|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NIŠU  ELEKTRONSKI FAKULTET |  |

Miljan Denić

**OBRADA TRANSAKCIJA, PLANOVI IZVRŠAVANJA TRANSAKCIJA, IZOLACIJA I ZAKLJUČAVANJE KOD POSTGRESQL BAZE PODATAKA**

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

|  |
| --- |
| Kandidat: |
|  |
| Miljan Denić, br. ind. 1517 |

Niš, maj 2023. godina

SADRžaj

[1. uvod 3](#_Toc135373315)

[2. Obrada transakcija 4](#_Toc135373316)

[2.1. Svojstva transakcija 4](#_Toc135373317)

[2.2. Pojam transakcije 4](#_Toc135373318)

[2.3. Upravljanje transakcijama 5](#_Toc135373319)

[2.4. Korišćenje SAVEPOINT komande 8](#_Toc135373320)

[2.5. Dvofazno potvrđivanje transakcija 10](#_Toc135373321)

[2.6. Podešavanje karakteristika transakcija 12](#_Toc135373322)

[3. Planovi izvršavanja transakcija 13](#_Toc135373323)

[3.1. Primer plana upita kod transakcija 13](#_Toc135373324)

[4. IZolacija 14](#_Toc135373325)

[4.1. Read Commited nivo izolacije 15](#_Toc135373326)

[4.2. Repeatable Read nivo izolacije 16](#_Toc135373327)

[4.3. Serializable nivo izolacije 17](#_Toc135373328)

[5. Zaključavanje 19](#_Toc135373329)

[5.1. Zaključavanja na nivou tabele 19](#_Toc135373330)

[5.2. Zaključavanja na nivou reda 21](#_Toc135373331)

[5.3. Zastoji 22](#_Toc135373332)

[5.4. Savetodavna zaključavanja 23](#_Toc135373333)

[6. Zaključak 25](#_Toc135373334)

[7. Literatura 26](#_Toc135373335)

# uvod

Uvođenjem baza podataka u poslovne i svakodnevne aktivnosti, postalo je neophodno da baze podataka budu u stanju da obrade veliku količinu podataka i da istovremeno budu pouzdane i bezbedne. U ovom kontekstu, transakcije predstavljaju mehanizam za upravljanje podacima koji omogućavaju uspešno i bezbedno izvršavanje više povezanih operacija.

U prvom delu seminarskog rada će biti reči o samoj obradi transakcija, biće opisano šta transakcija predstavlja, koja svojstva treba da zadovolji, koje su mogućnosti transakcija i kako možemo kontrolisati izvršenje transakcija.

Zatim će biti reči o planu izvršavanja transakcija koji se odnosi na planiranje i organizovanje načina na koji se transakcije izvršavaju u PostgreSQL-u. Postoji više načina za obradu transakcija, a svaki od njih ima svoje prednosti i nedostatke.

Nakon toga sledi nešto više o izolaciji u kontekstu baza podataka koja se odnosi na sposobnost sistema da omogući da transakcije rade nezavisno jedna od druge, tako da se promene koje jedna transakcija vrši ne ometaju rad drugih transakcija. Izolacija se postiže korišćenjem različitih nivoa izolacije koji će biti opisani i koji definišu kako transakcije vide i interaguju sa podacima koje deli sa drugim transakcijama. Nivoi izolacije obezbeđuju balans između sigurnosti i performansi, uzimajući u obzir zahteve konkurentnosti i doslednosti podataka.

Zaključavanje je još jedan ključni koncept u upravljanju transakcijama koji će biti opisan. Ovaj mehanizam sprečava da dve ili više transakcija pristupe istim resursima istovremeno, čime se obezbeđuje da transakcije budu izolovane i da se izvršavaju jedna za drugom.

U nastavku će biti detaljnije razmotren svaki od ovih koncepta, kako bi bolje razumeli kako Postgresql funkcioniše i kako se može optimizovati za najbolje performanse i pouzdanost. Za izvršavanje upita će se koristiti baza na PostgreSQL serveru verzije 15.2[1].

# Obrada transakcija

## Svojstva transakcija

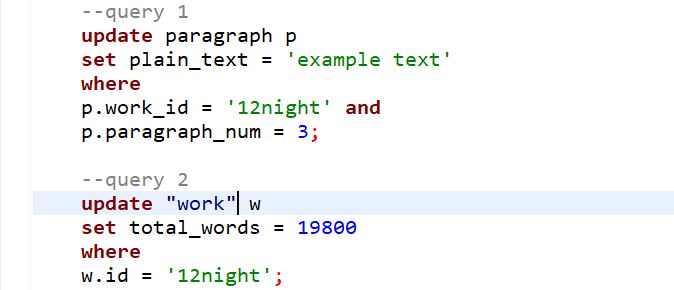
Koncept transakcija se sastoji od četiri ključne karakteristike, poznatih kao **ACID** svojstva[2]:

* **Atomicnost (atomicity)**: Transakcija se izvršava kao celina ili se ne izvršava uopšte. Ukoliko se neka od operacija ne izvrši uspešno, transakcija se poništava i sve se vraća u prethodno stanje.
* **Konzistentnost (consistency):** Transakcija mora da održava konzistentnost baze podataka. Ukoliko je baza bila konzistentna pre početka transakcije, mora biti konzistentna i nakon njenog završetka.
* **Izolacija (isolation):** Transakcije koje se izvršavaju istovremeno moraju biti izolovane jedna od druge. Ovo znači da se promene koje jedna transakcija izvršava ne smeju videti od strane druge transakcije, sve dok prva ne završi svoje izvršavanje.
* **Trajnost (durability):** Kada se transakcija završi, njeni efekti moraju ostati trajno sačuvani u bazi podataka, čak i u slučaju pada sistema ili nekog drugog kvara.

## Pojam transakcije

Tansakcije su osnovni koncept svih baza podataka. Osnovna ideja svih transakcija je da objedinjuju više koraka u jednu operaciju koja će se u potpunosti izvršiti ili se neće izvršiti uopšte. Međustanja između koraka nisu vidljiva drugim konkurentnim transakcijama i ukoliko se neka greška dogodi to će prekinuti izvršenje transakcije i nijedan od dotada izvršenih koraka neće imati uticija na bazu podataka.

Uzmimo za primer bazu podataka o Šekspiru sa kojom ćemo raditi. Baza ima tabelu paragrafa gde se nalaze dijalozi iz određenih dela koja se nalaze u drugoj tabeli. Želimo da promenimo tekst jednog paragrafa i da ažuriramo broj reči u delu u tabeli „work“ (tabela gde su smeštena sva dela). SQL naredba bi izgledala ovako:



Slika 1. Primer upita

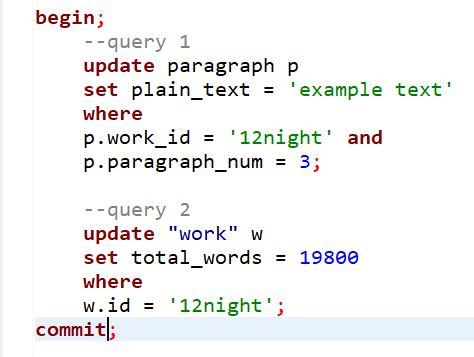
Detalji o samom upitu trenutno nisu previše bitni za objašnjenje poente. Bitno je napomenuti da su više nezavisnih update naredbi uključene kako bi se izvršila ova operacija. Bitno je da ukoliko se promeni tekst paragrafa da se promeni i broj reči u delu. Treba garantovati da ukoliko nešto ne bude bilo u redu tokom izvršenja nijedan korak neće imati uticija za bazu podataka. Grupisanje ovih naredbi u transakciju nam daje tu garanciju. Za transakciju kazemo da treba da bude atomična odnosno da treba da se izvrši u potpunosti ili da se uopšte ne izvrši.

Takođe, želimo garantovati da jednom kada se transakcija izvrši i potvrđena je od strane sistema baze i ukoliko dođe do pada servera ili sličnih okolnosti da su podaci trajno sačuvani i da neće biti izgubljeni nakon toga. Transakcija garantuje da su sve promene koje su napravljene transakcijom sačuvane u trajnu memoriju kao što je disk.

Još jedno bitno svojstvo transakcionih baza je povezano sa pojmom atomičnih ažuriranja. Kada imamo više transakcija koje se izvršavaju konkurentno, nijedna od njih ne bi trebala da vidi nekompletne promene koje su nastale tokom izvršenja drugih transakcija. Npr. ako menjamo tekst nekog paragrafa, taj tekst ne sme biti vidljiv drugim upitima sve dok transakcija nije u potpunosti završena. Dakle, transakcije moraju biti sve ili ništa ne samo u smislu njihovog trajnog efekta na baz podataka već i u smislu njihove vidljivosti dok se izvršavaju. Ažuriranja koja su napravljena tokom izvršenja transakcije su nevidljiva za druge transakcije dok se transakcija ne završi, nakon čega sva ažuriranja postaju vidljiva istovremeno.

## Upravljanje transakcijama

PostgreSQL baza podataka implementira transakcije kroz mehanizam loga transakcija. Svi SQL upiti koji se izvršavaju nad bazom podataka su deo transakcije, čak i ako to nije eksplicitno navedeno. Ukoliko se ne koristi BEGIN/COMMIT sintaksa, PostgreSQL implicitno pokreće transakciju pre svakog upita i zatim je potvrđuje (commit) ili poništava (rollback) nakon izvršavanja upita. Upravljanje transakcijama se može vršiti i eksplicitno, pomoću BEGIN, COMMIT i ROLLBACK naredbi.



Slika 2. Primer transakcije

**BEGIN naredba**

BEGIN komanda označava start transakcionog bloka[3]. Sve komande nakon BEGIN komande će biti izvršene u jednoj transakciji sve do unosa komandi COMMIT ili ROLLBACK. Normalno (bez BEGIN naredbe) PostgreSQL izvršava transakcije u „autocommit“ modu što znači da se svaka naredba izvršava u zasebnoj transakciji i potvrđivanje se vrši nakon svake naredbe ukoliko je izvršenje bilo uspešno, u suprotnom se vrši rollback.

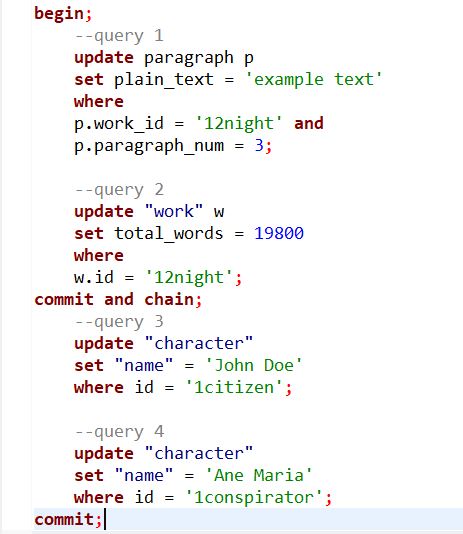
Naredbe se izvršavaju dosta brže u transakcionom bloku jer transakcione start/commit naredbe zahtevaju određene CPU i disk aktivnosti. Izvršenje više naredbi unutar transakcije je takođe korisno jer osigurava konzistenciju tokom pravljenja više povezanih promena. Ostale sesije će biti onemogućene da vide međustanja sve dok se sve povezane promene ne izvrše.

START TRANSACTION komanda vrši istu funkcionalnost kao i BEGIN naredba. Ako je specificiran nivo izolacije, čitanje ili pisanje nova transakcija će imati te karakteristike kao da je SET TRANSACTION komanda izvršena.

**COMMIT naredba**

COMMIT naredba potvrđuje trenutnu transakciju. Sve promene koje su napravljene u transakciji postaju vidljive drugima i garantovane ukoliko dođe do pada baze.

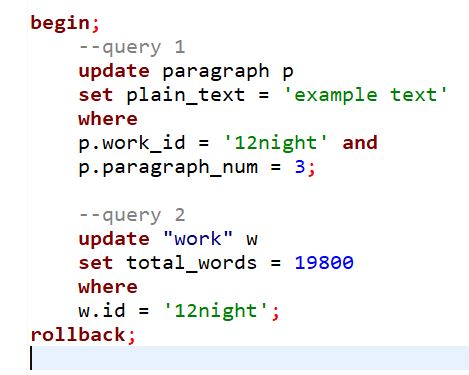
COMMIT naredba može ići sa AND CHAIN parametrom. Ako je AND CHAIN parametar specificiran, nova transakcija će krenuti odmah sa izvršenjem i imaće iste karakteristike kao prethodna koja je se upravo završila.

****

Slika 3. commit and chain primer komande

**ROLLBACK naredba**

ROLLBACK komanda vraća bazu na stanje pre početka transakcije i uzrokuje da se sve nastale promene ponište. Kao i COMMIT komanda tako i ROLLBACK komanda može ići sa parametrom AND CHAIN. U PostgreSQL bazi podataka, korišćenje ROLLBACK komande se odvija automatski u slučaju bilo kakvog problema u toku transakcije, što znači da se baza podataka vraća u stanje pre početka transakcije. To obezbeđuje da se podaci u bazi podataka ne oštete, a sve operacije se izvršavaju u sigurnom i pouzdanom okruženju.



Slika 4. Primer rollback naredbe

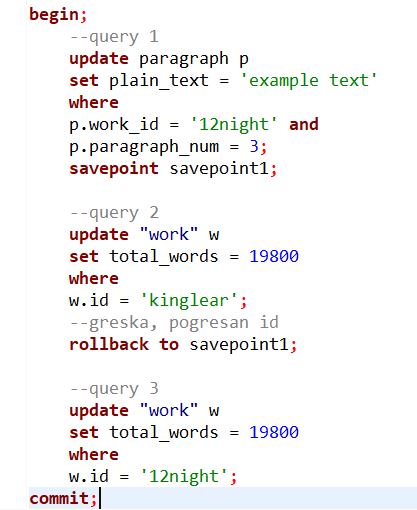
## Korišćenje SAVEPOINT komande

Moguće je kontrolisati naredbe u transakcijama na detaljnijem nivou korišćenjem savepoint-a (tačka čuvanja). Tačke čuvanja omogućuju da selektivno odbacimo delove transakcija, a da ostatak transakcije potvrdimo. Nakon definisanja SAVEPOINT, možemo se vratiti na tu tačku čuvanja koristeći ROLLBACK TO komandu.

Nakon vraćanja na određenu tačku čuvanja ona i dalje nastavlja da postoji tako da je moguće više puta se vratiti na istu tačku čuvanja. Takođe je moguće ukoliko je korisnik siguran da mu određena tačka čuvanja nije više potrebna ona može biti uklonjena tako da će se njenim uklanjanjem osloboditi određeni resursi na sistemu. Treba imati na umu da će se bilo uklanjanjem ili vraćanjem na određenu tačku čuvanja automatski izbrisati sve ostale tačke čuvanja koje su definisane nakon toga.

Sve ovo se dešava u transakcionom bloku tako da ništa od ovoga nije vidljivo ostalim sesijama u bazi podataka. Kada se ili ukoliko se uopšte potvrdi transakcioni blok potvrđene promene postaju vidljive ostalim sesijama, dok odbačene(rolled-back) promene nikad ne postanu vidljive.

Ako pogledamo primer sa slike ispod možemo da vidimo primer upotrebe savepoint komande. Recimo da je u transakciji promenjen tekst određenog paragrafa i da kasnije želimo promeniti broj reči u tom delu. Definisan je savepoint sa imenom savepoint1 nakon promene teksta paragrafa i nakon toga je greškom zamenjen broj reči u delu sa pogrešnim id-em. Koristeći savepoint savepoint1 i komandu „rollback to savepoint1“ moguće je poništiti promene nastale nakon savepoint1 i nastaviti sa izmenom broja reči u delu sa pravim id-em.



Slika 5. Primer upotrebe savepoint-a

Ovo je naravno bio uprošćen primer ali postoji dosta mogućnosti za kontrolu transakcija upotrebom tački čuvanja. ROLLBACK TO je zapravo jedini način da se povrati kontrola nad transakcionim blokom koji je doveo sistem u prekinuto stanje zbog greške, osim naravno potpunog vraćanja u početno stanje i ponovnog pokretanja transakcije.

**SAVEPOINT**

SAVEPOINT komandom se definiše nova tačka čuvanja unutar trenutne transakcije. Kao parametar ove komande se daje ime tačke čuvanja. Ako već postoje tačke čuvanja sa istim imenom, one se neće moći koristiti sve dok se ne oslobode novije tačke čuvanja sa istim imenom. SAVEPOINT je posebna oznaka koja omogućava da se sve komande koje se izvrše nakon što je kreirana odbace vraćajući stanje transakcije na ono što je bilo u trenutku kreiranja tačke čuvanja.

ROLLBACK TO SAVEPOINT je komanda kojom se odbacuje sve do tačke čuvanja. RELEASE SAVEPOINT je komanda kojom se uništava tačka čuvanja, zadržavajući efekte komandi koje su izvršene nakon kreiranja tačke čuvanja. Tačke čuvanja se mogu kreirati samo u transakcionom bloku i može postojati više tački čuvanja u okviru transakcije.

**ROLLBACK TO SAVEPOINT**

ROLLBACK TO SAVEPOINT komandom se odbacuju sve promene koje su nastale nakon postavljanja tačke čuvanja. Tačka čuvanja ostaje važeća i može se koristiti ponovo ako zatreba. ROLLBACK TO SAVEPOINT implicitno uništava sve tačke čuvanja kreirane nakon iskorišćene tačke čuvanja. Koristi se sa parametrom savepoint\_name koji označava ime tačke čuvanja na koju se vraćamo.

**RELEASE SAVEPOINT**

RELEASE SAVEPOINT uklanja tačku čuvanja prethodno definisanu u trenutnoj transakciji. Efekat uklanjanja tačke čuvanja je da više nije moguće vratiti se na stanje u trenutku definisanja te tačke čuvanja. Uklanjanje ne poništava efekte naredbi koje su se izvršile nakon kreiranja tačke čuvanja. Uklanjanjem tačke čuvanja kada više nije potrebna omogućavamo sistemu da oslobodi deo resursa sitema pre završetka transakcije.

Komanda se koristi uz parametar savepoint\_name koji označava ime tačke čuvanja koju uklanjamo. Navođenje imena tačke čuvanja koja ne postoji prouzrokuje grešku. Ako imamo više tački čuvanja sa istim imenom, samo ona najskorije definisana će biti uklonjena. Ponavljanjem komande možemo nastaviti brisanje unazad kreiranih.

## Dvofazno potvrđivanje transakcija

Dvofazno potvrđivanje (two-phase commit) je transakcijski protokol koji se koristi kako bi se osiguralo da sve baze podataka koje učestvuju u transakciji koja obuhvata više resursa (npr. različite baze podataka) slažu o uspešnom završetku transakcije, a ako transakcija ne uspe da se završi, onda da se sve baze podataka koje su učestvovale vrate u prethodno stanje.

U PostgreSQL bazi podataka, protokol dvofaznog potvrđivanja se sprovodi uz pomoć PREPARE TRANSACTION i COMMIT PREPARED komandi. Kada se izvrši PREPARE TRANSACTION, PostgreSQL šalje zahtev za potvrdu transakcije svim učesnicima transakcije. Svaka baza podataka odgovara sa svojim statusom da li se slaže sa završetkom transakcije ili ne. Ako sve baze podataka učesnice daju pozitivan odgovor, tada se izvršava COMMIT PREPARED komanda koja se sastoji od dva koraka: prvo se završava transakcija, a zatim se izvršava COMMIT na svim učesnicima transakcije. Ako bilo koja od baza podataka učesnica vrati negativan odgovor tokom faze potvrde transakcije, tada se izvršava ROLLBACK komanda na svim učesnicima transakcije.

Dvofazno potvrđivanje osigurava da se sve promene u transakciji čuvaju trajno u svim učesničkim bazama podataka ili se sve promene poništavaju ako se transakcija ne može uspešno izvršiti. To omogućava veću pouzdanost i sigurnost u aplikacijama koje koriste više baza podataka i transakcija koje se obavljaju na njima.

**PREPARE TRANSACTION**

PREPARE TRANSACTION komanda priprema trenutnu transakciju za dvofazno potvrđivanje. Nakon ove komande transakcija više nije povezana sa trenutnom sesijom. Umesto toga, stanje je u potpunosti sačuvano na disku i postoji velika verovatnoća da će biti potvrđeno uspešno čak i ako dođe do pada baze podataka pre nego što je stigao zahtev za potvrđivanjem.

Kada je transakcija pripremljena ona može kasnije biti potvrđena ili odbačena sa COMMIT PREPARED ili ROLLBACK PREPARED komandama. Ove komande mogu biti pozvane iz bilo koje sesije, a ne samo iz one koja je izvršila originalnu transakciju.

Sa strane sesije koja poziva, PREPARE TRANSACTION nema isto ponašanje kao ROLLBACK komanda. Nakon izvršenja komande nema više trenutnih transakcija i efekti pripremljenih transakcija više nisu vidljivi. Efekti postaju vidljivi tek kada se transakcija potvrdi. Ako PREPARE TRANSACTION komanda padne iz bilo kog razloga, ona postaje ROLLBACK i trenutna transakcija se poništava.



Slika 6. Primer PREPARE TRANSACTIO komande

PREPARE TRANSACTION nije namenjena za korišćenje u aplikacijama ili interaktivnim sesijama. Namenjena je da dozvoli eksternim transakcionim menadžerima da izvršavaju atomične globalne transakcije na više baza podataka ili drugih transakcionih resursa. Ova komanda mora biti korišćena unutar transakcionog bloka.

Trenutno nije dozvoljeno da se priprema transakcija koja radi sa operacijima koje uključuju privremene tabele, privermeni namespace neke sesije, kursore kreirane sa WITH HOLD ili izvršene LISTEN, UNLISTEN ili NOTIFY. Ove funkcionalnosti su previše usko vezane za trenutnu sesiju da bi bile korisne za pripremanje transakcije.

Sve trenutno dostupne pripremljene transakcije su izlistane u pg\_preprared\_xacts pogledu. Treba obratiti pažnju da pripremljene transakcije ne ostanu predugo u pripremljenom stanju. To utiče na mogućnosti VACUUM komande da oslobodi prostor na sistemu i u nekim slučajevima može dovesti do pada baze podataka. Namena ove komande jeste da se nakon pripremanja ili potvrdi ili odbaci nakon što se dobija informacija da li su ostale baze spremne za potvrđivanje.

**COMMIT PREPARED**

COMMIT PREPARED potvrđuje transakciju koja je u pripremljenom stanju. Koristi se sa parametrom transaction\_id koji označava identifikator transakcije koja se potvrđuje. Transakcija se može potvrditi samo od strane korisnika koji je izvršio transakciju originalno ili od strane super korisnika ali ne mora biti iz iste sesije kao transakcija.



Slika 7. Primer COMMIT PREPARED komande

**ROLLBACK PREPARED**

ROLLBACK PREPARED vrši poništavanje pripremljene transakcije. Kao i COMMIT PREPARED se koristi sa transaction\_id parametrom koji označava identifikator transakcije koja se potvrđuje. Takođe se može potvrditi samo od strane koristika koji je izvršio transakciju originalno ili od strane super korisnika ali ne mora biti iz iste sesije kao transakcija.



Slika 8. Primer ROLLBACK PREPARED komande

## Podešavanje karakteristika transakcija

Preko SET TRANSACTION komande možemo izvršiti podešavanje karakteristika trenutne transakcije. Ova komanda nema uticaja na bilo koje naredne transakcije. Komanda SET SESSION CHARACTERISTICS postavlja transakcione karakteristike i za naredne transakcije. Ove karakteristike mogu biti zamenjene SET TRANSACTION komandom za pojedinačnu transakciju.

Karakteristike koje se mogu podešavati su nivo izolacije transakcije, režim pristupa(čitanje/upis ili samo čitanje) i mogućnost odlaganja izvršavanja transakcije. Takođe, moguće je izabrati snapshot, ali samo za trenutnu transakciju.

Nivo izolacije transakcije određuje koje podatke transakcija može videti kada se druge transakcije izvršavaju konkurentno.

* **READ COMMITTED:** Naredba može da vidi samo redove potvrđene pre početka transakcije.Ovo je standardno podešeno.
* **REPEATABLE READ:** Ovaj nivo izolacije obezbeđuje da podatke transakcija čita samo jednom i da se tokom trajanja transakcije neće pojaviti nikakve promene tih podataka od strane drugih transakcija. Takođe, transakcija sa ovim nivoom izolacije sprečava čitanje svih novih redova koje druga transakcija doda u bazu podataka tokom trajanja transakcije.
* **SERIALIZABLE:** nivo izolacije u PostgreSQL bazi predstavlja najviši nivo izolacije i obezbeđuje potpuno izolovanu transakciju. Ovaj nivo izolacije garantuje da će se transakcije izvršavati kao da se izvršavaju jedna za drugom, a ne istovremeno.

SQL standard definiše još jedan nivo, a to je READ UNCOMMITTED koji se u PosgreSQL-u tretira kao READ COMMITTED.

Transakcioni režim pristupa određuje da li je transakcija čitanje/upis ili samo čitanje. Čitanje/upis je podrazumevano podešavanje. Kada je transakcija u režimu samo čitanja sledeće SQL komande su zabranjene: INSERT, UPDATE, DELETE, kao i COPY FROM ako tabela u koju se upisuje nije privremena tabela. Zatim su još zabranjene sve CREATE, ALTER i DROP komande, COMMENT, GRANT, REVOKE, TRUNCATE, EXPLAIN, ANALYZE i EXECUTE komande.

DEFERRABLE podešavanje transakcjie nema efekta ukoliko transakcija nije SERIALIZABLE i READ ONLY. Kada su sva tri podešavanja izabrana za transakciju, transakcija može da se blokira kada prvi put zatraži svoj snapshot, nakon čega može da se izvršava bez normalnog ograničenja SERIALIZABLE transakcije i bez ikakvog rizika od doprinošenja ili otkazivanja zbog neuspeha serializacije. Ovaj režim je pogodan za dugotrajne izveštaje ili sigurnosne kopije.

# Planovi izvršavanja transakcija

U PostgreSQL bazama plan upita predstavlja detaljno objašnjenje kako će baza podataka izvršiti određeni SQL upit. Plan upita opisuje različite korake koje će baza podataka primeniti da bi izvršila upit i dobila željene rezultate.

Plan upita je generisan od strane PostgreSQL planera upita koji je odgovoran za analizu upita i određivanje najefikasnijeg načina za povlačenje zahtevanih podataka. Planer upita uzima u obzir razne faktore prilikom generisanja plana upita, uključujući strukturu upita, podatke koji se zahtevaju, postojeće indekse i druge opcije za optimizaciju.

Plan upita se može videti koristeći EXPLAIN komandu u PostgreSQL-u. EXPLAIN komanda dozvoljava da se vide koraci koje će baza podataka primeniti kako bi izvršila upit, uključujući redosled u kojem će tabele biti skenirane, koji tipovi join operacija će biti korišćeni kao i bilo koja operacija sortiranja ili filtriranja koja će biti izvršena.

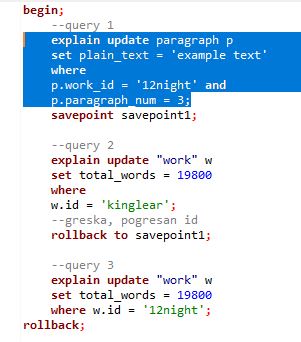
Razumevanje plana upita je bitno za optimizaciju upita jer omogućava da identifikujemo potencijalna uska grla i neefikasnosti u procesu izvršenja upita. Analizom plana upita i menjanjem upita ili šeme baze podataka mogu se poboljšati performanse upita i smanjiti količina vremena neophodna za povlačenje podataka iz baze podataka.

## Primer plana upita kod transakcija

Planovi izvršavanja transakcija se u suštini ne razlikuju od planova izvršavanja pojedinačnih upita. Kada se izvrši EXPLAIN na nivou transakcije, svaki pojedinačni upit unutar transakcije će generisati svoj plan izvršavanja, baš kao što bi to bio slučaj i da su izvršeni pojedinačno.

Međutim bitno je napomenuti da na plan izvršavanja mogu uticati kontekst i stanje baze podataka u trenutku izvršavanja transakcije. Na primer, ako se u transakciji menjaju podaci ili se drugi upiti izvršavaju paralelno, to može uticati na izbor plana izvršavanja za svaki pojedinačni upit unutar transakcije.

Takođe je važno napomenuti da EXPLAIN komanda sama po sebi ne izvršava upite, već samo generiše planove izvršavanja. To znači da će izvršavanje transakcije doneti stvarne rezultate i performanse u skladu sa tim planovima. Zato je korisno koristiti EXPLAIN za analizu i optimizaciju upita unutar transakcije, ali je takođe važno pratiti stvarne performanse i rezultate kada se transakcija izvrši.



Slika 9. Primer generisanja plana izvršenja upita

# IZolacija

Izolacija u PostgreSQL bazi podataka predstavlja nivo kontrole nad pristupom podacima tokom transakcija. To znači da pristup drugim transakcijama može biti ograničen na način da transakcija bude izvršena bez uticaja na druge transakcije. Ovaj koncept omogućava da baze podataka ostanu dosledne, pouzdane i stabilne čak i u uslovima velike konkurentnosti i pristupa različitih korisnika. PostgreSQL podržava različite nivoe izolacije transakcija, od kojih svaki pruža različitu vrstu kontrole nad pristupom podacima i time utiče na performanse sistema. Nivoi izolacije u PostgreSQL-u uključuju READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ i SERIALIZABLE.

Na svakom od ovih nivoa osim na SERIALIZABLE nivou može da se javi neki od sledećih fenomena. Oni su rezltat interakcije između konkurentnih transakcija[4].

* **Dirty read**
  + Transakcija čita podatke konkurentne nepotvrđene transakcije
* **Nonrepeatable read**
  + Transakcija ponovo čita podatke koje je prethodno pročitala i otkriva da su podaci modifikovani drugom transakcijom (koja je potvrđena nakon inicijalnog čitanja).
* **Phantom read**
  + Transakcija ponovo izvršava upit vraćajući skup redova koji zadovoljavaju uslov pretrage i otkriva da se skup redova koji zadovoljavaju uslov promenio zbog druge nedavno izvršene transakcije.
* **Serialization anomaly**
  + Rezultat uspešnog izvršavanja grupe transakcija nije u skladu sa svim mogućim redosledima pokretanja tih transakcija jedne po jedne.

Tabela 1. Nivoi izolacije

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nivo izolacije** | **Dirty read** | **Nonrepeatable read** | **Phantom read** | **Serialization anomaly** |
| **Read uncommitted** | Nije moguće | Moguće | Moguće | Moguće |
| **Read committed** | Nije moguće | Moguće | Moguće | Moguće |
| **Repeatable read** | Nije moguće | Nije moguće | Nije moguće | Moguće |
| **Serializable** | Nije moguće | Nije moguće | Nije moguće | Nije moguće |

U PosgreSQL-u mogu se zahtevati bilo koja od ovih četiri standardnih izolacionih nivoa ali su samo tri izolaciona nivoa implementirana. Read Uncommitted se ponaša kao Read Commited. To je iz razloga zato što je to najlogičniji način mapirati standardne izolacione nivoe na PosgreSQL arhitekturu.

Za podešavanje nivoa transakcione izolacije može se koristiti komanda SET TRANSACTION koja je ranije opisana.

## Read Commited nivo izolacije

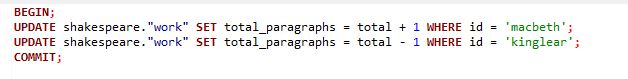
Podrazumevani izolacioni nivo u PostgreSQL-u je Read Committed. Kada transakcija koristi ovaj nivo izolacije, SELECT upit (bez klauzule FOR UPDATE/SHARE) vidi samo podatke koji su potvrđeni pre početka upita. Nikada ne vidi nepotvrđene podatke ili promene koje su potvrđene tokom izvršenja upita od strane konkurentnih transakcija. U suštini, SELECT upit vidi snapshot baze podataka u trenutku kada upit počinje da se izvršava. Međutim, SELECT vidi efekte prethodnih ažuriranja koja su izvršena unutar sopstvene transakcije, čak i ako nisu još uvek potvrđena. Treba napomenuti da dva uzastopna SELECT upita mogu videti različite podatke, čak i ako su unutar jedne transakcije, ako druge transakcije potvrde promene nakon što prvi SELECT počne i pre nego što drugi SELECT počne sa izvršenjem[5].

Komande UPDATE, DELETE, SELECT FOR UPDATE i SELECT FOR SHARE se ponašaju isto kao i SELECT u smislu traženja redova. Pronalaze samo one redove koji su bili potvrđeni u trenutku početka izvršavanja komande. Međutim, takav red bi mogao već biti izmenjen (ili obrisan ili zaključan) od strane neke druge paralelne transakcije u trenutku kada je pronađen. U ovom slučaju, sačekaće se da se prva ažurirajuća transakcija potvrdi ili poništi (ukoliko je još u toku). Ako se prva ažurirajuća transakcija poništi, njeni efekti se poništavaju i druga ažurirajuća transakcija će nastaviti sa ažuriranjem prvobitno pronađenog reda. Ako se prva ažurirajuća transakcija potvrdi, druga ažurirajuća transakcija će ignorisati red ako je obrisan od strane prve transakcije, inače će pokušati da primeni svoju operaciju na ažuriranu verziju reda. Uslov pretrage komande (WHERE klauzula) se ponovo izvršava da bi se videlo da li se ažurirana verzija reda i dalje poklapa sa uslovom pretrage. Ako je tako, druga ažurirajuća transakcija nastavlja sa svojom operacijom koristeći ažuriranu verziju reda. U slučaju SELECT FOR UPDATE i SELECT FOR SHARE, to znači da je to ažurirana verzija reda koja se zaključava i vraća klijentu.

Komanda INSERT sa klauzulom ON CONFLICT DO UPDATE ponaša se slično. U Read Committed modu, svaki red koji se predloži za unos će biti ili unešen ili ažuriran. Ako do konflikta dođe zbog druge transakcije čiji efekti nisu još uvek vidljivi za INSERT, UPDATE komanda će se izvršiti nad tim redom.

Komanda INSERT sa klauzulom ON CONFLICT DO NOTHING može sprečiti unos reda zbog ishoda druge transakcije čiji efekti nisu vidljivi za snapshot INSERT-a. Ovo važi samo u Read Committed modu.

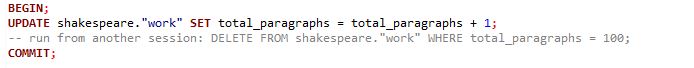
Zbog gore navedenih pravila, moguće je da ažurirajuća komanda vidi nekonzistentan snapshot. Može videti efekte istovremenih ažurirajućih komandi na istim redovima koje pokušava da ažurira, ali ne vidi efekte tih komandi na drugim redovima u bazi podataka. Ovo ponašanje čini Read Committed režim neprikladnim za komande koje uključuju složene uslove pretrage ali je u redu za jednostavne slučajeve.



Slika 10. Read Commited primer

Ako dve transakcije pokušavaju konkurentno da promene broj paragrafa dela sa id-em macbeth svakako nam je u cilju da druga transakcija vide promene koje je prva napravila. Pošto svaka komanda utiče samo na unapred određeni red, puštanje da vidi ažuriranu verziju reda ne stvara nekonzistentnost.

Složenija primena može da dovede do neželjenih rezulata. Uzmimo za primer DELETE komandu koja utiče na podatke koji se dodaju i brišu iz druge komande, pretpostavićemo da imamo redove sa vrednošću total\_paragraphs 99 i 100.



Slika 11. Read Commited primer

DELETE neće imati efekta iako postoji work.total\_paragraph = 100 red i posle ažuriranja. Ovo se dešava zato što je vrednost reda pre ažuriranja 99 preskočena, a kada se UPDATE završi i DELETE dobije lock, nova vrednost reda više nije 100 već 101 što više ne odgovara uslovu upita.

## Repeatable Read nivo izolacije

Nivo izolacije Repeatable Read vidi samo podatke koji su bili potvrđeni pre početka transakcije. Nikada ne vidi podatke koji nisu potvrđeni ili promene koje su potvrđene tokom izvršavanja konkurentnih transakcija. (Međutim, upit vidi efekte prethodnih ažuriranja izvršenih unutar svoje transakcije, iako još uvek nisu potvrđeni.) Ovo je snažnija garancija od one koja je zahtevana od strane SQL standarda za ovaj nivo izolacije i sprečava sve opisane anomalije osim serijalizacije. Kao što je pomenuto ranije, ovo je specifično dozvoljeno prateći standard, koji opisuje minimalan nivo zaštite koje svaki nivo izolacije mora da obezbedi.

Ovaj nivo se razlikuje od Read Committed po tome što uzastopni SELECT upiti unutar jedne transakcije vide iste podatke, odnosno ne vide promene koje su napravljene od strane drugih transakcija koje su bile potvrđene nakon što je trenutna transakcija počela. Aplikacije koje koriste ovaj nivo moraju biti spremne da više puta izvrše transakcije zbog potencijalnih grešaka u serijalizaciji.

UPDATE, DELETE, MERGE, SELECT FOR UPDATE i SELECT FOR SHARE naredbe se ponašaju isto kao i SELECT sa vidika pretrage redova. Pronalaze samo redove koji su bili potvrđeni pre početka transakcije. Međutim, takav red može već biti ažuriran, obrisan ili zaključan od strane druge istovremene transakcije u trenutku pronalaska. U tom slučaju, transakcija će sačekati da se prva ažurirajuća transakcija potvrdi ili poništi (ako je još uvek u toku). Ako se prva transakcija poništi, njeni efekti se poništavaju i transakcija sa ponovljivim čitanjem može nastaviti sa ažuriranjem originalno pronađenog reda. Međutim, ako se prva transakcija potvrdi i ažurira ili obriše red, tada će transakcija sa ponovljivim čitanjem biti poništena sa porukom.

*ERROR: could not serialize access due to concurrent update*

jer transakcija sa ponovljivim čitanjem ne može izmeniti ili zaključati redove koji su promenjeni od strane drugih transakcija nakon početka transakcije sa ponovljivim čitanjem.

Kada aplikacija primi ovu poruku o grešci, treba prekinuti trenutnu transakciju i ponovo je pokrenuti. U narednom pokušaju transakcija će videti prethodno potvrđenu promenu na početku izvršenja, tako da ne postoji konflikt u korišćenju nove verzije reda kao polazne tačke za izvršenje nove transakcije.

Treba napomenuti da samo ažurirajuće transakcije mogu zahtevati ponovno pokretanje, dok čitajuće transakcije nikada neće imati konflikte serijalizacije.

Režim Repeatable Read pruža strogu garanciju da svaka transakcija vidi potpuno stabilan prikaz baze podataka. Međutim, ovaj prikaz neće uvek biti konzistentan sa nekim serijskim izvršavanjem konkurentnih transakcija istog nivoa.

## Serializable nivo izolacije

Nivo izolacije Serializable pruža najstrožu izolaciju transakcija. Ovaj nivo simulira izvršavanje transakcija kao da su izvršene sekvencijalno, jedna za drugom. Međutim, kao i nivo Repeatable Read, aplikacije koje koriste ovaj nivo moraju biti spremne izvrše ponovo transakciju ukoliko dođe do serializacijske greške. Ovaj nivo izolacije se ponaša kao i Repeatable Read, osim što prati uslove koji mogu dovesti do toga da konkurentne serijske transakcije ne budu konzistentne sa svim mogućim sekvencijnim izvršavanjima tih transakcija. Ovaj nadzor ne uvodi nikakvo blokiranje osim onog koje već postoji kod Repeatable Read, ali postoji dodatni trošak zbog nadzora, a otkrivanje uslova koji mogu dovesti do anomalije serializacije će izazvati neuspeh serijalizacije.

Uzmimo za primer tabelu mojatabela koja inicijalno sadrži sledeće podatke. Pretpostavimo da serializable transakcija A izvršava upit:



Slika 12. Upit 1 nad tabelom mojatabela

i onda upisuje rezultat 30 kao vrednost u novi red sa vrednošću za klasu 2. Konkurentno, serializable transakcija B izvršava:



Slika 13. Upit 2 nad tabelom mojatabela

i dobija rezultat 300 koji upisuje u novi red sa vrenošću klase 1.

Tabela 2. mojatabela

|  |  |
| --- | --- |
| Klasa | Vrednost |
| 1 | 10 |
| 1 | 20 |
| 2 | 100 |
| 2 | 200 |

Onda obe transakcije pokušavaju da potvrde svoje operacije. Jednoj transakciji će biti dozvoljeno da potvrdi svoje promene dok će druga morati da odradi roll back sa porukom:

*ERROR: could not serialize access due to read/write dependencies among transactions*

Da je A se izvršila pre B, B bi izračunala sumu 330, a ne 300. Slično tome, drugi redosled bi rezultirao drugom sumom za transakciju A.

Kada se oslanjamo na serijalizabilne transakcije kako bismo sprečili anomalije, važno je da se podaci pročitani iz trajne korisničke tabele ne smatraju validnim sve dok transakcija koja ih je pročitala nije uspešno potvrđena. Ovo važi čak i za read-only transakcije, osim u slučaju *deferrable* read-only transakcija gde se podaci koji su pročitani smatraju validnim odmah po čitanju, jer takva transakcija čeka da dobije *snapshot* koji sigurno nema takvih problema pre nego što počne sa čitanjem podataka. U svim drugim slučajevima, aplikacije ne smeju da se oslanjaju na rezultate pročitane u toku transakcije koja je kasnije otkazana. Umesto toga, trebalo bi da pokušaju ponovno izvršavanje transakcije sve dok se ona uspešno ne izvrši.

Da bi se garantovala prava serijalizacija, PostgreSQL koristi *predikativno zaključavanje*, što znači da čuva zaključavanja koja mu omogućavaju da utvrdi kada bi upis imao uticaj na rezultat prethodnog čitanja iz konkurentne transakcije u slučaju da je prva izvršena. U PostgreSQL-u ova zaključavanja ne izazivaju blokiranje i stoga ne mogu da učestvuju u prouzrokovanju deadlock-a. Koriste se kako bi se identifikovale i označile zavisnosti među konkurentnim serijalizabilnim transakcijama koje u određenim kombinacijama mogu dovesti do serijalizacijskih anomalija. Za razliku od toga, transakcija koja koristi Read Committed ili Repeatable Read nivo izolacije kako bi obezbedila konzistentnost podataka može da zaključa celu tabelu, što može blokirati druge korisnike koji pokušavaju da koriste tu tabelu, ili može koristiti SELECT FOR UPDATE ili SELECT FOR SHARE koji ne samo da mogu blokirati druge transakcije, već mogu izazvati i pristup disku.

Konzistentna upotreba serijalizabilnih transakcija može olakšati razvoj. Garancija da će bilo koji skup uspešno potvrđenih konkurentnih serijalizabilnih transakcija imati isti efekat kao da su izvršene jedna za drugom znači da, ako možete dokazati da će jedna transakcija, kako je napisana, ispravno obaviti svoj posao kada se izvrši sama, možete imati poverenja da će ispravno obaviti svoj posao u bilo kojoj kombinaciji serijalizabilnih transakcija, čak i bez ikakvih informacija o tome šta bi druge transakcije mogle da urade, ili se neće uspešno potvrditi. Važno je da okruženje koje koristi ovu tehniku ima generalizovan način za rukovanje greškama serijalizacije (koje uvek vraćaju SQLSTATE vrednost '40001'), jer će biti veoma teško predvideti tačno koje transakcije mogu doprineti zavisnostima čitanja/upisa i moraju biti poništene radi sprečavanja serijalizacijskih anomalija. Praćenje zavisnosti čitanja/upisa ima svoju cenu, kao i ponovno pokretanje transakcija koje su završene greškom serijalizacije, ali u odnosu na cenu i blokiranje koje su uključeni u upotrebu eksplicitnih zaključavanja i SELECT FOR UPDATE ili SELECT FOR SHARE, serijalizabilne transakcije su najbolji izbor za performanse u određenim okruženjima.

# Zaključavanje

PostgreSQL pruža različite režime zaključavanja kako bi kontrolisao paralelni pristup podacima u tabelama. Ovi režimi mogu se koristiti za zaključavanje kontrolisano od strane aplikacije u situacijama kada MVCC ne daje željeno ponašanje. Takođe, većina PostgreSQL komandi automatski dobija zaključavanje odgovarajućih režima kako bi osigurale da se referencirane tabele ne brišu ili modifikuju na nepodržane načine tokom izvršavanja komande[6].

## Zaključavanja na nivou tabele

U nastavku je prikazan spisak dostupnih režima zaključavanja i konteksti u kojima se automatski koriste u PostgreSQL-u. Bilo koje zaključavanje se može koristiti komandom LOCK. Do neke mere, imena prikazuju tipičnu upotrebu svakog režima zaključavanja, ali semantika je ista za sve. Jedina prava razlika između jednog režima zaključavanja i drugog je skup režima zaključavanja sa kojima svaki od njih izaziva konflikt (tabela 3). Dve transakcije ne mogu istovremeno držati zaključavanje konfliktnih režima na istoj tabeli. Zaključavanja kod kojih ne dolazi do konflikta mogu biti korišćena istovremeno od strane više transakcija. Svi ovi režimi zaključavanja su nivoa tabele, čak i ako ime sadrži reč "row".

**ACCESS SHARE (AccessShareLock)**

SELECT naredba dobijaju zaključavanje ovog režima na referenciranim tabelama. Generalno, bilo koji upit koji samo čita tabelu i ne menja je dobija zaključavanje ove vrste.

**ROW SHARE (RowShareLock)**

SELECT naredba dobijaja zaključavanje ovog režima na svim tabelama na kojima je specificirana jedna od opcija FOR UPDATE, FOR NO KEY UPDATE, FOR SHARE ili FOR KEY SHARE.

**ROW EXCLUSIVE (RowExclusiveLock)**

Komande UPDATE, DELETE, INSERT i MERGE dobijaju zaključavanje ovog tipa na zadatim tabelama. Generalno, ovaj režim zaključavanja će biti dobijen od strane svake naredbe koja vrši izmene nad podacima u tabeli.

**SHARE UPDATE EXCLUSIVE (ShareUpdateExclusiveLock)**

Ovaj režim štiti tabelu od istovremenih promena šeme tabele i izvršenja VACUUM operacija. Dodeljuje se naredbama VACUUM (bez FULL), ANALYZE, CREATE INDEX CONCURRENTLY, CREATE STATISTICS, COMMENT ON, REINDEX CONCURRENTLY i određenim ALTER INDEX i ALTER TABLE varijantama.

**SHARE (ShareLock)**

Ovaj režim štiti tabelu od istovremenih promena podataka. Dodeljuje se naredbama CREATE INDEX (bez CONCURRENTLY).

**SHARE ROW EXCLUSIVE (ShareRowExclusiveLock)**

Ovaj režim štiti tabelu od istovremenih promena podataka i ima ekskluzivnost samo za sebe, tako da samo jedna sesija može da ga poseduje u isto vreme. Dodeljuje se naredbama CREATE TRIGGER i nekim oblicima ALTER TABLE.

**EXCLUSIVE (ExclusiveLock)**

Ovaj režim dozvoljava samo istovremena ACCESS SHARE zaključavanja, tj. samo čitanja sa tabele mogu da se izvršavaju paralelno sa transakcijom koja drži ovaj režim zaključavanja. Dodeljuje se naredbi REFRESH MATERIALIZED VIEW CONCURRENTLY.

**ACCESS EXCLUSIVE (AccessExclusiveLock)**

Ovaj režim garantuje da je vlasnik jedina transakcija koja pristupa tabeli na bilo koji način. Dodeljuje se naredbama DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL i REFRESH MATERIALIZED VIEW (bez opcije CONCURRENTLY). Mnoge forme naredbi ALTER INDEX i ALTER TABLE takođe dodeljuju zaključavanje na ovom nivou. Ovo je takođe podrazumevani režim zaključavanja za naredbe LOCK TABLE koje eksplicitno ne navode režim.

Jednom kada je zaključavanje dobijeno, obično se zadržava do kraja transakcije. Međutim, ako se zaključavanje dobije nakon postavljanja tačke čuvanja (savepoint), zaključavanje se oslobađa ako se tačka snimanja poništi. Ovo je u skladu sa principom da ROLLBACK poništava sve efekte komandi od tačke snimanja. Isto važi i za zaključavanje dobijeno unutar PL/pgSQL bloka greške. Prouzrokovana greška oslobađa zaključavanje dobijeno unutar bloka.

Tabela 3. Režimi zaključavanja koji izazivaju konflikte

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Zahtevani režim zaključavanja** | **Postojeći režim zaključavanja** | | | | | | | |
| **ACCESS SHARE** | **ROW SHARE** | **ROW EXCL.** | **SHARE UPDATE EXCL.** | **SHARE** | **SHARE ROW EXCL.** | **EXCL.** | **ACCESS EXCL.** |
| ACCESS SHARE |  |  |  |  |  |  |  | X |
| ROW SHARE |  |  |  |  |  |  | X | X |
| ROW EXCL. |  |  |  |  |  | X | X | X |
| SHARE UPDATE EXCL. |  |  |  |  | X | X | X | X |
| SHARE |  |  |  | X | X | X | X | X |
| SHARE ROW EXCL. |  |  | X | X | X | X | X | X |
| EXCL. |  | X | X | X | X | X | X | X |
| ACCESS EXCL. | X | X | X | X | X | X | X | X |

## Zaključavanja na nivou reda

Pored zaključavanja na nivou tabele, postoje i zaključavanja na nivou reda koja su navedena u nastavku zajedno sa kontekstima u kojima se automatski koriste u PostgreSQL-u. U tabeli 4 su prikazani svi mogući konflikti zaključavanja na nivou reda. Treba imati na umu da transakcija može držati konfliktna zaključavanja na istom redu, čak i u različitim podtransakcijama, ali osim toga, dve transakcije nikada ne mogu držati konfliktna zaključavanja u istom redu. Zaključavanja na nivou reda ne utiču na upite podataka. Blokiraju samo upise i zaključavanja na istom redu. Zaključavanja na nivou reda se oslobađaju na kraju transakcije ili tokom vraćanja na tačku čuvanja, baš kao i zaključavanja na nivou tabele.

**FOR UPDATE**

Korišćenje režima FOR UPDATE uzrokuje zaključavanje redova koji su povučeni SELECT naredbom, kao da su namenjeni za ažuriranje. To sprečava da ih druge transakcije zaključaju, izmene ili izbrišu dok se trenutna transakcija ne završi. Drugim rečima, druge transakcije koje pokušavaju da izvrše UPDATE, DELETE, SELECT FOR UPDATE, SELECT FOR NO KEY UPDATE, SELECT FOR SHARE ili SELECT FOR KEY SHARE nad ovim redovima će biti blokirane dok se trenutna transakcija ne završi. Obrnuto, SELECT FOR UPDATE će čekati da se izvrši konkurentna transakcija koja je izvršila bilo koju od tih naredbi nad istim redom, a zatim će zaključati i vratiti ažurirani red (ili nijedan red ako je red izbrisan). Međutim, u okviru transakcije sa nivoom izolacije REPEATABLE READ ili SERIALIZABLE, biće generisana greška ako red koji treba da se zaključa bude promenjen od početka transakcije. Režim zaključavanja FOR UPDATE takođe se dobija pri svakoj DELETE operaciji nad redom, kao i pri UPDATE koji menja vrednosti određenih kolona.

**FOR NO KEY UPDATE**

Ponaša se slično kao i FOR UPDATE, osim što je zaključavanje koje se dobija slabije. Ovo zaključavanje neće blokirati SELECT FOR KEY SHARE naredbe koje pokušavaju da dobiju zaključavanje na istim redovima. Ovaj režim zaključavanja se takođe dobija pri svakom UPDATE koji ne dobija FOR UPDATE zaključavanje.

**FOR SHARE**

Ponaša se slično kao i FOR NO KEY UPDATE, osim što dobija deljeno zaključavanje umesto ekskluzivnog zaključavanja na svakom dobijenom redu. Deljeno zaključavanje blokira druge transakcije da izvrše UPDATE, DELETE, SELECT FOR UPDATE ili SELECT FOR NO KEY UPDATE nad ovim redovima, ali im ne sprečava izvršavanje SELECT FOR SHARE ili SELECT FOR KEY SHARE naredbi.

**FOR KEY SHARE**

Ponaša se slično kao FOR SHARE, osim što je zaključavanje slabije: SELECT FOR UPDATE je blokiran, ali ne i SELECT FOR NO KEY UPDATE. Deljeno zaključavanje ključa blokira druge transakcije da izvrše DELETE ili bilo koju UPDATE koja menja vrednosti ključa, ali ne i druge UPDATE, niti sprečava SELECT FOR NO KEY UPDATE, SELECT FOR SHARE ili SELECT FOR KEY SHARE.

Tabela 4. Režimi zaključavanja koji izazivaju konflikte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Zahtevani režim zaključavanja** | **Postojeći režim zaključavanja** | | | |
| **FOR KEY SHARE** | **FOR SHARE** | **FOR NO KEY UPDATE** | **FOR UPDATE** |
| FOR KEY SHARE |  |  |  | X |
| FOR SHARE |  |  | X | X |
| FOR NO KEY UPDATE |  | X | X | X |
| FOR UPDATE | X | X | X | X |

## Zastoji

Upotreba eksplicitnog zaključavanja može povećati verovatnoću pojave zastoja(deadlock), u kojima dve ili više transakcije drže zaključavanja koja druga želi. Na primer, ako transakcija 1 stekne zaključavanje na tabeli A, a zatim pokuša da stekne zaključavanje na tabeli B, dok transakcija 2 već ima zaključavanje na tabeli B i sada želi zaključavanje na tabeli A, tada nijedna od njih ne može nastaviti dalje. PostgreSQL automatski otkriva situacije zastoja i rešava ih tako što prekida jednu od transakcija, omogućavajući drugim da se završe. Nikad se ne zna koja će transakcija biti prekinuta jer je teško predvideti.

Treba imati na umu da se zastoji mogu pojaviti i kao rezultat zaključavanja na nivou reda. Prema tome, mogu se desiti čak i ako se eksplicitna zaključavanja ne koriste. Razmotrićemo slučaj u kojem dve istovremene transakcije menjaju tabelu. Prva transakcija se izvršava:



Slika 14. Zastoj primer 1

Ovim se dobija zaključavanje na nivou reda na redu sa navedenim id-em dela. Zatim se druga transakcija izvršava:



Slika 15. Zastoj primer 2

Prva naredba UPDATE uspešno dobija zaključavanje na nivou reda, tako da uspeva da ažurira taj red. Međutim, druga naredba UPDATE vidi da je red koji pokušava da ažurira već zaključan, tako da čeka da se transakcija koja ga je zaključala završi. Sada se nastavlja izvršenje transakcije jedan.



Slika 16. Zastoj primer 3

Transakcija jedan pokušava da dobije zaključavanje na nivou reda, ali ne nije moguće jer transakcija dva već drži takvo zaključavanje. Zbog toga, transakcija jedan čeka da se transakcija dva završi. Kao rezultat, transakcija jedan je blokirana zbog transakcije dva, a transakcija dva je blokirana zbog transakcije jedan, što znači da je došlo do zastoja(deadlock).

PostgreSQL će detektovati ovu situaciju i prekinuti jednu od transakcija.

Najbolja odbrana protiv zastoja je da sve aplikacije koje koriste bazu podataka dobijaju zaključavanja na više objekata u pravilnom redosledu. U datom primeru, ako bi obe transakcije ažurirale redove u istom redosledu, zastoj se ne bi dogodio. Takođe, trebalo bi obezbediti da prvo dobijeno zaključavanje na objektu u transakciji bude najrestriktivniji režim koji je potreban za taj objekat. Ako ovo nije moguće unapred proveriti, onda se zastoji mogu rešiti u hodu ponovnim pokušajem transakcija koje se prekidaju zbog zastoja.

Sve dok se ne detektuje zastoj, transakcija koja pokušava da dobije zaključavanje na nivou tabele ili reda će čekati neograničeno dugo dok se ne oslobode konfliktna zaključavanja. To znači da nije dobra ideja da aplikacije drže otvorene transakcije tokom dužeg vremenskog perioda, na primer dok čekaju neki unos sa korisničke strane.

## Savetodavna zaključavanja

PostgreSQL pruža mogućnost kreiranja zaključavanja sa aplikaciono-definisanim značenjem koja se nazivaju "savetodavna zaključavanja" (advisory locks). Ova zaključavanja se ne sprovode automatski od strane sistema već je odgovornost na aplikaciji da ih ispravno koristi. Na primer, često se koriste za emuliranje pesimističkih strategija zaključavanja koje su tipične za tzv. "flat file" sisteme upravljanja podacima. Iako bi se ista svrha mogla postići korišćenjem polja u tabeli, savetodavna zaključavanja su brža, izbegavaju povećanje veličine tabele i automatski se oslobađaju od strane servera na kraju sesije.

Postoje dva načina za dobijanje savetodavnog zaključavanja u PostgreSQL-u: na nivou sesije ili na nivou transakcije. Kada se savetodavno zaključavanje dobije na nivou sesije, ono se zadržava sve dok se eksplicitno ne oslobodi ili dok se sesija ne završi. Za razliku od standardnih zahteva za zaključavanje, zahtevi za savetodavno zaključavanje na nivou sesije ne poštuju semantiku transakcija. Zaključavanje stečeno tokom transakcije koja kasnije bude poništena i dalje će biti zadržano nakon poništavanja transakcije, a isto tako, otključavanje je efektivno čak i ako transakcija koja je pozvala otključavanje kasnije ne uspe. Jedan proces može dobiti savetodavno zaključavanje više puta. Za svaki završeni zahtev za zaključavanje mora postojati odgovarajući zahtev za otključavanje pre nego što se zaključavanje zaista oslobodi. Sa druge strane, zahtevi za savetodavno zaključavanje na nivou transakcije se ponašaju slično kao obični zahtevi za zaključavanje: automatski se oslobađaju na kraju transakcije i ne postoji eksplicitna operacija otključavanja. Ovo ponašanje je često pogodnije za kratkotrajno korišćenje savetodavnog zaključavanja. Zahtevi za zaključavanje na nivou sesije i nivou transakcije za isti identifikator savetodavnog zaključavanja će blokirati jedni druge na očekivani način. Ako sesija već poseduje određeno savetodavno zaključavanje, dodatni zahtevi od strane iste sesije će uvek uspeti, čak i ako druge sesije čekaju na to zaključavanje. Ovo važi bez obzira da li postojeće zaključavanje i novi zahtevi su na nivou sesije ili nivou transakcije.

Kao i za sve zaključavanja u PostgreSQL-u, potpuni spisak savetodavnih zaključavanja koja trenutno poseduje bilo koja sesija može se pronaći u sistemu pregleda pg\_locks.

# Zaključak

U ovom seminarskom radu je fokus bio na obradi transakcija, planovima izvršavanja, izolaciji i zaključavanju u kontekstu baza podataka. Objašnjeno je da su transakcije osnovne jedinice izvršavanja u bazi podataka koje obezbeđuju ACID svojstva i da je upravljanje transakcijama važan aspekt pravilnog funkcionisanja baza podataka.

Ukratko su objašnjeni planovi izvršavanja transakcija koji omogućavaju optimizaciju upita putem analize i odabira najefikasnijeg načina izvršavanja. Prikazan je primer plana generisanja upita kod transakcija..

Obrađen je pojam izolacije koji je bitan za održavanje konzistentnosti i izbegavanje konflikata između konkurentnih transakcija. Razmatrani su različiti nivoi izolacije, poput Read Commited, Repeatable Read i Serializable, koji definišu vidljivost promena izvršenih u drugim transakcijama.

Takođe je objašnjen pojam zaključavanja koji je ključni mehanizam za održavanje konzistentnosti i sprečavanje konflikta prilikom pristupa deljenim resursima. Prikazano je zaključavanje na nivou tabele i reda, kao i uloga savetodavnog zaključavanja.

U ovom radu su samo obrađene osnove obrade transakcija, planova izvršavanja, izolacije i zaključavanja u PostgreSQL bazi podataka. Dalje istraživanje i detaljnije proučavanje ovih tema omogućava dublje razumevanje performansi, skalabilnosti i sigurnosti baza podataka.

Generalno, razumevanje obrade transakcija, planova izvršavanja, izolacije i zaključavanja je od suštinske važnosti za pravilno projektovanje, implementaciju i održavanje baza podataka. Korišćenje odgovarajućih mehanizama i tehnika može poboljšati performanse, konzistentnost i bezbednost sistema, što dovodi do efikasnijeg i pouzdanijeg rada aplikacija zasnovanih na bazama podataka.

# Literatura

1. https://github.com/catherinedevlin/opensourceshakespeare [Accessed: 5-Maj-2023].
2. https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions [Accessed: 10-Maj-2023].
3. https://www.postgresql.org/docs/current/sql-commands.html [Accessed: 12-Maj-2023].
4. https://www.postgresql.org/docs/current/mvcc.html [Accessed: 13-Maj -2023].
5. https://www.postgresql.org/docs/current/transaction-iso.html [Accessed: 15-Maj -2023].
6. <https://www.postgresql.org/docs/current/explicit-locking.html> [Accessed: 18-Maj -2023].