UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Mehanizmi transakcija

Sistemi za upravljanje bazama podataka

Student:

Dušica Milanović, br. ind. 1059

Mentor:

doc. dr Aleksandar Stanimirović

Niš, maj 2020.

SADRŽAJ

Uvod

Informacioni sistem omogućava organizaciju podataka koja obezbeđuje procese i informacije, korisne članovima informacionog sistema i njegovim klijentima. Kao takav, informacioni sistem treba da omogući da se svi ti procesi obavljaju što efekasnije i brže. Informacioni sistem je u stvari slika nekog realnog sistema čiji je cilj da unapredi taj realni sistem.

Postoje tri razloga za primenu informacionih tehnologija u nekoj organizaciji, koji su u direktnoj vezi sa osnovnim ulogama informacionog sistema, koje on može imati za tu organizaciju. To su: podrška poslovnim procesima i aktivnostima organizacije, podrška u donošenju odluka i podrška strategiji u realizaciji konkurentskih prednosti.

U ovom radu predstavljeni su mehanizmi transakcija i jedan od najpoznatijih primera koji ilustruje korišćenje transakcija je prenos novca sa jednog bankovnog računa na drugi.

Transakcija

Danas je veoma bitan i značajan koncept baze podataka po kojem je to, u stvari, zajednički resurs koga istovremeno (konkurentno) koristi veći broj programa, jer se pravi efekti baze podataka ispoljavaju kada se radi na mrežnom okruženju. DBMS je upravo tu da upravlja konkurentnim radom više korisnika i da obezbeđuje sinhronizaciju njihovog rada.

DBMS ima i funkciju da spreči štetne posledice (narušen integritet baze podataka, nekonzistentno stanje baze, itd.) pri promenama (transakcijama) koje se vrše nad bazom podataka u višekorisničkom okruženju, a za to koristi tehnike zaključavanja podataka, vremenskog markiranja, odlaganja i slično. U tom smislu posebno je značajno upravljanje istovremenim (konkurentnim) transakcijama.

Baze podataka kontinuirano skladište informacije koje opisuju trenutno stanje organizacije. Na primer, baza podataka banke skladišti trenutni bilans na svakom računu deponenta. Kada se u stvarno stanju dogodi nešto što menja stanje organizacije, mora da se uradi odgovarajuća promena informacija u bazi podataka. Sa dostupnim DBMS-om, ove promene se dešavaju u realnom vremenu, uz pomoć programa koji se nazivaju transakcije koje deluju kada dođe do promena u stvarnom svetu. Na primer, kada klijent stavlja novac u banku (događaj u stvarnom svetu), izvršava se transakcija depozita. Svaka transakcija mora biti uređena tako da održava preciznost veze između stanja baze podataka organizacije koje je kreira i stvarnog sveta. Pored toga što menja stanje baze podataka, transakcija sama po sebi može da inicira neke događaje u stvarnom svetu. Na primer, izdvojena transakcija kod bankomata, inicira događaj odliva novca.

U većini aplikacija baze podataka se koriste kako bi se modelovalo stanje nekog stvarnog preduzeća. U takvim aplikacijama transakcija predstavlja program koji posreduje sa bazom podataka, tako da podržava slaganje stanja preduzeća i stanja baze podataka. Praktično rečeno, transakcija ažurira bazu podataka tako da prikazuje događaje koji su se odrazili na stanje stvarnog preduzeća. Jedan primer bi bila transakcija polaganja novca u banku. Klijent daje novac blagajniku kao depozit. Transakcijom se ažurira informacija o računu klijenta u bazi podataka kako bi se prikazao depozit.

Definicija transakcije

Transakcija je program u izvršenju DBMS-a koji predstavlja logičku jedinicu obrade baze podataka. Sadrži jednu ili više operacija koje pristupaju bazi podataka (umetanje, brisanje, modifikacija, pretraživanje, itd.). Transakcija predstavlja jedinicu posla koja treba da se izvrši u realnom sistemu. Važno je istaći da talogička jedinica posla se izvršava do kraja ili se poništava u celini. Poenta transakcija je da ukoliko imamo niz naredbi koje treba da se izvrše, prvo ih proglasimo kao transakciju, a zatim ih smestimo ih sve u jednu transakciju. Ukoliko se iz nekog razloga neka naredba iz transakcije ne izvrši, sve prethodne naredbe će biti poništene. Drugim rečima, zahteva se da transakcija bude atomska (nedeljiva) i da sve instrukcije jedne transakcije moraju biti izvršene ili nijedna. U tom smislu, transakcija predstavlja osnovnu programsku jedinicu kojom se obezbeđuje očuvanje konzistentnosti baze.

Naravno, u jednom trenutku nad bazom podataka može se izvršiti više transakcija, i imajući u vidu gore rečeno, transakciona obrada označava grupisanje izmena sadržaja baze podataka u „paket“ koji se potom obrađuje kao neraskidiva celina. Obrada se uvek odvija tako da se uspešno izvrše ili sve transakcije, ili nijedna od njih.

Transakciona obrada je korisna u aplikacijama u kojima jedna akcija mora da se izvrši u vezi sa jednom ili više drugih akcija. Transakciona obrada je uobičajena u bankarskim, računovodstvenim i mnogim drugim aplikacijama.

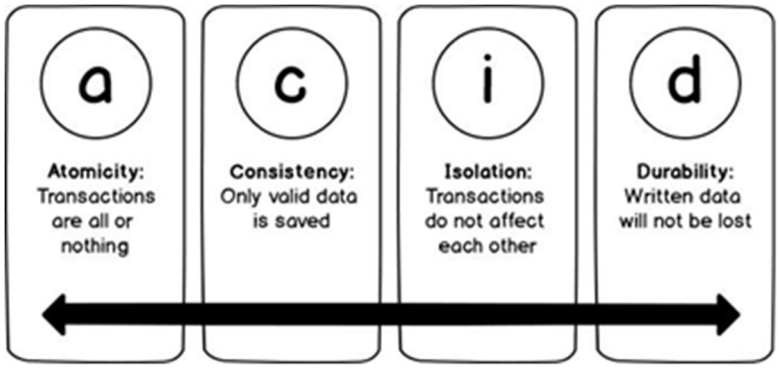
Na primer, kada u nekoj aplikaciji za bankarsko poslovanje premeštamo iznos sa jednog računa na drugi, ne bismo želeli da stanje na jednom računu povećamo, a da ga pri tom na drugom računu ne smanjimo. Zato ćemo te dve izmene grupisati u jednu transakciju.

Transakciona obrada je izuzetno važna u višekorisničkim aplikacijama. Kada više korisnika istovremeno unosi izmene u bazu podataka, više se ne možemo pouzdati u to da će uvek jedna izmena biti trajno upisana u bazu pre nego što započne naredna, osim ako određenu grupu izmena ne „uokvirimo“ u transakciju. Zbog toga bi u višekorisničkom okruženju trebalo da koristimo transakcionu obradu.

Osobine transakcija

Sve transakcije imaju sledeće osobine:

- Atomnost (atomicity),
- Konzistentnost (consistency),
- Izolacija (izolation) i
- Trajnost (durability).



Slika 1 – ACID svojstva transakcije

Atomnost (Atomicity) podrazumeva skup aktivnosti nad bazom podataka po principu „sve ili ništa“. Ili su sve aktivnosti uspešno obavljene ili je baza podataka ostala nepromenjena. DBMS, pored atomnosti transakcija, mora da garantuje atomnost svake aktivnosti (pojedinačne operacije). Primetimo da obični programi ne moraju imati osobinu atomnosti. Npr. ako je sistem pao u trenutku dok je program ažurirao neki fajl, kada smo podigli sistem, fajl je ostao delimično promenjen. Atomnost izvršenja znači da je svaka transakcija ili potvrđena, ili smo odustali od nje.

Konzistentnost znači da transakcija treba da prevede bazu podataka iz jednog u drugo konzistentno stanje. Za vreme obavljanja transakcije konzistentnost baze podataka može da bude narušena. Ukoliko u toku transakcione obrade dođe do greške, podaci moraju biti vraćeni u stanje pre početka transakcije. Transakcijom se mora pristupati i ažurirati BP tako da sva ograničenja integriteta BP ostanu zaštićena. Svaka realna organizacija je organizovano u odnosu na određena pravila koja ograničavaju moguća stanja organizacije.

Npr. broj studenata prijavljenih za nastavu ne može preći broj mesta namenjenih za tu nastavu. Kada takvo pravilo postoji, moguća stanja BP su ograničena na isti način. Ograničenja integriteta, u odnosu na pomenuta pravila, potvrđuju da broj registrovanih studenata koji se unosi u polje BP ne sme da pređe vrednost polja BP koja odgovara veličini sale. Dakle, kada se transakcija registracije završi, BP mora da zadovolji ovo ograničenje integriteta (pretpostavljajući da je ograničenje zadovoljeno na početku transakcije).

Izolacija znači da kada se dve ili više transakcija izvršavaju istovremeno, njihovi efekti moraju biti međusobno izolovani. Efekti koje izazovu transakcije koje se obavljaju istovremeno moraju biti jednaki efektima nekog njihovog serijskog (jedna posle druge) izvršenja. Zbog povećanja paralelizma u obradi transakcija dozvoljavaju se različiti nivoi izolovanosti.

Prilikom rasprave o konzistentnosti BP, usredsredili smo se na efekat jedne transakcije. Sledeće što ćemo ispitivati je efekat niza transakcija. Rekli smo da se niz transakcija izvodi sekvencijalno, ili serijski, ako se jedna transakcija niza izvrši pre nego što druga počne. Dobra strana kod serijskog izvršavanja je da ako su sve transakcije konzistentne i baza podataka je inicijalno u konzistentnom stanju.Serijsko izvršavanje čuva konzistentnost. Kada prva transakcija niza započne, BP je u konzistentnom stanju, a s obzirom da je transakcija konzistentna i nakon njenog izvršenja BP će biti u konzistentnom stanju. Zbog toga što je BP konzistentna pre početka druge transakcije, i druga će se obaviti korektno.

Serijsko izvršavanje je adekvatno za aplikacije čiji su zahtevi skromni. Međutim, mnoge aplikacije imaju stroge zahteve vremena odziva i pristupa, i često jedini način da se zahtevi zadovolje je konkurentno izvršavanje transakcija. Moderni sistemi mogu da istovremeno izvršavaju više od jedne transakcije, a ovaj metod izvršavanja nazivamo konkurentnim. Konkurentno izvršavanje je adekvatno kada se sistemom za obradu transakcija služi više korisnika. U tom slučaju biće dosta aktivnih, delimično završenih transakcija u svakom trenutku. Kada se transakcije izvršavaju konkurentno, konzistentnost svake od njih nije dovoljna da obezbedi da baza posle izvršenja obe transakcije u potpunosti prikazuje stanje preduzeća.

Trajnost znači da kada se transakcija završi (potvrđene promene), njeni efekti ne mogu biti izgubljeni, čak i ako se neposredno po njenom okončanju desi neki ozbiljan otkaz sistema. Ovaj zahtev sistema za obradu transakcija odnosi se na to da se informacije ne izgube. Sistem mora da osigura da transakcija, koja je jednom potvrđena, svoj efekat prenese na BP, pa čak i ako kompjuter, ili medijum na kojem je baza podataka smeštena, iznenada prestane da radi.

Npr. ako ste se uspešno prijavili za kurs, očekujete da sistem zapamti vašu registraciju pa čak i ako posle toga prestane da radi. Možemo da primetimo da obični programi ne moraju imati osobinu trajnosti. Npr. ako se pojave problemi pošto je program izvršio promenu nekog fajla, fajl može biti vraćen u stanje pre izvršene promene.

COMMIT I ROLLBACK

Obezbeđenje ACID (akronim od reči **A**tomicity, **C**onsistency, **I**solation, **D**urability) osobina transakcije se radi upotrebom određenih metoda i instrukcija:

- transakcija počinje sa **`BEGIN TRANSACTION`**,
- završava se sa **`COMMIT`**, čime se potvrđuju promene u bazi podataka ako su sve instrukcije uspešno izvršene,
- završava se sa **`ROLLBACK`**, ako sve instrukcije nisu uspešno završene.

U stvari, postupak transakcije počinje pozivanjem metode **`BeginTrans`**, čime se označava početak niza operacija koje čine jednu logičku jedinicu. Metoda **`CommitTrans`** preuzima sve izmene načinjene od poslednjeg mesta na kome je bila pozvana metoda **`BeginTrans`** i upisuje ih na disk. Metoda **`RollbackTrans`** deluje na suprotan način od **`CommitTrans`** – ona poništava sve izmene i vraća stanje kakvo je bilo pre poslednjeg poziva metode **`CommitTrans`**.

SUBP inicira iz bilo kog razloga **`ROLLBACK`** instrukciju, ako dođe do neplaniranog završetka transakcije.

S obzirom da jedan program može predstavljati kolekciju transakcija, transakcije mogu biti i ugnježdene. U tom slučaju, COMMIT instrukcija se izvršava od najnižeg do najvišeg nivoa, a dovoljno je da se ROLLBACK pojavi samo na jednom mestu i poništavaju se promene svih transakcija.

SUBP poseduje i održava dnevnik transakcija (tj. dnevnik aktivnosti, log file). Za svaku transakciju i za svaki objekat baze podataka koji je ona ažurirala čuva se:

- vrednost pre ažuriranja (before-image),
- vrednost posle ažuriranja (after-image).

Na naredbu **`ROLLBACK`**, SUBP koristi vrednosti pre za datu transakciju. Pre **`Commit`** naredbe sistem prvo upisuje vrednosti pre i posle u log fajl. Ako se prekine **`COMMIT`** naredba, mogu se pročitati vrednosti posle sa log fajla, što omogućava očuvanje konzistentnog stanja.

Naredbe za upravljanje transakcijama

SQL obezbeđuje sledeće naredbe za upravljanje transakcijama:

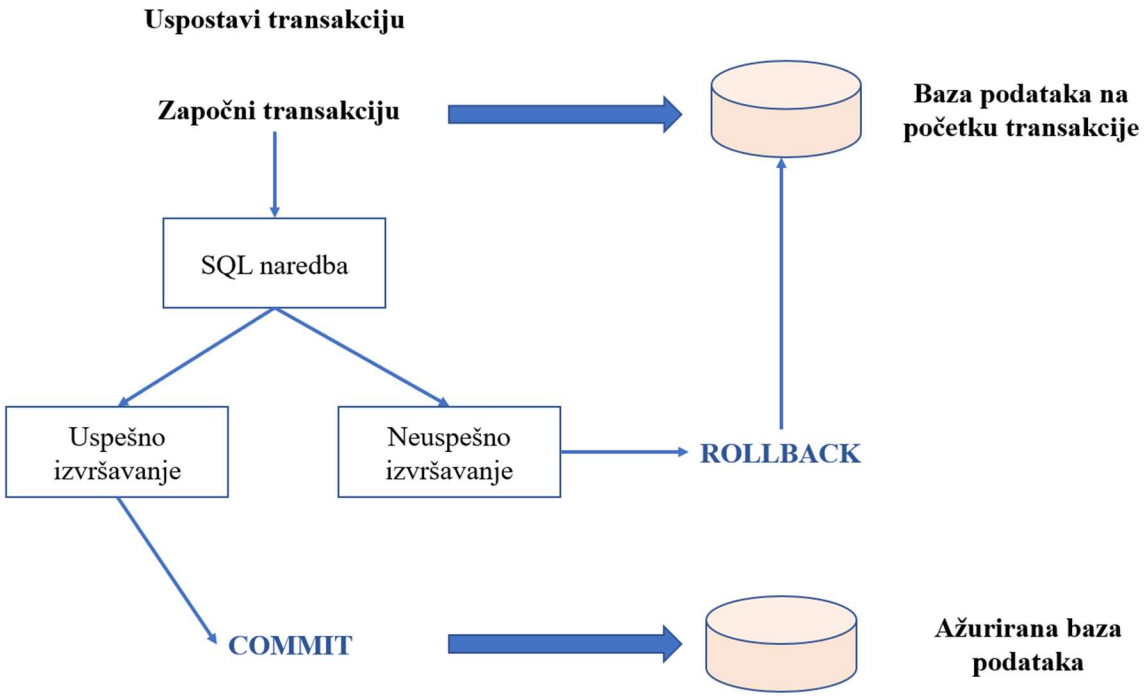
- **SET TRANSACTION** - kontroliše mnoge karakteristike promene podataka, najpre read/write karakteristike i nivo izolacije (izdvajanja) transakcija. Kada se transakcija izvršava sve karakteristike će biti uključene, i one koje su navedene u naredbi i one koje nisu. Karakteristike transakcije koje nisu navedene će preuzeti podrazumevane vrednosti. Sve karakteristike transakcije nisu raspoložive ni za jednu transakciju osim prve.
  Kada se izda, **SET TRANSACTION** postavlja svojstva sledeće dolazeće transakcije. Zbog toga, **SET TRANSACTION** je privremena naredba, koju bi trebalo izdavati posle jedne završene transakcije, a pre nego što sledeća transakcija počne (za početak transakcije i postavljanje njenih karakteristika u istom trenutku, koristiti **START TRANSACTION**). Može da bude primenjena više od jedne opcije sa ovom naredbom, ali samo u jednom načinu pristupa, izolacionom nivou, a dijagnostički broj (veličina) može da se specificira odvojeno.

- **START TRANSACTION** – **BEGIN TRAN**,
  MySQL, PostgreSQL i SQL Server podržavaju sličnu naredbu:
  BEGIN [TRAN[SACTION]] i njen sinonim **BEGIN [WORK]**. **BEGIN TRANSACTION** deklariše eksplicitnu transakciju, ali ona ne postavlja (određuje) izolacione nivoe. Većina platformi baza podataka sprovodi implicitnu kontrolu transakcija, korišćenjem nečega što se obično naziva autocommit način. U autocommit načinu, baza podataka tretira svaku naredbu kao transakciju kako unutar tako i izvan nje, upotpunjenu sa implicitnom naredbom **BEGIN TRAN** i **COMMIT TRAN**. MySQL, PostgreSQL i SQL Server, omogućavaju da se eksplicitno deklariše transakcija eksplicitno potvrdi, vrati na kontrolnu tačku ili povrati transakcija. Većina platformi koje su prethodno pomenute rade u autocommit načinu po dogovoru. Zbog toga, dobro je nepisano pravilo da se koriste samo eksplicitno deklarisane transakcije, ako se namerava da se tako radi sa svim transakcijama u sesiji. Drugim rečima, ne treba mešati implicitno deklarisane transakcije i eksplicitno deklarisane transakcije u jednoj sesiji. Svaka transakcija koja je eksplicitno deklarisana može da bude učinjena stalnom samo sa naredbom **COMMIT**. Slično, bilo koja transakcija koja je neuspešna ili je potrebno da bude odbačena mora da bude eksplicitno poništena sa naredbom ROLLBACK.
  Napomena: proverite da je **BEGIN** u paru sa bilo **COMMIT** ili **ROLLBACK**. Inače, DBMS neće moći da završi transakciju(e) dok ne naiđe na **COMMIT** ili **ROLLBACK**. Ne blagovremeno postavljanje **COMMITs** (ili **ROLLBACKs**) potencijalno može da dovede do ogromnih ili nedovoljno dobro planiranih transakcija. Dobra ideja je da se izda eksplicitni rollback ili commit posle jedne ili malog broja naredbi zato što transakcije koje duže traju mogu da zaključaju resurse, tako sprečavajući druge korisnike da pristupaju ovim resursima. Transakcije koje dugo rade ili vrlo velike grupne transakcije mogu da ispune povratne segmente ili dnevnike transakcija baze podataka

- **SET CONSTRAINTS –** nema ga SQL server,
- **SAVEPOINT – SAVE TRAN,**

- **RELEASE SAVEPOINT –** nema ga,
- **ROLLBACK** i
- **COMMIT** - Naredba **COMMIT** eksplicitno završava otvorenu transakciju i čini promene stalnim u bazi podataka. Transakcije mogu da budu otvorene implicitno kao deo naredbe **INSERT**, **UPDATE** ili **DELETE**, ili otvorene eksplicitno sa naredbom **START** (**BEGIN**). U bilo kom slučaju, eksplicitno izdavanje naredbe **COMMIT** će da završi otvorenu transakciju. Za jednostavne operacije, transakcije ćete da izvršavate (to jest, SQL kod rukuje ili menja podatke i objekte u bazi podataka) bez eksplicitnog definisanja transakcije. Naravno, sve transakcije su bolje upravljane eksplicitnim njihovim zaključivanjem sa naredbom **COMMIT**. Zato što vrste pa čak i cele tabele mogu da budu zaključane za vreme trajanja transakcije, veoma je značajno da se transakcije završe koliko je moguće brže. Tako, ručno izdavanje naredbe **COMMIT** sa transakcijom može da pomogne kontroli korisnika u pitanjima konkurentnosti i problemima zaključavanja na bazi podataka. Predlog je uvek korišćenje eksplicitnih transakcija sa **START TRAN**, na platformama baza podataka koje je podržavaju, za početak transakcije i **COMMIT** ili **ROLLBACK** za završavanje transakcija

Ove naredbe se koriste za početak i završetak transakcija, postavljanje njihovih svojstava, poštovanje ograničenja za vreme transakcije, određivanje mesta u okviru transakcije koja rade kao tačke na koje će se izvođenje transakcije da vrati u slučaju poništavanja nekih akcija. Na slici 2 je prikazana tipična transakcija.



Slika 2 – Osnovna SQL transakcija

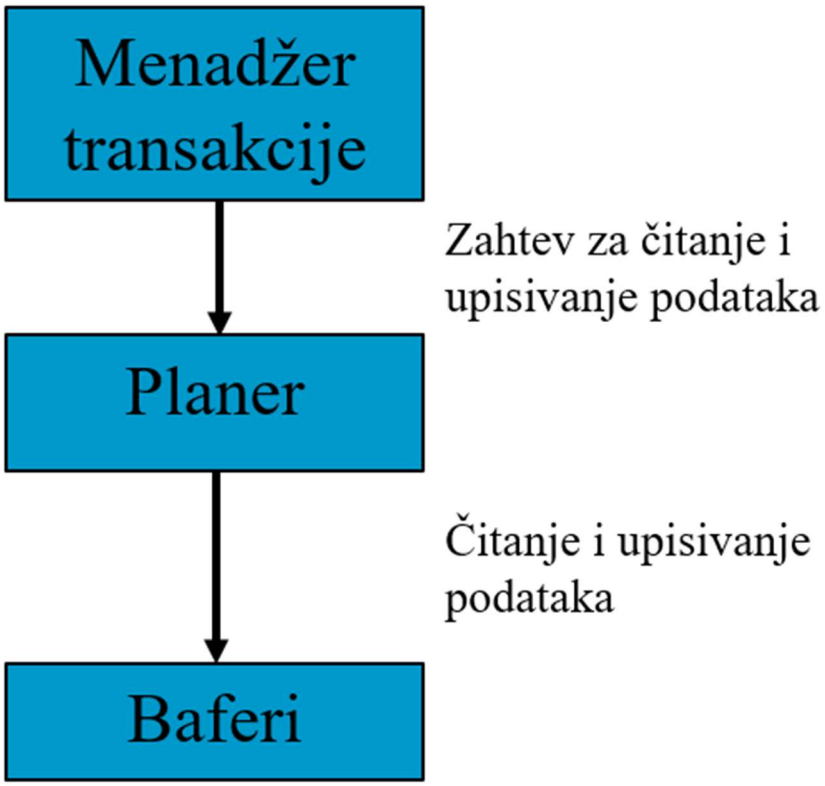
Konkurentno izvršavanje transakcija

Nad modernim bazama podataka transakcije se ne obavljaju u izolovanosti već konkurentno. Više transakcija mogu istovremeno zahtevati iste resurse, isti zapis baze podataka, itd. U takvim situacijama otvara se mogućnost da nekontrolisan međusobni uticaj transakcija dovede do nekonzistentnog stanja.

Već je nagovešteno da je jedan od zahteva SUBP da upravlja konkurentnim radom više korisnika, obezbeđuje sinhronizaciju njihovog rada... a sve u cilju sprečavanja štetnih posledica pri promenama koje se vrše nad bazom podataka u višekorisničkom okruženju.

Komponente SUBP koje učestvuju u ovom procesu su:

- Planer (Scheduler),
- Menadžer transakcija (Transaction manager).



Slika 3 – Komponente konkurentne obrade transakcija

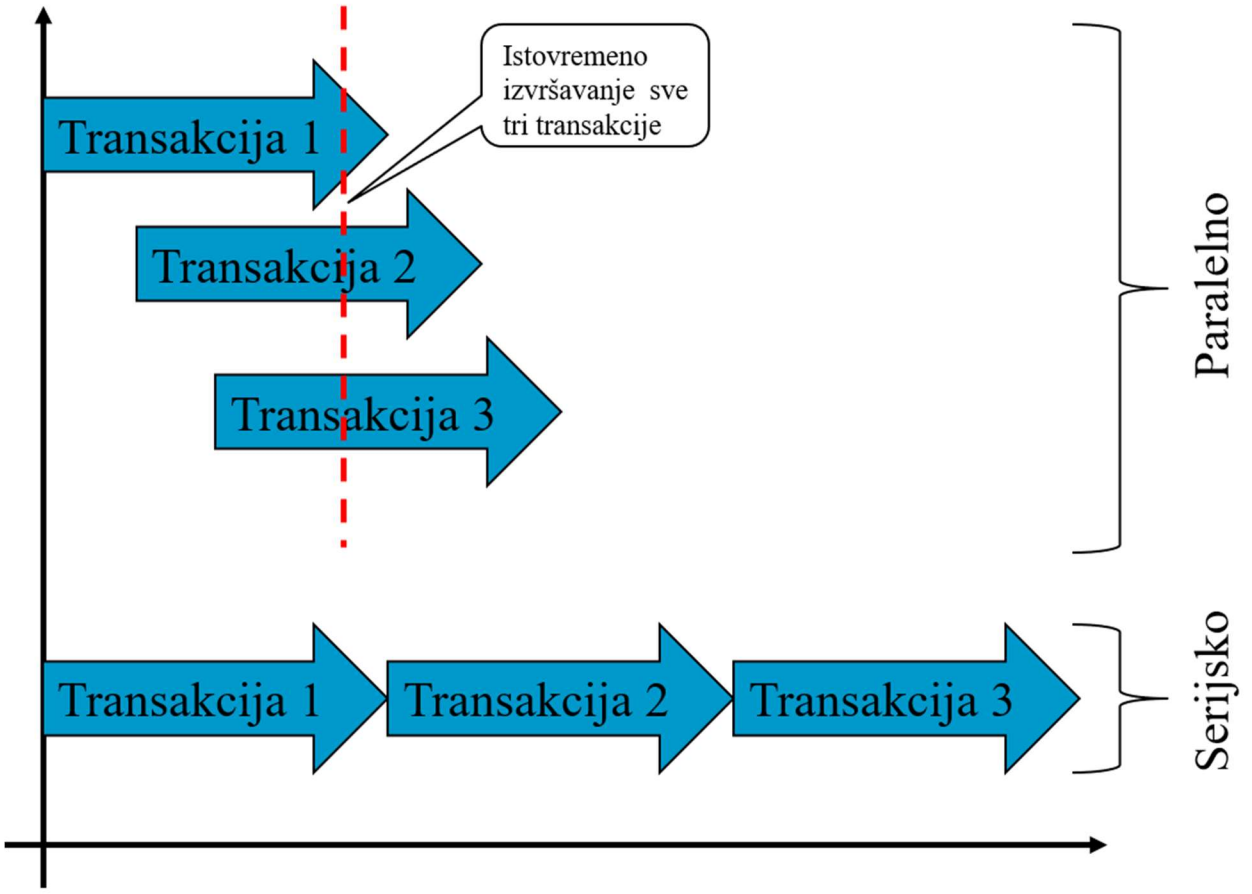
Planer vodi računa o redosledu akcija kod više konkurentnih transakcija. Ako čitanje ili upisivanje može da naruši integritet baze podataka, zahtev se ili vremenski odlaže ili se poništava cela transakcija.

Menadžer transakcija upravlja celokupnim izvršenjem transakcija.

Idealan slučaj izvršavanja transakcija je serijsko izvršavanje. Posledica je korektan rezultat. Serijsko izvršavanje transakcija, u stvari, znači da:

- nema preplitanja transakcija,
- prvo se završi jedna, zatim druga.

Konkurentno izvršavanje transakcija je serijabilno (linearno) ako daje isti rezultat kao i serijsko izvršavanje svih transakcija.



Slika 4 – Paralelno i serijsko izvršavanje transakcija

Interesantno je da kada se koriste transakcije, povećava se stepen integriteta izmena koje korisnik unosi, ali se smanjuje konkurentnost, odnosno mogućnost da pomoću date aplikacije veći broj korisnika istovremeno menja podatke, jer ima više zapisa koji su zaključani duže vreme.

Protokol zaključavanja

Velika je verovatnoća da, ako nema ograničenja, izvršavanje transakcija neće biti serijabilno, a provera serijabilnosti se teško može obaviti u realnom vremenu. Zato se vrši jedan način ograničavanja, tj. zaključavanje i otključavanje podataka.

U stvari, **mehanizam zaključavanja** (*locking*) predstavlja upravo nasilno ostvarivanje serijabilnosti. Transakcija zaključava objekat baze podataka kome je pristupila, čime je onemogućeno drugim transakcijama da nekorektno operišu.

Postoje dva osnovna **nivoa zaključavanja**, na nivou stranice i na nivou zapisa. Zaključavanje na nivou stranice je u stvari zaključavanje čitave stranice sa zapisima, a ne zaključavanje samo pojedinih zapisa, što je slučaj sa zaključavanjem na nivou zapisa. To znači da, na primer, kada se primenjuje zaključavanje na nivou zapisa, više korisnika može istovremeno da ažurira zapise u istoj tabeli, nasuprot slučaju kada bi bila zaključana cela tabela sa svim zapisima u njoj.

Zaključavanje na nivou zapisa obezbeđuje znatno bolje mogućnosti višekorisničkog pristupa podacima. Međutim, treba voditi računa da to ne znači uvek i bolje performanse. Kada se istovremeno ažurira veliki broj zapisa, zaključavanje na nivou stranice može da obezbedi bolje performanse. Ipak, u većini slučajeva, bolje mogućnosti višekorisničkog pristupa podacima, nadoknađuju nešto slabije performanse.

Postoje dva osnovna pristupa u strategiji zaključavanja objekata baze podataka, odakle se izdvajaju dve vrste zaključavanja:

- Ekskluzivno (exclusive lock ili write lock) ili pesimističko – XL,
- Deljivo (shared lock ili read lock) ili optimističko – SL.

Ekskluzivno zaključavanje znači da u svakom datom trenutku samo jedan korisnik može da menja objekat baze podataka. Drugim rečima, ako jedna transakcija postavi XL katanac na objekat baze podataka to znači da ne može ni jedna druga transakcija da postavi katanac, i ne može se postaviti ni jedan drugi katanac. U slučaju ako se primenjuje zaključavanje na nivou stranice, to je veliki problem, jer u svakom trenutku samo jedan korisnik može da ažurira bilo koji zapis koji se nalazi na zaključanoj stranici.

Deljivo zaključavanje omogućava da više korisnika istovremeno ažurira iste zapise uz manji broj sukoba oko zaključavanja zapisa. Drugim rečima, ako jedna transakcija postavi SL katanac na objekat baze podataka to znači da druga transakcija može da postavi SL katanac na isti objekat, i ni jedna druga ne može da postavi XL katanac na taj objekat. Međutim, time se povećava rizik nastajanja sukoba pri upisivanju podataka. Sukob pri upisivanju nastaje kada:

- prvi korisnik počinje da ažurira objekat baze podataka,
- drugi korisnik upisuje u bazu podataka izmene objekta koje je on uneo,
- prvi korisnik pokuša da u tom trenutku upiše svoje izmene.

Sukob pri upisivanju je štetan, jer on znači da prvi korisnik ažurira drugačiji objekat od onoga sa kojim je počeo da radi.

Verovatno je u izboru vrste zaključavanja prihvatljivije ekskluzivno zaključavanje, da bi se izbegli sukobi pri upisivanju. Međutim, uvek treba imati na umu korisnike koji duže vreme zaključavaju objekte baze podataka. U tom slučaju bi bilo dobro razmotriti upotrebu deljivog zaključavanja. Ovaj problem delimično može biti rešen i upotrebom ekskluzivnog zaključavanja sa vremenskim ograničavanjem. Na primer, nekom obrascu se može dodati mogućnost vremenskog ograničavanja tako da izmene koje korisnik nije snimio posle deset minuta automatski bivaju poništene.

Protokol zaključavanja obuhvata sledeća pravila:

1. Transakcija koja čita neki objekat baze podataka mora da postavi SL na taj objekat,
2. Transakcija koja želi da ažurira neki objekat mora da postavi XL. Ako je ta transakcija prethodno postavila SL treba da ga promeni u XL,
3. Ako transakcija nije postavila „katanac“, zato što je to pre uradila neka druga, prelazi u stanje čekanja,
4. Transakcija oslobađa XL i SL na kraju, sa **COMMIT** ili **ROLLBACK** naredbom.

Oba katanca XL i SL se postavljaju implicitno.

Kursori deklarisani sa opcijom WITH HOLD

U principu, izvršavanjem naredbi za kraj transakcije (**COMMIT** i **ROLLBACK**) zatvaraju se otvoreni kursori. S obzirom na to da je nekada zgodno da nakon uspešnog kraja transakcije otvoreni kursor i dalje ostane otvoren, moguće je navesti opciju **WITH HOLD** u deklaraciji kursora čime se postiže da on ostane otvoren i nakon uspešnog kraja transakcije.

Kursori u ugrađenom SQL-u mogu biti deklarisani korišćenjem opcije **WITH HOLD** na sledeći način:

DECLARE <naziv> **CURSOR WITH HOLD FOR**

<upit>

Na ovaj način kursor ostaje otvoren sve dok se eksplicitno ne zatvori, ili se ne izvrši naredba **ROLLBACK**. Prilikom izvršavanja naredbe **COMMIT**, otvoreni kursori koji su deklarisani sa opcijom **WITH HOLD** ostaju otvoreni. Na ovaj način nakon izvršavanja naredbe **COMMIT** naredni red u rezultatu se može pročitati naredbom **FETCH**.

Dakle, izvršavanje naredbe **COMMIT** zatvara sve kursore koji nisu definisani sa opcijom **WITH HOLD**. Transakcija se potvrđuje i sve promene se trajno zapisuju u bazi podataka u smislu da postaju dostupne svim konkurentnim ili narednim transakcijama. Dok, izvršavanje naredbe **ROLLBACK** zatvara sve kursore i poništava sve promene nastale tokom transakcije.

Oporavak baze podataka

Oporavak baze podataka (**RECOVERY**) predstavlja proces vraćanja baze podataka u korektno stanje. Sasvim je realno, i dešava se, da usled otkaza sistema mora da se uradi oporavak baze podataka. Uzroci otkaza mogu biti različiti: greške u programiranju, greške u operativnom sistemu, nestanak napajanja.

Proces oporavka se zasniva na redundansi podataka, tj. postojanje rezervnih kopija, koje mogu da se čuvaju na disku, traci, itd. Tako, u slučaju otkaza sistema, oštećena baza podataka se rekonstruiše u ispravno stanje na osnovu poslednje kopije, a nekonzistentno stanje se rešava tako što se poništavaju nekonzistentne promene, a transakcije se ponavljaju.

Trajnost podrazumeva da nijedna promena u bazi podataka koju je napravila započeta transakcija, ne sme biti izgubljena. Iz tog razloga, a pošto hard disk može da zakaže, baza podataka se mora čuvati na različitim hard diskovima. Jednostavan primer postizanja trajnosti jeste čuvanje različitih kopija baze podataka na različitim diskovima (koji se, po mogućstvu, napajaju iz različitih izvora energije). Pošto je istovremeni pad oba diska malo verovatan, velika je verovatnoća da će barem jedna kopija hard diska biti uvek dostupna. Duplikat, ili disk imidž, podržava ovakav pristup. Slika hard-diska (duplikat – mirrored disk) je sistem čuvanja po kome, kad god aplikacija zahteva da disk izvrši operaciju upisa, sistem simultano beleži istu informaciju na dva različita diska. Tako je prvi disk identična kopija, ili disk imidž, onog drugog.

U transakcijama koje obrađuju aplikacije, sistem disk imidža može da dostigne izuzetnu dostupnost sistema. Ukoliko jedan disk imidž padne, sistem može da nastavi dalje koristeći drugi, bez usporavanja ili zaustavljanja. Kada se zameni disk koji je zakazao, sistem mora da ponovo sinhronizuje oba. Nasuprot tome, kada se trajnost postiže samo pomoću LOG fajla (dnevnika), proces oporavka nakon pada hard diska može trajati znatno duže, a u toku tog perioda sistem je nedostupan korisnicima. Ipak, treba podsetiti da čak i kada transakcija u procesu koristi disk imidž, i dalje se mora koristiti dnevnik kako bi se postigla atomnost – na primer, da bi se poništila transakcija nakon pada.

Zaključak

Baze podataka i drugi sistemi za skladištenje podataka kod kojih je integritet podataka imperativ često koriste mehanizme transakcija kako bi očuvali taj integritet. Veoma je važno da sva obrada podataka ostavi skladište podataka u konzistentnom stanju.

Većina modernih sistema za upravljanje relacionim bazama podataka spada u kategoriju baza podataka koje podržavaju transakcije: transakcione baze podataka.

Ako nije došlo do grešaka u toku izvršavanja transakcije onda sistem komituje transakciju. Operacija komitovanja transakcije primenjuje sve promene izvršene na podacima u toku trajanja transakcije, i promene čini trajnim u bazi podataka. Ako je došlo do greške prilikom izvršavanja transakcije, ili ako korisnik zahteva operaciju rolbekovanja, sve izmene na podacima do kojih je došlo u toku trajanja transakcije se poništavaju. Nije moguće da se transakcija delimično komituje, jer bi to ostavilo sistem u nekonzistentnom stanju.

Literatura

[1] Slobodan Obradović, Siniša Ilić, *SQL Strukturirani upitni jezik u sistemima za upravljanje relacionim bazama podataka,* Beograd 2016

[2] Ivan Tot, Informacioni sistemi ya podršku odlučivanju, Beograd 2012

[3] Ivan Tot, Razvoj On-Line Analytical Processing (OLAP) sistema za podršku odlučivanju, Beograd 2010

[4] Vesna Marinković, Programiranje baze podataka, Matematički fakultet, Beograd

[5] https://www.link-elearning.com/site/kursevi/lekcija/7256