REPUBLIKA SLOVENIJA
MINISTRSTVO ZA IZOBRAŽEVANJE, ZNANOST IN ŠPORT

Univerza *v Ljubljani*

EVROPSKA UNIJA
EVROPSKI
SOCIALNI SKLAD
NALOŽBA V VAŠO PRIHODNOST

Management podatkovnih baz v resničnem/poslovnem okolju - 2. del

Tadej Borovšak, XLAB d.o.o.

Uvod

Današnjo delavnico bomo pričeli s pregledom osnovnih mehanizmov, ki jih sistemi za upravljanje podatkovnih baz uporabljajo pri svojem delovanju oz. pri shranjevanju podatkov. Te koncepte bomo v nadaljevanju uporabili kot temelje, na katerih bomo zgradili postopke za varnostno kopiranje podatkov. Delavnico bomo zaključili s posplošitvijo postopkov varnostnega kopiranja, nakar jih bomo uporabili za implementacijo ene izmed oblik distribuiranega sistema.

Principi delovanja sistemov za upravljanje s podatkovnimi bazami

Osnovne lastnosti poljubnega sistema za upravljanje s podatkovnimi bazami, ki hrani občutljive podatke, se večinoma opiše s kratico **ACID**. Kratica predstavlja štiri osnovne principe, ki zagotavljajo varnost podatkov: **Atomicity** (nedeljivost posamezne transakcije), **Consistency** (zagotavljanje veljavnosti hranjenih podatkov), **Isolation** (izločitev medsebojnih vplivov transakcij) in **Durability** (trajnost zapisa podatkov).

Nedeljivost transakcije (atomicity) Nedeljivost posamezne transakcije nam zagotavlja, da se transakcija izvede v celoti oz. se sploh ne izvede. Tipičen primer transakcije, za katero resnično želimo, da se ne izvede le delno, je prenos sredstev D med transakcijskima računoma A in B. Transakcija v tem primeru sestoji iz štirih operacij:

1. Preberi stanje SA iz računa A.
2. Shrani stanje SA - D na račun A.
3. Preberi stanje SB iz računa B.
4. Shrani stanje SB + D na račun B.

V kolikor ta transakcija ne bi bila nedeljiva, bi se lahko zgodilo, da se nekaj sredstev izgubi (če npr. uspeta le prvi dve operaciji) oz. pridela iz nič (če ne uspe druga operacija).

Veljavnost shranjenih podatkov (consistency) Ta lastnost se močno naslanja na nedeljivost transakcije in nam zagotavlja, da je sistem ob koncu poljubne transakcije v konsistentnem stanju. Če recikliramo prejšnji primer prenosa sredstev, lahko konsistentnost sistema definiramo kot nespremenljivost vsote sredstev na vseh računih, ker prenosi med računi ne spreminjajo te količine. Za naš zgornji primer bi bila enačba konsistence SA + SB = (SA - D) + (SB + D).

Izločitev medsebojnih vplivov transakcij Če še za trenutek ostanem v svetu transakcijskih računov, lahko to lastnost najlažje ilustriramo s primerom, ko želimo iz računa A v dveh vzporednih transakcijah prenesti na račun B D sredstev in na račun C E sredstev. Zaporedje operacij, ki ga želimo izvesti, je naslednje:

1. Preberi stanje SA1 iz računa A.
2. Shrani stanje SA1 - D na račun A.
3. Preberi stanje SB iz računa B.
4. Shrani stanje SB + D na račun B.
5. Preberi stanje SA2 iz računa A.
6. Shrani stanje SA2 - E na račun A.
7. Preberi stanje SC iz računa C.
8. Shrani stanje SC + E na račun C.

Naložbo sofinancirata Republika Slovenija in Evropska unija iz Evropskega socialnega sklada.

V zgornjem seznamu so operacije posamezne transakcije navedene v neprekinjenem zaporedju. Ker pa se transakciji izvajata sočasno, bi se lahko zaporedje operacij, v kolikor sistem ne bi omogočal izločitve medsebojnih vplivov transakcij, spremenilo v:

1. Preberi stanje SA1 iz računa A.
2. Preberi stanje SA2 iz računa A.
3. Shrani stanje SA1 - D na račun A.
4. Preberi stanje SB iz računa B.
5. Shrani stanje SB + D na račun B.
6. Shrani stanje SA2 - E na račun A.
7. Preberi stanje SC iz računa C.
8. Shrani stanje SC + E na račun C.

Brez večjih težav se prepričamo, da takšen vrstni red vodi v nekonsistentno stanje podatkovne baze.

Trajnost zapisa podatkov Zadnja lastnost nam zagotavlja, da je posledica transakcije (v našem primeru prenos sredstev) ob koncu transakcije trajno shranjena in zavarovana tudi v primeru okvare strojne opreme, na kateri gostuje podatkovna baza.

Zadnja lastnost je edina, pri kateri sistem potrebuje sodelovanje operacijskega sistema oz. strojne opreme, zato si bomo ta del ogledali malce podrobneje.

Zagotavljanje trajnosti hrambe podatkov

Da zagotovimo trajen zapis podatkov, moramo poskrbeti, da so podatki zapisani na disk in ne le v RAM oz. predpomnilnik. V najbolj enostavnem scenariju bi torej vsako transakcijo zapisali na disk, šele na to bi obvestili uporabnika, da je transakcija uspela.

Med tem ko bi zgoraj opisana metoda vsekakor delovala, pa je povsem neprimerna za uporabo na sistemih, do katerih dostopa več kot peščica uporabnikov. Zakaj? Ker je zapisovanje na disk ena izmed najbolj počasnih operacij, ki jih računalnik zna opraviti.

Počasnost operacije je posledica več faktorjev, najbolj pa na počasnost vplivata počasnost prenosa podatkov po vodilih do diska in pa velikost enote zapisa (memory page), ki je na običajnih računalnikih nastavljena na 4 kiB, kar pomeni, da spremembe le enega bajta pomeni zapis 4 kiB podatkov.

Tako kot večina sistemov za upravljanje podatkovnih baz, tudi PostgreSQL to težavo rešuje z zapisovanjem dnevnika sprememb. Spreminjanje podatkov v podatkovni bazi tako najprej navodila za spremembe zapiše v dnevnik (WAL, *write-ahead log*) in poskrbi, da je dnevnik shranjen na disk. Takoj zatem obvesti uporabnika, da je transakcija uspela, obenem pa zapiše spremembe še v pomnilnik. Sinhronizacija delovnega pomnilnika in stanja na disku se sedaj lahko zgodi tudi z zamikom, ker lahko v primeru okvare sistema spremembe razberemo iz dnevnika in jih apliciramo na obstoječe stanje.

Na prvi pogled izgleda, da smo z uvedbo pisanja dnevnika le dodali kompleksnost v sistem, še vedno pa moramo redno dostopati do diska. Vendar pa se zaradi samega načina zapisovanja učinkovitost sistema precej izboljša. Zapisovanje dnevnika je namreč zaporedno, kar je za velikostni red bolj učinkovito od naključnega pisanja, ki bi ga izvajala podatkovna baza sicer. Poleg tega pa se sedaj lahko več manjših transakcij, ki se izvedejo v kratkem časovnem obdobju, na disk zapiše naenkrat.

Na sliki 1 je shematsko prikazan potek izvedbe dveh operacij. V prvem koraku sistem prejme zahtevo za izvedbo operacije (1), nakar ustvari dnevniški vpis (2) in ga zapiše na disk (4). V tem trenutku je uporabnik obveščen, da je bila njegova sprememba uspešno zabeležena (4), obenem pa izvede še zapis v RAM. Na podoben način se izvede tudi naslednja operacija (koraki 5-8). In ker

sta se operaciji izvedli s kratkim časovnim razmikom, je sistem optimiziral sinhronizacijo RAM-a in diska ter obe operaciji sinhroniziral v enem samem koraku (9).

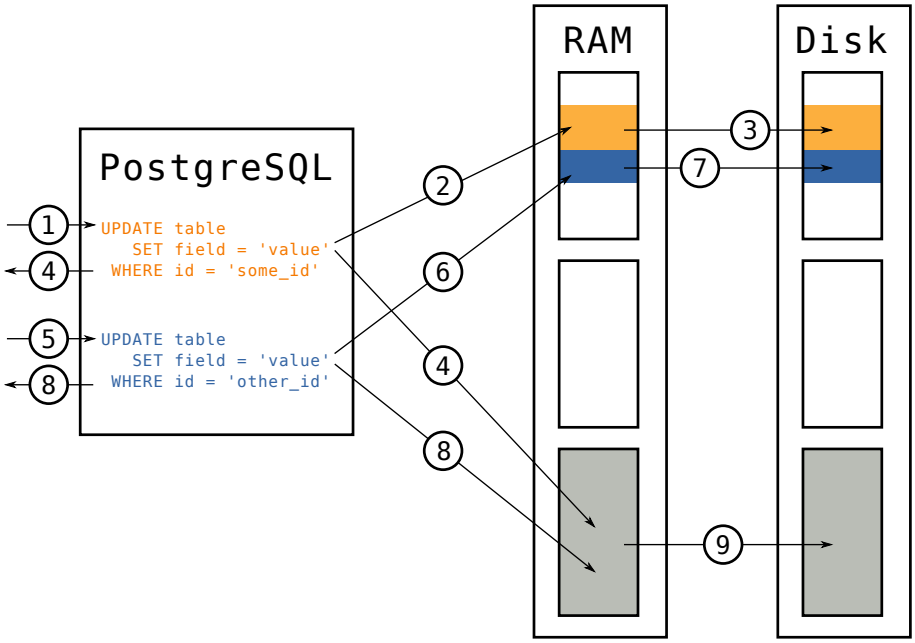

Figure 1: Zapis podatkov v bazo.

Dnevniške datoteke se po sinhronizaciji RAM-a in diska pobrišejo oz. reciklirajo, tako da ne zavzemajo večje količine prostora na disku (običajno do WAL datoteke veliko 16 MiB).

Sedaj, ko smo se seznanili z nekaterimi podrobnostmi delovanja baz, pa lahko nadaljujemo s pregledom postopkov za varnostno kopiranje oz. arhiviranje podatkov.

Varnostno kopiranje podatkov in rekonstrukcija baze

V grobem bi lahko postopke varnostnega kopiranja ločili v dve kategoriji: logične in fizične. Glavni kriterij, ki loči kategoriji, je odgovor na vprašanje "Ali lahko podatke, ki smo jih arhivirali na eni različici sistema, uporabim za obnovitev baze na novejšem sistemu?". Za logične postopke je odgovor pritrdilen, za fizične pa v veliki večini primerov nikalen.

Orodja za logično arhiviranje ustvarijo datoteko z SQL ukazi, ki jih lahko uporabimo za obnovitev baze. V primeru sistema PostgreSQL za ta namen uporabimo orodje `pg_dump`, ki na standardni izhod izpiše SQL ukaze za obnovitev. Program zna pripraviti tudi drugačne oblike podatkov, ki se jih lahko npr. ob rekonstrukciji podatkovne baze vzporedno nalaga v bazo, s čimer se skrajša čas obnovitve.

Fizično arhiviranje pa je v osnovi kopiranje datotek na datotečnem sistemu, ki vsebujejo podatke iz baze, tako da za to vrsto kopiranja ne potrebujemo posebnih orodij ampak lahko uporabimo obstoječe rešitve za arhiviranje podatkov (npr. program `tar`).

Za katero metodo se odločiti? Logične kopije podatkovne baze vsebujejo le podatke, brez meta-podatkov (npr. indeksov), zato so v primerjavi s fizičnimi kopijami običajno manjše, vendar pa ob obnovitvi baze potrebujejo več časa za popolno rekonstrukcijo, saj je potrebno meta-podatke ponovno ustvariti. Z logičnimi metodami je mogoče arhivirati tudi le dele podatkovne baze (posamezne relacije), kar z fizičnimi metodami ni mogoče.

Za produkcijske sisteme je bolj kot sama vrsta kopije pomembno to, da lahko iz varnostne kopije poustvarimo stanje, ki je čim manj oddaljeno od incidenta, ki je povzročil okvaro podatkovne baze. Večina varnostnih kopij je zato v poslovnih aplikacijah implementirana kot kontinuirano arhiviranje. Pa si to obliko oglejmo malo bolj podrobno.

Kontinuirano arhiviranje

Osnovno ideja te vrste varnostnega kopiranja je uporaba dveh ločenih metod arhiviranja:

1. Ustvarjanje osnovne kopije podatkov v rednih intervalih (npr. enkrat na dan).
2. Sprotno arhiviranje WAL zapisov, s katerimi lahko osnovno kopijo posodobimo do točke incidenta.

Časovni potek arhiviranja je prikazan na sliki 2, kjer so z modrimi bloki označeni časi izdelave posameznih WAL datotek, z rumeno pa je označen čas izdelave osnovne kopije vseh datotek, ki tvorijo podatkovno bazo.

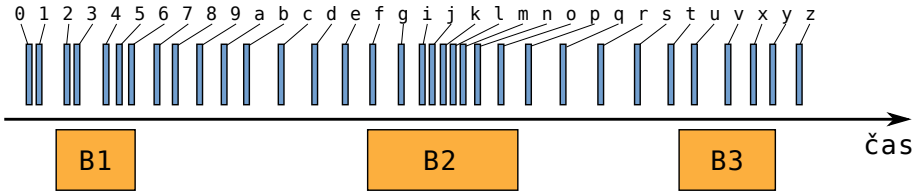

Figure 2: Prikaz komponent kontinuiranega arhiviranja.

Če predpostavimo, da se trenutno nahajamo na časovni osi tik za izdelavo WAL datoteke z, imamo za rekonstrukcijo na voljo kar tri sete datotek:

1. osnovno kopijo B1 in WAL datoteke 1, 2, 3, …, z;
2. osnovno kopijo B2 in WAL datoteke f, g, h, …, z ali
3. osnovno kopijo B3 in WAL datoteke u, v, x, y in z.

V vseh primerih pa je postopek rekonstrukcije enak:

1. Najprej delno rekonstruiramo bazo iz osnovne kopije Bx.
2. Nato uveljavimo še spremembe iz ustreznih WAL datotek.

Kontinuirano arhiviranje poleg polne rekonstrukcije omogoča tudi rekonstrukcijo stanja podatkovne baze za poljubno točko na časovni osi, kar dosežemo tako, da uveljavimo spremembe iz WAL datotek le do izbranega časa.

Se je pa potrebno zavedati, da je varnostno kopiranje (tako kot vsak računalniški proces) podvrženo dejavnikom iz okolja in se lahko zgodi, da ne uspe. Zato je nujno potrebno redno preverjanje uspešno postopka rekonstrukcije. Če npr. hranimo sedem dnevnih setov datotek za rekonstrukcijo, moramo testno rekonstrukcijo izvesti vsaj enkrat tedensko, da preprečimo morebitno izgubo podatkov.

Sedaj pa si oglejmo še pripravo distribuirane namestitve PostgreSQL sistema.

Distribuirane podatkovne baze

V tem delu si bomo ogledali postavitev distribuirane podatkovne baze z namenom povečanja zanesljivosti celotnega sistema. Povečanje zmogljivosti obdelave zahtev si bomo pogledali v zadnjem delu delavnice.

Prvi korak k povečanju zanesljivosti smo naredili z vpeljavo varnostnega kopiranja. Dodaten napredek naredimo, če uporabljamo kontinuirano arhiviranje, še vedno pa bo storitev, ki uporablja podatkovno bazo, med rekonstrukcijo baze iz varnostne kopije nedosegljiva.

Večini storitev, ki jih dnevno uporabljamo, takšen začasen izpad ne predstavlja večjih težav. Obstajajo pa situacije, kjer je nekaj ur nedosegljivosti storitve za seboj povleče hude posledice, in v teh primerih se običajno odločimo za postavitev replicirane gruče strežnikov.

Obstaja več načinov vzpostavitve gruče strežnikov, ki delujejo sinhrono:

1. hranjenje podatkov na diskovnem sistemu, ki je dosegljiv več strežnikom,
2. pošiljanje WAL segmentov med člani gruče ali
3. pošiljanje poizvedb vsem članom gruče.

Mi si bomo ogledali drugi način, ker se v praktični uporabi izkaže precej dobro in se ob izpadu glavnega strežnika ena izmed replik lahko vključi na njegovo mesto v nekaj sekundah.

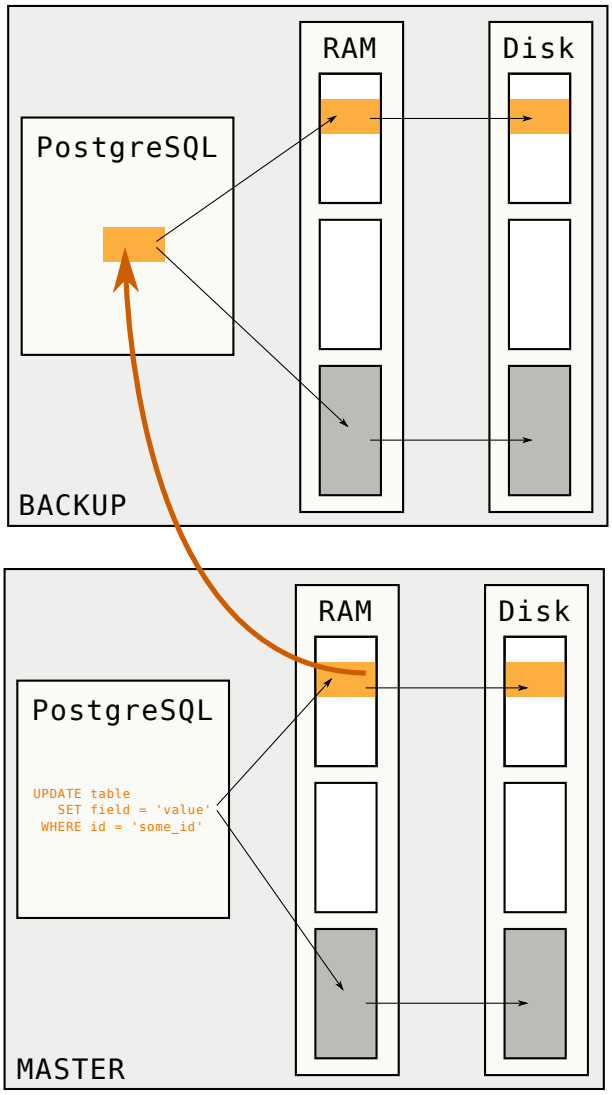

Figure 3: Pošiljanje WAL segmentov med instancami.

Takšen način replikacije je zgolj posplošitev zapisovanja dnevnika transakcij na disk, kar je jasno razvidno iz slike 3, ki močno spominja na sliko 2. Nova je le temno oranžna puščica, ki ponazarja

prenos WAL segmenta z enega strežnika na drugega.

Pošiljanje WAL segmentov je lahko bodisi asinhrono (privzeta nastavitev) ali pa sinhrono. Razlika med tema dvema načinoma je v tem, kdaj uporabnik dobi potrjeno transakcijo. V primeru asinhrone replikacije je transakcija potrjena tako, ko primarni (master) strežnik zapiše WAL segment na disk. V primeru sinhrone replikacije pa je transakcija potrjena šele, ko tudi rezervni strežnik potrdi zapis segmenta.

Seveda takšen način replikacije ni omejen le na dva strežnika, ampak jih lahko v gručo povežemo več. Običajen primer produkcijskega sistema je gruča treh strežnikov:

1. glavnega strežnika, ki sprejema poizvedbe,
2. vroče replike, ki sinhrono sprejema WAL segmente in
3. tople replike, ki WAL segmente sprejema sinhrono.

V primeru okvare primarnega strežnika nadzorni sistem pošiljanje poizvedb preusmeri na vročo repliko in njeno delovanje preklopi v primarni strežnik. Topla replika poveže na novi primarni strežnik in se pogreje do vroče replike. Nato pa se okvarjena instanca zamenja in se priklopi v gručo kot topla replika. Ta scenarij je shematsko prikazan na sliki 4.

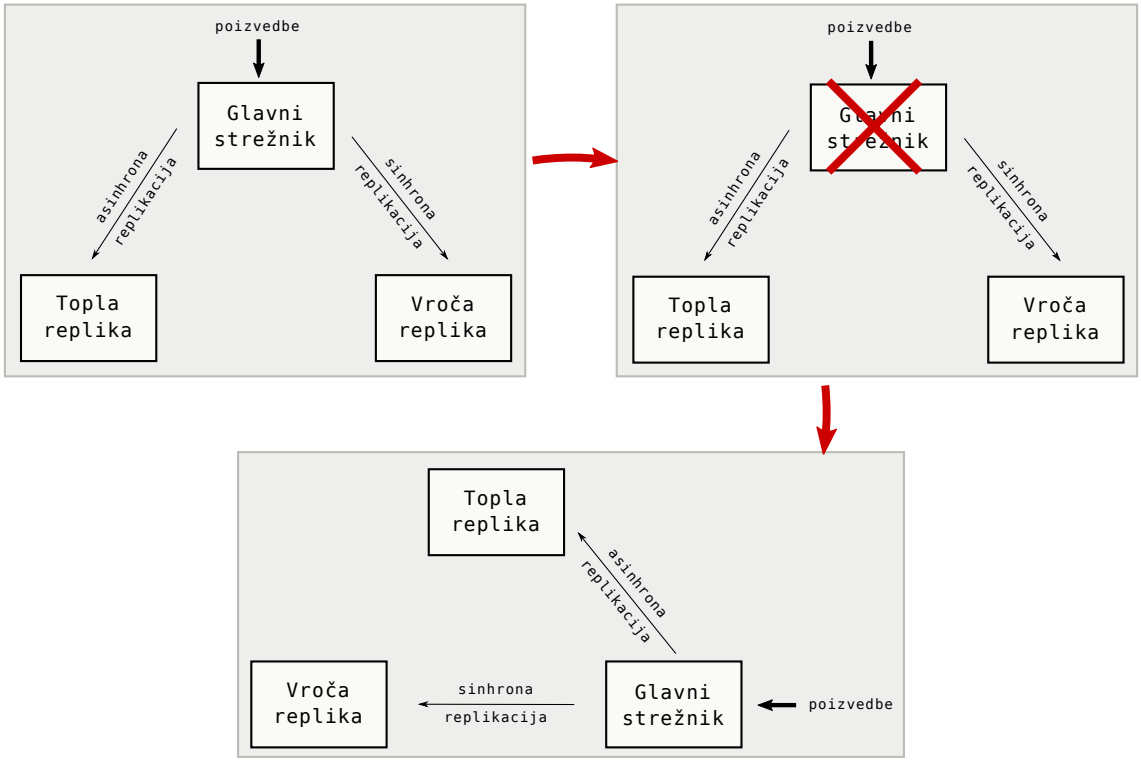

Figure 4: Preklop vlog strežnikov v gruči.