Elektronski fakultet Niš

Sistemi za upravljanje bazama podataka

Obrada transakcija, planovi izvršenja transakcija, izolacija i zaključavanje kod PostgreSQL baze podataka

Seminarski rad

Mentor:
Doc. dr Aleksandar Stanimirović

Student:
Julije Kostov 1026

Sadržaj

1. Uvod

Transakcije u bazama podataka predstavljaju logičku jedinicu posla koji se izvršava nad podacima u okviru sistema za upravljanje bazama podataka. Sastoje se od niza operacija koje se izvršavaju nad podacima. Transakcije moraju da omoguće oporavak od greške u toku izvršenja, kao i to da nakon izvršenja baza bude u konzistentnom stanju.

Kako bazu podataka može da koristi veći broj korisnika istovremeno, nad bazom se može izvršavati i veći broj transakcija. Transakcije moraju da omoguće izolaciju prilikom istovremenog pristupanja podacima.

Transakcija nad bazom podataka mora da bude atomična (eng. *atomicity*), konzistentna (eng. *consistency*), izolovana (eng. *isolated*) i trajna (eng. *durable*), drugim rečima transakcija treba da ima ACID svojstva. **Atomičnost** kod transakcija zahteva izvršenje svih operacija u okviru transakcije, tj. ili se sve operacije izvršavaju ili nijedna. **Konzistentnost** osigurava da transakcija konzistentnu bazu iz jednog stanja prevodi u drugo stanje koje je takođe konzistentno i poštuje ograničenja integriteta. **Izloacija** omogućava izvršenje konkurentnih transakcija tako da dobijeno stanje je jednako stanju kada se konkurente transakcije izvršavaju sekvencijalno. **Trajnost** obezbeđuje da se nakon izvršenja transakcije podaci trajno sačuvaju na disk [1].

Transakcije obezbeđuju izvršenje svih operacije ili nijedne u slučaju greške. One se koriste za grupe operacija gde njihovo delimično izvršenje ne ostavlja bazu u konzistentnom stanju. Primer transakcije je uplata 100 dinara sa jednog računa na drugi račun, što zahteva smanjenje iznosa sa jednog računa za 100 dinara i povećanje iznosa drugog računa za 100 dinara. Ako se jedna operacije ne izvrši zbog pada sistema ili konkurentnog pristupa podacima sistem nije u konzistentnom stanju. Korišćenjem transakcija izbegava se ovaj problem i omogućava se da se izvrše ili obe operacije ili nijedna.

Većina današnjih baza podataka koje imaju ACID svojstva podržavaju transakcije. Postoje različiti nivoi izolacije transakcije, a najveći je nivo serijalizacije koji garantuje da je izvršenje konkurentnih transakcija jednog njihovom sekvencijalnom izvršenju. Transakcija u SQL-u se sastoji od sledećih delova:

- Startovanje transakcije naredbom
- Niz operacija koje vrše manipulaciju nad podacima
- U slučaju izvršenja bez greške potvrđuje se transakcija
- U slučaju greške vrši se vraćanje nastalih promena na prethodno stanje

Nakon potvrđivanja transakcije promene u njoj se iz prostora transakcije upisuju u bazu i trajno pamte. Vraćanje transakcije odbacuje sve promene nastale u prostoru transakcije i ne menja bazu.

Postoji više načina za implementaciju transakcija u sistemu kao što su zaključavanje podataka ili kreiranje više različitih verzija.

U distribuiranim bazama podataka postoje i distribuirane transakcije kod kojih različiti čvorovi baze sadrže delove podataka kojima se manipuliše u transakciji. Jedan od načina za implementaciju distribuiranih transakcija je dvofazna potvrda (eng. *two-phased commit*).

Ovaj rad opisuje implementaciju transakcija u PostgreSQL bazi podataka u drugom poglavlju, a u trećem poglavlju dati su praktični primeri korišćenja transakcija.

2. PostgreSQL transakcije

PostreSQL omogućava različite načine za upravljanje konkurentnim pristupom podacima. Interno konzistencija se postiže korišćenjem multiverzionog modela (eng. *Multiversion Concurrency Control* ili *MVCC*). To znači da svaka SQL naredba koristi snepšot (eng. *snapshot*) podataka bez obzira na trenutno stanje podataka. Ovo omogućava da se naredba zaštiti od nekonzistentnih podataka koji se generišu promenama iz drugih transakcija koje se izvršavaju u tom trenutku. Ovim se postiže izolacija transakcija za svaku otvorenu sesiju ka bazi. MVCC se se ovde koristi umesto klasičnog zaključavanja podataka kako bi se smanjilo čekanja na otključavanje podataka [2].

Glavna prednost korišćenja MVCC modela za kontrolu konkurentnog pristupa umesto zaključavanja je što MVCC zaključavanja za čitanje podataka nisu u sukobu sa MVCC zaključavanjima za upis podataka. Time se postiže da čitanje ne blokira upis i obrnuto. PostreSQL podržava ovo i prilikom korišćenja najstrožeg nivoa izolacije pomoću serijskog snepšota izolacije (eng. *Serializable Snapshop Isolation* ili *SLL*).

PostgreSQL podržava i zaključavanje tabela i redova podataka, ali pravilno korišćenje MVCC-a dalje bolje performanse od zaključavanja.

2.1 MVCC

Prilikom početka svake transakcije, svakoj transakciji se dodeljuje identifikator (txid) od strane menadžera transakcija koji predstavlja 32 bitni ceo broj. Postgres upisuje informacije o transakciji za svaki red u bazi. Na osnovu ovih informacija određuje se da li je neki redi dostupan transakciji ili ne. Kada se izvrši dodavanje novog reda koristi se txid trenutne transakcije i upisuje se u polje xmin pored podataka o redu. Svaki red koji je potvrđen i koji ima xmin manji od trenutnog identifikatora transakcije dostupan je toj transakciji. Pre nego što se izvrši komanda COMMIT red nije vidljiv u okviru drugih transakcija. Sličan mehanizam koristi se prilikom ažuriranja i brisanja redova, u tom slučaju se upisuje vrednost u polje xmax. Polja xmin i xmax nisu vidljiva prilikom preuzimanja podataka i PostgreSQL ih interno koristi, ali eksplicitnim navođenjem moguće je dobiti informacije o njima. Primer komande:

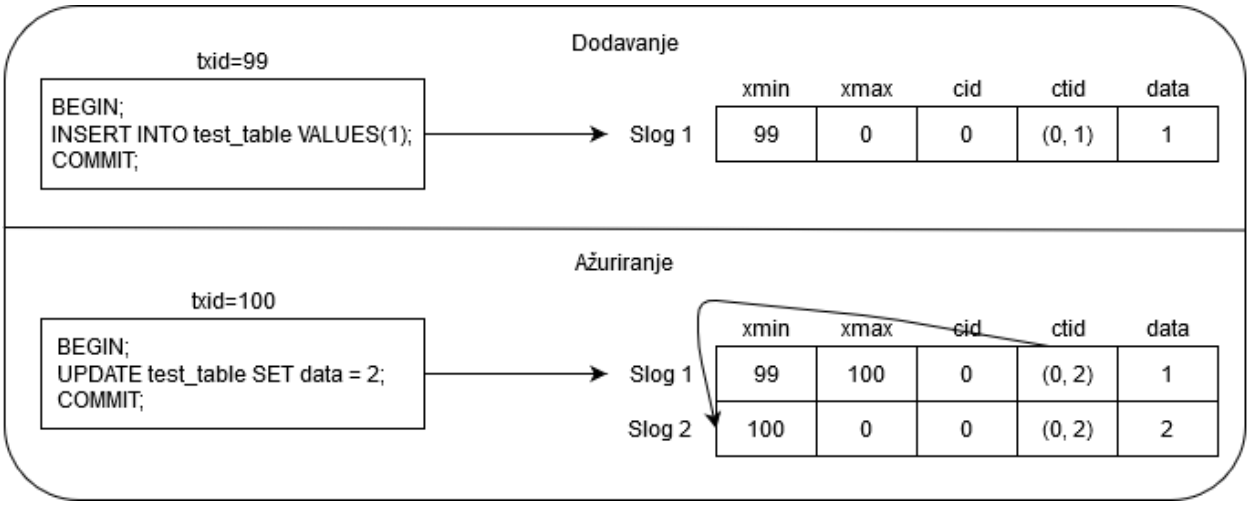
SELECT *, xmin, xmax FROM <ime_tabele>;

Za preuzimanje indentifikatora trenutne transakcije koristi se komanda:

SELECT txid_current();

Pored podataka za svaki red u tabeli u datoteku koja čuva podatke za svaki red se upisuje i **xmin** koji sadrži identifikator transakcije koja je dodala taj red, **xmax** koji sadrži indentifikator transakcije koja je ažurirala ili obrisala taj red, **cid** koji predstavlja broj komande u okviru transakcije, kreće od 0 za svaku transakciju i **ctid** koji sadrži identifikator slog podataka. Ukoliko

red nije ažuriran ili obrisan onda je xmax 0. Polje ctid se koristi kako bi se identifikovao validan podatak jer se prilikom ažuriranje vrši dodavanje novog sloga a prethodni se ne briše odmah već mu se menja ctid na novi slog. Kasnije se prilikom procesa čišćenja pronalaze mrtvi slogovi i oni se brišu. Primer promene slogova nakon izvršenja operacija prikazan je na slici 2.1.



Slika 2.1 Uticaj operacija u transakcijama na slogove

Sa slike 2.1 vidi se da se prilikom ažuriranja dodaje novi slog dok se ostali ne menja, osim podataka koji se koriste od strane Postgresa.

Kombinaciju statusa transakcija kao i vrednosti xmin i xmax PostgreSQL koristi kako bi obezbedio različite nivoe izolacija u transakcijama.

2.2 Izolacija transakcija

SQL standard definiše 4 nivoa izolacije transakcija. Najstroži nivo je nivo serijalizacije koji je definisan u uvodu ovog rada. Ostala tri nivoa definisana su pomoću pojava koje se ne smeju dogoditi prilikom interakcija konkurentnih transakcija.

Pojave koje mogu nastupiti prilikom interakcija konkurentnih transakcija:

- Prljavo čitanje (eng. *dirty read*) – jedna transakcija čita upisane podatke od strane druge transakcije koja nije potvrđena
- Neponovljivo čitanje (eng. *nonrepeatable read*) – transakcija ponovo čita podatke koje je već čitala a koje je u međuvremenu promenila druga transakcija koja je potvrđena
- Fantomsko čitanje (eng. *phantom read*) – transakcija ponovo izvršava upit koji vraća određene redove tabele koji se razlikuju od vraćenih redova prilikom prvog izvršenja zbog promena napravljenih od strane druge transakcije koja je potvrđena

- Anomalija serijalizacije – rezultat dobijen uspešnim izvršenjem grupe transakcija nije konzistentan sa rezultatom koji se dobija prilikom izvršenja jedne po jedne od tih transakcija u svim mogućim redosledima [2]

U tabeli 2.1 prikazan je SQL standard i nivoi izolacije implementirani u PostgreSQL bazi.

Nivo izolacije	**Prljavo čitanje**	**Neponovljivo čitanje**	**Fantomsko čitanje**	**Anomalija serijalizacije**
Nepotvrđeno čitanje	Dozvoljeno, ali ne u PG-u	Moguće	Moguće	Moguće
Potvrđeno čitanje	Nemoguće	Moguće	Moguće	Moguće
Ponovljivo čitanje	Nemoguće	Nemoguće	Dozvoljeno, ali ne u PG-u	Moguće
Serijalizacija	Nemoguće	Nemoguće	Nemoguće	Nemoguće

Tabela 2.1 Nivoi izolacije definisani SQL standardom i implementacija u PostgreSQL-u

Snepšot transakcije predstavlja skupo podataka koji sadrže informacije o aktivnim transakijama u datom trenutku. Transakcije u PostgreSQL-u imaju 4 stanja: IN_PROGRESS, COMMITED, ABORTED I SUB_COMMITED. Snepšot transakcije se dobija od strane menadžera transakcija. U zavisnosti od nivoa izolacije transakcije snepšot može da se kreira za svaku SQL komandu u okviru transakcije (READ COMMITED) ili pre izvršenja prve SQL komande transakcije (REPEATABLE READ i SERIALIZABLE).

2.2.1 Potvrđeno čitanje nivo izolacije

Potvrđeno čitanje (eng. *read* commited) predstavlja podrazumevani nivo izolacije u Postgres-u. Kada transakcija koristi ovaj nivo izolacije, SELECT upit vidi podatke koji su upisani i potvrđeni pre početka upita, a ne vidi podatke koji nisu potvrđeni, kao i potvrđene promene nad podacima u toku izvršenja upita od strane konkurentnih transakcija. Drugim rečima SELECT upit vidi snepšot podataka koji je kreiran prilikom početka upita. Podaci promenjeni u istoj transakciji u kojoj je SELECT upit su vidljivi. Osim toga ako je neka druga konkurentna transakcija završila upis i potvrdila promene i to je vidljivo u SELECT upitu. UPDATE, DELETE, SELECT FOR UPDATE i SELECT FOR SHARE komande se vide iste podatke kao i SELECT naredba što se tiče traženja podataka, međutim kako je moguće da je neki red već modifikovan, izbrisan ili zaključan od strane druge transakcije potrebno je sačekati da se transakcije koji su već napravile te promene potvrde ili odbace. Ukoliko se prva transakcija odbaci onda druga nastavlja ažuriranje, a ukoliko se prva potvrdi onda druga ignoriše red ukoliko ga je prva obrisala u suprotnom će pokušati da i ona izvrši ažuriranje reda. Ukoliko se prva transakcija potvrdi, WHERE deo druge se ponovo izvršava kako bi se proverilo da li red posle ažuriranja ispunjava uslove [2].

Ovako definisana pravila omogućuju da komanda za ažuriranje vidi nekonzistentan snepšot podataka, što se dešava zbog toga što komanda vidi promene od strane drugih transakcija nad

redovima koje ona želi da ažurira ali ne vidi promene nad ostalim redovima. Ovaj nivo je dovoljno dobar za komande koje nemaju složeno filtriranje. Primer:

UPDATE accounts SET balance = balance + 100 WHERE user_id = 1234;

UPDATE accounts SET balance = balance - 50 WHERE user_id = 1234;

Izvršenje ovih komandi u konkurentnim transakcijama omogućava da podaci budu u konzistentnom stanju koristeći potvrđeno čitanje nivo izolacije.

2.2.2 Ponovljivo čitanje nivo izolacije

Ponovljivo čitanje (eng. *repeatable* read) nivo izolacije vidi podatke koji su potvrđeni pre nego što je transakcija započeta i ne vidi nepotvrđene i promenjene podatke od strane drugih konkurentnih transakcija. Međutim promene nastale u okviru transakcije su vidljive u okviru nje. Iz tabele 2.1 vidi se da korišćenjem ovog nivoa izolacije nisu moguće sve pojave osim anomalije serijalizacije. Ovaj nivo se razlikuje od nivoa potvrđeno čitanje po tome što upit u transakciji vidi podatke od prve komande koja nije za kontrolu transakcije, a ne od starta trenutne komande u transakciji. SELECT ne vidi promene potvrđene od strane drugih transakcija nakon starta transakcije u kojoj se nalazi SELECT naredba. UPDATE, DELETE, SELECT FOR UPDATE i SELECT FOR SHARE komande se što se tiče pretraživanja podataka ponašaju kao i SELECT – vide podatke koji su potvrđeni pre starta transakcije. U slučaju da je red već ažuriran, obrisan ili zaključan od strane druge konkurentne transakcije tada prva transakcija čeka da se druga potvrdi ili ne potvrdi. Ukoliko se druga ne potvrdi tada prva nastavlja izvršenje. Ako se druga potvrdi i postoje podaci koji su ažurirani ili obrisani tada se prva ne potvrđuje i dolazi do greške. Kada dođe do ovakve greške aplikacije koje koriste ovaj nivo izolacije potrebno je da ponovo pokušaju izvršenje transakcije. Do ove greške dolazi samo ako transakcije ažurira podatke, dok se za transakcije koje samo čitaju podatke ova greška ne dešava [2].

2.2.3 Serijalizacija

Nivo serijalizacije (eng. serializable) predstavlja najveći nivo izolacije transakcija. Ovaj nivo simulira serijsko izvršenje svih potvrđenih transakcija, kao da su se izvršavale jedna nakon druge a ne konkurentno. Kao i kod ponovljivog čitanja aplikacije koje koriste ovaj nivo izolacije potrebno je da budu spremne da ponovo pokušaju izvršenje transakcije ukoliko dođe do greške zbog anomalije serijalizacije. Ovaj nivo radi tako što se vrši monitoring transakcija i proverava da li se dobijeni podaci ne slažu sa podacima svih mogućih kombinacija izvršenja transakcija. Kada se detektuje anomalija serijalizacije dolazi do greške serijalizacije i transakcija se ne potvrđuje.

Primer:

class	value
1	10
1	20
2	100
2	200

Tabela 2.2 Podaci tabele *test_table*

Komande koje se izvršavaju u konkurentnim transakcijama:

INSERT INTO test_table (2, (SELECT SUM(value) FROM test_table WHERE class = 1))

INSERT INTO test_table (1, (SELECT SUM(value) FROM test_table WHERE class = 2))

Ako se prva transakcija završi prva onda se u tabelu upisuju podaci (2, 30) i (1, 330), a ako se druga izvrši prva onda se upisuju podaci (1, 300) i (2, 330). Zbog ovoga ako transakcije koriste nivo serijalizacije onda se jedna transakcija potvrđuje a druga se ne potvrđuje i obaveštava se o nastaloj grešci.

Kako bi se smanjio broj grešaka koji se dobija korišćenjem nivoa serijalizacije koriste se ove tehnike [2]:

- Deklarisanje transakcije kao READ ONLY kada ona samo čita podatke
- Kontrola broja aktivnih konekcija korišćenjem ograničenog broja konekcija koje se ponovo koriste nakon oslobođenja
- Smanjenje komandi u okviru transakcije
- Korišćenje zaključavanja predikata koje ne vrši blokiranje već određuje da li rezultat može da utiče na stvaranje anomalije serijalizacije

2.3 Eksplicitno zaključavanje

Pored MVCC modela PostgreSQL pruža različite načina za zaključavanje podataka. PostgreSQL omogućava zaključavanje tabela, redova i strana.

2.3.1 Zaključavanje tabela

U listi ispod prikazani su mogući modovi zaključavanja kao i konteksti u kojima se oni koriste automatski od strane PostgreSQL-a. Ovi modovi se takođe mogu iskoristiti eksplicitnim navođenjem. Nakon što se izvrši zaključavanje određenim modom ono traje sve do završetka transakcije [2].

ACCESS SHARE

U konfliktu je samo sa ACCESS EXCLUSIVE modom zaključavanja.

SELECT komanda zaključava navedene tabele i u suštini svaki upit koji samo čita podatke bez modifikacije koristi ovaj mod zaključavanja.

ROW SHARE

U konfliktu je sa EXCLUSIVE i ACCESS EXCLUSIVE modovima zaključavanja.

SELECT FOR UPDATE i SELECT FOR SHARE komande koriste ovaj mod zaključavanja nad navedenim tabelama.

ROW EXCLUSIVE

U konfliktu je sa SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE modovima.

Komande UPDATE, DELETE i INSERT koriste ovaj mod zaključavanja, svaka komanda koja modifikuje podatke koristi ovaj mod zaključavanja.

SHARE UPDATE EXCLUSIVE

U konfliktu je sa SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE modovima.

Koriste ga komande VACUUM, ANALYZE, CREATE INDEX CONCURRENTLY, REINDEX CONCURRENTLY, CREATE STATISTICS, ALTER INDEX i ALTER TABLE.

SHARE

U konfliktu je sa ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE modovima. Ovaj mod štiti tabelu od konkurentnih promena nad podacima.

Koristi ga CREATE INDEX komanda.

SHARE ROW EXCLUSIVE

U konfliktu je sa ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE modovima. Ovaj mod štiti tabelu od konkurentnih promena i omogućava da ga samo jedna sesija koristi.

Komanda CREATE TRIGGER i neke forme ALTER TABLE komande ga koriste.

EXCLUSIVE

U konfliktu je sa ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE. Omogućava samo čitanje podataka paralelno sa transakcijom koja koristi ovaj mod.

Koristi ga REFRESH MATERIALIZED VIEW CONCURRENLTY.

ACCESS EXCLUSIVE

U konfktu je sa svim modovima. Obezbeđuje da pristup tabeli ima samo transakcija koja koristi ovaj mod zaključavanja.

Koriste ga DROP TABLE, TRUNCATE, REINDEX, CLUSTER VACUUM FULL, REFRESH MATERIALIZED VIEW komande. Takođe neke forme ALTER TABLE i ALTER INDEX komande ga koriste. Predstavlja podrazumevani mod za komandu LOCK TABLE bez eksplicitnog specificiranja moda.

Traženi mod zaključavanja	**Trenutni mod zaključavanja**							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

Tabela 2.3 Konflikti zaključavanja tabela

2.3.2 Zaključavanje redova

Pored zaključavanja na nivou tabela postoji i zaključavanje na nivou redova. Zaključavanje redova ne utiče na čitanje podataka već samo na upis i druga zaključavanja. U listi ispod prikazani su modovi zaključavanja redova kao i konteksti u kojima se oni koriste automatski od strane PostgreSQL-a [2].

FOR UPDATE

Redovi koji ulaze u SELECT upit se zaključavaju kada se oni modifikuju. Time se ne dozvoljava njihovo zaključavanje, ažuriranje ili brisanje od strane drugih transakcija dok se on koja ih je zaključala ne završi. Druge transakcije koje pokušaju da izvrše UPDATE, DELETE, SELECT FOR UPDATE, SELECT FOR NO KEY UPDATE, SELECT FOR SHARE ili SELECT FOR KEY SHARE komande nad ovim redovima se blokiraju dok se ne završi trenutna transakcija.

FOR UPDATE mod zaključavanja se koristi i prilikom brisanja određenih redova komandom DELETE, kao i prilikom ažuriranja određenih kolona UPDATE komandom.

FOR NO KEY UPDATE

Ponaša se slično kao i FOR UPDATE mod zaključavanja ali ima slabija ograničenja. Ovaj mod ne blokira SELECT FOR KEY SHARE komande koja pokuša da zaključa iste redove zaključane ovim modom. Ovaj mod se koristi kod UPDATE komande koje ne koriste FOR UPDATE mod.

FOR SHARE

Ponaša se slično kao FOR NO KEY UPDATE samo što umesto eksplicitnog zaključavanja koristi zaključavanje koje omogućava deljenje. Ovakvo zaključavanje blokira UPDATE, DELETE, SELECT FOR UPDATE i SELECT FOR NO KEY UPDATE komande nad zaključanim redovima ali ne blokira SELECT FOR SHARE i SELECT FOR NO KEY SHARE.

FOR KEY SHARE

Ponaša se slično kao i FOR KEY SHARE samo što je slabije zaključavanje koje ne blokira SELECT FOR NO KEY UPDATE ali blokira SELECT FOR UPDATE. Takođe ovaj mod blokira transakcije koje pokušavaju DELETE ili UPDATE komande koje menjaju vrednost ključa.

Traženi mod zaključavanja	Trenutni mod 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

Tabela 2.4 Konflikti zaključavanja redova

2.3.3 Zaključavanje strana

Pored zaključavanja tabela i redova postoje i deljena ili eksplicitna zaključavanja na nivou strana. Ova zaključavanja se koriste za kontrolu upisa i čitanja u stranama tabele. Ova zaključavanja se oslobađaju odmah nakon što je red pribavljen ili ažuriran. Obično se ne navode eksplicitno već je sam PostgreSQL zadužen za upravljanje ovakvim zaključavanjima [2].

2.4 Deadlock

Korišćenje eksplicitnih zaključavanja povećava šansu za stvaranje deadlock-ova, gde jedna transakcija koja zaključava podatke traži podatke koje je druga transakcija zaključala, a ona zahteva podatke koje je zaključala prva transakcija. PostgreSQL automatski detektuje deadlock-ove i rešava ih tako što jednu od transakcija prekida, ali koju transakciju prekida nije moguće proceniti.

3. Primeri korišćenja transakcija

Rad sa transakcijama izvršen je na udaljenoj virtuelnoj mašini gde je podignuta PostgreSQL baza podataka u okviru Docker kontejnera, komandom:

```
docker run -d --restart unless-stopped - POSTGRES_USER=root -e
POSTGRES_PASSWORD=dmbstest1232 -e POSTGRES_DB=dbms -p 5000:5432 postgres:12.2-alpine
```

Na podignutoj bazi kreirane su tabele *accounts* i *entries* naredbama:

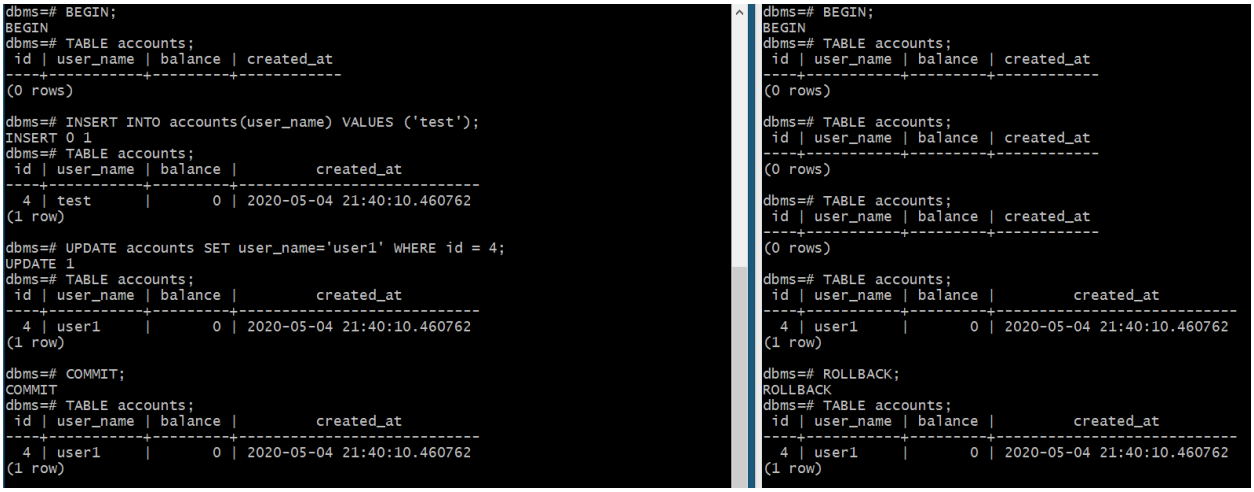
```
CREATE TABLE accounts (
   id SERIAL PRIMARY KEY,
   user_name TEXT NOT NULL,
   balance INT NOT NULL DEFAULT 0,
   created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE entries (
   id SERIAL PRIMARY KEY,
   amount INT NOT NULL DEFAULT 0,
   total INT NOT NULL DEFAULT 0,
   created_at TIMESTAMP NOT NULL DEFAULT NOW()
);
```

Naredba za početak transakcije je BEGIN; nakon koje slede naredbe koje je potrebno izvršiti u okviru transakcije. Transakcija se završava komandama COMMIT; što potvrđuje promene nastale komandama u okviru transakcije i trajno ih pamti ili ROLLBACK; što odbacuje sve promene nastale u okviru transakcije. Podrazumevani nivo izolacije prilikom startovanje transakcije bez navođenja je READ COMMITED. Kako bi promenili nivo izolacije transakcije potrebno je navesti nivo izolacije prilikom startovanja transakcije:

BEGIN TRANSACTION ISOLATION LEVEL <nivo_izolacije>;

Argument *nivo_izolacije* može da bude READ COMMITED, REPEATABLE READ i SERIALIZABLE. Pored toga postoji i READ UNCOMMITED koji je isti kao i READ COMMITED.

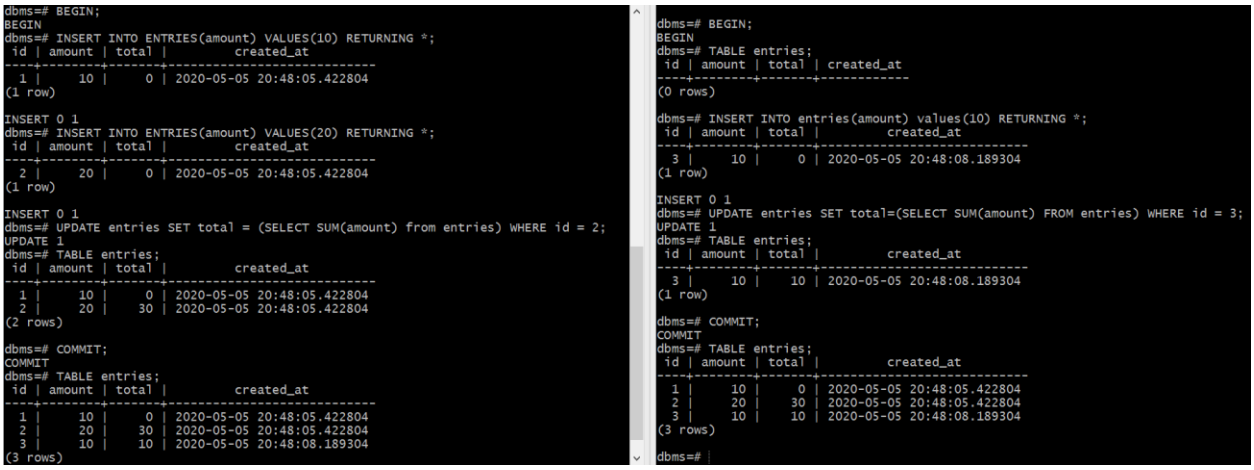
Primer transakcije:



Slika 3.1 Korišćenje transakcije

Na slici 3.1 vidi se da su otvorene dve sesije ka bazi. Obe sesije kreću komandom za startovanje transakcije, BEGIN. Na početku tabela *accounts* je prazna. Prva sesija izvršava komandu za unos i izmenu podataka. Nakon obe komande u drugoj sesiji se proverava stanje tabele i vidi se da je prazno. Nakon što prva sesija potvrdi podatke komandom COMMIT oni su vidljivi u okviru transakcije u drugoj sesiji. Druga sesija završava se komandom ROLLBACK što vraća odbacuje promene nastale u okviru transakcije, ali kako nema modifikacije podataka već samo čitanje stanje baze se ne menja.

Na slici 3.2 prikazane su transakcije sa podrazumevanim nivoom izolacije. U obe transakcije se pokušava manipulacija nad podacima:

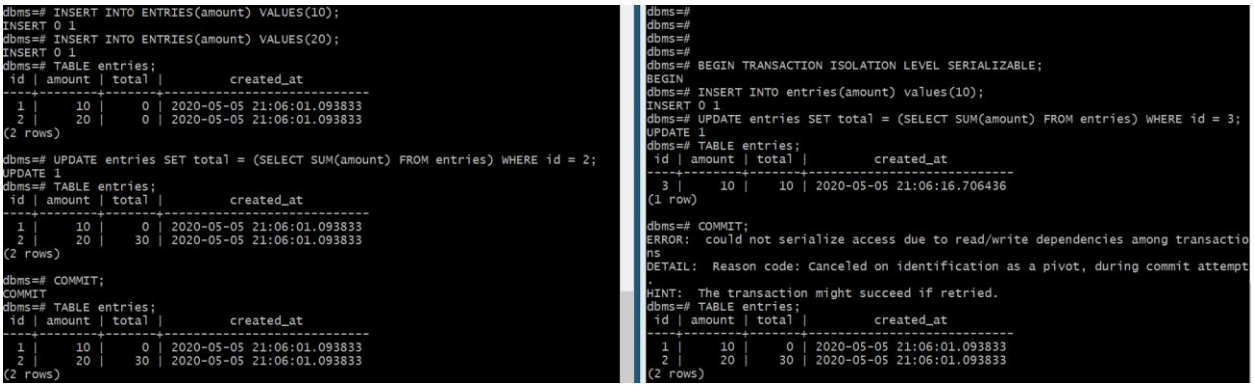


Slika 3.2 Transakcije koje pokušavaju modifikaciju podataka

Prva transakcije unosi dve vrednosti u tabelu pa vrši ažuriranje reda sa identifikatorom 2 tako što u njegovu kolonu *total* upisuje sumu vrednosti kolona *amount*. Druga transakcija nakon toga unosi jedan podatak i vrši ažuriranje reda sa identifikatorom 3 gre u *total* upisuje sumu vrednosti

kolone *amount*. Iako su podaci u prvoj transakciji kreirani pre ažuriranja u drugoj ona ne vidi te podatke što se može videti u rezultatu. Redovi sa identifikatorima 1 i 2 su kreirani pre reda sa identifikatorom 3 i ažuriranja što se može videti i po koloni *created_at.*

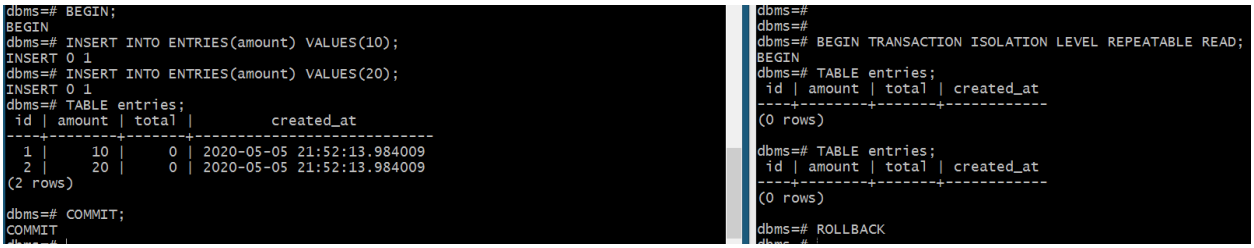
Na slici 3.3 prikazane su operacije u istom redosledu kao u primeru sa slike 3.2 ali nivo izolacije je najveći SERIALIZABLE:



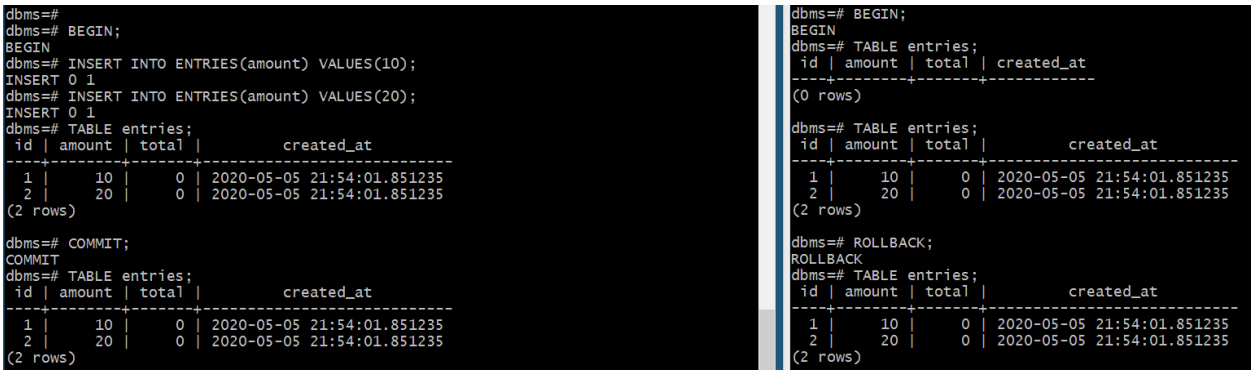
Slika 3.3 Transakcije sa SERIALIABLE nivoom izolacije

Sa slike 3.3 vide da je potvrđivanje prve transakcije prošlo, dok potvrđivanje druge nije i došlo je do greške. Prilikom korišćenja ovog nivoa izolacije više su moguće anomalije serijalizacije i zbog toga treba biti spreman za ponovno pokušavanje izvršenja transakcije. Ovde se javlja anomalija serijalizacije zbog toga što od redosleda izvršenja operacije u transakcijama zavisi krajnji rezultat, što se po ovom nivou izolacije ne dozvoljava.

Na slikama 3.4 i 3.5 prikazano je koje podatke vide transakcije sa READ COMMITED i REPEATABLE READ nivoima izolacije.



Slika 3.4 REPEATABLE READ nivo izolacije



Slika 3.5 READ COMMITED nivo izolacije

Na slici 3.4 prva transakcija je dodala podatke i potvrdila a zatim je druga transakcija sa REPEATABLE READ nivoom izolacije pročitala podatke i nije videla promene. Na slici 3.5 nakon što je prva transakcija unela podatke i potvrdila druga transakcija koja koristi READ COMMITED nivo izolacije u okviru transakcije vidi podatke koje je prva transakcija unela.

Pored izolacije moguće je eksplicitno zaključavanje tabela u okviru transakcije. Kako bi se zaključala tabela potrebno je koristiti komandu:

LOCK TABLE <ime_tabele> IN <mod_zaključavanja> MODE [NOWAIT];

Kada se u okviru transakcije zaključa tabela to zaključavanje traje sve dok se ne završi transakcija. NOWAIT specificira da je potrebno pokušati zaključati tabelu bez obzira na to da li je ona već zaključana. Ako je tabela zaključana i traženi nivo zaključavanja je u konfliktu sa trenutnim, zaključavanje ne uspeva i dolazi do greške. Ako se ne koristi NOWAIT u slučaju da postoji konflikt prilikom zaključavanja komanda LOCK TABLE čeka da se tabela otključa kako bi ona zatražila zaključavanje.

Na slici 3.6 je prikazan primer pokušaja eksplicitnog zaključavanja tabele *accounts* od strane dve transakcije.

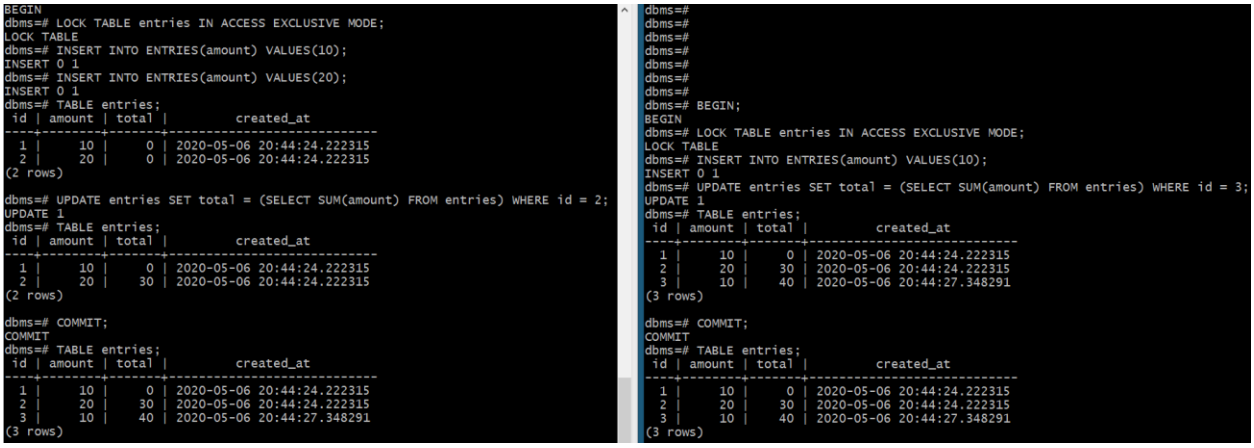
Slika 3.6 Eksplicitno zaključavanje tabele

Prva transakcija uspeva da zaključa tabelu a nakon toga druga pokušava da izvrši zaključavanje isto tabele ali sa NOWAIT i zbog toga dolazi do greške. Pošto se koristi najveći mod zaključavanja nije moguće ni čitanje podataka iz tabele što je prikazano na slici 3.7.

Slika 3.7 Pokušaj čitanja podataka nakon kada je tabela zaključana

Prva transakcija zaključava tabelu najvećim nivoom, a nakon toga druga transakcija pokušava da pročita podatke iz tabele i zbog toga što je prva transakcija već zaključala tabelu komanda čitanja se blokira i čeka da se tabele otključa. Nakon što se prva transakcija odbacuje tad se vrši i otključavanje tabele tek tad komanda za čitanje podataka iz druge transakcije nastavlja izvršenje i čita podatke.

Na slici 3.8 prikazan je primer kao sa slike 3.3, gde jedna transakcija zaključava tabelu unosi i modifikuje podatke, druga pokušava da zaključa tabeli i čeka sve dok je prva transakcija oslobodi, a nakon toga je zaključava unosi i modifikuje podatke.



Slika 3.8 Zaključavanje tabele i modifikacija podataka

4. Zaključak

U ovom radu opisane se transakcije, izolacija između transakcija i zaključavanje kod PostgreSQL baze podataka. Svaka transakcija koristi podrazumevani nivo izolacije, i svaka komanda nivo zaključavanja sa najmanje ograničenje, što za određene slučajeve korišćenja nije dovoljno dobro. Zbog toga je moguće eksplicitno definisati nivo izolacije ili zaključavanje što ima svoju cenu. Cena većeg nivoa izolacije ili zaključavanja dovodi do nastupanja greške i potrebe da se transakcije pokušaju ponovo, kao i zbog ograničenja zaključavanja do blokiranja naredbe dok se ne otključa određeni resurs. Zbog toga je potrebno pažljivo koristiti zaključavanja i nivoe izolacije, ali u slučajevima gde redosled operacija konkurentnih pristupa ili greška u sistemu može da dovede do nekonzistentnih podataka potrebno je na osnovu analize izabrati odgovarajući nivo izolacije i zaključavanja kako bi baza bila u konzistentnom stanju.

Izolacija transakcija daje bolje performanse od eksplicitnog zaključavanja pa se preporučuje korišćenje izolacije.

LITERATURA

[1] *„ACID“* - https://en.wikipedia.org/wiki/ACID, Poslednji pristup: 06.05.2020.

[2] *„PostgreSQL 12.2 Documentation”* - https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf, Poslednji pristup: 06.05.2020.