Univerzitet u Nišu

Elektronski fakultet

Katedra za računarstvo

*Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje*

*u Microsoft SQL Server bazi podataka*

****

**Mentor:** **Student:**

Prof. dr Aleksandar Stanimirović Jelena Tošić, 1116

Niš, 2021. godine

**Sadržaj:**

[1. Uvod 1](#_Toc71978401)

[2. Transakcije i njihova obrada u MSSQL-u 2](#_Toc71978402)

[2.1. Osnovni pojmovi i definicje 2](#_Toc71978403)

[2.2. Fizički integritet podataka prilikom izvršenja transakcija u MSSQL-u 3](#_Toc71978404)

[2.3. Kontrola obrade transakcija 4](#_Toc71978405)

[2.3.1. Različiti modovi za pokretanje izvršenja transakcija 4](#_Toc71978406)

[2.3.2. Mehanizam pražnjenja log bafera i okončanje obrade transakcija 11](#_Toc71978407)

[3. Zaključavanje i verzionisanje u MSSQL-u 13](#_Toc71978408)

[3.1. Različite vrste zaključavanja 13](#_Toc71978409)

[3.1.1. Obzervacija zauzetih brava na konkretnom primeru 17](#_Toc71978410)

[3.2. Kompatibilnost brava 18](#_Toc71978411)

[3.3. Granularnost i hijerahija zaključavanja 19](#_Toc71978412)

[4. Izolacija u MSSQL-u 21](#_Toc71978413)

[4.1. *Read Committed* nivo izolacije 21](#_Toc71978414)

[4.2. *Read Uncommitted* nivo izolacije 23](#_Toc71978415)

[4.3. *Repeatable Read* nivo izolacije 24](#_Toc71978416)

[4.4. *Serializable* nivo izolacije 25](#_Toc71978417)

[4.5. *Snapshot* nivo izolacije 26](#_Toc71978418)

[4.5.1. *Read commited snapshot* nivo izolacije 27](#_Toc71978419)

[4.6. Zaključak o na temu različitih izolacionih nivoa 27](#_Toc71978420)

[5. Zaključak 28](#_Toc71978421)

[6. Literatura 29](#_Toc71978422)

# Uvod

Za svako obavljanje poslova u sklopu SQL baze podataka, transakcije koje se mogu sastojati od mnogih pojedinačnih operacija, pa čak i drugih transakcija, igraju ključnu ulogu. One su ključne za održavanje integriteta podataka, kako za više povezanih operacija, tako i kada više korisnika istovremeno ažurira bazu podataka. 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 imati aplikacije koje efikasno koriste transakcije. Iz ovog razloga, *SQL Server Database Engine* koristi različite mehanizme za osiguravanje fizičkog integriteta svake transakcije i pruža informacije o tome kako aplikacije mogu efikasno da kontrolišu transakcije. [3]

SQL transakcije u početku deluju magično: čini se da omogućavaju ogroman *Ctrl-Z,* odnosno poništavanje pristupa podacima. Ipak, problemi sa konkurentnim operacijama i dalje postoje kada se istovremeno odvija više upita, pri čemu se može koristiti opcija za postavljanje nivoa izolacije, kojom se definiše stepen u kojem jedna transakcija mora biti izolovana od modifikacija resursa ili podataka izvršenih drugim transakcijama. [2]

Sa druge strane, zaključavanje je neophodno za uspešnu obradu transakcija SQL Servera i osmišljeno je da omogući SQL Serveru nesmetan rad u višekorisničkom okruženju. Zaključavanje je način na koji SQL Server upravlja konkurentnim izvršavanjem transakcija. U osnovi, brave (engl *lock*) su strukture u memoriji koje imaju vlasnike, tipove i heš resursa koji bi trebalo da zaštite. Takođe, neophodno je naglasiti da je mehanizam zaključavanja dizajniran tako da obezbedi integritet podataka u bazi, tako što primorava svaku SQL Server transakciju da prođe *ACID* test. [1]

Ovaj rad će posebno govoriti o konceptima povezanim sa transakcijama – planovima izvršavanja i obradi transakcija, zatim izolaciji i zaključavanju, kao i o tome kako se transakcije mogu koristiti u kontekstu *Microsoft Sql Server*[[1]](#footnote-1)baze podataka, (*MSSQL* u nastavku teksta), pri čemu će rad biti propraćen primerima korišćenjem relativno jednostavne demo baze.

# Transakcije i njihova obrada u MSSQL-u

## 2.1. Osnovni pojmovi i definicje

Transakcija u MSSQL-u predstavlja niz, odnosno sekvencu operacija koje se izvršavaju od strane *SQL Server Engine*-a kao logička jedinica posla. Kako bi transakcija mogla da se opiše atributom *logička jedinica* ona mora da zadovoljava *ACID* svojstva, i na taj način bude kvalifikovana kao transakcija. *ACID* svojstva čine:

* Atomičnost (*Atomicity*) – Transakcija mora predstavljati atomičnu jedinicu posla. Ili će biti izvršene sve modifikacije u sklopu posla ili nijedna od njih.
* Konzistentnost (*Consistency*) - Po završetku, transakcija mora sve podatke ostaviti u doslednom, konzistentnom stanju. U relacionoj bazi podataka, sva pravila moraju biti primenjena na modifikacije koje vrši data transakcija, kako bi se održao integritet podataka. Sve interne strukture podataka, poput indeksa B-stabla ili dvostruko povezane liste, moraju biti tačne na kraju transakcije. Na ovaj način se osigurava da baza podataka pravilno menja stanja u kojima se nalazi, nakom uspešno *commit*-ovane transakcije.
* Izolacija (*Isolation*) - Izmene izvršene konkurentnim transakcijama moraju biti izolovane od modifikacija izvršenih bilo kojim drugim konkurentnim transakcijama. Transakcija ili prepoznaje podatke u stanju u kome su se nalazili pre nego što ih je druga istovremena transakcija modifikovala ili ih vidi onakve kakvi su nakon završetka druge transakcije, ali ne prepoznaje srednje stanje. Ovo se naziva serijalizacijom, jer rezultuje sposobnošću ponovnog učitavanja početnih podataka i reprodukcije niza transakcija tako da podaci završe u istom stanju u kakvom su bili nakon izvršenja prvobitnih transakcija. Na ovaj način se postiže nezavisnost i transparentnost transakcija u MSSQL-u. SQL Server poseduje različite nivoe izolacije, koji mogu uticati na različito ponašanje transakcija prilikom njihovog izvršavanja, pri čemu će više reči o njima biti u poglavlju 4.
* Trajnost (*Durability*) - Nakon završetka transakcije, njeni efekti trajno postoje u sistemu. Izmene se nastavljaju čak i u slučaju kvara sistema. SQL Server 2014 (12.x) i novije verzije omogućavaju odložene trajne transakcije. Odložene trajne transakcije se *commit*-uju pre nego što se logovi transakcija sačuvaju trajno na disku.

## 2.2. Fizički integritet podataka prilikom izvršenja transakcija u MSSQL-u

Kako bi fizički integritet podataka bio očuvan, odgovornost celokupnog sistema za upravljanje bazom podataka, kao i jedne instance *SQL Server Engine*-a je da pruži odgovarajuće mehanizme za očuvanje integriteta podataka prilikom izvršenja svake transakcije. Na raspolaganju su sledeći mehanizmi SQL Server-a, koji osiguravaju *ACID* svojstva transakcija:

* Zaključavanje - Omogućava čuvanje izolacije transakcija (detaljniji opis u nastavku teksta).
* Logovanje – Osigurava trajnost transakcija. U MSSQL-u postoje dve vrste transakcija koje se razlikuju u trajnosti, odnosno trenutku u kome se transakcija *commit*-uje uzevši u obzir trenutak kada je izvršena perzistencija log zapisa. Na ovaj način razlikujemo potpuno trajne transakcije (engl. *delayed durable transactions*) koje čuvaju log zapise na disku pre nego što se izvrši njihov *commit*, i zakasnele trajne transakcije (engl. *delayed durable transactions*), koje koriste asinhron način izvršenja, drugačije zvan lenji *commit*, i koje vraćaju poruku o uspešnom izvršenju transakcije pre nego da je log zapis trajno sačuvan na disku. Potpuno trajne transakcije je neophodno koristiti u sistemima koji ne mogu da tolerišu gubitak podataka i kada usko grlo u sistemu nije posledica latence prezistencije logova transakcija. Sa druge strane, zakasanele trajne transakcije smanjuju latencu I/O operacija koje su posledica generisanja logova, tako što ih zadržavaju u memoriji i prenose na disk u *batch*-evima. [4]
* Funkcije za upravljanje transakcijama - Nameću atomičnost i konzistentnost transakcija. Nakon što je transakcija započeta, ona mora biti usešno završena (*commit*-ovana) ili će u suprotnom *SQL Server Database Engine* poništiti sve izmene koje je izvršila transakcija, pri čemu se vraća stanje baze u ono koje je bilo pre pokretanja date transakcije i unošenja novih promena.

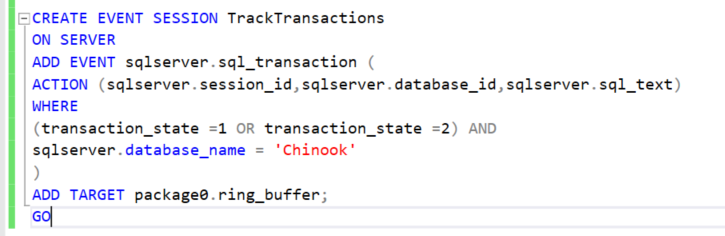
## 2.3. Kontrola obrade transakcija

Po *default*-u, upravljanje transakcijama se sprovodi u trenutku pokretanja konekcije, pri čemu se sve *Transact-SQL* naredbe izvršavaju u sklopu date konekcije sve dok se transakcija ne okonča. Aplikacije koje kontrolišu transakcije tako što najčešće samo navode kada transakcija počinje i kada se ona završava, pri čemu se u ovu svrhu mogu koristiti *Transact SQL* naredbe ili funkcije *database API*-ja.

### 2.3.1. Različiti modovi za pokretanje izvršenja transakcija

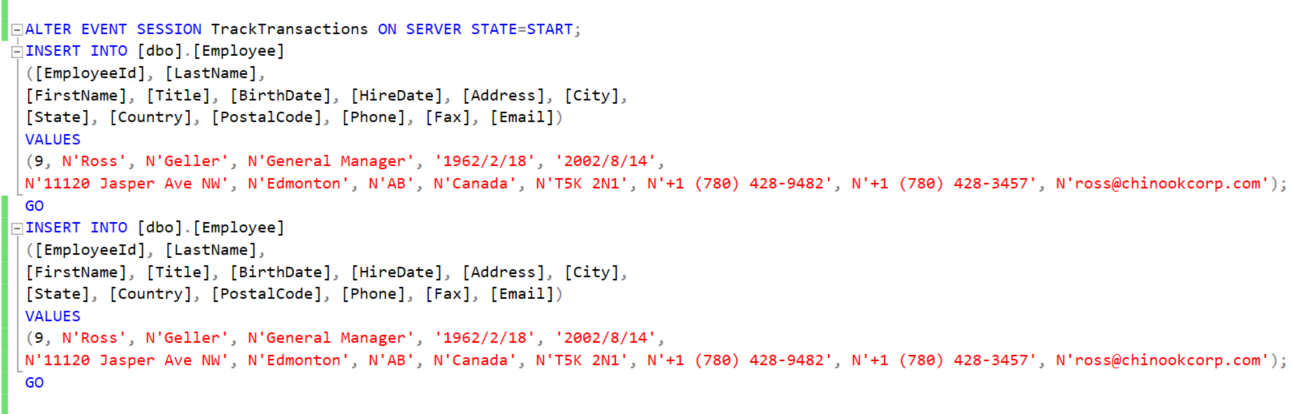
SQL Server može pokrenuti transakciju u jednom od tri različita moda:

1. *Autocommit* transakcije

*Autocommit* transakcije predstavljaju osnovni način obrade transakcija svaki put kada se pokrene konekcija ka *SQL Server Database Engine*-u (ukoliko naravno ovaj mod prethodno nije promenjen na neki od preostala dva).

Slika 1. Kreiranje extend event-a za praćenje commit-ovanih i rollback-ovanih transakcija u SQL Serveru

Svaka T-SQL nardeba je ili uspešno izvršena ili poništena, u zavisnosti od dobijenih rezultata, odnosno eventualnih grešaka. U ovom modu nije potrebno vršiti nikakvo upravljanje transakcijama s obzirom da će SQL Server to učiniti za nas. Za praćenje izvršenih i poništenih *default*-nih transakcija SQL Servera može se iskoristiti *extend event* prikazan na slici 1.

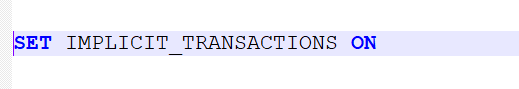
Zatim, u svrhe ilustracije rada *autocommit* transakcija može poslužiti sledeći primer koji unosi u bazu dva nova zapisa, pri čemu će prvi biti uspešno izvršen, dok će drugi rezultovati u grešci. Takođe, na samom početku upita, potrebno je pokrenuti prethodno kreirani *extend event.*

Slika 2. Ilustracija rada autocommit transakcija korišćenjem extend event-a

Slika 3. Rezultat izvršenja prethodnih transakcija

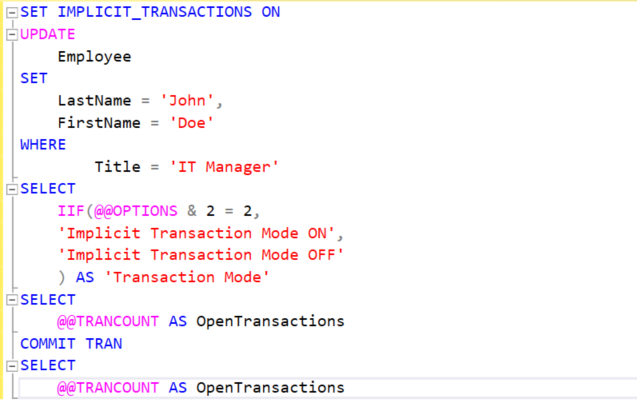
Slika 4. Prikaz status izvršenja autocommit transakcija

Kao što je prikazano na slici 4, vrednost u koloni *transaction\_type* je *System*, što označava da je *SQL Server Engine* izvršio date transakcije u *autocommit* modu. Takođe, primetno je da je druga transakcija bila poništena, sobzirom da je SQL Server poništio modifikacije koje su rezultovale greškom nakon unosa duplih zapisa sa istim primarnim ključem.

1. Implicitne transakcije

Slika 5. Postavljanje moda za izvršenje implicitnih transakcija

U implicitnom načinu obavljanja transakcija, *SQL Server Database Engine* generiše kontinualni lanac transakcija, tako što automatski pokreće novu transakciju nakon što je trenutna transakcija ili uspešno završena ili poništena. Mod za izvršenje implicitnih transakcija se može pokrenuti T-SQL naredbom sa slike 5.

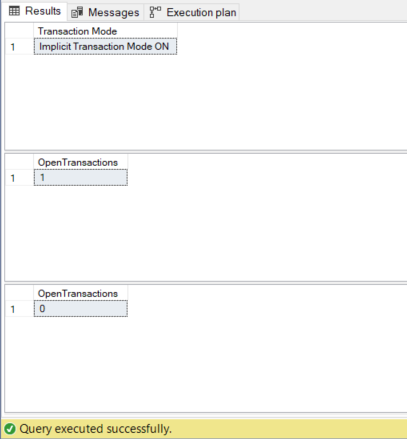
Sledeći primer ilustruje način za kreiranje implicitnih transakcija. Funkcija *@@TRANCOUNT* vraća broj *BEGIN TRANSACTION* naredbi u trenutnoj sesiji, pri čemu se ova funkcija može koristiti za brojanje otvorenih lokalnih transakcija. Ukoliko je transakcija poništena ovaj broj se postavlja na nulu. U suprotnom, svaka *BEGIN TRANSACTION* naredba će inkrementirati ovaj broj, dok će *COMMIT TRANSACTION* dekrementirati vrednost promenljive koja pamti broj otvorenih transakcija. [5]

Slika 6. Primer kreiranja implicitnih transakcija

SQL Nredba iz prethodnog primera:

*SELECT IIF(@@OPTIONS&2 = 0, 'Implicit Transaction Off', 'Implicit Transaction On') AS TranMode;* služi za utvrđivanje *default*-nog transakcionog moda.

Kod implicitnih transakcija, nova transakcija se startuje odmah nakon kompletiranja prethodne transakcije i zato nije potrebno navesti *BEGIN TRANSATION* naredbu, ali jeste potrebno eksplicitno navesti *COMMIT TRANSACTION* naredbu, koja će trajno primeniti promene nad podacima koje je modifikovala transakcija, odnosno *ROLLBACK* naredbu koja će poništiti dejstvo date transakcije.

Na sledećoj slici prikazan je rezultat nakon izvršenja prethodne implicitne transakcije i broj otovrenih transakcija u datim trenucima.

Slika 7. Rezultat izvršenja implicitne transakcije

Implicitne transakcije, nakon podešavanja implicitnog moda, će biti pokrenute ukoliko se pojavi neka od sledećih ključnih reči:

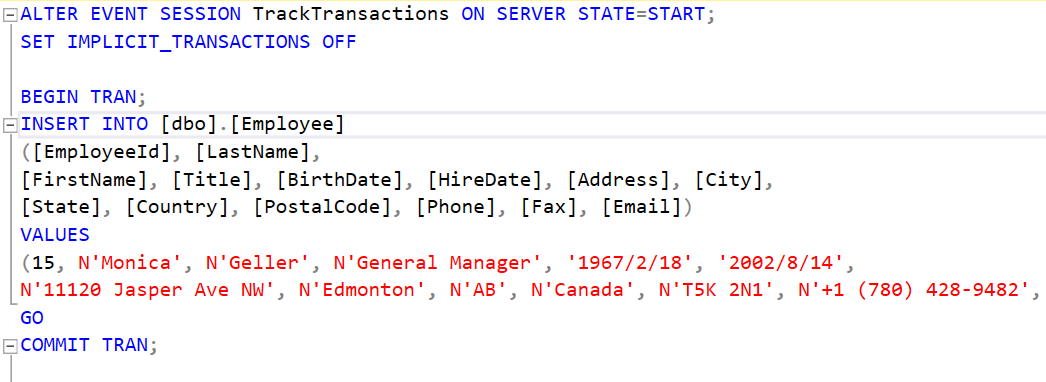
ALTER TABLE, BEGIN TRANSACTION, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE, UPDATE

1. Eksplicitne transakcije

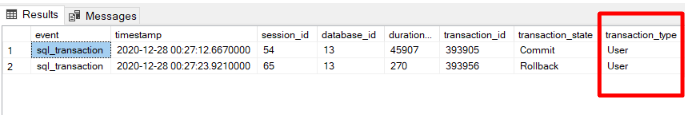
Eksplicitne transakcije su transakcije kod kojih se eksplicitno navodi početak i kraj date transakcije koristeći ili API funkcije ili T-SQL naredbe:

BEGIN TRANSACTION, COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION, ROLLBACK WORK

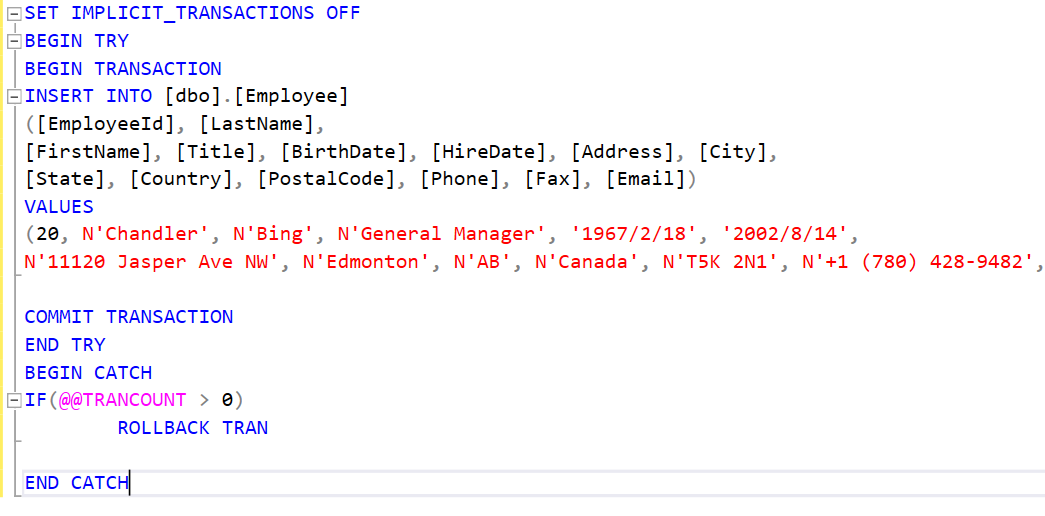
Kada se transakcija okonča, transakcioni mod se vraća u prethodno stanje koje je ili *autocommit* ili *implicit*.

Primer izvršenja jednostavne eksplicitne transakcije je dat na sledećoj slici, pri čemu je primetno da je eksplicitno navedena naredba za započinjanje i okončanje transakcije:

Slika 8. Primer izvršenja eksplicitnih transakcija

Nakon preuzimanja podataka iz *extend event*-a, možemo ponovo videti detalje izvršenih eksplicitnih transakcija. Za ove *insert* naredbe će sada kao tip transakcije stajati ‘*User’* umesto *’System’*. To znači da ovim transakcijama upravlja korisnik sesije umesto *SQL Servera*, što je bio slučaj sa *autocommit* transakcijama.

Slika 9. Podaci iz extend event-a nakon izvršenja eksplicitnih transakcija

I na kraju, eksplicitne transakcije je preporučljivo koristiti u sklopu *TRY-CATCH* blokova kao u primeru sa sledeće slike:

Slika 10. Primer izvršenja eksplicitne transakcije u okviru TRY CATCH bloka

Važno je pomenuti i sledeće podvrste transakcija:

* *Batch-scoped* transakcije – Primenljive su samo za fičer *SQL Servera* pod immenom MARS (engl. *Multiple Active Result Sets*), koji je dizajniran tako da dozvoli više *batch scoped* transakcija da budu izvršene u okviru jedne SQL konekcije. Transakcija sa *batch* opsegom koja nije *commit*-ovana ili poništena kada se *batch* kompletira automatski se poništava od strane *SQL Server-*a.
* Distribuiranetransakcije - Distribuirane transakcije obuhvataju dva ili više servera poznatih kao menadžeri resursa. Upravljanje transakcijom mora biti koordinisano između menadžera resursa pomoću serverske komponente koja se naziva menadžer transakcija. Svaka instanca *SQL Server Database Engine*-a može da funkcioniše kao menadžer resursa u distribuiranim transakcijama koje koordinišu menadžeri transakcija, kao što je *Microsoft Distributed Transaction Coordinator* (MS DTC). Sada, transakcija unutar jedne instance *SQL Server Database Engine*-a koja obuhvata dve ili više baza podataka zapravo je distribuirana transakcija. Korisniku ovakvo izvršenje distribuirane transakcije deluje kao da se izvršava lokalno. Na nivou aplikacije se distribuiranom transakcijom upravlja na sličan način kao i lokalnom transakcijom. Na kraju transakcije, aplikacija zahteva da se transakcija ili *commit*-uje ili poništi. Menadžer transakcija mora različito upravljati distribuiranim *commit*-om kako bi se smanjio rizik da neki kvar u mreži može uzrokovati uspešnim *commit*-om kod nekih od menadžera resursa, dok bi kod drugih došlo do poništavanja transakcije. To se postiže upravljanjem procesom *commit*-ovanja u dve faze (faza pripreme i faza *commit*-a), koja je poznata kao dvofazna predaja (2PC). [6] [3]
  + Faza pripreme – Kada menadžer transakcije primi zahtev za *commit*-om, on šalje pripremnu komandu svim ostalim menadžerima resursa koji su uključeni u obradu date transakcije. Zatim, svaki od menadžera resursa radi sve što je neophodno kako bi transakciju učinio trajnom, pri čemu svi baferi koji sadrže slike logova drugih transakcija bivaju preneti na disk. Nakon što svaki menadžer resursa završi fazu pripreme, potrebno je da pošalje nazad odgovor menadžeru transakcije kako bi ga obavestio da li je proces bio uspešan ili ne.
  + Faza *commit*-a – Ukoliko menadžer transakcije primi potvrdu od svih menadžera resursa da je faza pripreme uspešno okončana, on onda šalje COMMIT naredbu svim menadžerima resursa. Ukoliko svi menadžeri vrate poruku o uspešnom *commit*-u, tek tada će transakcioni menadžer javiti korisniku, odnosno aplikaciji da je transakcija uspešno okončana. U suprotnom, ukoliko bilo koji od menadžera resursa pošalje neuspelu poruku u fazi pripreme, transakcioni menadžer će poništiti transakciju, odnosno poslati komandu ROLLBACK koju treba da izvrše svi menadžeri resursa.

Slika 11. Izdavanje komande za pokretanje distribuirane transakcije

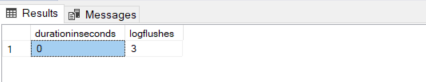
Treba napomenuti da će ona instanca *SQL Server Engine*-a koja izvrši naredbu sa prethodne slike biti zadužena za upravljanje distribuiranom transakcijom. [7]

### 2.3.2. Mehanizam pražnjenja log bafera i okončanje obrade transakcija

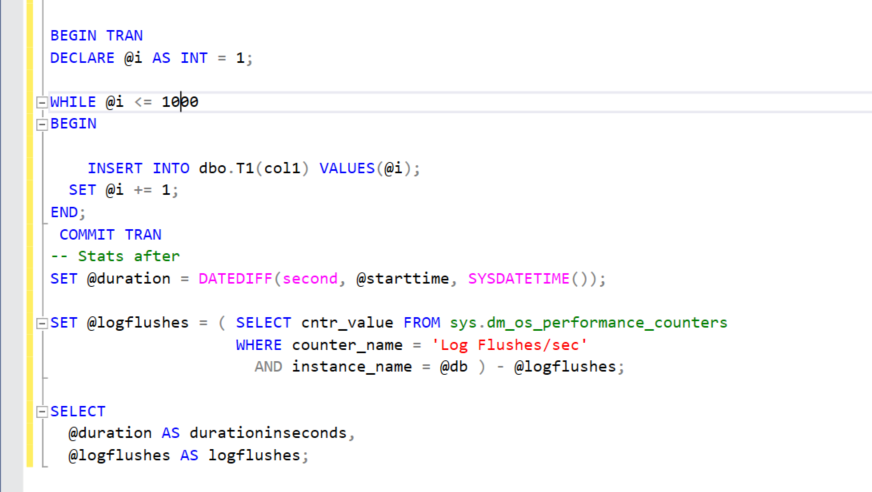
SQL Server, prilikom obrade transakcija koristi mehanizam ispiranja log bafera. Sve modifikacije transakcije bivaju upisane u ovaj bafer, pri čemu se zatim baferovani podaci šalju u log fajl u nekim od sledećih situacija:

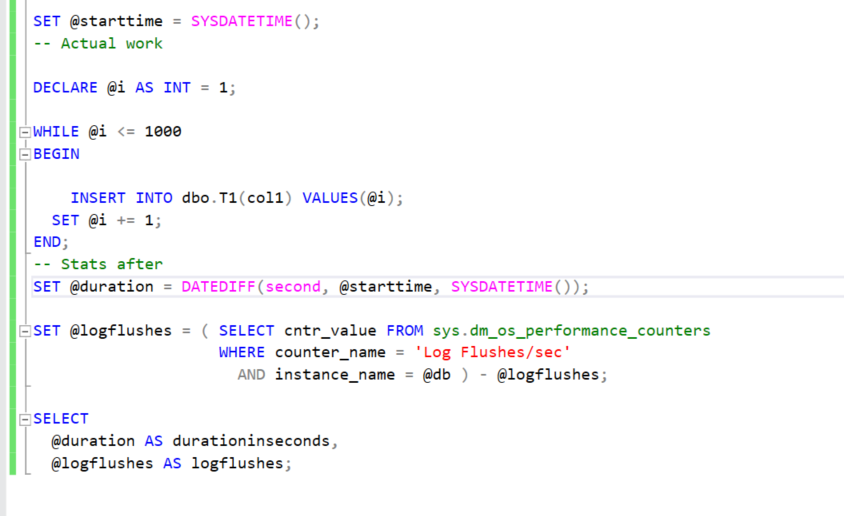
* U trenutku kada je transakcija uspešno završena
* Kada ovaj bafer dostigne veličinu od 60KB
* Nakon što je sys.sp\_flush\_log procedura izvršena
* Kada je CHECKPOINT proces kompletiran

Takođe treba imati na umu da će prilikom izvršenja transakcija u eksplicitnom modu prvo biti popunjena maksimalna veličina log bafera, pri čemu će tek onda biti izvršeno njegovo pražnjenje (čuvanje na disku), dok kod *autocommit* transakcije ovo neće biti slučaj (za slučaj unošenja 1000 novih zapisa, pražnjenje log bafera će se desiti upravo toliko puta).

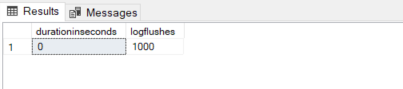
Primeri koji ilustruju prethodno objašnjeni način obrade transakcija u prethodno navedenim modovima, prikazani su na slikama u nastavku. Slika 13 prikazuje upit koji oslikava izvršenje eksplicitnih transakcija prilikom unosa 1000 novih zapisa u tabelu *Person,* a slika 12. prikazuje broj pražnjenja log bafera za dati broj eksplicitnih transakcija:

Slika 12. Broj pražnjenja log bafera prilikom korišćenja eksplicitnih transakcija

 Slika 13. Insertovanje korišćenjem eksplicitnih transakcija



Slika 14. Izvršenje istih insert naredbi u okviru autocommit moda

Slika 15. Broj pražnjenja log bafera u okviru autocommit moda

Na isti način prikazano je i izvršenje u *autocommit* modu pri čemu je evidentno da se za isti broj naredbi log bafer prazni čak 1000 puta, za razliku od 3, koliko je bio slučaj sa eksplicitnim transakcijama. [8]

I na kraju, potrebno je spomenuti i načine za okončavanje. Transakcija može biti okončana ili korišćenjem COMMIT ili ROLLBACK naredbe SQL-a. Ukoliko je u toku izvršenja generisana greška, ili ukoliko korisnik prekine izvršenje date transakcije, ona će biti poništena. Na ovaj način se poništavaju sve promene koje je unela data transakcija. Takođe bitna je i činjenica da će u ovom trenutnku i svi resursi koje je data transakcija držala u toku svog izvršenja, biti oslobođeni. U suprotnom, ukoliko je transakcija uspešno *commit*-ovana sve promene će biti trajno primenjene nad podacima u bazi. Takođe i u ovom slučaju svi resursi (poput brava) će biti oslobođeni.

# Zaključavanje u MSSQL-u

*SQL Server Database Engine* koristi dva mehanizma koja omogućavaju integritet transakcija i očuvanje konzistencije kada dva ili više korisnika pristupaju podacima u istom trenutku. Ta dva mehanizma su zaključavanje i verzionisanje.

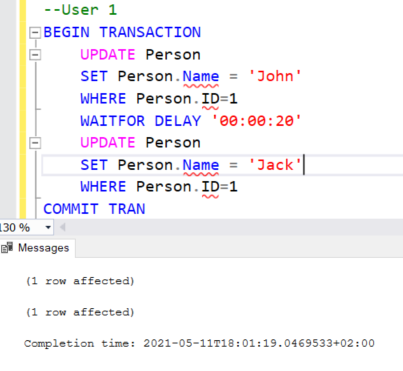
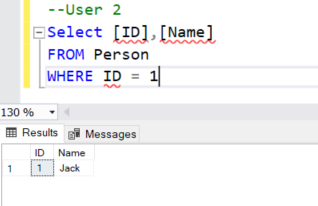
## 3.1. Različite vrste zaključavanja

Mehanizam zaključavanja (engl. *locking*) je esencijalni mehanizam koji omogućava SQL Serveru da na odgovarajući način upravlja transakcijama. Brave su memorijske strukture koje poseduju vlasnike, tipove i heševe resursa koji ih čuvaju. One su dizajnirane tako da osiguraju integritet podataka u bazi pri čemu primoravaju svaku SQL Server transakciju da prođe *ACID* test. Zaključavanje čini esencijalni deo zahteva za izolacijom i služi da zaključa objekte koji su pod uticajem date transakcije. Na ovaj način SQL Server može sprečiti druge transakcije da promene objekte koji su trenutno pod ključem. To znači da će ostale transakcije koje zahtevaju date objekte morati da čekaju sve dok su ti objekti pod ključem druge transakcije, pri čemu će one biti registrovane od strane SQL Servera koji će im dodeliti odgovarajući status. U trenutku kada je transakcija okončana, druge transakcije mogu dobiti mogućnost da menjaju prethodno zaključane objekte.

Svaka transakcija može da zahteva neku od sledećih vrsta brava nad resursima:

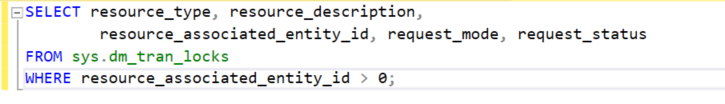
* Ekskluzivne brave (engl *exclusive lock*) (X) – Ovaj vid zaključavanja osigurava da će stranica ili zapis u bazi biti rezervisani isključivo za datu transakciju koja je nametnula ekskluzivno zaključavanje, sve do trenutka oslobađanja. Ekskluzivno zaključavanje se koristi kod transakcija koje menjaju podatke u bazi, odnosno u slučaju DML naredbi poput UPDATE, INSERT ili DELETE. Eksluzivno zaključavanje nad datim podacima moguće je uspostaviti samo u situaciji kada ne postoji neko drugo ekskluzivno ili deljeno zaključavanje koje je prethodno nametnuto od strane neke druge transakcije nad istim podacima. To upravo znači da samo jedna ekskluzivna brava može postojati nad stranicom ondnosno zapisom u datom trenutku.
* Deljene brave (engl. *shared locks*) (S) - Ovaj tip zaključavanja stranicu ili red da budu dostupni samo za čitanje, što znači da će bilo koja druga transakcija biti sprečena da modifikuje zaključani zapis sve dok je ovo zaključavanje aktivno. Ovo vodi ka tome da deljeno zaključavanje mogu nametnuti nekoliko transakcija istovremeno nad istom stranicom ili redom i na taj način nekoliko transakcija može deliti mogućnost čitanja podataka, jer sam proces čitanja neće uticati na stvarne podatke.
* Brave ažuriranja (engl. *update lock*) (U) - ova brava je slična ekskluzivnoj bravi, ali je dizajnirana da bude na neki način fleksibilnija. Zaključavanje ažuriranja može se nametnuti zapisu koji već ima deljenu bravu. U tom slučaju, zaključavanje ažuriranja će nametnuti još jedno deljeno zaključavanje na ciljnom redu. Ali, jednom kada je transakcija koja sadrži zaključavanje ažuriranja spremna za promenu podataka, zaključavanje ažuriranja će se transformisati u ekskluzivno zaključavanje. Važno je razumeti da je zaključavanje ažuriranja asimetrično u odnosu na deljeno zaključavanje. Iako se zaključavanje ažuriranja može nametnuti zapisu koji ima deljenu bravu, deljena brava ne može se nametnuti zapisu nad kojim je već uspostavljeno zaključavanje ažuriranja.
* Brave namere (engl. *intent locks*) (I) – Služe kao sredstvo koje neka transakcija koristi za svrhe obaveštavanja drugih transakcija o svojoj nameri da nabavi bravu. Svrha ovakvog zaključavanja je da osigura da će se modifikacija podataka izvršiti pravilno, sprečavanjem druge transakcije da preuzme zaključavanje nad sledećim objektom u hijerarhiji. U praksi, kada transakcija želi da dobije bravu nad zapisom, zahtevaće bravu namere nad tabelom, koja je viši hijerarhijski objekat. Uzimanjem ovakve brave, transakcija neće dozvoliti drugim transakcijama da steknu ekskluzivnu bravu nad tom tabelom (u suprotnom, ekskluzivna brava nametnuta od strane neke druge transakcije poništila bi zaključavanje nad zapisum u tabeli). Sa druge strane, brave namere mogu poboljšati performanse s obzirom da će *SQL Server Engine* proveriti postojanje namernih brava samo na nivou tabele, kako bi utvrdio da li neka transakcija može da zahteva bravu nad tom tabelom. Na taj način je eliminasana potreba da se proverava postojanje brave nad svakim zapisom u tabeli. Dalje, brave namere moguće je podeliti u sledeće dve kategorije, koje kombinuju različite aspekte prethodno navedenih brava i omogućavaju različite načine zaključavanja u SQL Serveru:
  + Regularne brave namere:
    - Ekskluzivna brava namere
    - Deljena brava namere
    - Namerna brava ažuriranja
  + Konverzione brave namere:
    - Deljene namerne ekskluzivne brave
    - Deljene namerne brave ažuriranja
    - Namerne ekskluzivne brave ažuriranja
* Šema brave (engl. *shema lock*) (Sch)- Koriste se kada se izvršava operacija koja zavisi od šeme tabele. Postoje dve vrste šema brava:
  + *Schema modification lock* – Zahtevaju se kada se izvršava neka DDL naredbi. One će sprečiti pristup tako zaključanom objektu, s obzirom da se struktura objekta menja. Ukoliko transakcija želi da menja tabelu, onda mora da dobije ovakvu bravu nad njom. *Index rebuild* proces je tipična operacija koja zahteva ovakvu vrstu brave nad tabelom u svrhu svog izvršenja.
  + *Schema stability lock* – Zahtevaju se prilikom kompajliranja i izvršenja upita koji zavise od šema, odnosno prilikom svake DML ili *select* naredbe kako bi osigurale integritet strukture tabele. Ovakve brave neće blokirati ostale transakcije ukoliko žele da pristupe podacima i kompatibilne su sa svim ostalim vrstama brava sem sa *Schema modification lock-*ovima.
* Brave za masovno ažuriranje (engl. *Bulk update locks*) (BU) – Ovakve brave se koriste u kombinaciji sa masovnim *insert* operacijama i to prilikom korišćenja TABLOCK argumenta. Drugi procesi ne mogu pristupiti datoj tabeli nad kojom je uspostavljena ovakva brava.

### 3.1.1. Obzervacija zauzetih brava na konkretnom primeru

Sledeći primer ima za cilj da ilustruje jednu od mogućih tehnika za posmatranje zauzetih odnosno zatraženih brava nad podacima kojima pristupaju dva korisnika u gotovo sličnim vremenskim trenucima.

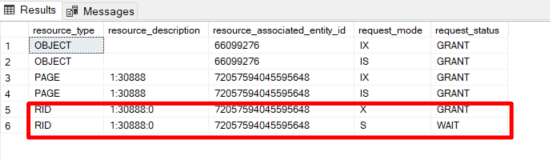
Slika 16. Naredbe koje izdaju korisnici User 1 i User 2 - ilustracija rada eksplicitnih brava

Uzevši u obzir da se prvo pokreće naredba koju je zadao korisnik 1, a ubrzo potom naredba koju izdaje korisnik 2, može se videti sledeći rezultat koji prikazuje da je prvi korisnik prvo ažurirao tabelu *Person* koristeći pauzu od 20 sekundi u okviru koje je naredba korisnika 2 bila prinuđena da sačeka, a zatim je prvi korisnik u okviru iste transakcije ponovo ažurirao ime osobe sa id-jem 1, pri čemu je nakon toga korisnik 2 dobio parvo pristupa i pročitao validno ime.

Kako bismo potvrdili pretpostavku o korišćenim eksplicitnim bravama (na šta je prethodno ponašanje upita ukazalo), možemo izvršiti sledeću naredbu koja pruža uvid u dešavanja u toku izvršenja prethodno izdatih naredbi datih korisnika.

Slika 17. Naredba koja koristi sys.dm\_tran\_locks za pregled trenutno zauzetih ključeva u toku izvršenja naredbi korisnika 1 i 2

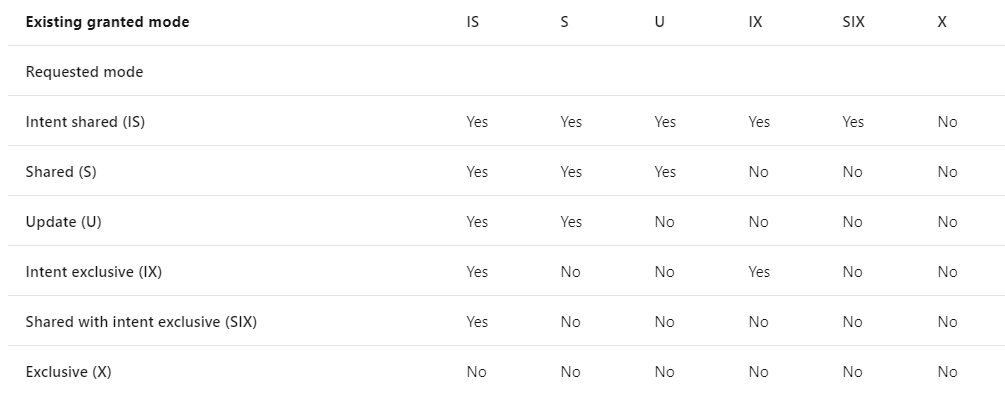
SQL Server pruža *Dynamics Management View* *sys.dm\_tran\_locks* za preuzimanje informacija o trenutnim aktivnim *lock*-ovima. U ovoj tabeli svaki red sadrži informacije o trenutnim aktivnim zahtevima upućenim *lock* menadžeru, a koji mogu biti ili GRANT (odobreni) ili WAIT (čekaju da budu odobreni). Tabela sadrži kolone koje opisuju ili resurs nad kojim je brava zahtevana ili kolone koje opisuju sam zahtev. *Resorce type* kolona se odnosi na tip resursa nad kojim SQL Server može uzeti bravu, pri čemu će više o ovome biti navedeno u narednom poglavlju. *Request mode* kolona prikazuje informacije o modu zaključavanja primenjenim nad datim resursom, a resource description pruža kratak opis resursa, pri čemu ova kolona nije uvek popunjena (u najčešćem broju slučajeva vrednost ove kolone upućuje na id vrste, stranice ili nekog drugog objekta) .[11]

Slika 18. Rezultat koji prikazuje naredba sa slike 17.

Kao što se može videti na slici 18. pribavljena je jedna ekskluzivna brava nad odgovarajućim resursom sa prikazanim brojem (RID je skraćenica za *row id*, pri čemu se ovo odnosi na hijerarhiju brava koje zaključavaju resurse na nivou jednog reda u tabeli) pri čemu je jedan zahtev označen sa X a drugi sa S, gde je X oznaka koju SQL Server koristi za obeležavanje ekskluzivnih brava, a S oznaka deljenih brava. Takođe, sa slike je jasno da je samo ekskluzivna brava u stanju GRANT dok je zahtev za istim resursom od strane korisnika 2 u stanju WAIT, sve dok se dati resurs ne bude oslobodio.

## Kompatibilnost brava

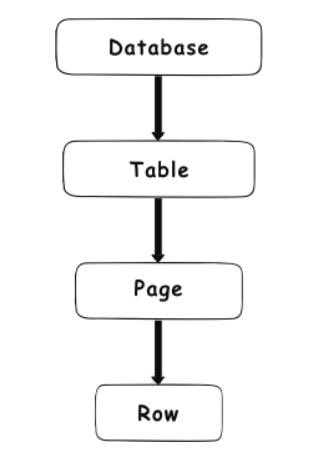
Kompatibilnost zaključavanja kontroliše da li više transakcija može istovremeno steći brave nad istim resursom. Ako je resurs već zaključan drugom transakcijom, novi zahtev za zaključavanje može se odobriti samo ako je način zahtevanog zaključavanja kompatibilan sa režimom postojećeg zaključavanja. U suprotnom transakcija koja zahteva novo zaključavanje će biti primorana da sačeka da se postojeća brava otpusti ili da istekne interval dozvoljenog vremena zaključavanja. Na primer, nijedan režim zaključavanja nije kompatibilan sa ekskluzivnim bravama. Dok se drži ekskluzivna (X) brava, nijedna druga transakcija ne može dobiti pravo na zaključavanje bilo koje vrste (deljena, ažurirana ili ekskluzivna) nad tim resursom sve dok se ne oslobodi ekskluzivna brava. Alternativno, ako je deljena (S) brava uzeta nad resursom, druge transakcije takođe mogu steći deljenu bravu ili bravu za ažuriranje (U) nad istim objektom, čak i ako prva transakcija nije završena. Međutim, druge transakcije ne mogu dobiti ekskluzivnu bravu sve dok se deljena brava ne oslobodi. [3]

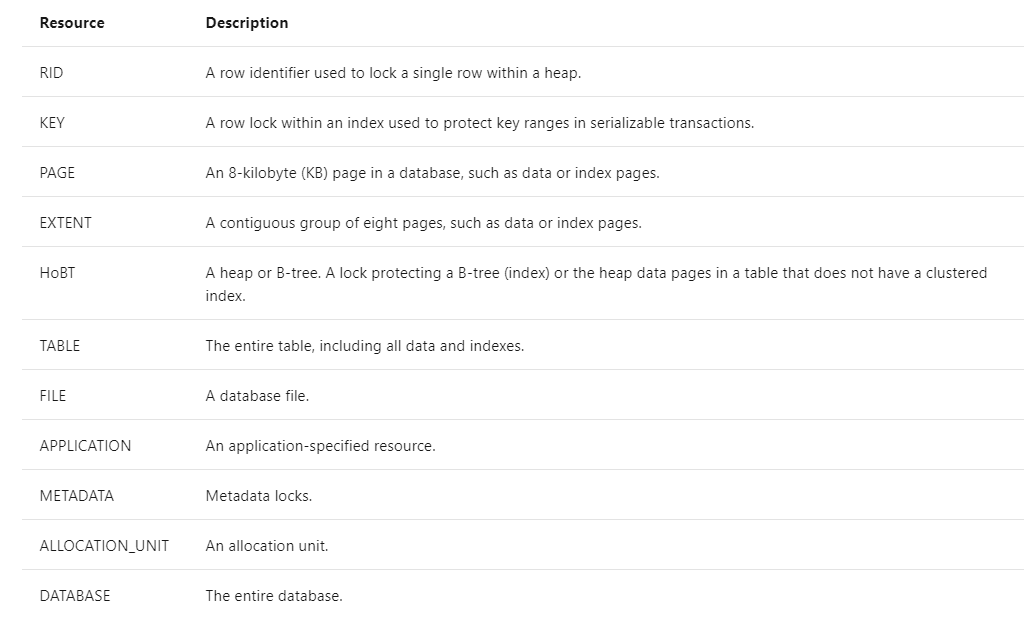
Slika 19. Kompatibilnost lock-ova u MSSQLu

## Granularnost i hijerahija zaključavanja

Multigranularnost predstavlja mehanizam zaključavanja koji koristi *SQL Server Database Engine* kako bi omogućio datoj transakciji da zaključa različite tipove resursa. Da bi smanjio troškove zaključavanja, SQL Server automatski zaključava resurse na nivou koji odgovara datom zadatku koji se trenutno izvršava. Zaključavanje na nivou manje granularnosti, kao što su redovi, povećava konkurentnost ali ima za posledicu veće troškove, jer se, shodno, mora držati više brava ako je zaključano više redova. Zaključavanje veće granularnosti, kao što su tabele, skupo je u smislu konkurentnosti, jer na primeru ekskluzivnog zaključavanja cele tabele SQL Server brani drugim transakcijama pristup bilo kom delu te tabele. Međutim, ovakvo zaključavanje vuče za sobom niže troškove, jer se smanjuje broj mogućih brava koje SQL Server treba istovremeno da održava.

*SQL Server Database Engine* često mora da stekne brave na više nivoa granularnosti da bi u potpunosti zaštitio dati resurs. Ova grupa brava koje se protežu na više nivoa granularnosti se naziva hijerarhija brava. Hijerarhija zaključavanja započinje na nivou baze podataka, koja je na najvišem nivou hijerarhije, i kreće se dalje preko nivoa tabele do nivoa stranice i jednog reda u tabeli. U osnovi, na nivou baze podataka uvek postoji deljena brava koja se nameće svaki put kada je transakcija povezana sa bazom podataka. Deljeno zaključavanje na nivou baze podataka nameće se da bi se sprečilo eventualno uništenje (DROP) baze podataka ili obnavljanje rezervne kopije baze podataka nad datom bazom koja se koristi od strane transakcije.

 Slika 20. Nivoi hijerarhije zaključavanja u Sql Server-u

Slika 20 ilustruje nivoe hijerarhije zaključavanja koje poseduje SQL Server, dok slika 21 prikazuje tabelu koja sadrži sve vrste resursa koje *SQL Server Engine* može zaključati.

Slika 21. Tabela resursa koje SQL Server Database Engine može zaključati

Konkretno, kada se izda naredba SELECT za čitanje nekih podataka, deljena brava (S) biće nametnuta na nivou baze podataka, deljena brava namere (IS) biće uzeta na nivou tabele i na nivou stranice, a deljena brava (S) na samom redu.

U slučaju DML naredbi, poput INSERT, DELETE, UPDATE, situacija je malo drugačija. Na nivou baze podataka biće zauzeta deljena brava (S), na nivou tabele i nivou stranice ekskluzivna brava namere (IXS) ili namerna brava ažuriranja (IU) i na nivou reda ekskluzivna ili brava ažuriranja (X ili U).

Na kraju, bitno je uočiti da će SQL Serverzauzimati brave sa vrha ka dnu, kako bi sprečio trku nad podacima (engl. *race condition*).

# Izolacija u MSSQL-u

Nivoi izolacije SQL Servera koriste se za određivanje stepena u kojem jedna transakcija mora biti izolovana od modifikacija resursa ili podataka izvršenih od strane drugih konkurentnih transakcija. SQL Server podržava nekoliko različitih nivoa izolacije:

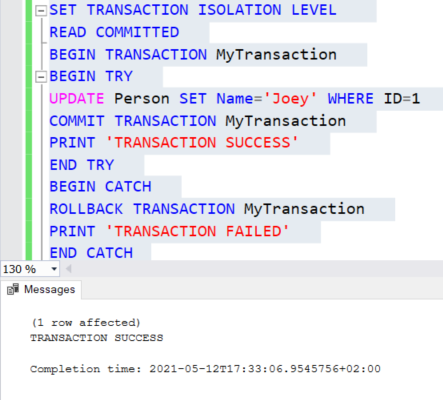
1. Read Commited
2. Read Uncommited
3. Repeatable Read
4. Serializable
5. SnapShot

Izmena nivoa izolacije transakcije se može izvršiti sledećom SQL naredbom:

Slika 22. Postavljanje nivoa izlacije

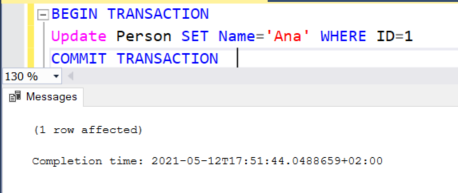
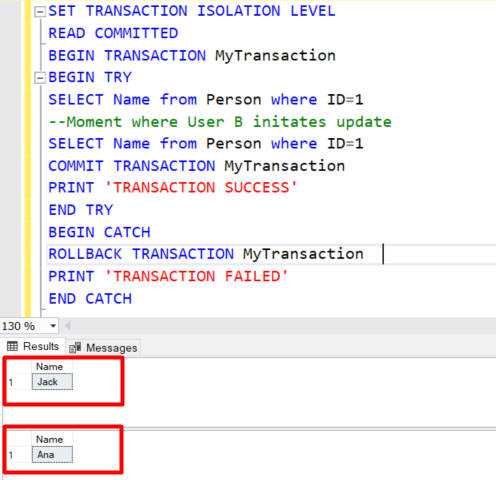
## 4.1. *Read Committed* nivo izolacije

Ovaj nivo izolacije predstavlja podrazumevani nivo izolacije transakcija u SQL Serveru, pri čemu on služi da štiti transakcije od takozvanih prljavih čitanja (engl. *dirty reads*). Kada je ovaj nivo izolacije na snazi, transakcija ne može čitati podatke koje druga, trenutna transakcija menja. Ovakav način zaključavanja podrazumeva da su podaci zaključani deljenom bravom, koja se drugačije naziva *read* *lock,* s obzirom da ona štiti podatke da ne budu modifikovani od strane nekog korisnika u trenutku kada drugi korisnik čita te podatke. [12][13][14]

Ukoliko imamo situaciju da korisnik A želi da pročita neki podatak koji korisnik B upravo menja, korisnik A se primorava da sačeka na okončanje transakcije drugog korisnika, pri čemu će korisnik A na kraju pročitati validne podatke. Primer izvršenja *read committed* transakcije prikazan je na slici u nastavku:

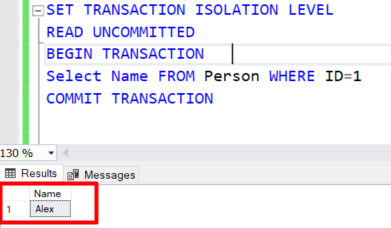
Slika 23. Read commited transakcije

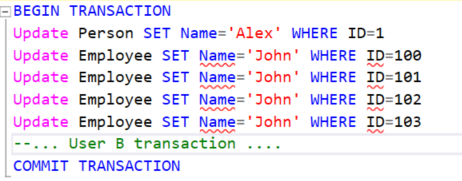
Potencijalni problemi koji mogu nastati u *read committed* izolacionom modu su fantomska čitanja i nekonzistentna analiza (na primer ukoliko se od korisnika A zahteva da čeka na čitanje ali ne i na modifikaciju ili unos podataka). Takođe treba voditi računa i o situacijama kada podaci mogu biti promenjeni od strane druge transakcije između izvršenja dve individualne naredbe u toku trenutne transakcije jer ovo može rezultovati u fantomskim podacima. Primer ovakve situacije prikazan je na slici u nastavku:



Slika 24. Ilustracija problema kod read commited izolacija transakcija

## 4.2. *Read Uncommitted* nivo izolacije

Kada je *read uncommitted* nivo izolacije postavljen, transakcija može čitati *uncommit-*ovane podatke, pri čemu će ovo često rezultovati u *dirty read* problemu. Uzmimo u obzir sledeću situaciju: korisnik A pokušava da pročita podatke iz reda koji upravo ažurira korisnik B, pri čemu je inicijalno stanje u tabeli *Person* za zapis sa ID-jem 1: *Ana.* Korisnik A u ovom slučaju čita još uvek ne sačuvane izmene (zapis sa ID-jem 1: *Alex*) koje unosi korisnik B u sklopu svoje transakcije.



Slika 25. Read uncommited nivo izolacije

*Read uncommitted* nivo transakcije se smatra najmanje restriktivnim nivoom transakcije koji postoji u SQL Serveru. Ono što je specifično u ovom nivou izolacije jeste činjenica da ovakve transakcije neće zahtevati uspostavljanje deljivih brava nad svojim resursima, s obzirom na činjenicu da su čitanja dozvoljena iako neka transakcija menja podatke koji trenutno bivaju čitani. Takođe ove transakcije neće biti blokirane ni od strane ekskluzivnih brava koje bi ih sprečile da čitaju podatke. Ova opcija *read uncommited* ima isti efekat kao da je postavljeno NOLOCK ograničenje nad svim tabelama u svim SELECT naredbama u okviru transakcije. [12][13][14]

## 4.3. *Repeatable Read* nivo izolacije

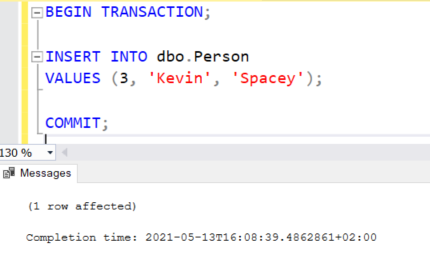
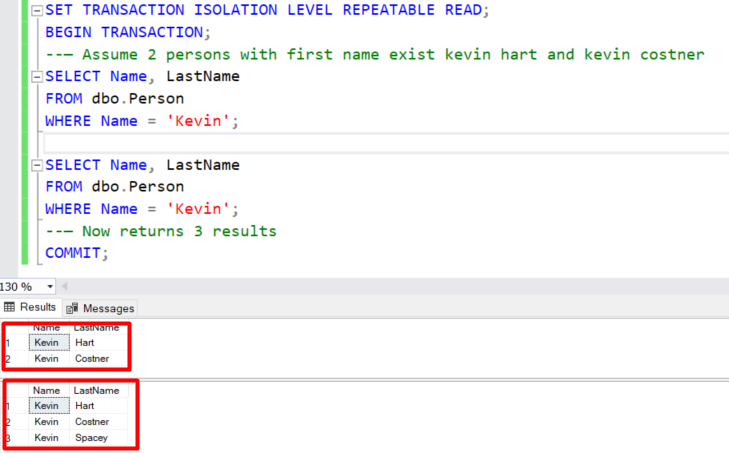
Ovaj nivo izolacije specificira da naredbe ne mogu čitati podatke koji su modifikovani, ali još uvek nisu *commit*-ovani od strane drugih transakcija i da nijedna druga transakcija ne može izmeniti podatke koje čita trenutna transakcija, sve dok se trenutna transakcija ne završi.

Deljene brave postavljaju se na sve podatke koje čita svaka naredba u trenutnoj transakciji i drže se sve dok se ta transakcija ne završi. Ovo sprečava druge transakcije da modifikuju bilo koje redove koje čita trenutna transakcija.

Prethodno navedeno znači da će *repeatable read* nivo izolacije raditi na sličan način kao i *read commited*, osnovni nivo izolacije, sem što će u slučaju kada korisnik A izvršava transakciju, korisnik B biti primoran da sačeka na njeno okončanje i u slučaju naredbe ažuriranja i u slučaju naredbe čitanja koju obavlja korisnik A u sklopu svoje transakcije.

Sa druge strane, ovakav način izolacije ne sprečava baš sve anomalije koje mogu nastati, s obzirom na činjenicu da su samo redovi koji se upravo čitaju zaključani. To znači da ostale transakcije mogu unositi nove redove (INSERT) koji se podudaraju sa uslovima pretraživanja naredba izdatih od strane trenutne transakcije, pri čemu može nastati anomalija. Ako trenutna transakcija zatim pokuša ponovo da izvrši datu naredbu, ona će preuzeti nove redove, što rezultuje u pojavi zvanoj u fantomsko čitanje. Jedan primer ovakvih anomalija prikazan je na slici 26, pri čemu će korisnik 1 pročitati 1 red više u odnosu na prvu SELECT naredbu, nakon što u međuvremenu korisnik 2 unese još jedan novi red u tabelu *Person.*

Takođe, budući da se deljene brave zadržavaju sve do kraja transakcije, umesto da se oslobađaju na kraju svake naredbe, konkurentnost je smanjena u odnosu na podrazumevani nivo izolacije READ COMMITTED. [12][13][14]

Slika 26. Anomalije koje mogu nastati u repeatable read izolacionom modu

## 4.4. *Serializable* nivo izolacije

*Serializable* nivo izolacije je u stanju da reši prethodne probleme *repeatable read* nivoa izolacije. Šta više, *serializable* nivo izolacije nameće sledeća ograničenja transakcijama:

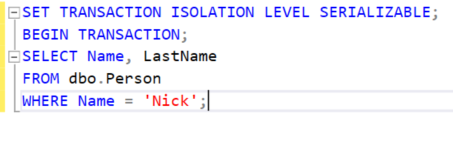
* Naredbe ne mogu čitati podatke koje trenutno ažurira neka druga transakcija (i još uvek nije izvršila *commit*)
* Nijedna druga transakcija ne može da menja podatke koje trenutno čita neka transakcija, sve dok se ona ne kompletira.
* Druge transakcije ne mogu da vrše unos (INSERT) novih redova u okviru određenog opsega ključeva kojima pristupa bilo koja naredba izdata od strane tekuće transakcije, sve dok se ona ne završi

Poslednja stavka jeste upravo ta koja omogućava ovom nivou izolacije da reši probleme vezane za fantomska čitanja *repeatable read*-a. Naime SQL Server koristi posebnu vrstu zaključavanja pod nazivom *range lock* koji sprečava umetanje novih redova u određenom opsegu.

Ukoliko se na primer u tabeli Person nalaze sledeća imena:



Slika 27. Podaci iz tabele Person

I ukoliko se zatim izvrši sledeća naredba:

Slika 28. Transakcija koja koristi serializable nivo izolacije

Serializable nivo izolacije će sprečiti bilo koju drugu transakciju da unese novu osobu koja se zove *Nick*. Uzevši u obzir da postoji indeks nad kolonama *Name* i *LastName,* SQL Server će zauzeti *range lock* između vrsta *Nick* i *Steve*, što će sprečiti bilo koju vrstu INSERT naredbi između ove dve vrednosti. Sa druge strane, ova vrsta brave neće sprečiti druge korisnike da čitaju ove vrednosti. Takođe ovo znači da iako transakcija samo čita osobu čije je ime *Nick*, neka druga transakcija neće moći da, na primer, unese osobu sa imenom *Rick*. Nakon što je transakcija okončana i *range lock*-ovi će biti otpušteni. [12][13][14]

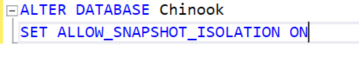
Na kraju, *Serializable* nivo izolacije predstavlja najrestriktivniji nivo izolacije koji postoji u SQL Serveru, a samim tim nosi za sobom i posledicu da će nivo konkurentnosti između konkurentnih transakcija biti nizak.

## 4.5. *Snapshot* nivo izolacije

Ovaj nivo izolacije navodi da će podaci pročitani bilo kojom naredbom u transakciji biti transakcijski dosledna verzija podataka koja je postojala na početku transakcije. Transakcija može prepoznati samo modifikacije podataka koje su izvršene pre njenog početka. Izmene podataka izvršene od strane drugih transakcija nakon početka izvršenja trenutne transakcije, nisu vidljive naredbama koje se izvršavaju u tekućoj transakciji. Efekat je kao da izrazi u transakciji dobijaju snimak (engl. *snapshot*) podataka onakvih kakvi su postojali na početku transakcije. Na ovaj način svaka transakcija radi nad sopstvenom kopijom podataka.

*Snapshot* izolacioni nivo ne zahteva postavljanje brava prilikom čitanja podataka. Takođe, ovakve transakcije koje čitaju podatke iz neke tabele neće blokirati druge transakcije koje žele da upisuju nove podatke, a ni obrnuto.

Ovakav vid izolacije se smatra najboljim u situacijama kada se koriste *read-only* naredbe. Ali u situacijama kada više transakcija pokušava da izmeni isti podatak, umesto blokiranja, doći će do toga da će jedna od transakcija pući u momentu kada pokuša da *commit*-uje podatke. [12][13][14]

Pre započinjanja transakcija koje koriste opisani nivo izolacije potrebno je navesti sledeću naredbu:

Slika 29. Postavlajnje snapshot izolacionog nivoa

### 4.5.1. *Read commited snapshot* nivo izolacije

Ukoliko je postavljena ova opcija, ona će uzrokovati da će se *read commited* transakcije ponašati poput *snapshot* transakcija, umesto da koriste mehanizam zaključavanja (što je slučaj kod regularnog *read commited* izolacionog nivoa).

## 4.6. Zaključak o na temu različitih izolacionih nivoa

U jednom trenutku se može postaviti samo jedna opcija nivoa izolacije koje nudi SQL Server, i ona ostaje postavljena za tu konekciju ka SQL Serveru sve dok se izričito ne promeni. Niži nivo izolacije povećava mogućnost da više korisnika istovremeno pristupa podacima, ali povećava i broj konkurentnih efekata, kao što su prljava čitanja ili izgubljena ažuriranja itd. Suprotno tome, viši nivo izolacije smanjuje razne tipove konkurentnih efekata na koje bi korisnici mogli da naiđu, ali zahteva više sistemskih resursa i povećava šanse da će jedna transakcija blokirati drugu, i samim tim primorati korisnike da čekaju.

# Zaključak

Na kraju, najbolja praksa je da transakcje koje se kreiraju u SQL Serveru budu što kraće. Kada se transakcija započne, sistem za upravljanje bazom podataka (DBMS) mora da zadrži mnogo resursa do kraja transakcije kako bi zaštitio svojstva atomičnosti, konzistencije, izolacije i trajnosti (*ACID*) te transakcije. Ako su podaci izmenjeni, izmenjeni redovi moraju biti zaštićeni ekskluzivnim bravama koje sprečavaju čitanje tih redova od strane bilo koje druge transakcije, a ekskluzivne brave se moraju držati sve dok transakcija ne bude uspešno izvršena ili poništena. Takođe, u zavisnosti od izabranog nivoa izolacije transakcije, SELECT izrazi mogu imati brave, koje se moraju zadržati dok transakcija ne bude *commit*-ovana ili poništena. Naročito u sistemima sa velikim brojem korisnika, transakcije moraju biti što kraće kako bi se smanjila trka oko zaključavanja resursa između konkurentnih transakcija. I na kraju, dugotrajne, neefikasne transakcije možda neće predstavljati problem kod malog broja korisnika, ali su gotovo neprihvatljive u sistemu sa hiljadama korisnika, kakvi su često sistemi kojima upravlja SQL Server.

# Literatura

[1] Sve o zaključavanju u SQL Serveru <https://www.sqlshack.com/locking-sql-server/>

[2] Postavljanje nivoa izolacije u Microsoft Sql Server-u <https://contentlab.io/transaction-isolation-levels-in-microsoft-sql-server/>

[3] Zaključavanje transakcija i verzionisanje <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver15>

[4] Kontrolisanje trajnosti transakcija <https://docs.microsoft.com/en-us/sql/relational-databases/logs/control-transaction-durability?view=sql-server-ver15>

[5] Brojanje otvorenih transakcija @@TRANCOUNT <https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql?view=sql-server-ver15>

[6] Razumevanje transakcija u SQL Serveru <https://www.c-sharpcorner.com/UploadFile/84c85b/understanding-transactions-in-sql-server/>

[7] Distribuirane transakcije u MSSQL-u <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/begin-distributed-transaction-transact-sql?view=sql-server-ver15>

[8] Pražnjenje Log bafera [https://sqlperformance.com/2018/11/sql-performance/understanding-log-buffer-flushes](https://sqlperformance.com/2018/11/sql-performance/understanding-log-buffer-flushes%20)

[9] Sve o zaključavanju u SQL Server-u <https://www.sqlshack.com/locking-sql-server/>

[10] Brave u SQL Serveru <https://vaishaligoilkar3322.medium.com/lock-in-sql-server-bb7ea3e64e9f>

[11] sys.dm\_tran\_locks <https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-tran-locks-transact-sql?view=sql-server-ver15>

[12] Nivoi izolacije kod MSSQL-a [https://contentlab.io/transaction-isolation-levels-in-microsoft-sql-server/](%20https:/contentlab.io/transaction-isolation-levels-in-microsoft-sql-server/)

[13] Nivoi izolacije u SQL Server-u <https://www.sqlservercentral.com/articles/isolation-levels-in-sql-server>

[14] Postavljanje nivoa izolacije <https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver15>

1. U trenutku pisanja ovog rada najnovija verzija SQL Servera je Microsoft SQL Server 2019, RTM 15.0.2000.5. [↑](#footnote-ref-1)