supplierid";
               break;

case 7:  str = "SELECT Customers.c_companyname, Orders.o_orderid " +
               "FROM Customers " +
               "LEFT JOIN Orders ON customers.c_customerid=Orders.c_customerid " +
               "ORDER BY Customers.c_companyname";
               break;

case 8:  str = "SELECT Customers.c_companyname, Orders.o_orderid " +
               "FROM Customers " +
               "INNER JOIN Orders ON Customers.c_customerid=Orders.c_customerid " +
               "ORDER BY Customers.c_companyname";
          break;

    }

                return str;
            }

//---Za update stavke, ko je zastavica 0, torej ko se poizvedbe prvič izvajajo-----
public static String VrniUpdateZast0(int a){
String str="";
switch (a) {
case 1:  str = "update orderdetails " +
               "set od_discount=0.2 " +
               "where o_orderid=2";
   break;

case 2:  str = "update orders " +
                "set o_shipname='NewShip' " +
               "where o_shipname='Erna'";
   break;
```

```
                case 3:  str = "update products " +
                               "set p_unitprice=p_unitprice+10 " +
                               "where p_productname='Sayer'";
                    break;


                case 4:  str = "UPDATE customers " +
                               "SET c_contractname='Ferina'" +
                               "WHERE c_contractname='Ferin'";
                    break;

                case 5:  str = "UPDATE products " +
                               "SET p_unitprice=p_unitprice+0.2";
                    break;
}
return str;
}
```

//---Za update stavke, ko je zastavica 1, torej ko se poizvedbe izvajajo z nasprotnimi vrednostmi kot pri zastavica=0, da podatkovni bazi povrnejo nazaj prejšnje stanje in da lahko ponovno izvedemo prvotne poizvedbe -----

```
public static String VrniUpdateZast1(int a){
String str="";
switch (a) {

                case 1:  str = "update orderdetails " +
                               "set od_discount=0.0 " +
                               "where o_orderid=2";
                    break;

                case 2:  str = "update orders " +
                               "set o_shipname='Erna' " +
                               "where o_shipname='NewShip'";
                    break;

                case 3:  str = "update products " +
                               "set p_unitprice=p_unitprice-5 " +
                               "where p_productname='Sayer'";
                    break;


                case 4:  str = "UPDATE customers " +
                               "SET c_contractname='Ferin'" +
                               "WHERE c_contractname='Ferina'";
                    break;


                case 5:  str = "UPDATE products " +
                               "SET p_unitprice=p_unitprice-0.1";
                    break;
 }
return str;
}
```

```
public static void main(String[] args)throws IOException, SQLException {
Connection connect = null;
Statement stmt = null;
Scanner read =null;
String b="";
int nakljucno=0;
String querry="";
long startTime;
long stopTime;
long skupniCas=0;
String insert =null;
PreparedStatement stmt1 = null;
ResultSet rs=null;
String update;

connection(); // klic metode connection
int[] t1 = {0,0,0,0,0}; //tabela stanj uporabljena pri update stavkih, za kontrolo izvajanja
 connect = DriverManager.getConnection("jdbc:sqlite::memory:"); //povezava na bazo - v pomnilniku
stmt = connect.createStatement();

//-------------------------------------------------kreiranje tabel-----------------------------------------------
        stmt.executeUpdate("drop table if exists CATEGORIES;");
        //creating table CATEGORIES
        stmt.executeUpdate("  CREATE TABLE CATEGORIES (" +
                           "  CA_CATEGORYID INT      NOT NULL, " +
                           "  CA_CATEGORYNAME  VARCHAR(300)  NOT NULL, " +
                           "  CA_DESCRIPTION   VARCHAR(300), " +
                           "  CA_PICTURE       VARCHAR(300),  " +
                           "  PRIMARY KEY(CA_CATEGORYID));");

        stmt.executeUpdate("drop table if exists CUSTOMERDEMOGRAPHICS;");
         //creating table CUSTOMERDEMOGRAPHICS
        stmt.executeUpdate("CREATE TABLE CUSTOMERDEMOGRAPHICS(" +
                           "  CD_CUSTOMERTYPEID  INT     NOT NULL," +
                           "  CD_CUSTOMERDESC    VARCHAR(100)," +
                           "  PRIMARY KEY (CD_CUSTOMERTYPEID));");

        stmt.executeUpdate("drop table if exists CUSTOMERS;");
        //creating table CUSTOMERS
        stmt.executeUpdate("CREATE TABLE CUSTOMERS( " +
                           "  C_CUSTOMERID INT          NOT NULL," +
                           "  C_COMPANYNAME VARCHAR(300) NOT NULL," +
                           "  C_CONTRACTNAME  VARCHAR(300)," +
                           "  C_CONTRACTTITLE  VARCHAR(300)," +
                           "  C_ADDRESS       VARCHAR(300)," +
                           "  C_CITY          VARCHAR(300)," +
                           "  C_REGION        VARCHAR(300)," +
                           "  C_POSTALCODE     INT," +
                           "  C_COUNTRY       VARCHAR(300)," +
                           "  C_PHONE         INT," +
                           "  C_FAX           INT, " +
                           "  PRIMARY KEY (C_CUSTOMERID));");

        stmt.executeUpdate("drop table if exists CUSTOMERCUSTOMERDEMOGRAPHICS;");
```

```
        //creating table CUSTOMERCUSTOMERDEMOGRAPHICS
        stmt.executeUpdate("CREATE TABLE CUSTOMERCUSTOMERDEMOGRAPHICS (" +
                           "  C_CUSTOMERID INT    NOT NULL," +
                           "  CD_CUSTOMERTYPEID  INT       NOT NULL, " +
                           "  PRIMARY KEY(C_CUSTOMERID, CD_CUSTOMERTYPEID)," +
                           " FOREIGN KEY (CD_CUSTOMERTYPEID) REFERENCES
CUSTOMERDEMOGRAPHICS (CD_CUSTOMERTYPEID)," +
                           "FOREIGN KEY (C_CUSTOMERID)  REFERENCES CUSTOMERS
(C_CUSTOMERID));");



        stmt.executeUpdate("drop table if exists EMPLOYEES;");
        //creating table EMPLOYEES
        stmt.executeUpdate("CREATE TABLE EMPLOYEES(" +
                           "  E_EMPLOYEEID       INT             NOT NULL," +
                           "  E_LASTNAME         VARCHAR(300)        NOT NULL," +
                           "  E_FIRSTNAME        VARCHAR(300)        NOT NULL," +
                           "  E_TITLE          VARCHAR(300)," +
                           "  E_TITLEOFCOURTESY  VARCHAR(300)," +
                           "  E_BIRTHDATE        DATETIME," +
                           "  E_HIRE_DATE        DATETIME," +
                           "  E_ADDRESS          VARCHAR(300)," +
                           "  E_CITY           VARCHAR(300)," +
                           "  E_REGION           VARCHAR(300)," +
                           "  E_POSTALCODE       INT," +
                           "  E_COUNTRY          VARCHAR(300)," +
                           "  E_HOMEPHONE        INT," +
                           "  E_EXTENSION        INT," +
                           "  E_PHOTO          VARCHAR(300)," +
                           "  E_NOTES          VARCHAR(300)," +
                           "  E_REPORTSTO        INT," +
                           "  E_PHOTOPATH        VARCHAR(300)," +
                           "  PRIMARY KEY  (E_EMPLOYEEID));");

          stmt.executeUpdate("drop table if exists REGION;");
          stmt.executeUpdate("CREATE TABLE REGION(" +
                             "  R_REGIONID     INT      NOT NULL," +
                             "  R_REGIONDESCRIPTION     VARCHAR(300)," +
                             "  PRIMARY KEY (R_REGIONID));");

        stmt.executeUpdate("drop table if exists TERRITORIES;");
        stmt.executeUpdate("CREATE TABLE TERRITORIES(" +
                           "  T_TERRITORYID   INT      NOT NULL," +
                           "  R_REGIONID            INT            NOT NULL," +
                           "  T_TERRITORYDESCRIPTION  VARCHAR(300)    NOT NULL," +
                           "  PRIMARY KEY (T_TERRITORYID)," +
                           "FOREIGN KEY (R_REGIONID)  REFERENCES REGION
(R_REGIONID));");
```

```
        stmt.executeUpdate("drop table if exists EMPLOYEETERRITORIES;");
        //creating table EMPLOYEETERRITORIES
        stmt.executeUpdate("CREATE TABLE EMPLOYEETERRITORIES(" +
                           "  E_EMPLOYEEID   INT   NOT NULL," +
                           "  T_TERRITORYID  INT       NOT NULL," +
                           "  PRIMARY KEY (E_EMPLOYEEID, T_TERRITORYID)," +
                           "FOREIGN KEY (E_EMPLOYEEID) REFERENCES EMPLOYEES
(E_EMPLOYEEID)," +
                           "FOREIGN KEY (T_TERRITORYID) REFERENCES
TERRITORIES (T_TERRITORYID));");

        stmt.executeUpdate("drop table if exists SHIPPERS;");
        stmt.executeUpdate("CREATE TABLE SHIPPERS(" +
                           "  S_SHIPPERID    INT   NOT NULL," +
                           "  S_COMPANYNAME    VARCHAR(300)  NOT NULL," +
                           "  S_PHONE   INT," +
                           " PRIMARY KEY (S_SHIPPERID));");

        stmt.executeUpdate("drop table if exists ORDERS;");
        stmt.executeUpdate("CREATE TABLE ORDERS( " +
                           "  O_ORDERID    INT    NOT NULL," +
                           "  E_EMPLOYEEID      INT," +
                           "  S_SHIPPERID       INT," +
                           "  C_CUSTOMERID      INT," +
                           "  O_ORDERDATE       DATETIME," +
                           "  O_REQUIREDDATE    DATETIME," +
                           "  O_SHIPPEDDATE     DATETIME," +
                           "  O_FREIGHT         VARCHAR(300)," +
                           "  O_SHIPNAME        VARCHAR(300)," +
                           "  O_SHIPADDRESS     VARCHAR(300)," +
                           "  O_SHIPCITY        VARCHAR(300)," +
                           "  O_SHIPREGION      VARCHAR(300)," +
                           "  O_SHIPPOSTALCODE   INT," +
                           "  O_SHIPCOUNTRY     VARCHAR(300)," +
                           "  PRIMARY KEY (O_ORDERID)," +
                        "FOREIGN KEY (C_CUSTOMERID) REFERENCES CUSTOMERS
(C_CUSTOMERID)," +
                        "FOREIGN KEY (E_EMPLOYEEID) REFERENCES EMPLOYEES
(E_EMPLOYEEID)," +
                        "FOREIGN KEY (S_SHIPPERID)  REFERENCES SHIPPERS
(S_SHIPPERID));");

        stmt.executeUpdate("drop table if exists SUPPLIERS;");
        stmt.executeUpdate("CREATE TABLE SUPPLIERS(" +
                           "  SU_SUPPLIERID      INT     NOT NULL," +
                           "  SU_COMPANYNAME    VARCHAR(300)  NOT NULL," +
                           "  SU_CONTACTNAME     VARCHAR(300)," +
                           "  SU_CONTACTTITLE    VARCHAR(300)," +
                           "  SU_ADDRESS         VARCHAR(300)," +
                           "  SU_CITY           VARCHAR(300)," +
                           "  SU_REGION          VARCHAR(300)," +
                           "  SU_POSTALCODE      INT," +
                           "  SU_COUNTRY         VARCHAR(300)," +
                           "  SU_PHONE          INT," +
                           "  SU_FAX            INT," +
```

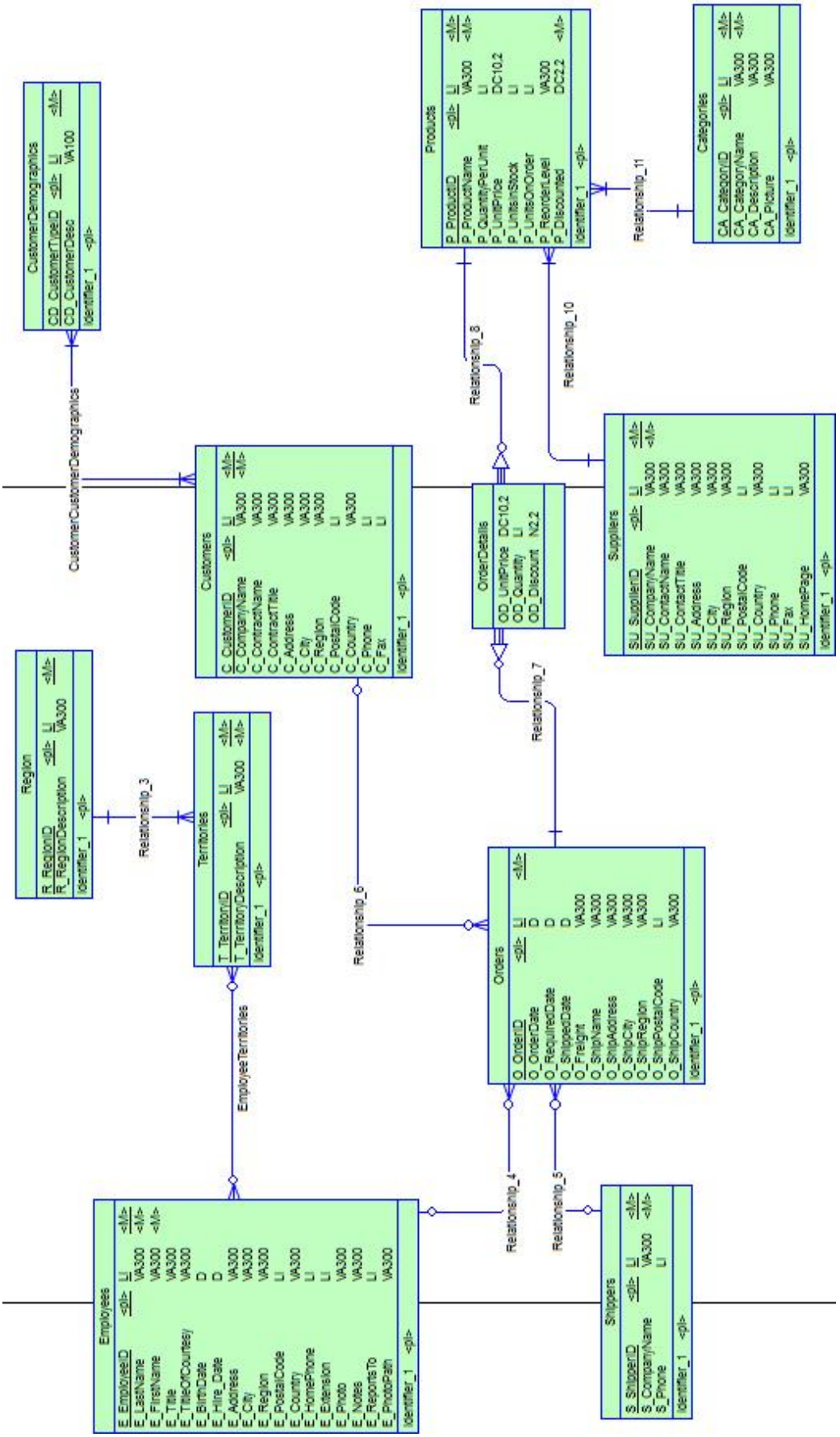
```
                            "  SU_HOMEPAGE        VARCHAR(300)," +
                            "  PRIMARY KEY (SU_SUPPLIERID));");


        stmt.executeUpdate("drop table if exists PRODUCTS;");
        stmt.executeUpdate("CREATE TABLE PRODUCTS(" +
                            "  P_PRODUCTID        INT                NOT NULL," +
                            "  SU_SUPPLIERID      INT                NOT NULL," +
                            "  CA_CATEGORYID      INT                NOT NULL," +
                            "  P_PRODUCTNAME      VARCHAR(300)       NOT NULL," +
                            "  P_QUANTITYPERUNIT  INT," +
                            "  P_UNITPRICE        DECIMAL(10,2)," +
                            "  P_UNITSINSTOCK     INT," +
                            "  P_UNITSONORDER     INT," +
                            "  P_REORDERLEVEL     VARCHAR(300)," +
                            "  P_DISCOUNTED       DECIMAL(2,2)            NOT NULL," +
                            "  PRIMARY KEY (P_PRODUCTID)," +
                            "FOREIGN KEY (CA_CATEGORYID)  REFERENCES
CATEGORIES (CA_CATEGORYID)," +
                            "FOREIGN KEY (SU_SUPPLIERID)  REFERENCES SUPPLIERS
(SU_SUPPLIERID));");

        stmt.executeUpdate("drop table if exists ORDERDETAILS;");
        //creating table ORDERDETAILS
        stmt.executeUpdate("CREATE TABLE ORDERDETAILS(" +
                            "  O_ORDERID    INT                    NOT NULL," +
                            "  P_PRODUCTID  INT                    NOT NULL," +
                            "  OD_UNITPRICE  DECIMAL(10,2)," +
                            "  OD_QUANTITY   INT," +
                            "  OD_DISCOUNT   DECIMAL(2,2)," +
                            "  PRIMARY KEY (O_ORDERID, P_PRODUCTID)," +
                            "FOREIGN KEY (O_ORDERID) REFERENCES ORDERS
(O_ORDERID)," +
                            "FOREIGN KEY (P_PRODUCTID) REFERENCES PRODUCTS
(P_PRODUCTID));");



// indeksi za polnjenje baze podatkov. Indekse sem kreiral nad prazno bazo

/* stmt.executeUpdate("CREATE INDEX CATEGORIESID ON CATEGORIES
(CA_CATEGORYID ASC);");
 stmt.executeUpdate("CREATE INDEX CUSTOMERCUSTOMERDEMOGRAPHICSID ON
CUSTOMERCUSTOMERDEMOGRAPHICS (C_CUSTOMERID, CD_CUSTOMERTYPEID
ASC);");
 stmt.executeUpdate("CREATE INDEX CUSTOMERDEMOGRAPHICSID ON
CUSTOMERDEMOGRAPHICS (CD_CUSTOMERTYPEID ASC);");
stmt.executeUpdate("CREATE INDEX CUSTOMERSID ON CUSTOMERS (C_CUSTOMERID
ASC);");
stmt.executeUpdate(" CREATE INDEX EMPLOYEESID ON EMPLOYEES (E_EMPLOYEEID
ASC);");
stmt.executeUpdate("CREATE INDEX EMPLOYEETERRITORIESID ON
EMPLOYEETERRITORIES (E_EMPLOYEEID, T_TERRITORYID ASC);");
```

```
stmt.executeUpdate("CREATE INDEX ORDERDETAILSID ON ORDERDETAILS (O_ORDERID,
P_PRODUCTID ASC);");
stmt.executeUpdate("CREATE INDEX ORDERSID ON ORDERS (O_ORDERID ASC);");
stmt.executeUpdate("CREATE INDEX PRODUCTSID ON PRODUCTS (P_PRODUCTID ASC);");
stmt.executeUpdate("CREATE INDEX REGIONID ON REGION (R_REGIONID ASC);");
stmt.executeUpdate("CREATE INDEX SHIPPERSID ON SHIPPERS (S_SHIPPERID ASC);");
stmt.executeUpdate("CREATE INDEX SUPPLIERSID ON SUPPLIERS (SU_SUPPLIERID
ASC);");
stmt.executeUpdate("CREATE INDEX TERRITORIESID ON TERRITORIES (T_TERRITORYID
ASC);");*/



//-----------------------------------------polnjenje baze----------------------------------------------

Statement st = connect.createStatement();
read=new Scanner(new BufferedReader(new FileReader("NorthWind-podatki_v3MYSQL.sql")));
read=read.useDelimiter("INSERT");
startTime = System.nanoTime();
          while (read.hasNext())
          {
              b="INSERT";
              b=b+read.next();
              b=b.substring(0,b.length());
              // System.out.println(b);
             st.executeUpdate(b);
          }

stopTime = System.nanoTime();
read.close();
System.out.println("Polnjenje baze: "+(stopTime-startTime)/1000000000+" s");


//-----Indeksi za poizvedbe-----------------------
/*stmt.executeUpdate("CREATE INDEX INDEXC_COMPANYNAME ON CUSTOMERS
(C_COMPANYNAME);");
stmt.executeUpdate("CREATE INDEX INDEXO_ORDERDATE ON ORDERS
(O_ORDERDATE);");
stmt.executeUpdate("CREATE INDEX INDEXE_LASTNAME ON EMPLOYEES
(E_LASTNAME);");
stmt.executeUpdate("CREATE INDEX INDEXE_FIRSTNAME ON EMPLOYEES
(E_FIRSTNAME);");
stmt.executeUpdate("CREATE INDEX INDEXOD_UNITPRICE ON ORDERDETAILS
(OD_UNITPRICE);");
stmt.executeUpdate("CREATE INDEX INDEXP_PRODUCTNAME ON PRODUCTS
(P_PRODUCTNAME);");*/

//---------------------------------Indexi za update---------------------------------
stmt.executeUpdate("CREATE INDEX OSHIPNAME ON ORDERS (O_SHIPNAME);");
stmt.executeUpdate("CREATE INDEX PRODUCTNAME ON PRODUCTS
(P_PRODUCTNAME);");
stmt.executeUpdate("CREATE INDEX CONTRACTNAME ON CUSTOMERS
(C_CONTRACTNAME);");
```

```
//-----------------------------------Poizvedba---------------------------------------
stmt = null;
startTime = System.nanoTime();
                    for(int i=0;i<1000;i++){
                       nakljucno=(int)(Math.random()*(9-1)+1);
                       querry=VrniPoizvedbo(nakljucno);
                         try {
                               stmt = connect.createStatement();
                               rs = stmt.executeQuery(querry);
                              }
                         catch (SQLException ex) {
                                  ex.printStackTrace();
                             }

                        }
stopTime = System.nanoTime();
skupniCas=skupniCas+(stopTime-startTime);
System.out.println("Excetution time:"+(skupniCas/1000000000)+" s");

stmt.close();
connect.close();

//-----------------------------------Poizvedbe+updati---------------------------------------
stmt = null;
startTime = System.nanoTime();
                    for(int i=0;i<1000;i++){
                              nakljucno=(int)(Math.random()*(6-1)+1);
                              querry=VrniPoizvedbo(nakljucno);
                              if(t1[nakljucno-1]==0){
                                        update=VrniUpdateZast0(nakljucno);
                                        t1[nakljucno-1]=1;
                              }else{
                                        update=VrniUpdateZast1(nakljucno);
                                        t1[nakljucno-1]=0;
                                    }
                              try {
                                        stmt = connect.createStatement();
                                         stmt.executeQuery(querry);
                                         stmt = connect.createStatement();
                                         stmt.executeUpdate(update);
                                 }
                                 catch (SQLException ex) {
                                        ex.printStackTrace();
                              }
                       }
                    stopTime = System.nanoTime();
                    skupniCas=skupniCas+(stopTime-startTime);
                    System.out.println("Excetution time: "+(skupniCas/1000000000)+" s");
                    connect.close();

                    }
          }
```

Priloga 2

KONCEPTUALNI MODEL PODATKOVNE BAZE NORTHWIND



Priloga 3

INDEKSI SCENARIJA B

Indeksi, ki sem jih uporabil pri poizvedbah za pohitritev izvajanja:

1. CREATE INDEX INDEXC_COMPANYNAME ON CUSTOMERS (C_COMPANYNAME);

2. CREATE INDEX INDEXO_ORDERDATE ON ORDERS (O_ORDERDATE);

3. CREATE INDEX INDEXE_LASTNAME ON EMPLOYEES (E_LASTNAME);

4. CREATE INDEX INDEXE_FIRSTNAME ON EMPLOYEES (E_FIRSTNAME);

5. CREATE INDEX INDEXOD_UNITPRICE ON ORDERDETAILS (OD_UNITPRICE);

6. CREATE INDEX INDEXP_PRODUCTNAME ON PRODUCTS (P_PRODUCTNAME);

Priloga 4

INDEKSI SCENARIJA C

Indeksi, ki sem jih uporabil pri posodobitvah za pohitritev izvajanja:

1. CREATE INDEX OSHIPNAME ON ORDERS (O_SHIPNAME);
2. CREATE INDEX PRODUCTNAME ON PRODUCTS (P_PRODUCTNAME);
3. CREATE INDEX CONTRACTNAME ON CUSTOMERS (C_CONTRACTNAME);