UNIVERZITET U NIŠU ELEKTRONSKI FAKULTET

Milan Stanković

**SEMINARSKI RAD**

**Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključvanje kod PostgreSql**

Sistemi za upravljanje bazama podataka

|  |  |
| --- | --- |
| Student: | Mentor: |
| Milan Stanković, br. ind. 1276 | Prof. dr Aleksandar Stanimirović |

Niš, 2022. god.

Sadržaj

[1. Uvod 3](#_Toc105013828)

[2. Transakcije 4](#_Toc105013829)

[2.1 Stanja transakcija 4](#_Toc105013834)

[2.2 Operacije transakcija 5](#_Toc105013838)

[2.3 ACID svojstva transakcije 6](#_Toc105013843)

[2.4 Implementacija transakcije kod PostgreSQL-a 7](#_Toc105013849)

[3. Konkurentnost kod transakcija 11](#_Toc105013850)

[3.1 Izolacija transakcija kod PostgreSQL-a 12](#_Toc105013854)

[3.2 Eksplicitna zaključavana transakcija kod PostgreSQL-a 20](#_Toc105013855)

[3.2.1 Table-Level Locks (Zaključavanja na nivou tabele) 21](#_Toc105013856)

[3.2.2 Row-Level Locks (Zaključavanje na nivou reda) 22](#_Toc105013857)

[3.2.3 DeadLock-ovi (Zastoji) 24](#_Toc105013858)

[3.2.4 Advisory Locks (Zaključavanja na nivou aplikacije) 25](#_Toc105013859)

[4 Zaključak 27](#_Toc105013860)

[5 Literatura 28](#_Toc105013861)

# Uvod

Baze podataka i drugi sistemi za skladištenje podataka kod kojih je integritet podataka vrlo značajan često imaju mogućnost da rade sa transakcijama kako bi očuvale taj integritet.

Prilikom čitanja ili upisa podataka u bazu potrebno je voditi računa o tome da sva obrada nad podatacima ostavi bazu podataka u konzistentom stanju. Neophodno je obezbediti ispravnost podataka u svakom trenutku, čak i u slučaju u kome se upiti izvršavaju istovremeno, odnosno u kome je prisutna konkurentnost. U bilo kojoj bazi podataka, loše upravljanje transakcijama često dovodi do problema sa performansama u sistemima koji imaju mnogo korisnika. Kako se povećava broj korisnika koji pristupaju podacima, postaje važno što efikaksnije korišćenje transakcija.

Pojedinačne transakcije su akcije koje se mogu sastojati od pojedinačnih operacija, skupa operacija pa čak i od drugih transakcija. One su ključne za održavanje samog integriteta podataka, obezbeđuju da se skup operacija izvršava kao celina čime direktno utiču na to da stanje u bazi bude konzistentno.

Ovaj rad govoriće o konceptu transakcija koji se koristi kod različitih sistema, planu izvršavanja i obradi transakcija, biće navedene i opisane neke od najznačajnijih naredbi koje se koriste u okviru transakcija kod PostgreSQL-a sa konkretnim primerima iz realnog sveta. Takođe biće objašnjen koncept ACID svojstva, koja svaka transakcija treba da zadovolji kao i pojava konkurentnosti kod transakcija.

Pored transakcija opisaće se koncept izolacije, odnosno operacije koje pružaju postavljanje nivoa izolovanosti kojim se definiše stepen u kojem jedna transakcija mora biti izolovana od modifikacije nad podatacima izvršene od strane druge transakcije. Takođe navešćemo nivoe izolacije koji se koriste kod PostgreSQL-a.

Na kraju biće opisan mehanizam zaključavanja odnosno ekspicitnog zaključavanja, koji je neophodan za uspešnou obradu transakcija. Biće nabrojani različti režimi zaključavanja koje PostgreSQL koristi za kontrolu istovremenog pristupa podacima u tabelama u situacijama kada se izolacijom ne postiže željeno ponašanje.

# Transakcije

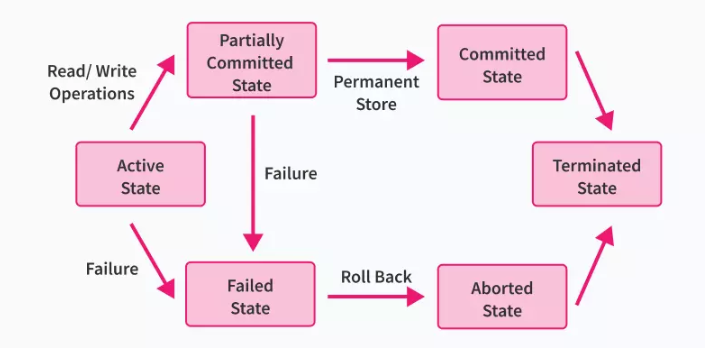
U situacijama kada jedan ili više korisnika pokušava da pristupi i promeni podatke u istoj tabeli u istom trenutku, dolazi do preplitanja akcija i mogućnosti nekonzistentne promene istih podataka ili netačnosti podataka. U cilju rešavanja ovog problema SQL podržava korišćenje transakcija da bi se obezbedilo konkurentno izvođenje akcija, ali i u cilju oporavka baze podataka.

Transakcija se definiše kao jedinica posla, koju čine jedna ili više SQL naredbi koje izvršavaju odgovarajući skup akcija. Transakcije mogu sadržati operacije čitanja, pisanja, brisanja, ažuriranja ili njihove kombinacije nad podacima u tabeli.



## Stanja transakcija

Tokom trajanja transakcije, potrebno je proći mnogo stanja. Ova stanja ažuriraju sistema i govore o trenutnom stanju transakcije. Takođe govore korisniku kako da planiraju dalju obradu transakcije. Ova stanja odlučuju o sudbini transakcije, da li će se ona izvršiti ili će se prekinuti.



Slika 1. Stanja transakcija u DBMS

Stanja kroz koje prolazi jedna transakcija navedene su i pojašnjene u delu ispod:

* **Aktivno stanje (Active State)** - Kada su instrukcije transakcije pokrenute, transakcija je u aktivnom stanju. Ako se sve operacije čitanja ili upisa izvode bez ikakvih grešaka, onda ide u "delimično predano stanje" (Partially Commited State); ako bilo koja instrukcija ne uspe, prelazi u „neuspešno stanje“ (Failed State).
* **Delimično predano stanje (Partially Commited State)** - Nakon što su sve operacije čitanja i pisanja završene, promene koje su prethodno napravljene u glavnoj memoriji sada postaju trajne u bazi podataka, nakon čega će stanje napredovati u predano stanje (Commited State), ali će u slučaju neuspeha otići u neuspešno stanje.
* **Neuspešno stanje (Failed State)** - Ako bilo koja operacija tokom transakcije ne uspe zbog nekih softverskih ili hardverskih problema, onda prelazi u neuspešno stanje. Pojava greške tokom transakcije čini trajnu promenu podataka u bazi podataka. Promene napravljene u podacima lokalne memorije se vraćaju u prethodno konzistentno stanje.
* **Prekinuto stanje (Aborted State)** - Ako transakcija ne uspe tokom svog izvršenja, prelazi iz neuspešnog u prekinuto stanje i pošto su u prethodnim stanjima sve promene izvršene samo u glavnoj memoriji ili lokalnom baferu, ove neizvršene promene se ili brišu ili vraćaju nazad (RollBack). Transakcija u ovom trenutku može ponovo da se pokrene i počne iznova iz aktivnog stanja.
* **Izvršeno stanje (Commited State)** - Ako transakcija uspešno završi sve skupove operacija, sve promene napravljene tokom delimično izvršenog stanja se trajno čuvaju u bazi podataka i za transakciju se navodi da je završena, tako da transakcija može napredovati da bi konačno bila prekinuta u završnom stanju (Terminated State).
* **Završeno stanje (Terminated State)** - Ako se transakcija prekine nakon vraćanja unazad (rollback) ili transakcija dođe iz stanja izvršenja, onda baza podataka dolazi u konzistentno stanje i spremna je za dalje nove transakcije pošto je prethodna transakcija sada prekinuta.



## Operacije transakcija

Za sve transakcije, čitanje i pisanje predstavljaju osnovne operacije baze podataka. U nastavku ćemo nabrojati neke od osnovnih operacija za rad sa transakcijama čija se sintaksa može razlikovati u odnosu na korišćeni sistem. Operacije niskog nivoa koje se obavljaju u transakciji su :

* **Begin\_transaction** − Naredba koji specificira početak izvršenja transakcije.
* **Read\_item** ili **Write\_item** – Operacije baze podataka koje se mogu preplitati sa operacijama glavne memorije kao deo transakcije.
* **Commite** – Signal koji označava da je transakcija u celini uspešno izvršena i da neće biti poništena.
* **Rollback** – signal za navođenje da je transakcija bila neuspešna i da su sve privremene promene u bazi podataka poništene. Transakcija koja je bila izvršena u potpunosti (commitovana) se ne može vratiti nazad.
* **Savepoint** - definisanje tačke u transakciji na koju se možemo kasnije vratiti

Transakcija koja uključuje samo preuzimanje podataka bez ikakvog ažuriranja podataka naziva se transakcija samo za čitanje. Svaka operacija visokog nivoa može se podeliti na niz zadataka ili operacija niskog nivoa. Na primer, operacija ažuriranja podataka može se podeliti na tri zadatka :

* **read\_item()** − čita podatak iz skladišta u glavnu memoriju.
* **modify\_item()** − menja vrednost podatka koji se nalazi u glavnoj memoriji.
* **write\_item()** − upisuje izmenjenu vrednost podatka iz glavne memorije u skladište.

Operacije Begin, Commit, Rollback i Savepoint su operacije za kontrolu transakcije i mogu se kombinovati samo sa operacijama DML (insert, update, delete), ne mogu se koristiti sa operacijama create table ili drop jer se one automatski commit-uju u bazi.



## ACID svojstva transakcije

ACID je akronim koji se odnosi na skup od 4 ključna svojstva koja definišu transakciju: Atomičnost, Konzistentnost, Izolacija i Trajnost. Ako operacija baze podataka ima ova svojstva ACID, ona se može nazvati ACID transakcijom, a sistemi za skladištenje podataka koji primenjuju ove operacije nazivaju se transakcioni sistemi. Transakcije kod PostgreSQL-a ispunjavaju sva četiri ACID svojstva. ACID transakcije garantuju da svako čitanje, pisanje ili modifikacija tabele ima sledeća svojstva:

* **Atomičnost** **(Atomicity)** - Korisnici bi trebalo da posmatraju izvođenje svake transakcije kao atomske. Sve akcije u transakciji (za čitanje, pisanje, ažuriranje ili brisanje podataka) se tretiraju kao jedna celina. Ili se izvršava cela naredba, ili se nijedna naredba iz te celine ne izvršava. Ovo svojstvo sprečava gubitak i oštećenje podataka tako da korisnici ne bi trebalo da brinu o efektima nekompletnih transakcija (recimo, kada se dogodi pad sistema).
* **Konzistencija (Consistency)**  - Osigurava da transakcije unose promene u tabele samo na unapred definisane, predvidljive načine. Konzistentnost transakcija osigurava da oštećenje ili greške u vašim podacima ne stvaraju neželjene posledice po integritet vaše tabele. Drugim rečima, ako je svaka transakcija konzistentna, a stanje baze podataka je bilo takođe u konzistentnom stanju pre izvršenja transakcije, tada stanje baze podatka mora ostati u tom stanju i posle završetka transakcija. Garantovanje ovog svojstva transakcija je odgovornost korisnika.
* **Izolacija (Isolation)** – DBMS prepliće akcije nekoliko transakcija zbog obezbeđivanja boljih performansi, ali i zbog usaglašenosti sa realnim sistemom. Kada više korisnika istovremeno čita i piše iz iste tabele, izolacija njihovih transakcija osigurava da istovremene transakcije ne ometaju ili utiču jedna na drugu. Svaki zahtev se može pojaviti kao da se javlja jedan po jedan, iako se zapravo dešava istovremeno, odnosno uticaj na bazu podataka mora da bude isti kao da se transakcije izvršavanju sekvencijalno.
* **Trajnost (Durability)** - Kada DBMS informiše korisnika da je transakcija uspešno završena (izvršen commit), njeni efekti bi trebalo da budu sačuvani, čak i ako sistem padne, pre nego što su se sve njene promene upisane na disku. Drugim rečima, rezultati transakcija koje su uspešno završene i kao takve potvrđene, pamti se u bazi. Ovo svojstvo se naziva trajnost.

Transakcije mogu da budu nekompletne iz četiri razloga:

* Prvo, transakcija može da bude prekinuta (odbačena) od strane DBMS-a, zato što su se nastale neke anomalije tokom izvršavanja. Ako je transakcija prekinuta od DBMS-a iz nekih internih razloga, ona se automatski restartuje i izvršava ponovo.
* Drugo, sistem može da padne (na primer, zato što je prekinuto snabdevanje strujom) dok je jedna ili više transakcija u izvršavanju.
* Treće, transakcija može da naiđe na neočekivanu situaciju (na primer, čitanje nepredviđene vrednosti podataka ili nije sposobna da pristupi nekom disku) i "reši" da prekine samu sebe.
* Četvrto, može da dođe do fizičkog oštećenja na memorijskim uređajima, kada podaci na njima više nisu dostupni.

Transakcija koja je prekinuta može da ostavi bazu podataka u nekonzistentnom stanju. DBMS mora da odstrani efekte nepotpunih transakcija iz baze podataka, to jest, mora da obezbedi atomičnost transakcija: bilo da su sve akcije transakcija završene ili nisu. DBMS obezbeđuje atomičnost transakcija poništavanjem akcija nekompletnih transakcija. Da bi bio u stanju da ovo čini, DBMS održava dnevnik (log), svih upisivanja u bazu podataka. Dnevnik se, takođe, koristi da obezbedi trajnost: Ako sistem padne pre neko što su promene učinjene kompletnim (transakcije upisane na disk), dnevnik se koristi da zapamti i vrati ove promene kada se sistem restartuje. DBMS komponenta koja omogućava atomičnost i trajnost se naziva rukovalac oporavkom (recovery manager).

Konzistentnost baze podataka sledi iz transakcione atomičnosti i izolacije.

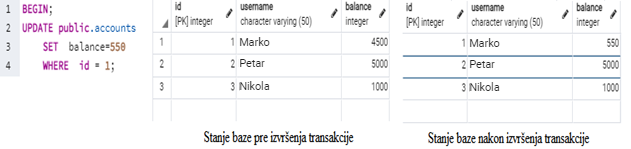
ACID transakcije obezbeđuju najveću moguću pouzdanost i integritet podataka. Takođe osiguravaju da vaši podaci nikada ne padnu u nekonzistentno stanje zbog operacije koja se samo delimično izvršila.



## Implementacija transakcije kod PostgreSQL-a

U PostgreSQL-u, kao i u bilo kojoj drugoj bazi podataka usklađenoj sa ACID svojstvima, svaki iskaz predstavlja transakciju i pokreće se u transakciji: ako iskaz izazove grešku, PostgreSQL poništava sve njene efekte. Kod PostgreSQL-a, svaki iskaz izvan eksplicitno označene transakcije se zapravo izvršava u sopstvenoj transakciji sa jednom naredbom. Da biste eksplicitno pokrenuli blok transakcije, možete koristiti ili BEGIN ili START TRANSACTION komande (one su sinonimi). Da biste izvršili transakciju, izdajte komandu COMMIT.

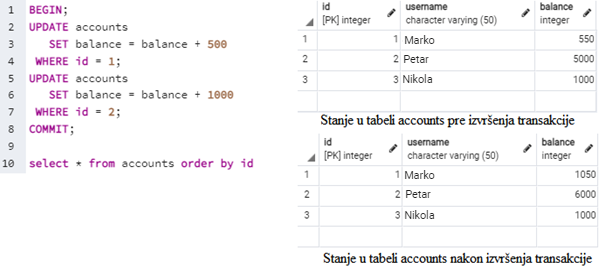
Kao što smo rekli naredba koja označava početak transakcije kod PostgreSQL-a je BEGIN (takođe možemo započeti transakciju naredbama BEGIN TRANSACTION, BEGIN WORK, START), ukoliko ne navedemo početak transakcije ovom naredbom sistem baze podataka neće moći da prepozna iskaz kao transakciju.



Slika 2. primer transakcije bez commitovanja

U primer koji smo naveli na slici iznad započeli smo transakciju naredbom BEGIN, u okviru koje vršimo izmenu vrednosti polja “balance” sa odgovarajućim id-jem. Nakon uspešnog izvršenja transakcije (što se vidi na osnovu rezultata) sve promene koje su izvršene u okviru nje vidljive su samo u okviru te sesije ali ne i izvan nje, tako da ukoliko u okviru druge sesije zatražimo podatke dobićemo ne ažuriranu verziju tih podataka. Sve izmene biće izgubljene nakon osvežavanja baze, ili u slučaju prestanka rada sistema. Razlog za to je što nije izvršena naredba COMMIT na kraju transakcije.

Naredba COMMIT (ili COMMIT WORK ili END TRANSACTION) se koristi za čuvanje promena i njihovo prikazivanje u bazi podataka. Nakon Commit naredbe sve promene su trajne i dostupne za sve sesije čak i u slučaju prestanka rada sistema.



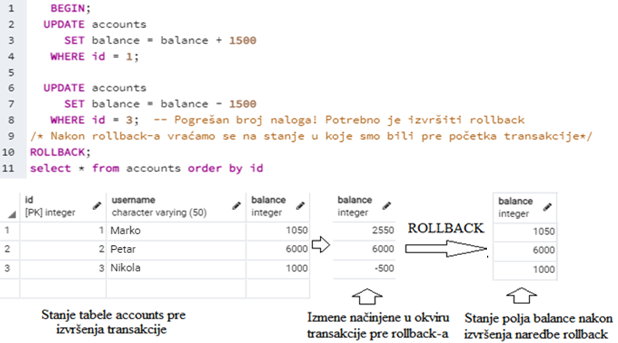
Slika 3. Primer uspešno izvršene transakcije

Na slici 3. Prikazan je primer u okviru kog se izvršavaju dve naredbe kao deo jedne te iste transakcije. Na osnovu navedenog rezultata vidimo da je su naredbe uspešno izvršene. Kako bi smo završili transakciju i trajno sačuvali u bazi podataka sve izmene koje su ostvarene u toku te transakcije izvršavamo naredbu COMMIT.

Kao što je već rečeno, unutar transakcije, svi ili nijedan iskaz će biti predati bazi podataka. Prekid izvršenja izjave i modifikacija napravljenih unutar transakcije umesto da se primene na bazu podataka poznato je kao povratak ili ROLLBACK transakcije.

Transakcije se mogu vratiti automatski ili ručno. PostgreSQL automatski poništava transakciju ako jedna od izjava u okviru ne rezultira greškom. Takođe poništava transakciju ako bi došlo do greške u serijalizaciji ako to ne dozvoljava izabrani nivo izolacije.

Da biste ručno vratili izjave koje su izvršene tokom trenutne transakcije, možete koristiti naredbu ROLLBACK. Ovo će poništiti sve izjave unutar transakcije, vraćajući bazu u stanje u kome se nalazila na početak transakcije

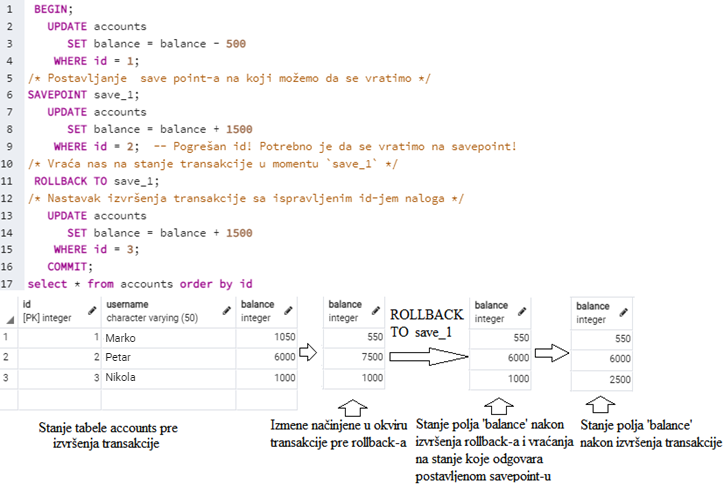


Slika 4. Primer transakcije poništene naredbom Rollback

Na primeru iznad pokazano je kako funkcioniše naredba rollback. Iako su izvršene određene promene nad podacima iz baze podataka u okviru transakcije (pre izvršenja naredbe rollback), sve te promene biće poništene nakon izvršenja ove naredbe, odnosno izmenjeni podaci neće biti upisane u bazu.

Podrazumevano, komanda ROLLBACK vraća transakciju na mesto gde je bila kada su prvi put pozvane komande BEGIN ili START TRANSACTION. Ali šta ako želimo samo da poništimo neke izjave unutar transakcije?

Iako ne možemo da navedete proizvoljna mesta na koja bi se vratili kada se izvrši komanda ROLLBACK, možemo da se vratitmo na bilo koju tačku čuvanja (SAVEPOINT) koju smo postavili tokom transakcije. Moguće je unapred označiti mesta u transakciji pomoću komande SAVEPOINT, a zatim kada treba izvršiti ROLLBACK umesto vraćanja na sam početak transakcije možemo se vratiti na te specifične lokacije koje smo postavili. Ove tačke čuvanja vam omogućavaju da kreirate srednju tačku vraćanja, sa koje bi ste mogli da nastavite izvršavanje transakcije.



Slika 5. Primer transakcije sa Savepoint-om

Na slici 5. prikazan je upit izvršenja transakcije koja koristi SAVEPOINT. U okviru transakcije kreirana je tačka čuvanja “save\_1” u okviru koje se pamti stanje podataka koji su bili izmenjeni do tog trenutka. Kada dođe do neke greške prilikom izmene podataka možemo pozvati nardebu ROLLBACK TO save\_1 čime ćemo poništiti sve izmene koje su izvršene i vratićemo se u stanje u koje su se podaci nalazili u trenutku kreiranja tačke čuvanja save\_1. Nakon vraćanja na tačku čuvanja možemo nastaviti sa izvršavanjem transakcije preskačući deo koji je pogrešno izvršen i koji je izazvao rollback.

Potrebno je naglasiti da neuspeh transakcije ni na koji način ne utiče na sadržaj baze podataka koju je transakcija trebalo da promeni. To je zato što je više koraka radnji spojeno u jedan korak, što implicira da ako korak ne uspe, nijedna promena se ne ažurira u bazi podataka.

# Konkurentnost kod transakcija

U ovom poglavlju biće opisano ponašanje PostgreSQL sistema baze podataka kada dve ili više sesija pokušavaju da pristupe istim podacima u isto vreme. Ciljevi u toj situaciji su da se omogući efikasan pristup svim sesijama uz održavanje striktnog integriteta podataka.

PostgreSQL pruža bogat skup alata za programere za upravljanje istovremenim pristupom podacima. Za raliku od ostalih sistema baza podataka koji koriste standardna zaključavanja, kod PostgreSQL konzistentnost podataka se održava korišcenjem MVCC-a (Multiversion Concurrency Control). To znači da dok se izvršavaju upiti nad bazom svaka transakcija vidi verziju podataka kakva je bila u nekom prethodnom trenutku, bez obzira na trenutno stanje podataka. Ovo sprečava izjave da pregledaju nedosledne podatke proizvedene istovremenim transakcijama koje izvode izmene nad istim redovima podataka, obezbeđujući izolaciju transakcija za svaku sesiju baze. MVCC minimizira sukob oko zaključavanja, čime pruža bolje performanse u višekorisničkom okruženju.

Glavna prednost MVCC modela u odnosu na standardna zaključavanja je ta što lock-ovi stečeni za upite čitanja podataka nisu u sukobu sa lock-ovima stečenim za upis podataka, tako da čitanje nikad ne blokira upis podataka i obrnuto. PostgreSQL održava ovu garanciju čak i kada obezbeđuje najstroži nivo izolacije transakcija kroz korišćenje inovativnog nivoa izolacije serijskih snimaka (Serializable Snapshot Isolation level - SSI)

Mogućnosti zaključavanja na nivou tabele i reda su takođe dostupne u PostgreSQL-u za aplikacije kojima generalno nije potrebna potpuna izolacija transakcija i preferiraju da eksplicitno upravljaju određenim tačkama sukoba. Ali korišcenje MVCC-a će generalno obezbediti bolje performanse od zaključavanja.

U nastavku biće detaljno opisani mehanizmi za izolacije i zaključavanja koji se takođe koriste za kontrolu konkurentnosti prilikom izvršavanja transakcija.



## Izolacija transakcija kod PostgreSQL-a

PostgreSQL je u potpunosti kompatibilan sa ACID-om i tako implementira izolaciju transakcija. Istovremeno izvršavanje transakcija predstalja veliki problem. Korišćenjem niova izolovanosti definiše se nivo do kojeg transakcija treba da bude izolovana od izmene podataka izvršenih od drugih konkurentnih transakcija.

U zavisnosti od slučaja korišćenja moguće je izabrati različiti nivo izolacije od istovremene aktivnosti. Jednostavan slučaj upotrebe koji zahteva izolaciju su online rezervne kopije. Aplikacija za rezervnu kopiju za PostgreSQL je pg\_dump, a uloga ove aplikacije je da napravi snimak (sanpshot) za celu vašu bazu podataka i izvezite je kao rezervnu kopiju. Ovo zahteva da čitanja koja izvodi pg\_dump budu potpuno izolovana od bilo koje istovremene aktivnosti pisanja u sistemu, a ovo se dobija korišćenjem nova izolacije repeatable read ili serijalizacijom.

Izolacija transakcija je definisana SQL standardom i implementirana u PostgreSQL-u. SQL standard definiše četiri nivoa izolacije transakcija, a to su:

* **Read uncommitted**
* **Read committed**
* **Repeatable read**
* **Serializable**

Najstrožije nivo izolacije je Serializable, što je definisano standardom koji kaže da svako istovremeno izvršavanje skupa transakcija koje se mogu serializovati garantuje da će proizvesti isti efekat kao i njihovo pokretanje jedne po jedne u nekom redosledu. Ostala tri nivoa su definisana u terminima fenomena, koji su rezultat interakcije između istovremene transakcije, koje se ne smeju dešavati na svakom nivou.

U zavisnosti od toga koji nivo izolacije koristimo mogu se javiti različiti fenomeni. Fenomeni koji su zabranjeni na različitim nivoima su:

**Dirty read** – Do dirty read-a dolazi kada transakcija čita podatke koje je zapisala druga istovremena nezavršena transakcija.

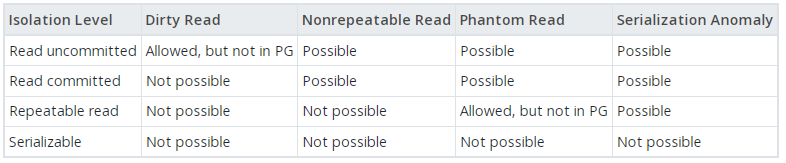
**Nonrepeatable read** – javlja se u slučaju kada transakcija ponovo čita podatke koje je prethodno pročitala i otkriva da su podaci modifikovani od strane druge transakcije (koja je izvršena od početnog čitanja).

**Phantom read** – Do Phantom read-a dolazi kada transakcija ponovnim izvršavanjem upita dobije skup redova koji zadovoljavaju uslov pretrage ali se taj dobijeni skup redova razlikuje od prethodno dobijenog izvršavanjem tog istog upita, što je posledica izmena izvršenih nad tim redovima od strane neke druge istovremene transakcije.

**Serialization anomaly** - Rezultat uspešnog izvršavanja grupe transakcija nije u skladu sa rezultatom dobijinim izvođenjem tih transakcija jedne po jedne u svim mogućim redosledima, odnosno dolazi do nekonzistentnosti rezultata.

Gore navedena četiri nivoa izolacije samo definišu koji od nabrojanih fenomeni se ne smeju desiti, a ne koji se fenomeni moraju desiti.

SQL standard i nivoi izolacije transakcija implementirani u PostgreSQL opisani su u tabeli prikazanoj na slici ispod.



Slika 6. Nivoi izolacije transakcija kod PostgreSQL

U PostgreSQL-u možemo zahtevati bilo koji od četiri standardna nivoa izolacije transakcija, ali interno se primenjuju samo tri različita nivoa izolacije a to su: Read commited, Repeatable read, Serializable read. PostgreSQL-ov režim Read Uncommitted se ponaša kao Read Committed. To je zato što je to jedini razuman način da se standardni nivoi izolacije mapiraju na viševerzionu arhitekturu kontrole konkurentnosti PostgreSQL-a.

Na osnovu tabele prikazane na slici 6. možemo primetiti da PostgreSQL u slučaju implementacije Repeatable Read izolacije ne dozvoljava Phantom Read. Takođe vidimo da ni u jednom slučaju nivoa izolacije Dirty Read se ne može desiti u PostgreSQL-u i da u slučaju Serializable nivoa izolacije ni jedna anomlija nije dozvoljena.

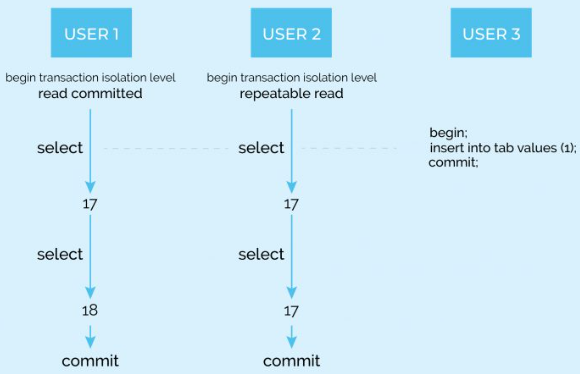
Postavljanje nivoa izolacije vrši se unutar transakcije pre bilo koje DML naredbe (insert, update, delete…), jer u suprotnom neće imati nikakav efekat. Postavljeni nivo izolacije odnosi se samo na tu transakciju unutar koje je postavljen i menja se podrazumevano postavljeni nivo. Naredba kojom se postavlja nivo izolacije kod PostgreSQL-a je:

*SET TRANSACTION ISOLATION LEVEL* (ili BEGIN TRASACTION ISOLATION LEVEL) i navođenja imena željenog izolacionog nivoa .

U nastavku će biti opisani pomenuti nivoi izolaci koji se koriste kod PostgreSQL-a.

**Read Committed –** Read Committed je podrazumevani nivo izolacije u PostgreSQL-u. Kada transakcija koristi ovaj nivo izolacije, upit SELECT (bez klauzule FOR UPDATE/SHARE) vidi samo podatke izmenjene pre nego što je upit počeo. Uzastopne komande SELECT mogu videti različite podatke, čak i ako se nalaze unutar jedne transakcije, ukoliko druge transakcije sačuvaju izvršene promene nakon što prvi SELECT započne i pre nego što počne drugi SELECT.

**Repeatable read** – Kao i kod Read Commited nivoa, transakcija vidi samo podatke izmenjene pre nego što je upit počeo. On je viši nivo (drugi po redu) u odnosu na Read Commited i garantuje da jednom pročitani podaci unutar transakcije ne mogu biti izmenjeni izvan te transakcije, dok se ona ne završi. Na slici ispod prikazana je razlika u pristupu podacima unutar transakcije između ova dva nivoa.



Slika 7. Poređenje pristupa podacima unutar tranksacije kod read commited i repeatable read niova

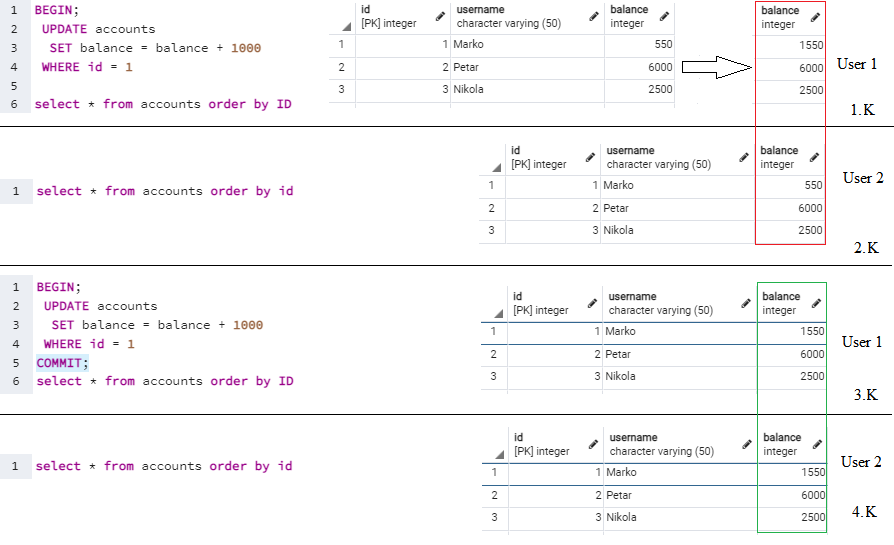
**Serializable** – predstavlja najviši nivo izolacije, gde su transakcije u potpunosti izolovane međusobno. Ovaj nivo izolacije dozvoljava da se transakcije izvode istovremeno, stvarajući efekat da se transakcije izvode u serijskom redosledu. Transakcije obezbeđuju zaključavanje za operacije čitanja i pisanja. Serializable je podrazumevani nivo izolacije po SQL standardu.

U nastavku biće prikazani pojedini primeri upotrebe nivoa izolacije kod PostgreSQL.

1. Slučaj Dirty Read-a - kada jedna (nedovršena) transakcija ubaci redove u tabelu, a druga (takođe nedovršena) transakcija pokuša da pročita sve redove iz tabele.

U slučaju da se dozvoli dešavanje Diry Read-a prva transakcija može vratiti vrednosti koje je izmenila, a druga transakcija bi onda čitala „fantomske“ redove koji nikada nisu postojali, zbog toga je potrebno sprečiti da se desi Dirty Read.

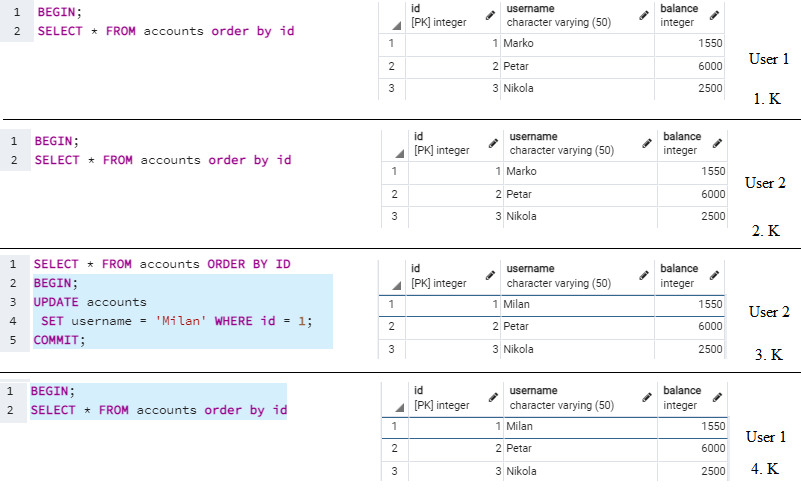
Dirty read može se desiti samo u slučaju korišćenja Read Uncommited niova izolacije, ali budući da ,kao što smo već rekli, PostgreSQL ne podržava ovaj nivo izolacije ni u kom slučaju ne može doći do njega. Na sledećoj slici prikazaćemo primer pokušaja izazivanja Dirty read-a kod Postgre-a, pri čemu će transakcije koristiti podrazumevani nivo izolacije (Read commited, tako da nije potrebno postaviti nivo), i videti kako se sistem ponaša u ovom slučaju.



Slika 8. Primer pokušaj stvaranja Dirty reada kod PostgreSQL-a

Kao što možemo videti na slici 8. u prvom koraku (1.K) korisnik pod nazivom User 1 izvršava neke izmene nad podacima u okviru transakcije. Izmene nad podacima su vidljive samo lokalno u okviru te sesije, jer nije izvršena naredba commit koja bi učinila te izmene trajnim. U drugom koraku (2.K), korisnik User 2 pribavlja podatke iz baze (koje je korisnik 1 izmenio u prvom koraku), ali kao što vidimo pribavljeni podaci imaju originalne vrednosti koje su bile i pre nego što je korisnik 1 izmenio. U trećem koraku, korisnik 1 je izvršio naredbu COMMIT, čime je trajno sačuvao izvršene izmene iz koraka 1.. U poslednjem koraku korisnik 2 ponovo zahteva podatke iz baze i kao rezultat dobija ažurirane vrednosti podataka. PostgreSQL nije dozvolio da dođe do čitanja izmenjenih podataka koji nisu commit-ovani, čime je sprečen Dirty Read.

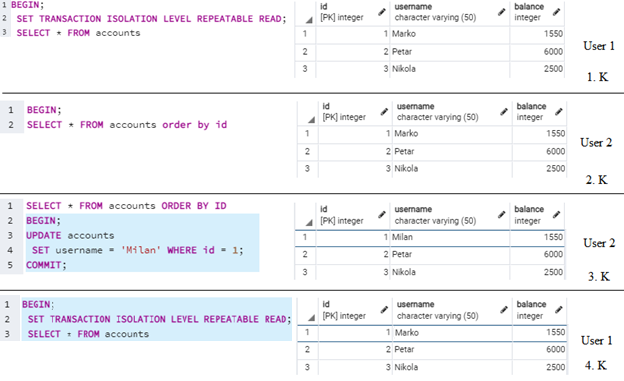
1. Slučaj pojave i rešavanja Nonrepeatableread-a – Kao što smo rekli Non-repeatable read se dešava kada transakcija pročita red, a zatim ga ponovo pročita malo kasnije, ali dobije drugačiji rezultat. Primer pojave non-repeatable čitanja prikazan je na slici ispod:

Slika 9. Problem Non-repeatable čitanja podataka

Na osnovu prikazanog primera vidimo da su rezultati izvršenih upita iz prvog i drugog koraka (1.K,2.K) za oba korisnika isti. U oba slučaja započeta je transakcija i izvršen je upit čitanja, ali transakcije nisu završene naredbom commit. U trećem koraku, korisnik 2, izvršio je izmene nad podacima, nakon commit-ovanja te izmene su postale trajne i vidlje u ostalim sesijama. Nakon toga, korisnik 1 ponvo u okviru iste transkcije izvršava operaciju čitanja podataka (kao i u prvom koraku), ali je dobio drugačiji rezultat u odnosu na prvo izvršenje. Potrebno je napomenuti da smo u ovom slučaju koristili potrazumevani nivo izolacije koji koristi PostgreSQL, pa se zbog toga i desio Non-repetable read.

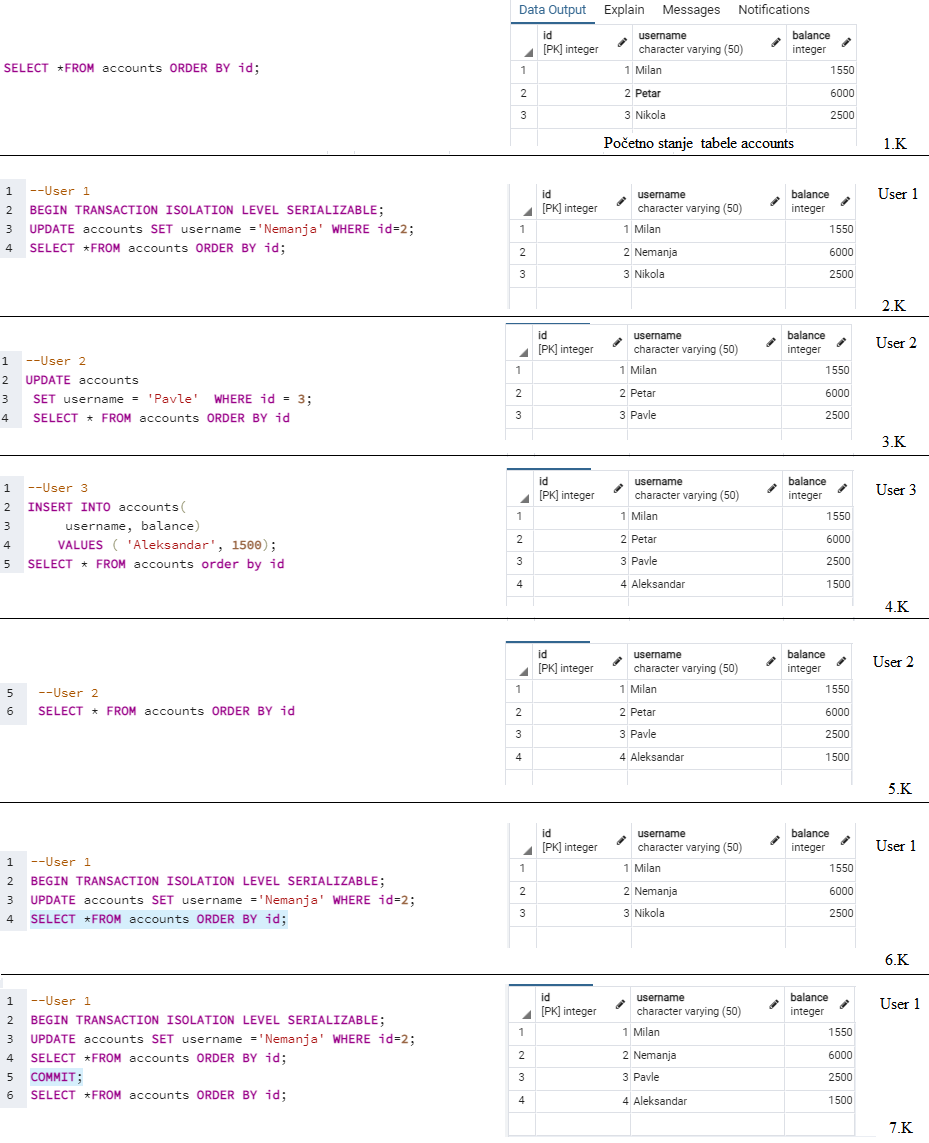
Kako bi smo rešili ovaj problem potrebno je pre izvršenja upita postaviti nivo izolacije na Repeatable read. PostgreSQL će tada osigurati da će drugo (ili bilo koje) čitanje takođe vratiti isti rezultat kao prvo čitanje.

Na slici 10. biće prikazan primer istog scenarija ali sa nadograđenim nivoom izolacije, kako bi se rešio problem Non-repeatable read-a:



Slika 10. Rešenje problema Non-repeatable read-a korišćenjem Repeatable nivoa izolacije

1. Slučaj primene Serializable nivoa izolacije podataka – On predstavlja jednu vrstu proširene verzije Repeatable nivoa izolacije. Kao što smo rekli korišćenjem Serilazied nivoa, ne možemo modifikoati podatke dok druga transakcija čita te iste podatke. Za Select-only transakcije koritimo Serializable nivo kada ne želimo izmene načinjene od drugih transakcija tokom transakcije koju izvršavamo. Za Update i Delete transakcije, Serializable izolacija sprečava istovremenu modifikacije istog reda, stoga ga treba koristiti oprezno. Na slici ispod biće prikazan primer Serializable nivoa izolacije.



Slika 11. Primer korišćenja Serializable nivoa izolacije

Na slici 11. u prvom koraku vidimo početno stanje tabele accounts. U drugom koraku (2.K) podešavamo nivo izolacija na Serializable za transakciju koju izvršava korisnik 1, a zatim vrši izmenu polja username čiji je id=2 i izmena je uspešno obavljena. Paralelno sa tim korisnik 2 u okviru svoje transakcije vrši izmenu polja sa id-jem 3, dok korisnik 3 ( u četvrtom koraku) vrši dodavanje u tabelu. Izmene koje su napravili korisnik 2 i korisnik 3 nad tabelom, su međusobno vidlji između ova dva korisnika, takođe su ovim korisnicima vidljive i izmene koje je u drugom koraku napravio korisnik . Međutim izmene napravljene od strane korisnika 2 i 3 nisu vidljive korisniku 1, kao što vidimo u koraku 6 (6.k). Tako kada korisnik 1 izvrši naredbu commit (kao u koraku 7.k) izmenje načinjene od ostalih korisnika postaće vidljive i ovom korisniku. Ovo je način na koji funkcioniše Serializable nivo izolacije.

## Eksplicitna zaključavana transakcija kod PostgreSQL-a

PostgreSQL zaključavanje je jedna od kritičnih tema kod PostgreSQL-a, posebno za programere koji rade sa bazama podataka. Zaključavanje kod PostgreSQL-a pomaže nam sa konkurentnim pristupom ili modifikacijama objekata baze podataka izazivanjem zaključavanja čim se naredba izvrši. Tipovi zaključavanja zavise od tipa naredbe koja se izvršava.

PostgreSQL obezbeđuje različite režime zaključavanja radi upravljanja istovremenih pristupa podacima u tabelama. Ovi režimi se mogu koristiti za zaključavanja koje kontroliše aplikacija u situacijama kada MVCC ne daje željni efekat. Takođe, većina PostgreSQL naredbi automatski obezbeđuje zaključavanje u odgovarajućem režimu, da bi obezbedile da referencirane tabele ne budu izbrisane ili izmenjene dok se naredba izvršava. (Npr. naredba TRUNCATE ne može biti izvršena bezbedno istovremeno sa drugim operacija na istoj tabeli, tako da ovaj slučaj zahteva ACCESS EXCLUSIVE režim zaključavanja tabele da bi se to izvelo.)

PostgreSQL podržava tri mehanizma zaključavanja:

* **Table Level Locks** – zaključavanje na nivou tabele
* **Row Level Locks** – zaključavanje na nivou reda
* **Advisory Locks** – zaključavanje na nivou aplikacije

Neka zaključavanja su izazvana implicitno od strane PostgreSQL-a, npr. kada se pokrene neki SQL upit. Implicitno zaključavanje znači da će se zaključavanje podrazumevano isključiti kada se transakcija izvrši. Zaključavanja takođe mogu biti eksplicitno izazvana od strane korisnika korišćenjem izjave WITH LOCK. Postgre sintaksa za eksplicitno zaključavanje tabele bi bila:

*Lock table name\_of\_table IN [Mode of locking] [NOWAIT]*

Table level i Row-level zaključavanja mogu biti eksplicitna ili implicitna, dok Advisory zaključavanja mogu biti samo eksplicitna.

Prilikom korišćenja zaključavanja moguće je doći do međusobnog konflikta, jer neki režimi zaključavanja su u sukobu sa drugim. Npr.kada se nekom transakcijom izazove ACESS EXCLUSIVE režim zaključavanja, druga transakcija ne može pribaviti ni jedno drugo zaključavanje, iz razloga što režim ACCESS EXCLUSIVE je u sukobu sa svim drugim tipovima zaključavanja. Tako da u tom slučaju druga transakcija treba da sačeka u red da prva transakcija završi sa zaključavanjem.

Transakcija ne može biti u konfliktu sama po sebi. To znači da transakcija može koristiti ACCESS EXCLUSIVE i ACCESS SHARE režime zaključavanja istovremeno. Nekonfliktna zaključavanja se mogu držati od strane više transakcija istovremeno.

Zaključavanje se oslobađa kada transakcija izvrši trajno upisivanje podataka u bazu korišćenjem naredbe commit ili se vrati na početno stanje izvršavanjem naredbe rollback.

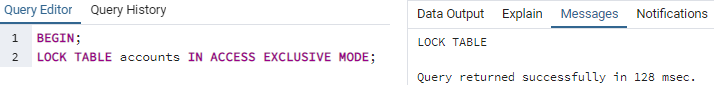
### Table-Level Locks (Zaključavanja na nivou tabele)

Table-level locks su automatski izazvana zaključavanja od strane PostgreSQL-a, ali mogu biti izazvana i eksplicitno korišćenjem LOCK komande. Table-level locks kao što sam naziv govori odnose se na celu tabelu.

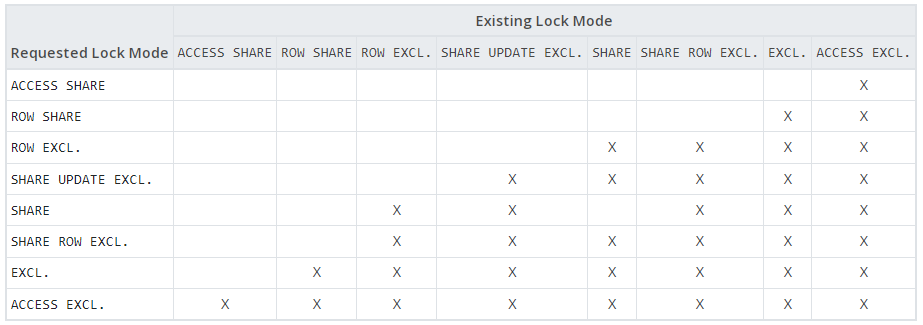
Dostupni režimi zaključavanja na nivou tabele kod PostgreSQL-a su:

* **ACCESS SHARE** – Naredba SELECT izaziva zaključavanje ovog tipa, odnosno svaki upit koji samo čita podatke iz tabele i ne menja ih izazvaće ACCESS SHARE zaključavanje. U konfliktu je samo sa ACCESS EXCLUSIVE režimom.
* **ROW SHARE** – Ovaj režim koristi se prilikom izvršavanja naredbi SELECT FOR UPDATE i SELECT FOR SHARE. U konfliktu je sa EXCLUSIVE i ACCESS EXCLUSIVE režimima.
* **ROW EXCLUSIVE** – Koristi se prilikom izvršavanja naredbi UPDATE, DELETE i INSERT. U konfliktu je sa režimima SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, i ACCESS EXCLUSIVE.
* **SHARE UPDATE EXCLUSIVE** – Ovaj model štiti tabelu od konkurentnih izmena i vakuma. Koristi se kod naredba VACUMM, ANALYZE, CREATE INDEX CONCURRENTLY i nekih oblika ALTER TABLE naredbi. U konfliktu je sa SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, i ACCESS EXCLUSIVE režimima
* **SHARE** – Koristi se kod CREATE INDEX naredbe. Štiti tabelu od konkurentnih izmena podataka. U konfliktu je sa ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, and ACCESS EXCLUSIVE.
* **SHARE ROW EXCLUSIVE** – Ovaj režim štiti tabelu od istovremenih izmena podataka i samoisključiv je, tako da ga može držati samo jedna sesija istovremneo. Izazivaju ga naredbe CREATE TRIGGER i neki oblici ALTER TABLE naredbe. U konfliktu je sa ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, i ACCESS EXCLUSIVE režimima.
* **EXCLUSIVE** - Samo operacije čitanja mogu da se pridruže paralelno transakciji koja koristi ovaj režim zaključavanja. Koristi se kod REFRESH MATERIALIZED VIEW CONCURRENTLY naredbe. Ovaj režim dozvoljava samo istovremeno korišćenje ACCESS SHARE režima.
* **ACCESS EXCLUSIVE** – Koristi se kod naredba ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACCUM FULL i REFESH MATERIALIZED VIEW. Ovo je podrazumevani režim zaključavanja za LOCK TABLE naredbu koja ne specificira konkretni režim. Ovaj mod je u konfliktu sa svim ostalim režimima. On garantuje da je transakcija koja ga koristi ujedno i jedina transakcija koja pristupa tabeli u tom trenutku na bilo koji način.

Primer sintakse zaključavanja tabele korišćenjem Access Exslusive režima:



Jednom stečeno, zaključavanje se obično drži do kraja transakcije. Ali ako je zaključavanje stečeno nakon uspostavljanja tačke čuvanja, zaključavanje se odmah otpušta ako se vrati na tačku čuvanja. Ovo je u skladu sa principom da ROLLBACK poništava sve efekte komandi od tačke čuvanja.



Slika 12. Međusobni konflikti između režima za zaključavanja na nivou tabele

### Row-Level Locks (Zaključavanje na nivou reda)

Pored zaključavanja na nivou tabele, posoje i zaključavanje na nivou reda, koja su navedeni u redosledu u kome se koriste automatski od strane PostgreSQL-a .

Transakcija može držati međusobno konfliktna zaključavanja u istom redu, čak i u različitim podtransakcijama, ali osim toga dve transakcije nikad ne mogu zaključati isti red koristeći međusobno konfliktne režime.

Zaključavanja na nivou reda ne blokiraju upite podataka, samo blokiraju druge transakcije koje menjaju iste redove. Zaključavanja na nivou reda se oslobađaju na kraju transakcije ili tokom vraćanja na Save Point, isto kao kod zaključavanja na nivou tabele.

U starijim verzijama PostgreSQL-a postojale su samo dve tipa zaključavanja na nivou reda, dok sada se koriste četiri režima zaključavanja i to su:

* **FOR UPDATE** – Dovodi do toga da redovi koji su preuzeti naredbom SELECT budu zaključani kao da je reč o slučaju ažuriranja. Ovo ih sprečava da budu zaključani, modifikovani ili obirsani od strane drugih transakcija, dok se trenutna transakcija ne završi. SQL naredba navedena ispod izaziva ekskluzivno zaključavanje na nivo reda bez stvarne izmene.

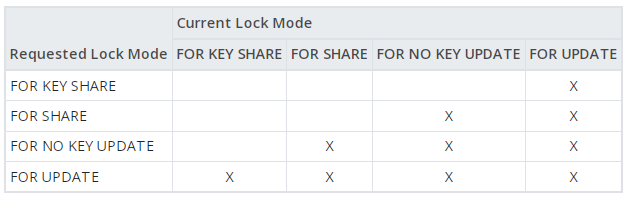
*select \* from table where id > 10 and id < 15 FOR UPDATE;*

Dakle, druga transakcija ne može istovremeno da drži ovu vrstu zaključavanja na redovima čiji je id između 10 i 15, ali može da ih čita.

* **FOR NO KEY UPDATE** – Ponaša se slično kao FOR UPDATE, ali je manje restriktivna. Ovaj režim zaključavanja neće blokirati SELECT FOR KEY SHARE naredbe koje pokušavaju da ostvare zaključavanje na istim redovima.
* **FOR SHARE** – Dozvoljava drugoj transakcijia da obezbede isti tip zaključavanja ali ne može modifikovati ili brisati selektovane redove, ili obezbediti neki od exclusive lock-ova nad tim podacima. Primer SQL naredba koja izaziva shared zaključavanje na nivou reda navedena je ispod:

*select \* from table where id > 10 and id < 15 FOR SHARE;*

* **FOR KEY SHARE** – Ponaša se slično kao FOR SHARE, ali je manje restriktivna: SELECT FOR UPDATE se blokira, ali ne i SELECT FOR NO KEY UPDATE. Ovaj režim blokira druge transakcije od izvršavanja DELETE ili bilo koje vrste UPDATE naredbe koje bi izmenile vrednosti ključa. Ne sprečavaju SELECT FOR NO KEY UPDATE, SELECT FOR SHARE ili SELECT FOR KEY SHARE

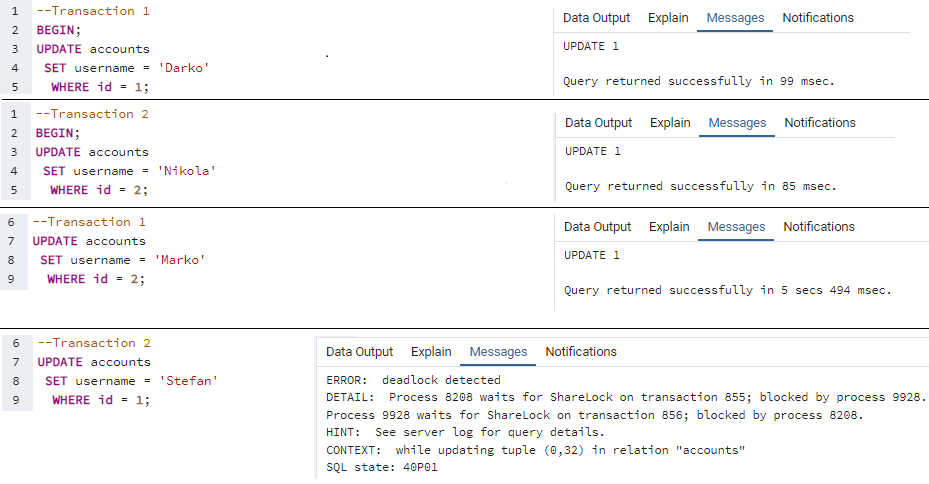


Slika 13. Međusobni konflikti između režima za zaključavanja na nivou reda

Neprikladno korišćenje zaključavanja na nivou redova može izazvati pojavu određenih zastoja odnosno DeadLock-ova prilikom izvršavanja transakcija.

### DeadLock-ovi (Zastoji)

U istovremenim sistemima gde su resursi zaključani, dva ili više procesa mogu završiti u stanju u kojem svaki proces čeka na drugi. Ovo stanje se naziva zastoj ili DeadLock. Zastoji su važni problemi koji se mogu desiti u bilo kojoj bazi podataka. Korišćenje eksplicitnih zaključavanja može uvećati verovatnoću pojave zastoja. Kada se desi zastoj, PostgreSQL ga detektuje, zaustavlja jednu od transakcija koje su izazvale zastoj, dozvoljavajući drugoj (ili drugim) transakcijama da nastave izvršavanje. Primer DeadLock-a biće prikazan na slici ispod a zatim će biti pojašnjen kako bi razumeli šta se dogodilo.



Slika 14. Izazivanja DeadLock-a kod PostgreSQL

Kao što vidimo na primeru navedenom na slici 14. u prvom trenutku transakcija 1 koristi zaključavanje na nivou reda nad red čiji je id=1 i ažurira polje username sa tim id-jem. Nakon toga, transakcija 2 takođe koristi zaključavanje na nivou reda nad red čiji je id=2 i ažurira polje u tom redu. Nakon što je transakcija 2 ažurirala polje, transakcija 1 pokušava takođe da pristupi i izmeni to isto polje sa id=2 koje je zaključano od strane transakcije 2, ali mora da čeka dok se ne izvrši transakcija 2. Međutim druga transakcija takođe u nastavku pokušava da ažurira polje koje je prethodno zaključala i ažurirala transakcija 1 (polje sa id=1), ali takođe mora da sačeka da se transakcija 1 završi kao bi se polje oslobodilo. U ovom trenutku ove dve transakcije se čekaju međusobno, nakon čega dolazi do detektovanja DeadLock-a od strane PostgreSQL-a. PostgreSQL rešava DeadLock (u ovom slučaju), tako što zaustavlja drugu transakciju, poništavajući izmene koje je napravila, a prva transakcija se izvršava nesmetano. Prva transakcija dobija poruku o uspešno izvršenoj izmeni podataka, dok druga transakcija dobije poruku o DeadLocku.

Najbolja odbrana protiv DeadLock-ova je njihovo izbegavanje, a to se može postići vođenjem računa o tome da sve aplikacije koje koriste bazu podataka preuzimaju zaključavanje podataka u konzistentnom redosledu. U primeru iznad, da su obe transakcije izmenjivale podatke u istom redosledu ne bi došlo do pojave deadlock-a. Sve dok se ne otrkije deadlock, transakcija koja traži zaključavanje na nivou tabele ili reda čekaće neograničeno vreme da se otpuste konfliktna zaključavanja.

### Advisory Locks (Zaključavanja na nivou aplikacije)

PostgreSQL Advisory Locks su zaključavanja na nivou aplikacije koja se eksplicitno zaključavaju i otključavaju pomoću koda aplikacije korisnika. Možemo ih koristiti, npr. da ograničimo istovremenost izvršavanja određenog dela koda u našoj aplikaciji koja radi sa podacima iz određene tabele, dok drugi delovi aplikacije koji pristupaju toj istoj tabeli ne bi primetili. Zaključavanja na nivou aplikacije mogu biti korisne kod strategija zaključavanja koje se ne mogu ostvariti korišćenjem MVCC modela. Pomoću Advisory Lock-a možemo kontrolisati sinhronizaciju koda koji ne upravlja podacima iz baze podataka (npr. kreranje API poziva, pristup podacima sa diska…). Zaključavanja na nivou aplikacije mogu biti ostvarena na nivou sesije ili na nivou transakcije.

Nivo sesije znači da se zaključavanje dobija za povezivanje sa bazom podataka. Zahtevi za zaključavanje se mogu gomilati, tako da ako se zatraži zaključavanje istog broja dva puta u okviru iste veze, biće dodeljeno dva puta, a zatim pre nego što druge veze budu mogle da pristupe, potrebno je obaviti otključavanje i to onoliko puta koliko je izvedeno zaključavanje.

Zahtevi za za zaključavanje na nivou transakcije se, s druge strane, ponašaju više kao redovni zahtevi za zaključavanje, oni se automatski oslobađaju na kraju transakcije I ne postoji eksplicitna operacija otključavanja. Ovo ponašanje je često pogodnije od ponašanja na nivou sesije za kratkoročno korišćenje advisory lock-a.

Advisory lock-ovi kao i regularna zaključavanja koja PostgreSQL nudi ( zaključavanja na nivou tabele i reda) čuvaju se u deljenoj memoriji u pool-u čiju veličinu definišu konfiguracione promenljive *max\_locks\_per\_transaction* i *max\_connections*. Potrebno je voditi računa o tome da se ne iskoristi ova meorija jer u tom slučaju server neće moći odobriti ni jedno zaključavanje. Ovim se nameće gornja granica broja zaključavanja koje server može da odobri.

Sintaksa koja se koristi za dobijanje zaključavanja je:

*pg\_try\_advisory\_lock(key);* - Ovo je funkcija koju koristi Postgre, kojom se pokušava da se obezbedi zaključavanje. Ako je zaključavanje uspešno dodeljeno, Postgre će vratiti kao rezultat “true”, uspurotnom će vratiti “false” ako zaključavanje nije dodeljeno. Argument Key je indetifikator za zaključavanje.

Ako smo npr. izvršili: *Select* *pg\_try\_advisory\_lock(1)* i uspešno smo obezedili zaključavanje za našu transakciju, druga sesija koja pokuša da pozove istu ovu naredbu dobiće kao rezultat “false”. Međutim, ta druga sesija bi možda mogla da obezbedi zaključavanje sa drugim ključem, kao npr: *Select* *pg\_try\_advisory\_lock(2).*

Sintaksa koja se koristi za oslobađanje obezbeđenog zaključavanja:

*Select pg\_advisory\_unlock(n)*– Pozivanjem ove funkcije biće oslobođena jedna rerferenca za zaključavanje čiji je id = n.

*Select pg\_advisory\_unlock\_all() –* Pozivanjem ove funkcije oslobodiće se svi advesory lock-ovi koje su se koristili u trenutnoj sesiji u kojoj su bili zauzeti.



Slika 15. Primer zaključavanja i otključavanja kod Advisery lock-a

Na slici 15. Prikazan je konkretan primer korišćenja funkcija za dobijanje zaključavanja kod Advisesry lock-a, kao i način na koji se vrši oslobađanje svih ili pojedinačnih referenci za zaključavanje.

# Zaključak

Ključna stvar prilikom razvoja aplikacija i baze podataka je da se olakša pristup zajedničkim podacima velikom broju korisnika u isto vreme. Istovremeni pristup podacima bi bio relativno lak kada bi svi korisnici samo čitali podatke, jer u tom slučaju ne bi postojala mogućnost međusobnog mešanja ili gubitka podataka. Međutim, kada više korisnika istovremeno pristupa bazi podataka, a najmanje jedan korisnik vrši ažuriranje podataka, može doći do smetnje, što dalje može dovesti do nedoslednosti podataka.

Kako bi se sprečila nekonzistentnost podataka u okviru baze, mogu se koristiti različiti mehanizmi koje PostgreSQL nudi. PostgreSQL obezbeđuje konkurentan pristup bazi podataka korišćenjem transakcija koje ispunjavaju ACID svojstva. Upotrebom transakcija, korisniku se pruža mogućnost potpune kontrole načina na koji će se pristupati podacima u bazi. U zavisnosti od slučaja korišćenja PostgreSQL daje mogućnost odabira različitih nivoa izolacije od istovremene aktivnosti, koje korisnik može da definiše na početku transakcija. Takođe pored određivanja nivoa izolacje, korisniku se stavlja na raspolaganje korišćenje eksplicitnog zaključavanja podataka, iako većina PostgreSQL naredbi automatski obezbeđuje zaključavanje u odgovarajućem režimu.

U okviru ovog rada teorijski smo obradili sve ove navedene pojmove, uz navođenje nekih konkretnih primera i slučajeva korišćenja, pružajući uvid u to kakve sve mogćnosti pruža PostgreSQL sistem svojim korisnicima, kako bi obezbedili konzistentnost podataka.

# Literatura

1. The Art of PostgreSQL by Dimitri Fontaine - <https://dokumen.pub/the-art-of-postgresql-2nbsped.html>
2. Transakcije kod DBMS-a - <https://www.scaler.com/topics/dbms/transaction-in-dbms/#operations-in-transaction>
3. Stanja transakcija kod DBMS-a - <https://www.geeksforgeeks.org/transaction-states-in-dbms/>
4. PostgreSQl transakcije - <https://www.geeksforgeeks.org/postgresql-transactions/>
5. Postgre transactions and how to use them by Felix Schildorfer - <https://arctype.com/blog/posgres-transaction/>
6. <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-transaction/>
7. Understanding and using transactions in PostgreSQL -<https://www.prisma.io/dataguide/postgresql/inserting-and-modifying-data/using-transactions>
8. How to work with postgre transactions by Tushar Ajua -<https://www.enterprisedb.com/postgres-tutorials/how-work-postgresql-transactions>
9. Transaction isolation - <https://www.postgresql.org/docs/14/transaction-iso.html>
10. Transaction isolation levels with PostgreSQL - <https://mkdev.me/posts/transaction-isolation-levels-with-postgresql-as-an-example>
11. Transaction isolation in Postgres - [https://medium.com/@darora8/transaction-isolation-in-postgres ec4d34a65462](https://medium.com/@darora8/transaction-isolation-in-postgres%20ec4d34a65462)
12. Transactions and isolation level - <https://pgdash.io/blog/postgres-transactions.html>
13. Postgre example of serializable isolation level concurrency control <https://www.dbrnd.com/2019/07/postgresql-example-of-serializable-isolation-level-concurrency-control-of-transaction/>
14. PostgreSQL concurrency isolation and locking -[https://tapoueh.org/blog/2018/07/postgresql-concurrency-isolation-and- locking/](https://tapoueh.org/blog/2018/07/postgresql-concurrency-isolation-and-%20locking/)
15. PostgreSQL locking - <https://muatik.medium.com/postgresql-locking-b8657221825>
16. Explicit locking in Postgre-<https://www.postgresql.org/docs/current/explicit-locking.html>
17. Everything you need to know about PostgreSQL locks -<https://postgreshelp.com/postgresql-locks/>
18. <https://postgrespro.com/blog/pgsql/5968005>
19. Postgre Advisory locks - <https://www.kostolansky.sk/posts/postgresql-advisory-locks/>
20. Advisory locks with examples - <https://www.netguru.com/blog/advisory-locks>