|  |  |  |
| --- | --- | --- |
| Unigrb | Univerzitet u Nišu ELEKTRONSKI FAKULTET | logo_1960_4 |

**Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje kod MS SQL baze podataka**

SEMINARSKI RAD

|  |  |
| --- | --- |
| Mentor: | Student: |
|  |  |
| Prof. dr Aleksandar Stanimirović | Stefan Mladenović1694 |

Niš, maj 2024. God.

[1. Uvod 2](#_Toc566240589)

[1.1. Pojam transakcije 3](#_Toc1484152169)

[2. Obrada transakcija 5](#_Toc569151564)

[2.1. Režimi transakcija u SQL Serveru 5](#_Toc1955954468)

[2.1.1. Autocommit transakcije 5](#_Toc395695911)

[2.1.2. Implicit transakcije 5](#_Toc185895522)

[2.1.3. Explicit transakcije 6](#_Toc1604565288)

[2.1.4. Batch-scoped transakcije 7](#_Toc32076037)

[2.2. Komande za upravljanje transakcijama 8](#_Toc19191548)

[2.2.1. BEGIN, COMMIT i ROLLBACK komande 8](#_Toc2101911904)

[2.2.2. TRANCOUNT globalna promenljiva 11](#_Toc1409690961)

[2.2.3. SAVE komanda 13](#_Toc641837295)

[3. Planovi izvršenja transakcija 15](#_Toc1048005356)

[4. Izolacija transakcija 18](#_Toc1851563443)

[4.1. Izolacioni nivoi 18](#_Toc149726714)

[4.1.1. Read Uncommitted 18](#_Toc1498046657)

[4.1.2. Read Committed 19](#_Toc1168740146)

[4.1.3. Repeatable Read 21](#_Toc995974552)

[4.1.4. Snapshot 21](#_Toc1281146938)

[4.1.5. Serializable 22](#_Toc1154095491)

[5. Zaključavanje 23](#_Toc1444664835)

[5.1. Hijerarhija i granularnost zaključavanja 24](#_Toc1046740740)

[5.2. Režimi zaključavanja 25](#_Toc1617070923)

[5.2.1. Shared locks 25](#_Toc523560313)

[5.2.2. Exclusive locks 25](#_Toc548205361)

[5.2.3. Update locks 26](#_Toc591741212)

[5.2.4. Intent locks 27](#_Toc248068498)

[5.2.5. Schema locks 28](#_Toc382233103)

[5.2.6. Bulk update locks 28](#_Toc1994735728)

[5.2.7. Key-range locks 28](#_Toc1724322420)

[5.3. Kompatibilnost režima zaključavanja 29](#_Toc1378716525)

[5.4. Eskalacija zaključavanja 29](#_Toc822181961)

[5.4.1. Uslovi za pokretanje eskalacije 32](#_Toc703303294)

[6. Zaključak 33](#_Toc330149766)

[7. Literatura 34](#_Toc610823822)

# Uvod

Transakcije u SQL Server-u kao i u drugim sistemima za upravljanje bazama podataka predstavljaju ključni mehanizam za održavanje integriteta podataka i omogućavaju pouzdano izvršavanje operacija nad bazom podataka. Ovaj rad se fokusira na različite aspekte obrade transakcija, uključujući njihove režime izvršenja, izolacione nivoe, planove izvršenja i zaključavanja transakcija.

U prvom delu rada razmatra se pojam transakcije, gde se objašnjavaju osnovni principi i značaj transakcija u upravljanju bazama podataka. Detaljno se istražuju različiti režimi transakcija koje SQL Server podržava, kao i komande koje se koriste za upravljanje transakcijama.

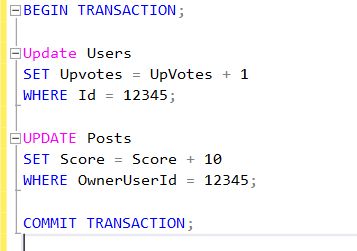
Dalje, rad obuhvata izolaciju transakcija, koja je od suštinskog značaja za sprečavanje problema poput *dirty read,* nepostojećih podataka i *phantom read.* Rad detaljno opisuje različite izolacione nivoe koje SQL Server nudi, uključujući *Read Uncommitted, Read Commited, Repetable Read, Snapshot i Serializable.*

Na kraju rada, prikazani su režimi zaključavanja koji su implementirani u SQL Serveru kako bi se omogućila efikasna izolacija i upravljanje konkurentnim pristupom resursima. Ovi mehanizmi zaključavanja uključuju *shared, exclusive, update, intent, schema, bulk update*  i *key-range* režime zaključavanja.

Ovaj rad pruža detaljan pregled i analizu ključnih aspekata obrade transakcija u SQL -Serveru, uključujući primere koji dodatno pojašnjavaju opisane mehanizme. Primeri koji su demonstrirani u daljem tekstu su izvršavani u okviru SQL Servera verzije 16.0.1.

## 1.1. Pojam transakcije

Transakcija predstavlja logičku jedinicu koja obavlja jednu ili više aktivnosti u bazi podataka. One mogu uključivati čitanje, pisanje, brisanje, ažuriranje podataka ili kombinaciju više operacija. [1]



Slika 2.1. Primer transakcije

Glavna ideja transakcija jeste da se dati skup operacija izvrši atomično, odnosno sve izmene se poništavaju ako je neka operacija iz skupa izazvala grešku. Na taj način se vrši očuvanje integriteta podataka. Kao što je dato na primeru, gde će na disku biti sačuvane promene samo ako se obe operacije izvrše uspešno.   
 Upravo ACID svojstva garantuju da se ovaj primer izvrši u potpunosti. ACID (a*tomicity*, *consistency*, *isolation* i *durability*) predstavlja skraćenicu za atomičnost, konzistentnost, izolaciju i trajnost.

* **Atomičnost**: Ovo svojstvo upravo osigurava da transakcija bude izvršena kao celina. U suprotnom ni jedna promena neće biti sačuvana, transakcija će biti poništena i stanje baze će se vratiti na stanje pre izvršenja transakcije.
* **Konzistentnost**: Svi podaci moraju ostati konzistentni nakon izvršenja transakcije poštujući definisana pravila, ograničenja, kaskade i okidače.
* **Izolacija**: Transakcije koje se izvršavaju istovremeno moraju biti izolovane jedna od druge. To znači da jedna transakcija ne sme videti promene druge transakcije sve dok ona ne bude kompletno izvršena.
* **Trajnost**: Nakon izvršenja transakcije promene postaju trajne u bazi podataka.

Ova svojstva pružaju osiguranje pouzdanosti i integriteta podataka unutar MS SQL baze podataka.

Transakcije u MSSQL Serveru koristi napredne mehanizme za upravljanje zaključavanjem i verzionisanjem podataka, što omogućava konkurentne pristupe i održavanje performansi na visokom nivou. Zaključavanje podataka omogućava MSSQL Serveru da kontroliše pristup resursima koji se koriste tokom izvršenja transakcija, dok se verzionisanje podataka koristi za upravljanje promenama u podacima tokom izvršenja transakcija. Ovaj pristup omogućava da se čitanja izvršavaju bez zaključavanja, koristeći *snapshot* verzije podataka koje su bile važeće na početku transakcije.

Ovi mehanizmi su ključni za održavanje ACID svojstva transakcije, čime se osigurava da baza ostane u konzistentnom stanju čak i u slučaju grešaka i prekida.

# Obrada transakcija

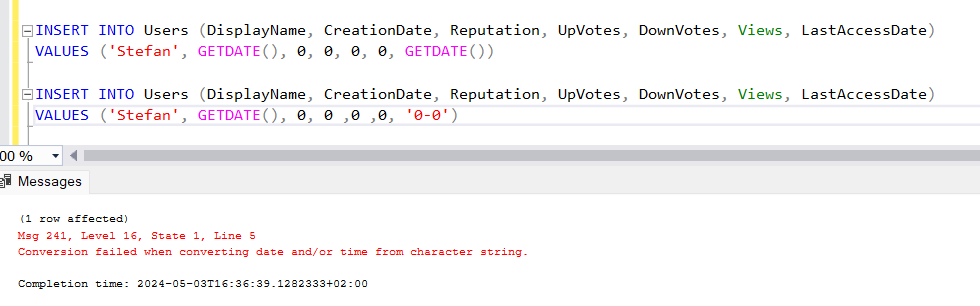
## 2.1. Režimi transakcija u SQL Serveru

MS SQL Server podržava tri različita režima izvršenja transakcija i to su:

1. *Autocommit* transakcije
2. *Implicit* transakcije
3. *Explicit* transakcije
4. *Batch-scoped* transakcije [1]

### 2.1.1. *Autocommit* transakcije

*Autocommit t*ransakcije predstavljaju podrazumevani režim izvršenja transakcija u SQL Serveru. U ovom režimu, svaka pojedinačna SQL naredba automatski formira transakciju i automatski se komituje odmah nakon izvršenja. To znači da svaka naredba poput INSERT, UPDATE ili DELETE, automatski pokreće transakciju i zatvara je nakon izvršenja bez eksplicitnog definisanja početka i kraja transakcije.

Slika 2.2. Primer *autocommit* transakcije

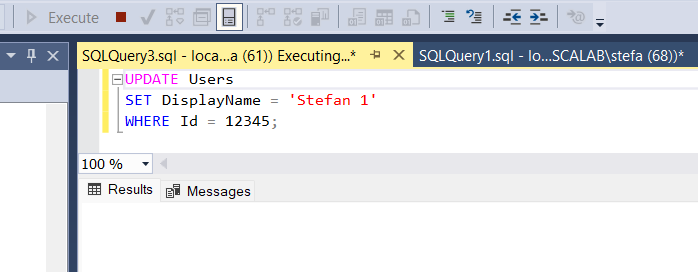
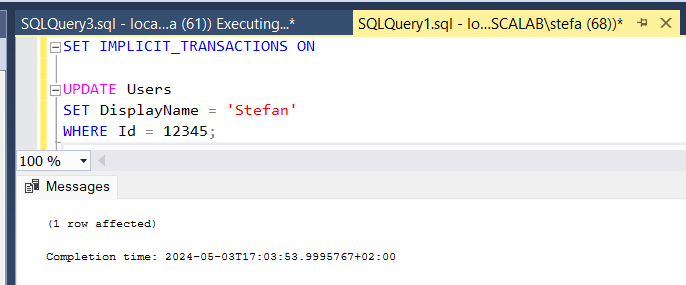
Na primeru je prikazano da se ove dve operacije posmatraju kao nezavisne transakcije. Prva transakcije je uspešno izvršena i promena je sačuvana u bazi, dok druga transakcija javlja grešku zbog nevalidanog formata datuma.

### 2.1.2. *Implicit* transakcije

Kod ovog režima transakcija, SQL Server podrazumeva početak transakcije implicitno, ali ne podrazuvema komande za potvrdu ili poništavanje transakcije, već korisnik mora eksplicitno da naglasi kada će se transackija potvrditi ili poništiti.

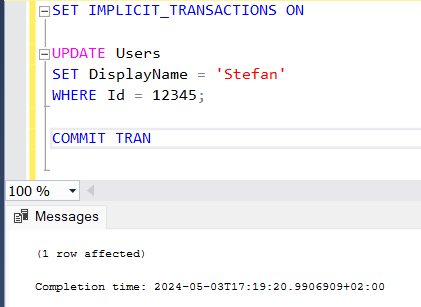
Ukljičivanje i isključivanje ovog moda vrši se pomoću komande:  
  
 *SET IMPLICIT\_TRANSACTIONS ON/OFF*

U implicitnom režimu transakcija, objekti baze podataka uključeni u transakciju biće zaključani dok se ne izvrše komande za potvrdu ili poništavanje.



Slika 2.3. Primer zaključavanja *implicitne* transakcije

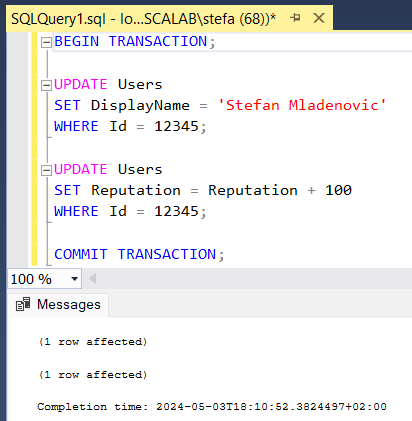
Na primeru je demonstrirano da druga transakcija ne može da se izvrši, dok se prva transakcija ne potvrdi ili ne prekine. Da bi se transakcija sačuvala u bazi potrebno je izvršiti potvrdu transakcije komandom *COMMIT TRAN*.



Slika 2.4. Primer potvrde implicitne transakcije

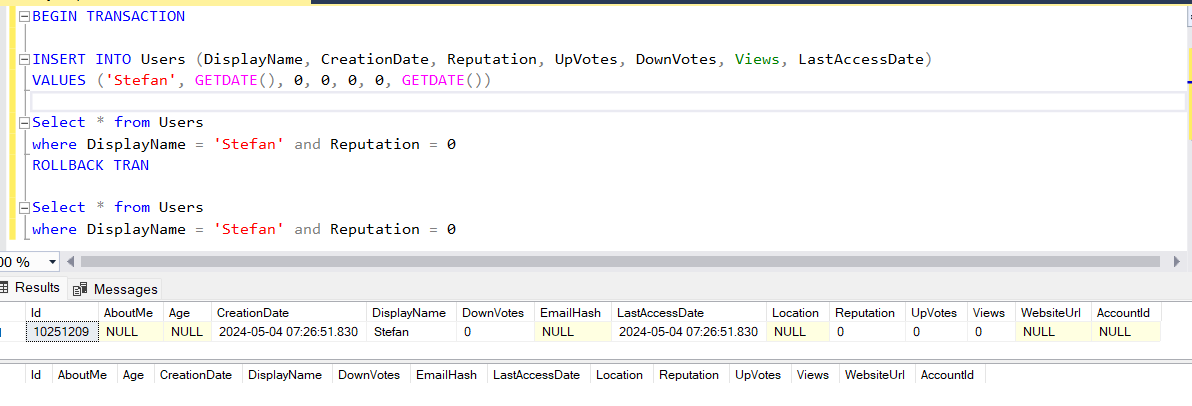
### 2.1.3. *Explicit* transakcije

U ovom režimu transakcija, moramo definisati početne i završne tačke transakcije, odnosno moramo navesti početak sa *BEGIN TRANSACTION* naredbom i kraj transakcije ili sa *COMMIT TRANSACTION* ili sa *ROLLBACK TRANSACTION* naredbom.

  
Slika 2.5. Primer eksplicitne transakcije

U ovom primeru smo koristili *COMMIT* naredbu kojom se potvrđuju promene koje su izvršene u toku transakcije. Nakon potvrđivanja transakcije *COMMIT* naredba nam garantuje da će promene biti zapisane trajno na disku.

Kada hoćemo da poništimo transakciju koristimo naredbu *ROLLBACK* na kraju transakcije. Nakon čega promene, koje su se desile u okviru transakcije, neće biti sačuvane u bazi, već će biti vraćene na stanje pre izvršenja transakcije.

Slika 2.6. Primer *rollback transaction* naredbe

Na ovom primeru je demonstrirano kako su promene koje su se desile u okviru transakcije poništene.

### 2.1.4. *Batch-scoped* transakcije

*Batch-scoped* transakcije su primenljive samo u *Multiple Active Result Sets (MARS)* sesijama. *MARS* omogućava aplikacijama da održavaju više aktivnih upita istovremeno unutar jedne sesije. Ova funkcionalnost je posebno korisna u scenarijima gde aplikacija treba da izvršava nezavisne upite i ažurira podatke istovremeno bez potrebe za otvaranjem više konekcija. [1]

Kada se u okviru *MARS* sesije započne transakcija, bilo eksplicitno ili implicitno, ta transakcija postaje *batch-scoped.* To znači da je opseg transakcije ograničen na taj *batch* SQL naredbi u kojem je transakcija započeta.

Glavna karakteristika *batch-scoped* transakcija je automatsko poništavanje od strane SQL Servera ako nisu eksplicitno potvrđene ili poništene do kraja *batch-a* (skup SQL naredbi). Na ovaj način SQL Server garantuje da neće sačuvati nepotpune promene u bazi podataka, čime se održava integritet i konzistentnost podataka.

## 2.2. Komande za upravljanje transakcijama

### 2.2.1. BEGIN, COMMIT i ROLLBACK komande

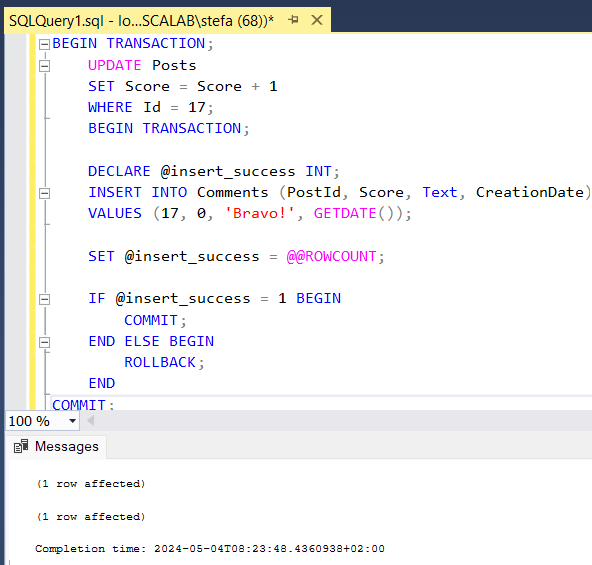
***BEGIN TRANSACTION***

Kao što smo već napomenuli, ova naredba se koristi za označavanje početka eksplicitne transakcije u SQL Serveru i signalizira sistemu za upravljanje bazom podataka da sledeće operacije treba tretirati kao deo jedne transakcije. Ukoliko je na ovaj način definisan sam početak transakcije, na kraju transakcija se mora zatvoriti sa *COMMIT* ili *ROLLBACK*.

Sintaksa komande izgleda ovako:  
  
BEGIN {TRAN | TRANSACTION} [ { transaction\_name | @tran\_name\_variable} [ WITH MARK [ ‘deskripcija’]]]

* *BEGIN TRAN* ili *BEGIN TRANSACTION* su ključne reči koje se koriste za označavanje početka transakcije.
* *Transaction\_name* i *@tran\_name\_variable* su opcioni parametri i omogućavaju imenovanje transakcije. Transakcija može biti imenovana eksplicitno ili korišćenjem promenljive koja sadrži ime transakcije. Dodeljivanje imena može biti korisno u kasnijim koracima za referenciranje trasakcije.
* WITH MARK [ ‘deskripcija’] je dodatna opcija koja omogućava obeležavanje transakcije. Opciono može biti navedena deskripcija kojom se opisuje transakcija. [2]

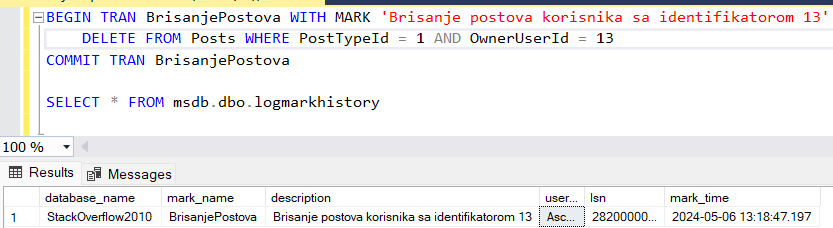
Korišćenjem ove komande možemo definisati i ugnježdene transakcije. Kad određenu transakciju ubacimo unutar druge transakcije, ta druga transakcija postaje roditeljska transakcija. U slučaju da se unutrašnja transakcija završi sa *COMMIT,* promene će biti zadržane samo ako se i spoljna transakcija završi sa *COMMIT.* Ukoliko se spoljna transakcija završi sa *ROLLBACK,* sve promene iz obe transakcije će biti poništene.



Slika 2.7. Primer ugnježdene transakcije

U ovom primeru je demonstrirano korišćenje ugnježdene transakcije. U okviru roditeljske transakcije ažurira se tabela *Posts.* a nakon toga u ugnježdenoj transakciji vrši se dodavanje novog sloga u tabeli *Comments.* U slučaju uspešnog dodavanja ugnježdena transakcija se potvrđuje, dok se u suprotnom poništavaju promene načinjene i u ugnježdenoj kao i u roditeljskoj transakciji. Nakon toga se potvrđuje i roditeljska transakcija.

SQL Server omogućava da označimo i dodamo opis određenoj transakciji u logu transakcija. Na taj način možemo lakše pratiti istoriju transakcija. U slučaju da je neka transakcija izvršila pogrešnu promenu podataka na osnovu log-a možemo identifikovati tačno vreme izvršenja, a nakon toga po potrebi vratiti bazu na stanje pre izvršenja same transakcije korišćenjem backup-ova. [2]

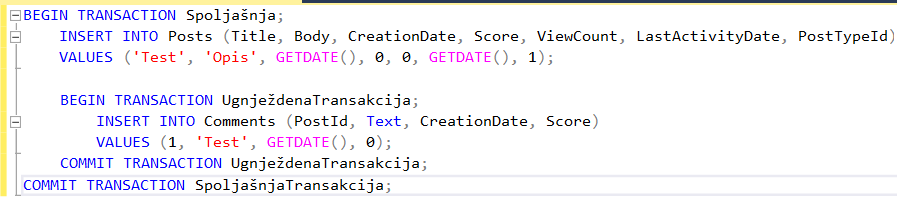
Slika 2.8. Primer markiranja transakcija

***COMMIT TRANSACTION***

*COMMIT* označava kraj uspešno izvršene transakcije u implicitnom ili eksplicitnom režimu transakcija. Naredba se koristi da se trajno sačuvaju sve promene napravljene tokom trenutne transakcije i nakon *COMMIT* naredbe sve promene postaju vidljive i ostalim korisnicima. Ovom naredbom se takođe potvrđuju i ugnježdene transakcije.  
  
Sintaksa izgleda ovako:

COMMIT [ {TRAN | TRANSACTION} [transaction\_name | @tran\_name\_variable]] [ WITH (DELAYED\_DURABILITY = {OFF | ON})]

* *COMMIT –* predstavlja ključnu reč za potvrđivanje promena u okviru trenutne transakcije.
* *TRAN* ili *TRANSACTION –* predstavljaju opcione ključne reči koje mogu biti korišćene uz *COMMIT* dase potvrdi transakcija koja je prethodno započeta sa *BEGIN TRAN* ili *BEGIN TRANSACTION*.
* *Transaction\_name* i *@tran\_name\_variable* – predstavlja opcioni deo koji omogućava navođenje imena transakcije koja se potvrđuje. Može da se navede eksplicitno ime transakcije ili putem promenljive koja sadrži ime konkretne transakcije.
* *WITH (DELAYED\_DURABILITY = { OFF | ON })* - omogućava da dodatno uključimo ili isključimo kontrolu *delayed durability* za transakciju. Ovo omogućava da SQL Server može da potvrdi transakciju bez čekanje na potpuno upisivanje logova na disku, što može dovesti do poboljšanja performansi, ali i rizik od gubljenja informacija u slučaju pada sistema. [3]

Slika 2.9. Primer potvrđivanja transakcija

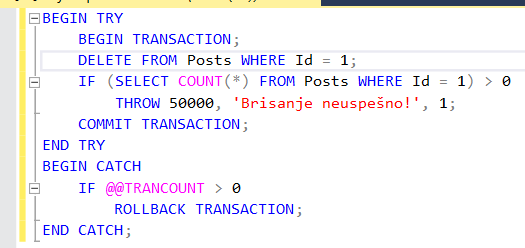
***ROLLBACK TRANSACTION***

*ROLLBACK* komanda u SQL Server-u koristi se za poništavanje promena koje su se dogodile unutar jedne transakcije, odnosno vraća bazu podataka u stanje pre početka transakcije. Ova komanda se koristi u slučaju grešaka ili neočekivanih situacija tokom izvršenja određene transakcije kako bi sprečila zadržavanje nepostojećih ili delimično promenjenih podataka u bazi.

Sintaksa izgleda ovako:

ROLLBACK {TRAN | TRANSACTION} [transaction\_name | @tran\_name\_variable | savepoint\_name | @savepoint\_variable]

* *ROLLBACK –* predstavlja ključnu reč za poništavanje promena u okviru transakcije ili dela transakcije.
* *TRAN* ili *TRANSACTION –* predstavljaju opcione ključne reči koje mogu biti korišćene uz *ROLLBACK* dase poništi transakcija koja je prethodno započeta sa *BEGIN TRAN* ili *BEGIN TRANSACTION*.
* *transaction\_name, @tran\_name\_variable, savepoint\_name, @savepoint\_variable –* Ovo je opcioni deo koji omogućava navođenje imena transakcije koja se poništva ili imena *tačke čuvanja* do koje se vraća transakcija. Moguće je eksplicitno navesti ime transakcije koje se poništava, ime *tačke čuvanja* ili korisiti promenljivu koja sadrži to ime. [4]

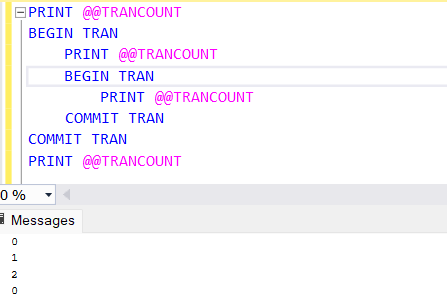


Slika 2.10. Primer korišćenja *ROLLBACK* komande

Dobra prakse je da se *ROLLBACK* komanda koristi u kombinaciji sa *TRY* i *CATCH* blokovima za obradu grešaka. Na primeru je prikazano kako u slučaju greške ili nepotpunog izvršenja određene naredbe možemo da poništimo promene koje su nastale prilikom izvršenja transakcije. Pre izvršenja *ROLLBACK* komande vršimo proveru aktivnog broja transakcija, odnosno da li uopšte postoji aktivna transakcija pomoću *@@TRANCOUNT* globalne promenljive. Na taj način se osiguravamo da transakcija nije još zatvorena i izbegavamo mogućnost pojave nove greške.

### 2.2.2. TRANCOUNT globalna promenljiva

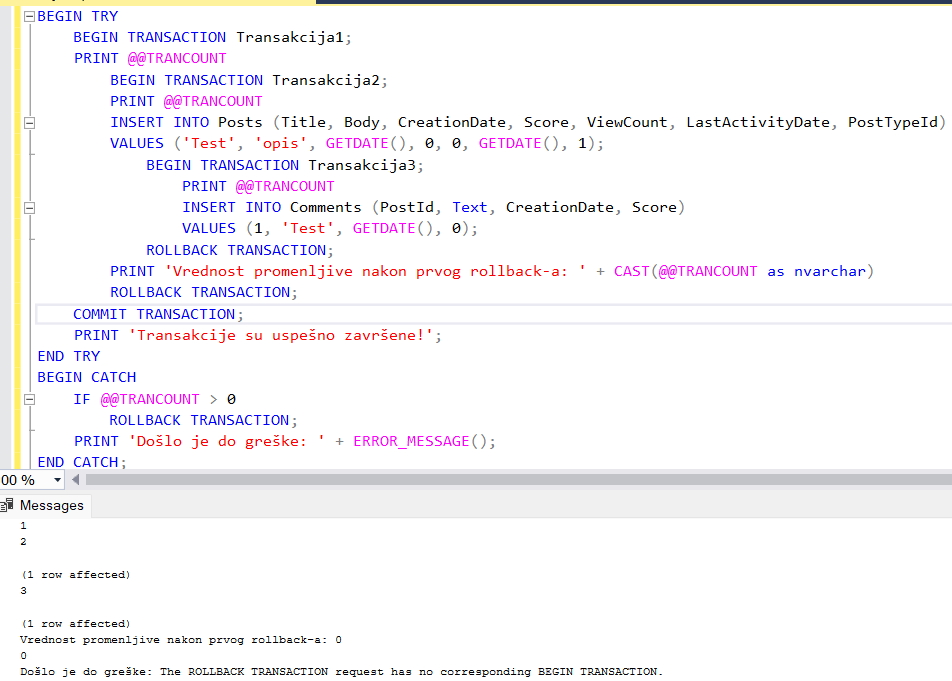
@@TRANCOUNT predstavlja globalnu promenljivu u Microsoft SQL Server-u koja sadrži broj trenutno otvorenih transakcija za trenutnu konekciju.



Slika 2.11. Primer rada *@@TRANCOUNT* globalne promenljive

Prilikom otvaranja svake nove transakcije vrši se uvećavanje @@*TRANCOUNT* promenljive za jedan, a prilikom potvrđivanja svake od transakcija vrši se dekrementiranje promenljive za jedan. Kao što je na primeru demonstrirano.

Naredba *ROLLBACK*, bez obzira da li navodi ime transakcije ili ne, izvršiće poništavanje svih promena napravljenih u okviru te transakcije i resetovati *@@TRANCOUNT* na 0. To je zato što SQL Server tretira sve ugnježdene transakcije kao deo jedne transakcije, čak i ako navedete ime transakcije u *ROLLBACK* naredbi, i dalje će sve transakcije biti poništene i brojač će biti vraćen na 0. [5]

Slika 2.11. Primer resetovanja @@TRANCOUNT globalne promenljive

### 2.2.3. *SAVE* komanda

Komanda *SAVE* u SQL Server-u predstavlja *tačku čuvanja* unutar trenutne transakcije. Tačnije *tačka čuvanja* može se zatim koristiti kako bi se poništio deo transakcije bez uticaja na celu transakciju. U prethodnom delu smo objasnili da korišćenjem samo *ROLLBACK* komande u okviru transakcije nije moguće poništiti samo ugnježdenu transakciju.   
  
Sintaksa za *SAVE* komandu izgleda ovako:

SAVE {TRAN | TRANSACTION} { savepoint\_name | @savepoint\_variable}

Gde:

* *Savepoint\_name* predstavlja opcioni argument koji označava ime tačke čuvanja koju želimo da kreiramo. Ovo ime je proizvoljno i treba da bude jedinstveno unutar trenutne transakcije. Ime je bitno kako bi kasnije mogli da se referenciramo na tu tačku prilikom poništavanja dela transakcije.
* *@savepoint\_variable* je takođe opcioni argument koji omogućava korišćenje promenljive umesto eksplicitno navedenog imena *tačke čuvanja.* Na ovaj način se omogućava dinamičko imenovanje *tačke čuvanja* u skriptama. [6]

Kao što smo napomenuli pomoću *tačke čuvanja* možemo vratiti stanje transakcije na stanje pre njenog kreiranja pomoću naredbe *ROLLBACK TRANSACTION <ime tačke čuvanja>.* Ovo omogućava programerima da podele transakciju na više logičkih delova i da vrše kontrolisani *rollback* samo delova transakcije koji su neophodni.  
  
 *Rollback* operacije su skupe u pogledu performansi, posebno ako se transakcija sastoji od velikog broja operacija. Korišćenjem *SAVE* komande, možete smanjiti uticaj na performanse tako što ćete izbegavati *rollback* celokupne transakcije.



Slika 2.12. Primer korišćenja *SAVE TRANSACTION* komande

Na ovom primeru je ilustrovano kako možemo iskoristi *SAVE TRANSACTION* komandu i njen uticaj na *@@TRANCOUNT* promenljivu. U prvoj transakciji je prikazana vrednost promenljive nakon kreiranja prve transakcije, zatim ažurirana kolona *Reputation* za korisnika sa identifikatorom 5 i prikazana vrednost ažuriranog podatka. Nakon kreiranja *tačke čuvanja,* otvorena je druga transakcija, prikazana je uvećana vrednostpromenljive i ažurirana kolona *Reputation* za istog korisnika, kao i prikazana uvećana vrednost. Nakon toga odrađeno je vraćanje na *tačku čuvanja* i prikazana vrednost promenljive koja u ovom slučaju nije promenjena. Možemo videti da *rollback* na tačku čuvanja ne utiče na *@@TRANCOUNT* promenljivu. Zatim vidimo da je vrednost *Reputation* kolone za tog korisnika vraćena na vrednost pre *tačke čuvanja* (što je ujedno i bio cilj)*.* Pošto nakon vraćanja na tačku čuvanja vrednost promenljive nije umanjena, odnosno druga transakcija nije završena, moramo izvršiti zatvaranje te transakcije (u ovom slučaju je potvrđena). Nakon toga vidimo da je ostala otvorena samo prva transakcija, koja je kasnije poništena i @@*TRANCOUNT* promenljiva vraćena na nulu (odnosno sve transakcije su zatvorene).

# Planovi izvršenja transakcija

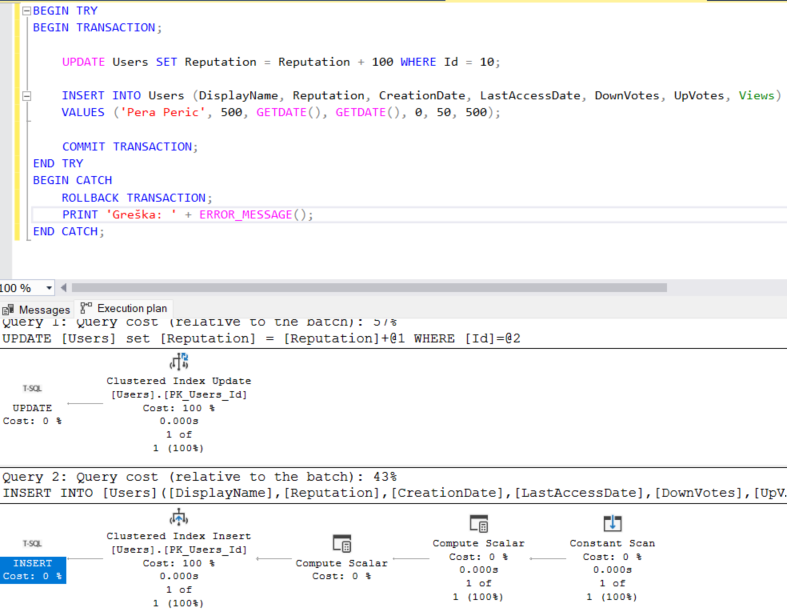
Plan izvršenja predstavlja detaljno opisan plan kako SQL Server izvršava datu transakciju ili upit. Sam plan sadrži redosled izvršenja operacija, pristup tabelama i indeksima, kao i vreme i resurse koji su utrošeni na izvršenja svakog od koraka.

Plan izvršenja se generiše od strane SQL Server optimizatora upita, koji procenjuje moguće metode izvršavanja date SQL operacije i bira najefikasniju na osnovu procene troškova. Troškovi se izračunavaju na osnovu faktora kao što su I/O operacije, upotreba CPU-a, pristup memoriji itd.

Neki od ključnih delova planova izvršenja su:

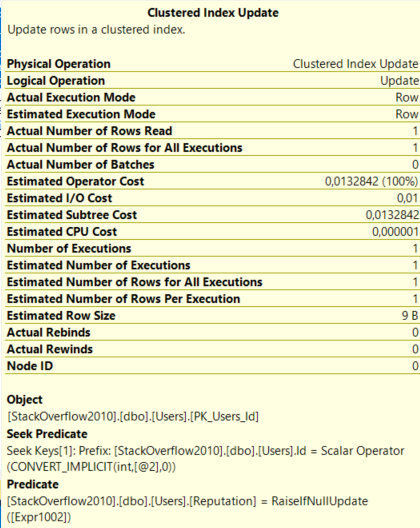
1. Operatori: Izvršni plan obuhvata niz operacija koje će se izvršiti kako bi se došlo do konačnog rezultata upita. Ovi operatori uključuju skeniranje tablica, sortiranje, grupisanje, spajanje (*join*), filtriranje i druge.
2. Tipovi pristupa podacima: Izvršni plan može pokazati kako se pristupa podacima, kao što su potpuno skeniranje tablica, indeksirani pristupi ili kombinacije.
3. Primeri korišćenja indeksa: Izvršni plan obično prikazuje da li su indeksi korišćeni i koji su indeksi izabrani za svaku operaciju. Korišćenje odgovarajućih indeksa može značajno poboljšati performanse upita.
4. Redosled operacija: Redosled operacija u izvršnom planu može imati značajan uticaj na performanse upita. Na primer, ako se prvo izvrši skeniranje velike tabele, a zatim se primene filteri, to može rezultirati većim brojem nepotrebnih operacija.
5. Zaključavanje i konkurentnost: Plan izvršenja takođe uključuje strategije za zaključavanje resursa radi održavanja integriteta podataka. SQL Server optimizuje tip i opseg zaključavanja kako bi smanjio uticaj na ostale konkurente pristupe.
6. Podupiti i ugnježdene transakcije: SQL Server Optimizator rukuje podupitima i ugnježdenim transakcijama optimizujući svaku komponentu individualno i obezbeđujući da cela transakcija bude efikasna.
7. Filtriranje i sortiranje: Plan izvršenja može uključivati operacije filtriranja i sortiranja kako bi se izdvojili ili uredili rezultati upita.
8. Statistike i estimacije: SQL Server koristi statistike o podacima kako bi procenio broj redova i veličinu rezultata upita. Plan izvršenja može sadržati estimacije broja redova i procene troškova operacija kako bi se donele informisane odluke o izboru plana izvršenja.

Razumevanje plana izvršenja je vrlo bitno za optimizaciju samih upita. Analizom plana mogu se identifikovati uska grla i neefikasnosti koje mogu dovesti do vrlo loših performansi. Nakon toga mogu se preduzeti odgovarajući koraci, kao što su uvođenje indeksa, refaktorisanje upita, uvođenje privremenih tabela, ažuriranje statistika podataka itd.

Slika 3.1. Prikaz plana izvršenja

Na slici je grafički prikaz plana izvršenja u okviru *Microsoft SQL Server Management Studio* alata. U konkretnom primeru imamo transakciju koja sadrži dve naredbe: naredbu ažuriranja i naredbu dodavanja. Plan izvršenja koji je generisan za dati primer prikazuje korake koji su izvršeni za obe naredbe pojedinačno, ali daje cenu koja prikazuje koliko je procentualno koja naredba utrošila resursa.

Prelaskom miša preko svake ikone u grafičkom prikazu ili odabirom *Properties* opcije iz padajućeg menija, koji se otvara desnim klikom na ikonu, možemo videti detalje za konkretni operator u planu izvršenja.



Slika 3.2. Detaljan opis *Clustered Index Update* operatora u planu izvršenja

Na slici je dat opis operatora *Clustered Index Update* iz plana izvršenja za prethodni primer. Ovde možemo uočiti neke od elemenata opisa operatora kao što su:

* Stvarni broj pročitanih redova (*Actual Number of Rows Read*): Ova metrika pokazuje koliko tačno redova je pročitano iz baze podataka tokom izvršenja operacije. To je korisno za procenu stvarne količine podataka koja je obrađena.
* Stvarni broj redova za sva izvršenja (*Actual Number of rows for all executions*): Ova metrika ukazuje na ukupan broj redova koji su vraćeni za sve izvršene operacije. Ona pruža informacije o ukupnom obimu rezultata tokom svih izvršenja upita.
* Procenjeni trošak operatora (*Estimated operator cost*): Ova metrika predstavlja procenjeni trošak izvršenja određenog operatora u izvršnom planu. To obuhvata troškove CPU-a, I/O operacija i drugih faktora relevantnih za tu operaciju.
* Procenjeni trošak I/O operacija (*Estimated I/O cost*): Ova metrika predstavlja procenjeni broj I/O operacija (npr. čitanje i pisanje na disk) koji će biti izvršeni tokom operacije.
* Procenjeni trošak CPU-a (*Estimated CPU Cost*): Ova metrika predstavlja procenjeni trošak vremena koji je potreban CPU-u za izvršenje operacije.
* Procenjeni broj izvršenja (*Estimated Number of Executions*): Ova metrika pokazuje koliko puta se očekuje da će se određena operacija izvršiti tokom izvršenja upita.
* Procenjeni broj redova za sva izvršenja (*Estimated number of rows for all executions*): Ova metrika ukazuje na ukupan broj redova koji se očekuje da će biti vraćeni za sve izvršene operacije tokom izvršenja upita. [7]

Ovi elementi su korisni prilikom analize samog plana i donošenja odluka za dalje korake prilikom optimizacije samih upita.

# Izolacija transakcija

Izolacija transakcija u SQL Serveru odnosi se na nivo zaštite podataka kod konkurentnog pristupa od strane više transakcija istovremeno. Ako dve transakcije istovremeno pokušavaju da menjaju vrednost istog podatka, jednoj od njih može biti ograničen pristup dok se druga ne izvrši. Ovaj pristup omogućava da baze podataka ostanu dosledne, pouzdane i stabilne čak i u uslovima velike konkurentnosti i pristupa različitih korisnika.

SQL Server podržava pet različita izolaciona nivoa:

1. *Read Uncommitted*
2. *Read Committed*
3. *Repeatable Read*
4. *Snapshot*
5. *Serializable*

Podrazumevani izolacioni nivo je *Read Committed.* Sintaksa za postavljanje nekog od izolacionih nivo izgleda ovako:  
  
 SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE}

## 4.1. Izolacioni nivoi

U ovom delu rada biće opisane karakteristike i ograničenja svih izolacionih nivoa.

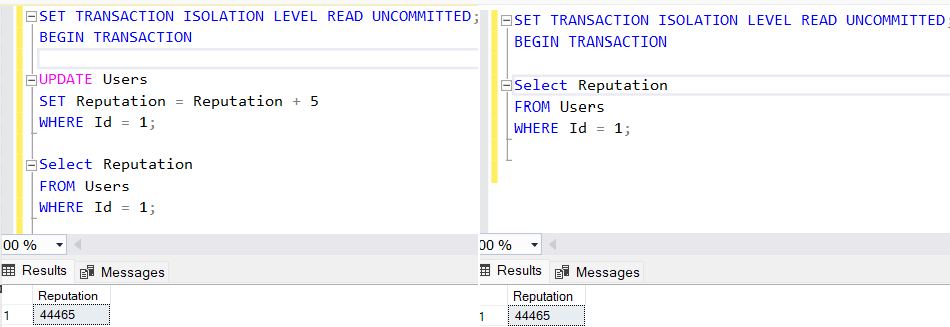
### 4.1.1. *Read Uncommitted*

Ovaj izolacioni nivo omogućava da jedna transakcija može čitati podatke koji su izmenjeni od strane drugih transakcija, ali još nisu potvrđene. Predstavlja najmanje restriktivni nivo izolacije.

Transakcije koje se izvršavaju na ovom nivou ne primenjuju *shared locks* kako bi sprečile druge transakcije da menjaju podatke koje trenutna transakcija čita. Takođe, transakcije nisu blokirane *exclusive locks* prilikom modifikacije podataka, čime se omogućava drugim transakcijama da čitaju modifikovane podatke koji još nisu potvrđeni. Kada je ova opcija postavljena, moguće je čitati nepotvrđene modifikacije, koje se nazivaju *dirty read*.

Problem može biti da se podaci koji se čitaju u okviru jedne transakcije mogu promeniti ili čak obrisati od strane drugih pre nego što je ta transakcija završena.

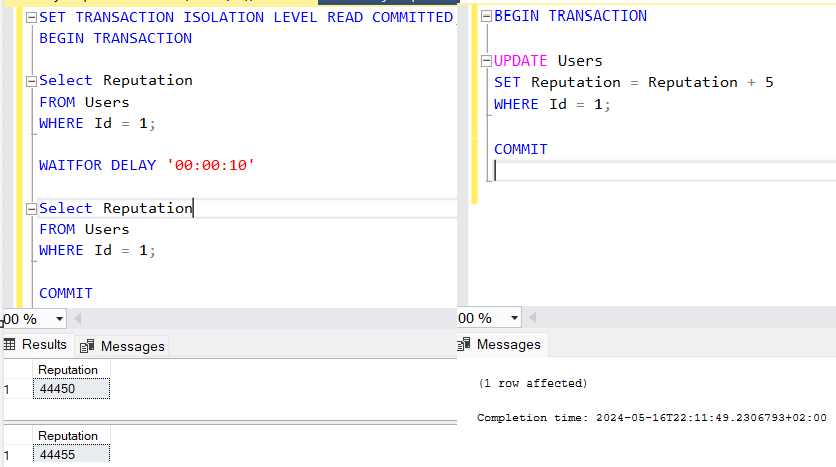
Ova opcija ima isti efekat kao postavljanje NOLOCK na svim tabelama u svim SELECT izjavama u transakciji. [8]

Slika 4.1. Primer *Read Uncommitted* transakcije

U ovom primeru imamo dve transakcije koje nisu potvrđene, prva koja ažurira i čita *Reputation* kolonu korisnika sa identifikatorom 1, a druga transakcija samo čita vrednost *Reputation* istog korisnika. Možemo videti da druga transakcija može pročitati izmenjeni podatak od strane prve transakcije iako prva transakcija nije potvrđena.

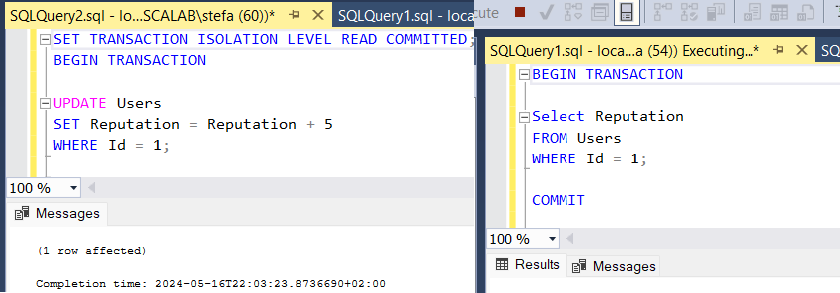
### 4.1.2. *Read Committed*

Ovo je podrazumevani izolacioni nivo u SQL Server-u. Transakcije na ovom nivou primenjuju *exclusive locks* prilikom modifikovanja podataka, čime se onemogućava da druge transakcije vrše čitanje podataka koji nisu još potvrđeni. *Read Committed* nivo sprečava *dirty read* pojavu. Međutim, podaci mogu biti promenjeni od strane drugih transakcija između pojedinačnih naredbi unutar trenutne transakcije, što rezultira neponovljivom čitanju ili pojave *phantom read.*

Slika 4.2. Primer simulacije pojave neponovljivog čitanja

Na ovom primeru je demonstrirana pojava neponovljivog čitanja*.* U prvoj transakciji vršimo čitanje iste kolone dva puta i možemo videti da se njena vrednost promenila u toku njenog izvršenja. Ovde smo iskoristili *timeout* kako bi drugom transakcijom izmenili vrednost kolone *Reputation* i na taj način demonstrirali pojavu neponovljivog čitanja.

Ponašanje *Read Commited* zavisi od podešavanja *READ\_COMMITTED\_SNAPSHOT* opcije baze podataka. Ako je *READ\_COMMITTED\_SNAPSHOT* podešen na *off, shared locks* sprečavaju druge transakcije da menjaju redove dok trenutna transakcija izvršava operaciju čitanja. *Shared locks* takođe blokiraju naredbu da čita redove koji su izmenjeni od strane drugih transakcija dok se druge transakcije ne izvrše. U suprotnom, ako je *READ\_COMMITTED\_SNAPSHOT* podešen na *on,* tada će biti primenjeno verzionisanje redova. Odnosno, ako transakcija čita određeni podatak, ona će pročitati vrednost tog podatka kakav je bio na početku tog izaraza, bez obzira da li ga neka druga transakcija trenutno menja ili ga je već promenila, ali promena nije potvrđena.

Slika 4.3. Primer izvršenja čitanja kod *Read Commited* izolacionog nivoa

Na primeru je prikazano podrazumevano ponašanje kod *Read Commited* izolacionog nivoa. Naime, u prvoj transakciji se vrši ažuriranje *Reputatiton* kolone bez potvrđivanja transakcija, a u drugoj transakciji se pokušava čitanje vrednosti iste kolone. Vidimo da druga transakcija ne može da se izvrši, a razlog tome je što prva transakcija nije završena, baš kako smo i objasnili u prethodnom pasusu.

### 4.1.3. *Repeatable Read*

Kod ovog izolacionog nivoa, naredbe ne mogu čitati podatke koji su izmenjeni od strane druge transakcije koja još uvek nije završena. Takođe, nijedna druga transakcija ne može menjati podatke koji su pročitani od strane trenutne transakcije dok se ona ne završi.

*Repetable Read* nivo postavlja *shared locks* na svim podacima koje pročita svaka naredba u transakciji i zadržavaju se sve dok se ta transakcija ne završi. Na ovaj način se sprečava da ostale transakcije menjaju bilo koji red koji je pročitala trenutna transakcija. Tako da ovaj nivo sprečava problem neponovljivog čitanja*.* Ali kod ovog nivoa se može javiti *phantom read* pojava u slučaju kada druge transakcije izvrše dodavanje novih redova čije vrednosti kolona odgovaraju kriterijumima naredbama čitanja trenutne transakcije. Tada će se pojaviti veći broj rezultata od očekivanih u trenutnoj transakciji. [8]

|  |  |
| --- | --- |
|  |  |

Slika 4.4. Primer izvršenja čitanja kod *Repetable Read* izolacionog nivoa

Na ovom primeru je demonstrirano kako prva transakcija, koja nije potvrđena, postavlja *shared locks* na red koji čitai zbog toga druga transakcija ne može da nastavi izvršenje sve dok se ne potvrdi ili poništi prva transakcija.

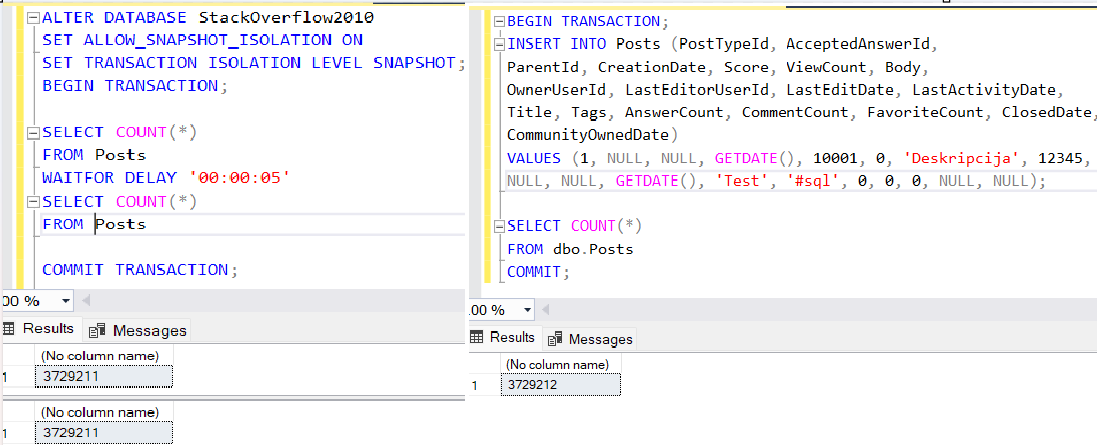
Korišćenjem ovog nivoa izolacije ograničava se konkuretni pristup podacima znatno više nego kod podrazumevanog *Read Commited.*

### 4.1.4. *Snapshot*

U ovom izolacionom nivou, podaci koje pročita bilo koja naredba u transakciji biće konzistentni u okviru te transakcije kao na početku transakcije. U slučaju promena od strane drugih transakcija nakon početka trenutne transakcije, te promene nisu vidljive naredbama koje se izvršavaju u trenutnoj transakciji. *Snapshot* transakcije ne zahtevaju zaključavanje prilikom čitanja podataka i ne blokiraju druge transakcije prilikom upisivanja novih podataka. Takođe, transakcije koje upisuju nove podatke ne blokiraju *snapshot* transakcije pri čitanju podataka.

Može se desiti da dođe do zaključavanja podataka u slučaju *rollback-a.* Odnosno, transakcija koja vrši *rollback* može da izvrši zaključavanje podataka koje je modifikovala tokom izvršenja, ali ako i druge transakcije istovremeno pokušaju da čitaju te podatke biće blokirane. Nakon što se izvrši *rollback,* podaci će biti otključani i transakcije odblokirane. [8]

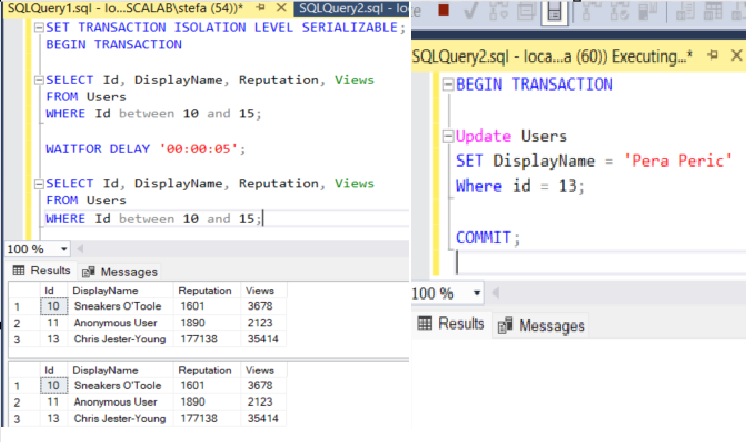
Pre početka transakcije koja koristi *Snapshot* izolacioni nivo, opcija *ALLOW\_SNAPSHOT\_ISOLATION* mora biti postavljena na *ON.*

Slika 4.5. Primer izvršenja čitanja kod *Snapshot izolacionog nivoa*

Na ovom primeru imamo dve transakcije koje se pokreću jedna za drugom. Na prvoj transakciji je postavljen S*napshot* izolacioni nivo i ona izvršava dve naredbe čitanja broja redova u tabeli *Posts* u intervalu od 5 sekundi, dok druga transakcija koja se pokreće odmah nakon prve izvršava upisivanje u tabelu *Posts* i čita broj redova u toj tabeli. Ono što možemo uočiti jeste da je prva transakcija u oba čitanja pročitala isti broj redova, iako je druga transakcija dodala još jedan red u toj tabeli. Na taj način smo dokazali gore navedenu funkcionalnost ovog izolacionog režima.

### 4.1.5. *Serializable*

Kod ovog nivoa izolacije, naredbe ne mogu čitati podatke koji su izmenjeni, ali još nisu potvrđeni od strane drugih transakcija. Takođe, nijedna druga transakcija ne može izmeniti podatke koje je pročitala trenutna transakcija dok se trenutna transakcija ne završi. Štaviše, druge transakcije ne mogu umetnuti nove redove sa ključevima koji bi upadali u opseg ključeva pročitanih od strane bilo koje naredbe u trenutnoj transakciji dok se ona ne završi.



Slika 4.6. Primer izvršenja čitanja kod *Serializable* izolacionog nivoa

Na ovom primeru je demonstrirano kako izolacioni nivo postavlja *range lock* na redovima čiji ključevi upadaju u opseg vrednosti koje odgovaraju kriterijumima za pretragu prve transakcije. Prva transakcija vrši čitanje korisnika sa identifikatorima od 10 do 15, dok druga transakcija pokušava da ažurira vrednost kolone *DisplayName* za korisnika sa identifikatorom 13. Druga transakcija se neće izvršiti sve dok se prva transakcija ne potvrdi.

Ovo predstavlja jedan od najrestriktivniji nivo izolacije jer zaključava cele opsege ključeva i drži ih zaključanim dok se transakcije ne izvrši. Na osnovu toga možemo zaključiti da je nivo konkurentnosti ovog izolacionog nivoa niži.

# Zaključavanje

U sistemima za upravljanje relacionim bazama podataka kontrola konkurentnosti je ključna za održavanje doslednosti i integriteta podataka. Mehanizam zaključavanja je ključni deo kojim SQL Server upravlja konkurentnim pristupom podacima. Kontrolišući pristup podacima, zaključavanja sprečavaju istovremeno ažuriranje podataka i osiguravaju da se transakcije izvršavaju na način koji poštuje ACID svojstva.

Ovaj mehanizam predstavlja suštinski deo izolacije i služi za zaključavanje objekata koji su pogođeni nekom od transakcija. SQL Server sprečava druge transakcije da naprave bilo kakve promene podataka u objektima koji se koriste od strane trenutne transakcije. Druge transakcije se dodaju u listu čekanja i nastavljaju svoje izvršenje kada se podaci oslobode od strane prethodne transakcije koja ih je koristila.

Nije tako često da aplikacije zahtevaju zaključavanja eksplicitno, već se zaključavanja specificiraju interno od strane SQL Query procesora. U zavisnosti od naredbe i izolacionog režima procesor zahteva određenu vrstu zaključavanja određenog resursa. Mehanizam zadužen za obavljanje zaključavanja naziva se *lock manager.*

Zaključavanja u SQL Server-u mogu se specificirati putem režima zaključavanja i granularnosti zaključavanja.

## 5.1. Hijerarhija i granularnost zaključavanja

SQL Server ima definisanu hijerarhiju zaključavanja koju koristi prilikom čitanja ili menjanja podataka. Hijerarhija zaključavanja je definisana tako da je na najvišem nivou baza podataka, a na najnižem nivou red. Hijerarhija zaključavanja se odnosi na strukturu koja definiše kako različiti nivoi zaključavanja komuniciraju ili se nasleđuju. Na primer, ako postoji zaključavanje na nivou tabele, to može uticati na niže nivoe poput stranica ili redova unutar te tabele.

Na osnovu zadatka koji SQL Server treba da obavi bira nivo granularnosti nad kojim će se izvršiti zaključavanje. Ako SQL Server treba da izvrši zaključavanje na nivou manje granularnosti, kao što je red, povećava se konkurentnost, ali ako je nivo granularnosti veći, kao što je tabela, onda je ograničenje veće i konkurentnost manja. [9]



Slika 5.2. Hijerarhija zaključavanja

Kod SQL Server uvek postoji *shared* zaključavanje na nivou baze podataka koje se postavlja kad god je transakcija povezana sa bazom, pri čemu se vrši protekcija da baza ne bude obrisana u međuvremenu. Npr, ako imamo SELECT naredbu u okviru transakcije, *shared* zaključavanje će biti postavljeno na nivou baze podataka, i*ntent shared* zaključavanje će biti postavljeno na nivou tabele i stranice, a *shared* zaključavanje na nivou samog reda.

## 5.2. Režimi zaključavanja

Na osnovu režima zaključavanja SQL Server određuje na koji način će konkurente transakcije pristupati resursima.

U SQL Server-u postoje 7 režima:

1. *Shared locks*
2. *Exclusive locks*
3. *Update locks*
4. *Intent locks*
5. *Schema locks*
6. *Bulk update locks*
7. *Key-range locks*

### 5.2.1. *Shared locks*

Ova vrsta zaključavanja, rezerviše stranicu ili red da bude dostupna samo za čitanje, što znači da će bilo kojoj drugoj transakciji biti onemogućeno menjanje zaključanog zapisa sve dok je zaključavanje aktivno. Takođe, nekoliko transakcija mogu izazvati zajedničko zaključavanje u isto vreme na istoj stranici ili redu i na taj način nekoliko transakcija može deliti mogućnost čitanja podataka, jer sam proces čitanja neće ni na koji način uticati na stvarne podatke stranice ili reda.

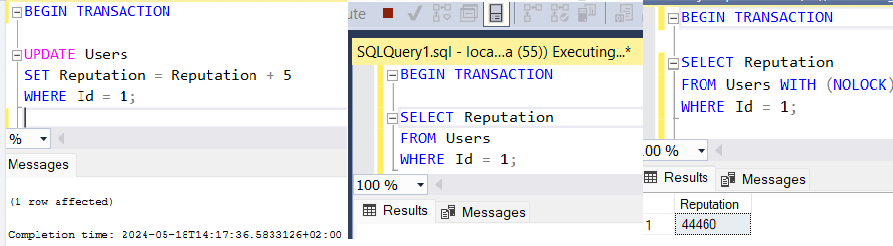
Zajednička zaključavanja na resursu se oslobađaju čim se operacaija čitanja završi, osim ako nivo izolacije transakcije nije postavljen na *Repetable Read* ili viši, ili ako se koriste *hintovi* kako bi se zadržalo zajedničko zaključavanje tokom trajanja transakcije. [9]

### 5.2.2. *Exclusive locks*

Kada je uključena ova vrsta zaključavanja, obezbeđuje da će stranica ili red biti rezervisani isključivo za transakciju koja je nametnula ekskluzivno zaključavanje, sve dok transakcija drži zaključavanje.

Ekskluzivno zaključavanje će biti aktivirano od strane transakcije kada želi da izmeni podatke stranice ili reda korišćenjem naredbi brisanja, dodavanja ili ažuriranja. Stranica ili red može da se ekskluzivno zaključa samo ako na nju već nije primenjeno ekskluzivno zaključavanje. To znači da možemo imati samo jednu vrstu zaključavanja na određenom resursu.

Naredbe čitanja u konkurentnim transakcijama mogu se jedino obavljati uz korišćenje *NOLOCK hint-a* ili kod transakcijama sa *Read Uncommitted* izolacionog nivoa.

Slika 5.1. Primer ekskluzivnog zaključavanja

Na primeru je demonstrirano ekskluzivno zaključavanje, prva transakcija postavlja ekskluzivno zaključavanje nad zapisom sa identifikatorom 1 u tabeli *Users.* Druga transakcija koja se podrazumevano izvršava u *Read Commited* izolacionom nivou ne može da pročita zapis zato što je prva postavila ekskluzivno zaključavanje. Dok treća transakcija može da pročita zbog korišćenja *hint-a* koji omogućava čitanje ne potrvrđenih promena, kao kod izolacionog nivoa *Read Uncommitted.*

### 5.2.3. *Update locks*

Ova vrsta zaključavanja se postavlja od strane SQL Server Engine-a kada se priprema ažuriranje podataka. Kod ove vrste zaključavanja samo jedna transakcija može držati zaključavanje nad datim resursom u isto vreme, za razliku od zajedničkog zaključavanja *(shared locks).* Kada procesor bude spreman da izmeni resurs, ova vrsta zaključavanja prelazi u eksluzivno zaključavanje. Ova vrsta zaključavanja je kompatibilna sa zajedničkim zaključavanjem.

Ako više transakcija drži *shared lock* i jedna od njih *update lock,* ta transakcija mora da čeka da se oslobode *shared lock* ostalih transakcija kako bi prešla u *exclusive lock.*

Problem kod ovakvog zaključavanja može se javiti prilikom transakcija sa nivoom izolacije *Repetable Read* i *Serializable.* Ako dve transakcije drže *shared lock* nad nekim podatkom i istovremeno obe hoće da izmene taj podataka, gde obe zahtevaju prelazak na *exclusive lock,* u tom slučaju doći će do *deadlock-a*.

Ova vrsta zaključavanja se takođe može nametnuti korišćenjem *UPDLOCK hint-a* u upitima.

### 5.2.4. *Intent locks*

SQL Server koristi ovu vrstu zaključavanja kako bi zaštitio postavljanje *shared locks ili exclusive locks* na resurs niži u hijerarhiji zaključavanja. Odnosno *intent locks* se postavlja na višem nivou (na primer, na nivou tabele) kako bi se osiguralo da ne dođe do *sukoba* na nižem nivou (na nivou stranice ili reda) dok se zaključavanje na višem nivou ne oslobodi. Na primer, *shared intent locks*  se zahteva na nivou tabele pre nego što se zatraži *shared locks* na stranicama ili redovima te iste table*.* Takođe, ova vrsta zaključavanja sprečava druge transakcije da modifikuju resurs višeg nivoa na način koji bi poništio zaključavanje na nižem nivou.

*Intent locks* poboljšava se efikasnost mehanizma baze podataka u detekciji sukoba zaključavanja na višem nivou granularnosti. Zato što u tom slučaju mehanizam baze podataka ispituje da li postoji *shared locks* samo na nivou tabele kako bi odredio da li transakcija može bezbedno da pribavi zaključavanja na toj tabeli, bez potrebe da proverava na nivou reda ili stranice.

Ova vrsta zaključavanja uključuje kombinaciju više zaključavanja:

|  |  |
| --- | --- |
| Režim zaključavanja | Opis |
| *Intent shared (IS)* | Kada je postavljneno *intent shared locks* (IS) zaključavanje, to signalizira SQL Server-u da transakcija planira da čita određene resurse niže u hijerarhiji zaključavanja korišćenjem pojedinačnih *shared locks* na nižim nivoima u hijerarhiji. |
| *Intent exclusive (IX)* | Kada je postavljeno *intent exclusive locks* (IX) zakjučavanje, to signalizira SQL Server-u da transakcija ima nameru da modifikuje neke resurse niže u hijerarhiji korišćenjem pojedinačnih *exclusive* locks (*X*) na nižim nivoima u hijerarhiji. |
| *Shared with intent exclusive (SIX)* | Kada je postavljeno s*hared with intent exclusive (SIX)*, to signalizira SQL Server-u da transakcija namerava da čita sve na nižem hijerarhijskom nivou i stoga pribavlja *shared locks* na svim resursima nižim u hijerarhiji, a zatim da modifikuje deo tih resursa, ali ne sve. U tom procesu, transakcija će pibaviti *IX* zaključavanje na onim resursima nižim u hijerarhiji koji treba da se modifikuju. U praksi, to znači da kada transakcija pribavi *SIX* zaključavanjena tabeli, pribaviće i *IX* zaključavanje na modifikovanim stranicama i *X* zaključvanje na modifikovanim redovima. |
| *Intent update (IU)* | *Intent update (IU)* može se pribaviti samo na nivou stranice, i čim se izvrši operacija ažuriranja, ono se konvertuje u *intent exclusive* (*IX*). |
| *Shared intent update (SIU)* | Ovo je nešto specifičnije zaključavanje jer predstavlja kombinaciju *S* i *IU* zaključavanja. Tipičan primer ovog zaključavanja je kada transakcija koristi upit izvršen sa *hint-om PAGELOCK* i upit za ažuriranje. Nakon što transakcija pribavi *SIU* zaključavanje na tabeli, upit sa *hint-om PAGELOCK* će pribaviti zajedničko (*S*) zaključavanje, dok će upit za ažuriranje pribaviti *IU* zaključavanje. |
| *Update intent exclusive (UIX)* | Kada se istovremeno pribave *U* zaključavanje i zaključavanje *IX* na resursima niže u hijerarhiji tabele, *UIX zaključavanje* će biti postavljeno na nivou tabele kao posledica. |

Tabela 5.1. *Intent locks* kombinacije zaključavanja. [9]

### 5.2.5. *Schema locks*

Kod SQL Server-a postoje dva tipa zaključavanja šeme:

* Schema modification lock (Sch-M)
* Schema stability lock (Sch-S)

Schema modification lock (Sch-M) - kao što naziv zaključavanja kaže, ova vrsta zaključavanja se koristi prilikom modifikacije strukture baze podataka. Sch-M zaključavanje biće postavljeno kada se izvrši DDL naredba, i sprečiće pristup podacima zaljučanog objekta jer se struktura objekta menja. SQL Server dozvoljava samo jedno Sch-M zaključavanje nad jednim objektom istovremeno. Da bi se izmenila određena tabela, transakcija mora da sačeka pribavljanje Sch-M zaključavanje, i tek nakon toga može da se izvrši modifikacija tabele. Nakon završetka modifikacije oslobađa se ova vrsta zaključavanja. Na ovaj način se sprečavaju sve operacije nad tom tabelom dok se modifikacija ne izvrši.

Schema stability lock (Sch-S) - zaključavanje će biti postavljeno dok se kompajlira i izvršava upit koji zavisi od šeme. Ova vrsta zaključavanja neće blokirati druge transakcije da pristupe podacima objekata i kompatiblina je sa svim režimima zaključavanja osim sa zaključavanjem za modifikaciju šeme Sch-M. Ova vrsta zaključavanja biće postavljena od strane svake DML i SELECT naredbe kako bi se osigurao integritet strukture tabele, odnosno da se tabele ne menjaju u toku izvršenja upita.

### 5.2.6. *Bulk update locks*

Bulk update zaključavanje je namenjeno za uptorebu pri operacijama masovnog uvoza podataka kada se koristi BULK INSERT naredba ili korišćenjem hint-a TABLOCK. Kada se postavi ova vrsta zaključavanja, druge transackije neće moći pristupati tabeli tokom izvršavanja masovnog uvoza podataka. Ali ova vrsta zaključavanja ne sprečava paralelno izvršavanje masovnog učitavanja od strane više transakcija, osim u slučaju postojanja klasterizovanog indeksa.

### 5.2.7*. Key-range locks*

Ova vrsta zaključavanja se koristi kod Serializable izolacionog nivoa. Vrše protekciju redova koji su implicitno uključeni u skup zapisa koji se čitaju od strane jedne transakcije. Nivo izolacije zahteva da svaki upit izvršen tokom transakcije mora da dobije isti skup redova svaki put kada se ponovi upit u okviru trsansakcije, odnosno da se ne javi pojava phantom read. Zaključavanje vrši protekciju kako ne bi došlo do umetanja novih redova koji bi upali u kriterijum za čitanje koji je nametnula prva transakcija sa Serializable izolacionim nivoom.

Key-range zaključavanje se postavlja na indeksu, navodeći početnu i završnu vrednost ključa. Ovo zaključavanje nakon toga blokira svaki pokušaj dodavanja, ažuriranja ili brisanja bilog kog reda sa vrednošću ključa koji upada u taj opseg. [9]

## 5.3. Kompatibilnost režima zaključavanja

Kompatibilnost zaključavanja određuje da li više transakcija može istovremeno da zaključa isti resurs. Ako je resurs već zaključan, novi zahtev za zaključavanje može biti odobren samo ako je kompatibilan sa postojećim zaključavanjem. Na primer, *exclusive* zaključavanje ne dozvoljava drugim transakcijama da zaključavaju taj resurs dok se *exclusive* zaključavanje ne oslobodi. S druge strane, *shared* zaključavanje dozvoljava drugim transakcijama da istovremeno postave *shared* zaključavanje nad istim resursom.

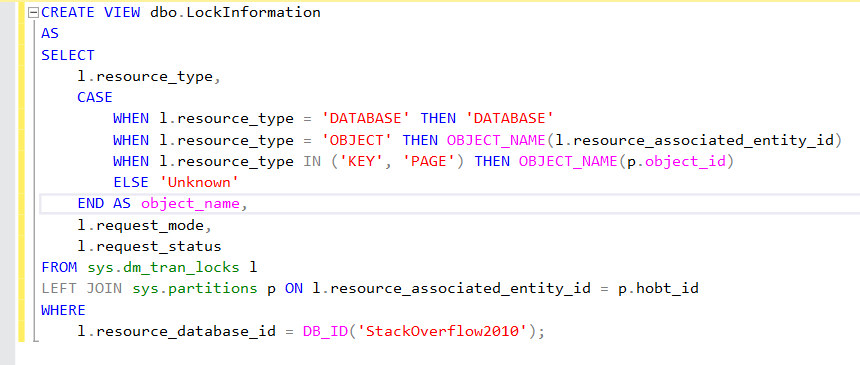
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Postojeći dodeljeni način | IS | S | U | IX | SIX | X |
| Zahtevani način |  |  |  |  |  |  |
| *Intent shared (IS)* | Da | Da | Da | Da | Da | Ne |
| *Shared (S)* | Da | Da | Da | Ne | Ne | Ne |
| *Update (U)* | Da | Da | Ne | Ne | Ne | Ne |
| *Intent exclusive (IX)* | Da | Ne | Ne | Da | Ne | Ne |
| *Shared with intent exclusive (SIX)* | Da | Ne | Ne | Ne | Ne | Ne |
| *Exclusive (X)* | Ne | Ne | Ne | Ne | Ne | Ne |

Tabela 5.2. Tablica kompatibilnosti između različitih režima zaključavanja [9]

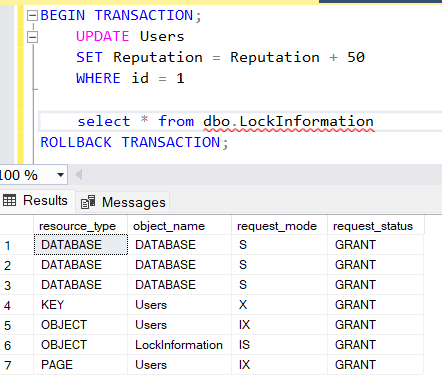
## 5.4. Eskalacija zaključavanja

SQL Server je uveo funkciju eskalacije zaključavanja kako bi sprečio situaciju u kojoj zaključavanje koristi previše resursa. Eskalacija zaključavanja predstavlja proces u kojem se manja zaključavanja, poput zaključavanja na redovima, prevode na zaključavanja većih granularnosti kao što su zaključavanja na nivou tabele. SQL Server mehanizam automatski odlučuje kada je potrebno izvršiti eskalaciju. Prilikom odlučivanja SQL Server uzima u obzir broj aktivnih zaključavanja, ukupni broj zaključavanja u transakciji i količinu memorije koja se koristi za ta zaključavanja. Obično se eskalacija pokreće kada može dovesti do poboljšanja performansi ili kada je potrebno smanjiti prekomernu upotrebu memorije za zaključavanje.[10]

Zaključavanja u SQL Server-u mogu biti prilično skupa. Za svako zaključavanje koje *Lock Manager* izvrši, SQL Server mora rezervisati određenu količinu memorije. Pa kako broj zaključavanja raste više memorije se zauzima.

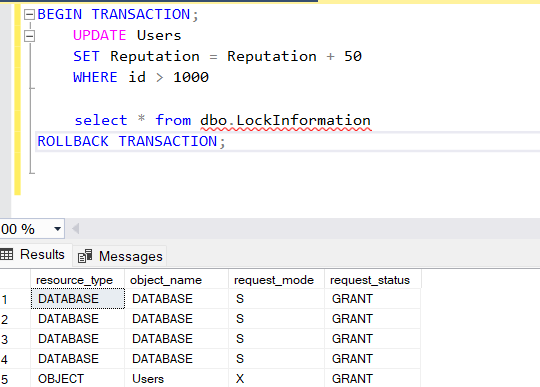
Slika 5.2. Pogled za očitavanje primenjenih zaključavanja

U okviru ovog poglavlja, koristićemo ovaj pogled za očitavanje primenjenih zaključavanja nad bazom podataka. Pogled prikazuje granularnost zaključavanja, tip zaključavanja, režim zaključavanja i status.



Slika 5.3. Prikaz primenjenih zaključavanja kod ažuriranja jednog reda

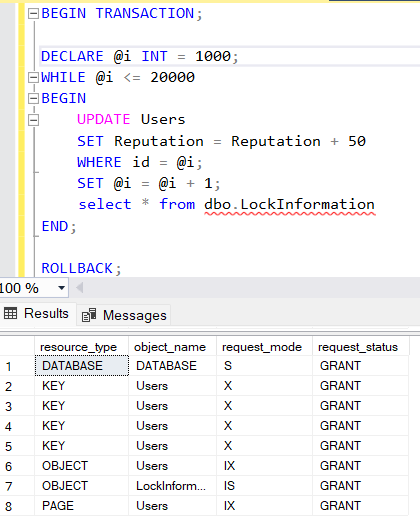
Na ovom primeru imamo ažuriranje jedne vrste u okviru *Users* tabele i prikazivanje primenjenih zaključavanja. Možemo uočiti da podrazumevano imamo *shared* zaključavanja nad bazom podataka koja su nametnuta od strane Management Studio-a prilikom otvaranja sesije. Zaključavanje PAGE i KEY sa *IX* režimom zaključavanja nad tabelom *Users* je povezano sa naredbom UPDATE i ukazuje da se vrši zaključavanje na nižem nivou granularnosti. Zatim, primećujemo zaključavanje tipa KEY na tabeli *Users*, gde se koristi eksplicitno zaključavanje *X* sa dozvolom (GRANT). Pri čemu ovo zaključavanje odgovara naredbi ažuriranja odgovarajućeg reda. Ovde vidimo da nije bilo potrebno pokretanje eskalacije.



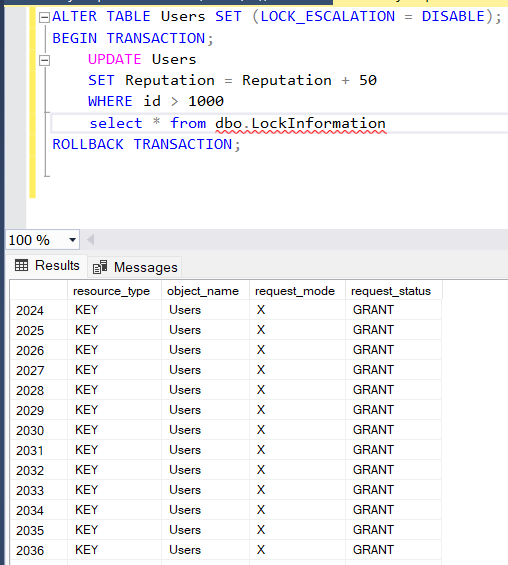
Slika 5.4. Primer aktiviranje eskalacije zaključavanja

Ovaj primer demonstrira kako je nad tabelom *Users* primenjeno ekskluzivno zaključavanje. Za razliku od prethodnog primera na ovom primeru imamo ažuriranje nekoliko desetina hiljada redova, što je uzrokovalo pokretanju eskalacije zaključavanja od strane SQL Servera. Tj. umesto da imamo za svaki red po jedno zaključavanje na nivou reda, SQL Server je podigao nivo granularnosti na nivo cele tabele.

Eskalaciju zaključavanja možemo izbeći ako bi izvršenje prethodnog primera izvršili na način dat na slici 5.5. Korišćenjem WHILE petlje podelili smo izvršenje naredbe UPDATE, i na taj način izbegli aktiviranje eskalacije.



Slika 5.5. Primer bez aktiviranje eskalacije zaključavanja



Slika 5.6. Primer deaktivacije eskalacije zaključavanja nad tabelom

Na primeru sa slike 5.6. je prikazano kako možemo deaktivirati opciju eskalacije i na taj način postići isti efekat kao na primeru sa slike 5.5.

Eskalacija zaključavanja u SQL Server-u ima svoje prednosti i mane. Prednosti uključuju efikasnije korišćenje memorije, poboljšanje performansi i jednostavnije upravljanje zaključavanjima. Dok sa druge strane, mane uključuju smanjenu konkurentnost, potencijalno povećanje vremena čekanja i rizik od smanjenja performansi u specifičnim scenarijima.

### 5.4.1. Uslovi za pokretanje eskalacije

Eskalacija zaključavanja je mehanizam kojim SQL Server pokušava da optimizuje performanse i efikasno koristi memoriju. Ona se pokreće kada se dostigne određeni prag memorije ili broj zaključavanja, sa ciljem smanjenja opterećenja sistema.

Memorija za zaključavanje je ograničena na 60 procenata vidljivog bafera. SQL Server može koristiti najviše 60% ukupne bafer memorije za zaključavanja. Ako je eskalacija zaključavanja onemogućena i memorija za zaključavanja premaši 60% ukupne bafer memorije, svi pokušaji da se dodeli dodatna zaključavanja će propasti i biće generisane greške. Prag za pokretanje eskalacije zaključavanja je postavljen na 40 procenata memorije za zaključavanje. Dakle, eskalacija će se desiti kada 40% od tih 60% bafer memorije bude korišćeno za zaključavanja.

Nakon što se odradi provera da li je dostignut prag memorije, SQL Server proverava koliko zaključavanja postoji na trenutnoj tabeli ili indeksu. Ako broj prelazi 5000 redova , doći će do pokretanja eskalacije zaključavanja. [10]

# Zaključak

U ovom radu detaljno smo istražili ključne aspekte transakcija u SQL Serveru, uključujući njihove režime, komande za upravljanje, planove izvršenja, izolaciju i zaključavanje. Razmatrali smo kako različiti režimi transakcija poput *autocommit*, implicitne, eksplicitne i *batch-scoped,* omogućavaju kontrolu nad načinom izvršavanja i potvrđivanja transakcija.

Posebnu pažnju smo posvetili izolacionim nivoima, od *Read Uncommitted* do *Serializable*, objašnjavajući kako svaki nivo utiče na konzistentnost i performanse baza podataka. Ovi nivoi izolacije pružaju različite nivoe zaštite od anomalija, omogućavajući balans između tačnosti podataka i efikasnosti sistema.

Takođe smo se bavili različitim tipovima zaključavanja, kao što su *shared*, *exclusive*, *update* i *intent locks*, naglašavajući njihovu ulogu u sprečavanju konflikata između konkurentnih transakcija. Razumevanje hijerarhije i granularnosti zaključavanja je ključno za optimizaciju performansi baza podataka, kao i za minimiziranje zastoja.

Primene ovih koncepata demonstrirane su na StackOverflow2010 ([11]) bazi, što je omogućilo praktičnu demonstraciju teorijskih aspekata. Kroz ove primere, pokazali smo kako se različiti režimi i tehnike mogu koristiti u praksi da bi se poboljšala efikasnost i pouzdanost SQL Server sistema.

Na kraju, kroz analizu kompatibilnosti režima zaključavanja i eskalacije zaključavanja, istakli smo značaj pažljivog dizajniranja transakcionih sistema kako bi se postigla optimalna kombinacija integriteta podataka i performansi.

Sve ove komponente zajedno čine kompleksan mehanizam, ali neophodan za efikasno upravljanje transakcijama u SQL Serveru. Ovaj rad pruža pregled tih komponenti i može poslužiti kao osnova za dalje istraživanje i implementaciju naprednih tehnika u radu sa bazama podataka.

# Literatura

[1]<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql> [Poslednji put posećeno 3.5.2024.]

[2]<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/begin-transaction-transact-sql> [Poslednji put posećeno 6.5.2024.]

[3]<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/commit-transaction-transact-sql> [Poslednji put posećeno 6.5.2024.]

[4]<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/rollback-transaction-transact-sql> [Poslednji put posećeno 6.5.2024.]

[5]<https://learn.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql> [Poslednji put posećeno 7.5.2024.]

[6]<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/save-transaction-transact-sql> [Poslednji put posećeno 7.5.2024.]

[7]<https://www.sqlshack.com/execution-plans-in-sql-server/> [Poslednji put posećeno 10.5.2024.]

[8]<https://learn.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql> [Poslednji put posećeno 12.05.2024.]

[9]<https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver16> [Poslednji put posećeno 18.05.2024.]

[10]<https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/performance/resolve-blocking-problems-caused-lock-escalation> [Poslednji put posećeno 23.05.2024.]

[11]<https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/> [Poslednji put posećeno 20.05.2024]