Logo

Description automatically generatedLogo, company name

Description automatically generatedUNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

KATEDRA ZA RAČUNARSTVO

**Sistemi za upravljanje bazama podataka – Seminarski rad**

**Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje kod MS SQL Server-a**

Student:

Nemanja Raković, 590

Profesor:

Doc. dr Aleksandar Stanimirović

Sadržaj

[Sadržaj 2](#_Toc105489805)

[1. Uvod 4](#_Toc105489806)

[2. Microsoft SQL Server 5](#_Toc105489807)

[3. Transakcije 6](#_Toc105489808)

[3.1 ACID svojstva transakcija 6](#_Toc105489809)

[3.1.1 Atomičnost 6](#_Toc105489810)

[3.1.2 Konzistentnost 6](#_Toc105489811)

[3.1.3 Izolacija 7](#_Toc105489812)

[3.1.4 Trajnost 7](#_Toc105489813)

[3.2 Osnove rada sa transakcija u MS SQL Serveru 7](#_Toc105489814)

[3.2.1 Eksplicitne transakcije 7](#_Toc105489815)

[3.2.2 Autocommit transakcije 9](#_Toc105489816)

[3.2.3 Implicitne transakcije 10](#_Toc105489817)

[3.2.4 Transakcije na nivou paketa 10](#_Toc105489818)

[3.2.5 Distribuirane transakcije 10](#_Toc105489819)

[3.2.5 Obrada grešaka 10](#_Toc105489820)

[3.3 Raspoređivanje transakcija 11](#_Toc105489821)

[3.4 Anomalije izazvane preplitanjem 12](#_Toc105489822)

[3.4.1 Izgubljeno ažuriranje (WW) 12](#_Toc105489823)

[3.4.2 Prljavo čitanje (WR) 12](#_Toc105489824)

[3.4.3 Neponovljivo čitanje (RW) 12](#_Toc105489825)

[3.4.4 Fantomsko čitanje (RW) 13](#_Toc105489826)

[3.4.5 Promašeno i duplirano čitanje (WR) 13](#_Toc105489827)

[3.5 Mehanizmi kontrole konkurentnosti u MS SQL Serveru 13](#_Toc105489828)

[3.5.1 Pesimistična kontrola konkuretnosti 14](#_Toc105489829)

[3.5.2 Optimistična kontrola konkuretnosti 14](#_Toc105489830)

[3.6 Nivoi izolacije 14](#_Toc105489831)

[4. Zaključavanje 17](#_Toc105489832)

[4.1 Naredbe 17](#_Toc105489833)

[4.2 Kategorije brava 18](#_Toc105489834)

[4.2.1 Exclusive (X) 18](#_Toc105489835)

[4.2.2 Shared (S) 18](#_Toc105489836)

[4.2.3 Update (U) 18](#_Toc105489837)

[4.2.4 Intent (I\*) 19](#_Toc105489838)

[4.2.5 Range 19](#_Toc105489839)

[4.2.5 Schema 20](#_Toc105489840)

[4.3 Kompatibilnost brava 20](#_Toc105489841)

[4.4 Eskalacija brava 22](#_Toc105489842)

[5. Deadlock 23](#_Toc105489843)

[5.1 Klasično uzajamno blokiranje 23](#_Toc105489844)

[5.2. Izvori zastoja u SQL Serveru 24](#_Toc105489845)

[5.2.1 Neoptimalno skeniranje 24](#_Toc105489846)

[5.2.2 Istovremeno ažuriranje istog slog koriščnjem pretrage po ključu 25](#_Toc105489847)

[5.2.2 Istovremeno ažuriranje istog slog koriščnjem pretrage po ključu 25](#_Toc105489848)

[5.3 Rešavanje zastoja 26](#_Toc105489849)

[7. Literatura 27](#_Toc105489850)

1. Uvod

U današnje doba je prepoznat značaj podataka kao potencijalno najvrednijeg resursa neke organizacije. S obzirom da se količina informacija koja je dostupna eksponencijalno uvećava, potrebno je imati alate koji mogu da na efikasni i korisniku efikasni način upravljaju masivnim, kompleksnim skupovima podataka i iz njih izvuku korisne informacije. U suprotnom, ti isti podaci mogu postati smetnja, s obzirom da cena njihovog skladištenja i održavanja može nadmašiti vrednost koju pružaju.

Gotovo sve organizacije, od lokalnih prodavnica do multinacionalnih korporacija i vlada, poseduju informacioni sistem, u čijoj srži se nalaze baze podataka, i sistemi za njihovo upravljanje. Baze podataka predstavljaju kolekciju informacija vezanih za određeni subjekat, namenu ili pojavu unutar jedne ili više povezanih organizacija. Sistem za upravljanje bazama podataka (*DBMS*) je softver namenjen da pomogne u održavanju i korišćenju velikih količina podataka.

DBMS ima nekoliko ključnih prednosti u odnosu na naivni pristup čuvanja podataka u datotekama i pisanja specifičnog aplikacionog koda za njihovo korišćenje. DBMS garantuje apstraktni pogled na podatke koji je nezavisan od stvarnog mehanizma skladištenja, sofisticirane algoritme i strukture za efikasno skladištenje i pristup podataka, integritet smeštenih podataka, kontrolu pristupa, korisničke interfejse za administraciju, konkurentni pristup bazi, konzistentni oporavak od pada sistema i funkcije koje su zajedničke velikom broju aplikacija koje koriste bazu te time omogućava brži razvoj klijentskog softvera.

Kako bi DBMS ispunio gorepomenute uslove, mora da poseduje mehanizme za konkurentnu obradu višekorisničkih zahteva. Svaki korisnički zahtev koji se treba izvršiti, DBMS interpretira kao jedan niz operacija (upis i čitanje), odnosno kao jednu transakciju. Transakcija je osnovni element konkurentnog izvršenja i oporavka kod sistema za upravljanje bazama podataka. Da bi DBMS na najefikasniji način iskoristio procesorske resurse, transakcije se ne mogu i ne smeju izvršavati jedna po jedna, tj. sekvencijalno, već mora doći do izvesnog preplitanja (*interlacing*) između operacija koje pripadaju različitim transakcijama. Kod ovakvog preplitanja treba voditi računa da krajnji rezultat bude ekvivalentan po bazu, odnosno treba postići efekat kao kad bi se date transakcije izvršavale sekvencijalno.

Jedna od osnovnih tehnika kojom se DBMS-ovi koriste u rešavanju problema performansi su indeksi. Indeksne strukture se prvenstveno koriste da se smanji broj pristupa sporijoj memoriji (disku, u nastavku *I/O pristupi/operacije*). Koriste se prilikom evaluacije i optimizacije upita i kod samog pribavljanja podataka.

U nastavku rada biće prikazano kako Microsoft SQL Server radi vrši obradu i planiranje transakcija, kao i koje se tehnike izolacije i zaključavanja primenjuju da bi se taj rad omogućio.

2. Microsoft SQL Server

SQL Server je sistem za upravljanje relacionim bazama podataka koji razvija i održava kompanija Microsoft. Jedno je od popularnijih rešenja dostupnih na današnjem tržištu iz ponude relacionih sistema. Primarna funkcija mu je skladištenje i pribavljanje podataka koje koriste druge aplikacije, kako na istoj mašini tako i preko mreže. Komunikacija se obavlja preko Tabular Data Stream (TDS) protokola, čiji se paketi dalje mogu slati raznovrsnim protokolima kao što su TCP/IP, poruke, web servisi, itd.

Microsoft koristi sopstvenu implementaciju glavne celine ANSI SQL 92 standarda pod nazivom Transact-SQL (T-SQL). Takođe ima mnoštvo proširenja specifičnih za SQL Server, uglavnom vezanih za funkcije i proceduralne strukture.

Microsoft u ponudi ima gomilu edicija sa različitim skupovima funkcionalnosti. Pored standardnog interfejsa ka bazi podataka, u ponudi usluga se takođe mogu naći servisi za broker poruka, servisi za mašinsko učenje, replikaciju, generisanje izveštaja (poslovna inteligencija), obaveštenja, integracije sa sistemima za masovno skladištenje, tekstualnu pretragu…

Takođe dolazi u paru sa grafičkim alatom SQL Server Management Studio (SSMS) koji omogućava konfigurisanje, administraciju i upravljanjem svim komponentama SQL Servera kroz grafički i skriptni korisnički interfejs. Postoji i integracija sa Visual Studiom IDE-om. Azure SQL Server je PaaS u sklopu Azure platforme za računarstvo u oblaku, koja je uglavnom kompatibilna sa on-premise verzijom kojoj se u ovom radu posvećuje pažnja.

Verzija sistema koja se koristi u ovom radu je Microsoft SQL Server 2019 Developer Edition.

3. Transakcije

Koncept **transakcije** se nalazi u osnovi konkuretnog izvršenja i oporavka od pada sistema kod sistema za upravljanje bazama podataka i definiše se kao bilo koje pojedinačno izvršenje nekog korisničkog programa unutar sistema. Transakcija može čitati ili upisivati objekte unutar baze podataka. Pored toga, transakcija mora specificirati svoju konačnu naredbu kao **potvrdu** (*commit*) ili kao **poništenje** (*rollback*) svih naredba izvršenih do tada.

3.1 ACID svojstva transakcija

ACID je akronim nastao od 4 svojstva koja svaka transakcija mora da sledi:

* Atomičnost (**A**tomicity)
* Konzistentnost (**C**onsistency)
* Izolacija (**I**solation)
* Trajnost (**D**urability)

3.1.1 Atomičnost

**Atomičnost** (atomicity) je garancija da će se sve transakcije izvršiti kao jedna celina, po „sve ili ništa“ principu. Sve promene izvršene u transakciji će biti primenjene ili poništene u celini. Najbolji primer atomičnosti je promena dva povezana kvantiteta u bazi, kao što su npr. štedni i tekući račun u banci. U transferu novca sa tekućeg na štedni račun je potrebno izvršiti dve operacije: smanjenje stanja na tekućem računa i povećanje stanja na štednom računu za isti iznos.

Atomičnost transakcije garantuje da će ili obe operacije biti uspešno izvršene ili da će obe biti poništene. Time se sistem ne ostavlja u nekonzistentom stanju gde je stanje izmenjeno samo na jednom računu. Ovo je zadatak sistema za oporavak. Mogući razlozi za obustavljanje transakcije mogu biti pad sistema, nailazak na neočekivani problem ili direktna naredba od strane korisnika.

3.1.2 Konzistentnost

**Konzistentnost** (consistency) je osiguranje da će baza podataka ostati u doslednom stanju nakon završetka transakcije, bez obzira na rezultat transakcija (uspeh ili neuspeh), odnosno zahtev da uspešno izvršena transakcija prevodi bazu podataka iz jednog konzistentnog stanja u drugo. Postoje granice do kojih sistem za upravljanje bazom podataka može garantovati konzistentnost. Sistem ne može „razumeti“ načine na koje podaci mogu postati nekonzistentni osim onih koji su mu zadati samom šemom.

U gorepomenutom primeru transfera između bankovnih računa se implicitno podrazuvema očuvanje ukupnog iznosa novca između računa prilikom transfera. Ukoliko korisnički program, na primer, poveća iznost na štednom računu za više nego što je tekući smanjen, sistem neće uočiti ovu grešku i obustaviti transakciju. Kontrola konzistentnosti sa semantičkog aspekta je krajnjoj liniji odgovornost korisnika.

3.1.3 Izolacija

**Izolacija** (*isolation*) osigurava da su promene izvršene u transakciji nevidljive ostalim transakcijama sve dok transakcija ne bude uspešno završena. Svojstvom izolacije se takođe može postići postići efekat nalik da se svaka transakcija izvršava nezavisno od ostalih, iako se odvija preplitanje operacija iz više transakcija tj. transakcije se konkretno izvršavaju. Ukoliko imamo dve transakcije T1 i T2 koje se konkurentno izvršavaju, konačni efekat mora da bude takav kao da su se T1 i T2 sekvencijalno izvršile u nekom konkretnom redosledu (T1 → T2 ili T2 → T1).

Izolacija je zahtev koji se najčešće ne primenjuje u potpunosti po definiciji iznad, već se kontroliše **nivoima izolacije transakcije** (*transaction isolation level*).

3.1.4 Trajnost

**Trajnost** (*durability*) je garancija da su sve promene izvršene transakcijom trajne. Po završetku transakcije, sistem za upravljanje bazom podataka kroz svoje mehanizme oporavka osigurava da se promene iz transakcije neće izgubiti. SQL Server postiže trajnost korišćenjem ***write-ahead*****transakciononog loga** na disku. Transakcija se ne smetra završenom dok se svi logovi promena generisanim transakcijom ne upišu u log datoteteku smeštenu na trajnom skladištu. Ako ova operacija iz nekog razloga ne uspe, transakcija se poništava.

3.2 Osnove rada sa transakcija u MS SQL Serveru

Korisnički programi uglavnom kontrolišu transakcije izdavanjem naredbi za početak i kraj transakcija korišćenjem API funkcija ili T-SQL naredbama. Sistem takođe mora biti sposoban da korektno obradi grešeke koje mogu prouzrokovati prekid transakcije pre završetka. Transakcijama se podrazumevano upravlja na nivou konekcije ka bazi.

3.2.1 Eksplicitne transakcije

Eksplicitna transakcija je ona u kojoj se eksplicitno definišu početak i kraj transakcije korišćenjem:

* @@TRANCOUNT – Funkcija koja vraća broj trenutno aktivnih transakcija.
* BEGIN { TRAN | TRANSACTION } – Naredba kojom se eksplicitno započinje lokalna transakcija. Moguće je dodeliti transakciji identifikator ili promenljivu kao i opis. Inkrementira @@TRANCOUNT za 1.
* BEGIN DISTRIBUTED TRANSACTION – Isti sintaksni detalji kao za regularnu transakciju. Kreira se distribuirana transakcija kojom upravlja Microsoft Distributed Transaction Coordinator (MS DTC).
* SAVE { TRAN | TRANSACTION } – Kreira marker unutar transakcije koja definiše lokaciju na koju transakcija može da se povrati u slučaju uslovnog prekida. Transakcija je tada i dalje aktivna i mora biti završena potvrdom ili poništavanjem. Imena markera se mogu ponavljati, u tom slučaju se prilikom ROLLBACK naredbe transakcija vraća na poslednji definisan marker sa tim imenom.
* COMMIT { TRAN | TRANSACTION } – Naredba kojom se označava kraj implicitne ili eksplicitne transakcije. Dekrementira @@TRANCOUNT za 1. Nakon ove naredbe se za tu transakciju više ne može izvršiti ROLLBACK naredba.
* COMMIT [ WORK ] – Identična naredbi iznad, sa izuzetkom da ne prihvata dodatne parametre, kao npr. ime.
* ROLLBACK { TRAN | TRANSACTION } – Naredba kojom se obustavlja transakcija i poništavaju promene. Promene ugnježdenih transakcija se poništavaju čak i u slučaju da su one potvrđene. Može se navesti ime transakcije ali to mora biti krajnja spoljna transakcija. Postavlja @@TRANCOUNT na 0. Za slučaj da se navede ime save markera, @@TRANCOUNT se ne menja.
* ROLLBACK [ WORK ] - Identična naredbi iznad, sa izuzetkom da ne prihvata dodatne parametre, kao npr. ime.

Primer upotrebe osnovnih naredbi:

begin transaction T1

delete from dbo.Customers

where CustomerId = 11

save tran savepoint

update dbo.Customers

set LastName = 'Codd'

where CustomerId = 12

commit transaction T1

Primer funkcionisanja @@TRANCOUNT funkcije:

print @@trancount

begin tran

print @@trancount

begin tran

print @@trancount

rollback

print @@trancount

-- Rezultat izvršenja

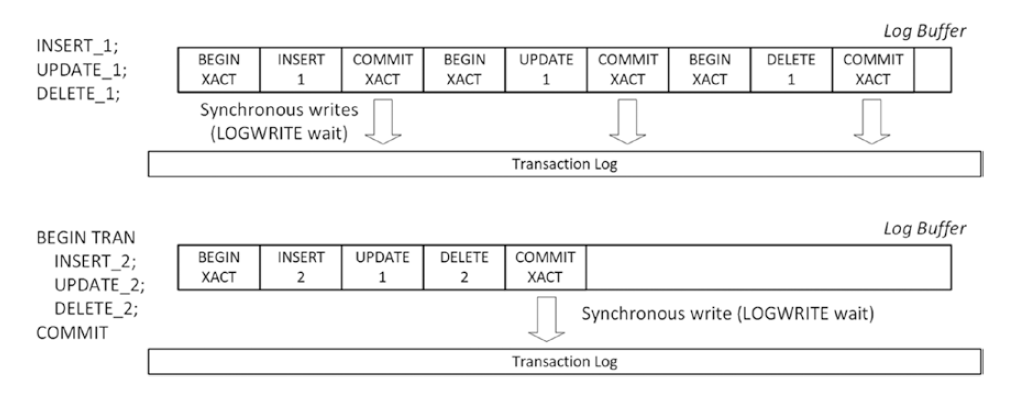
-- 0

-- 1

-- 2

-- 0

Unutar eksplicitnih transakcija se mogu koristiti gotovo sve T-SQL naredbe osim onih vezanih za kreiranje i izmenu same baze, *backup/restore* naredbi i naredbi vezanih za *full-text* operacije.



Slika 1 - Poređenje upisa u log za autocommit i eksplicitne transakcije

3.2.2 Autocommit transakcije

Podrazumevani mod rada sa transakcijama je *autocommit* mod upravljanja, pri kome se za svaku pojedinačnu T-SQL naredbu vrši *commit* ako je uspešno izvršena ili *rollback* ako je došlo do greške. Ako drugačije nije naglašeno, konekcije se otvaraju sa ovim podešavanjem.

3.2.3 Implicitne transakcije

Ako je konekcija podešena na mod implicitnih transakcija, SQL Server će automatski startovati novu transakciju kada se trenutna transakcija okonča. Ovim se generiše konituirani lanac transakcija. Korisnik ne čini ništa direktno da pokrene transakciju, već se prva transakcija za konekciju automatski startuje izvršavanjem neke od sledećih naredbi: ALTER TABLE, FETCH, REVOKE, CREATE, GRANT, SELECT, DELETE, INSERT, TRUNCATE TABLE, DROP, OPEN, UPDATE.

3.2.4 Transakcije na nivou paketa

Ako je omogućeno podešavanje za MARS (*multiple active results set*) na nivou konekcije, implicitne transakcije automatski postaju transakcije na nivou paketa. Transakcije koje nisu završene u trenutku kada se paket naredbi završi se automatski poništavaju od strane SQL Servera.

3.2.5 Distribuirane transakcije

Distribuirane transakcije obuhvataju dva ili više servera koji se nazivaju menadžeri resursa. Upravljanje distirbuiranom transakcijom mora biti koordinisano između menadžera resursa. Svaka instanca SQL servera može funkcionisati kao menadžer resursa, koristeći serversku komponentu koja se zove menadžer transakcija (MS DTC).

Transakcija koja se izvršava unutar jednog servera ali obuhvata više od jedne baze podataka se takođe interno tretira kao distribuirana transakcija, iako se korisniku prikazuje kao lokalna transakcija. Distribuirane transakcije se koordinišu korišćenjem standardnog *two-phase commit* protokola (2PC).

3.2.5 Obrada grešaka

U slučaju bilo kakve greške koja sprečava kompletiranje transakcije, SQL Server će automatski poništiti transakciju i osloboditi sve njome zauzete resurse. Ako se klijent izloguje sa servera, transakcija se automatski poništava. Isto važi i za nenamerno prekidanje konekcije (greška u mreži, itd..) sa tim što se prvo čeka obaveštenje od strane mreže da je konekcija prekinuta.

U slučaju greške u vremenu izvršavanja (npr. kršenje ograničenja poput stranog ključa), podrazumevano podešavanje je da se poništi samo ta naredba. Korišćenjem SET XACT\_ABORT ON naredbe se omogućava da bilo koja greška automatski poništi celu transakciju. Takođe je moguće granularnije kontrolisati efekat grešaka korišćenjem standardnih TRY CATCH THROW konstrukcija u T-SQL jeziku.

3.3 Raspoređivanje transakcija

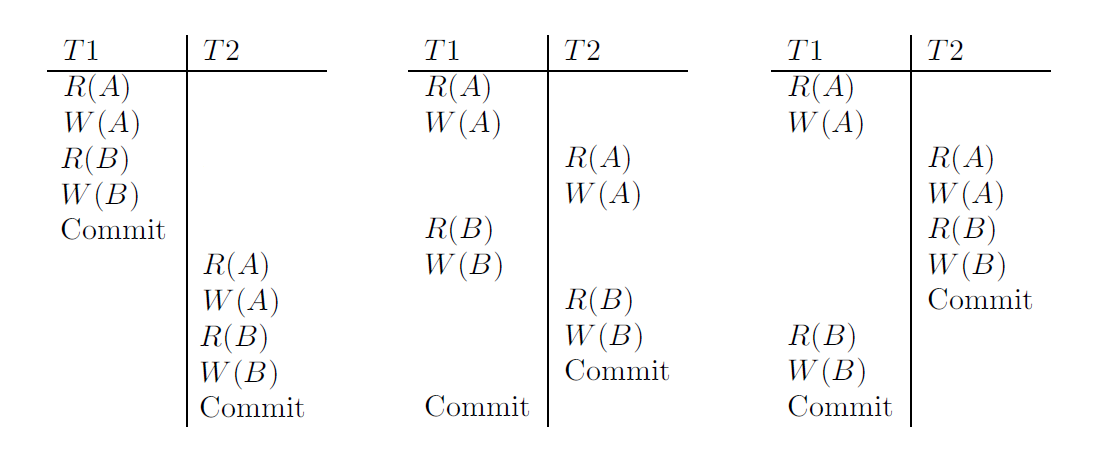
Sistem koji se oslanja na transakcije podrazumeva sledeća bitna ograničenja:

* Transakcije međusobno interaguju isključivo putem čitanja/upisa objekata u bazi podataka, odnosno ne smeju razmenjivati poruke.
* Baza podataka je fiksni skup nezavisnih objekata.

Budući da drugo ograničenje nije primenjivo u realističnom sistemu, potrebno je olakšati ga kao što je to pomenuto kod nivoa izolacije transakcije.

**Raspored** (*schedule*) predstavlja listu naredbi (čitanje, upis, potvrda, poništenje) iz skupa transakcija pod uslovom da se sve naredbe iz iste transakcije moraju u rasporedu naći u istom redosledu kao u samoj transakciji. Redosled predstavlja potencijalnu sekvencu izvršavanja više transakcija. Ako se naredbe unutar rasporeda ne prepliću, takav raspored se naziva *serijski*. Mnogo češći slučaj je da je da postoji izvesno preplitanje naredbi različitih transakcija, u tom slučaju se kao *serijabilni* (*serializable*) raspored smatra onaj pri kome je rezultat izvršenja više isprepletenih transakcija ekvivalentan određenom sekvencijalnom kompletnom izvršenju svake od njih. Ova definicija ne pokriva slučaj obustavljenih transakcija. Sistem za upravljanje bazama podataka neće nužno uvek funkcionisati na ovaj način; korisniku se nudi opcija da izabere neki neserijabilni raspored ili je kontrola konkurentnosti implementirana tako da osigurava da rezultat izvršenog neserijabilnog rasporeda bude jednak rezultatu nekog serijabilnog rasporeda.

Ako sa A i B označimo neke objekte u bazi, a sa R i W operacije čitanja i upisa, respektivno, na sledećim dijagramima se sa leva na desno vide primeri serijskog, serijabilnog i neserijabilnog rasporeda:



Slika 2- Serijski, serijabilni i neserijabilni raspored

3.4 Anomalije izazvane preplitanjem

Raspored koji uključuje dve ili više uspešno izvršene, potvrđene transakcije koje bi trebalo da očuvaju konzistentnost ipak mogu dovesti bazu podataka u nekonzistentno stanje. Da bi dve naredbe nad istim objektom izazvale konflikt, potrebno je da barem jedna od njih bude upis. Postoje tri mogućnosti za konflikte, zavisno od redosleda naredbi:

3.4.1 Izgubljeno ažuriranje (WW)

U slučaju da dve ili više transakcije pročitaju isti objekat iz baze a zatim upišu novu vrednost na osnovu na osnovu prethodno pročitane vrednosti. Ovi konflikti spadaju u grupu „upis-upis”, odnosno ***write-write (WW)****.* Nijedna transakcija nije „svesna“ da se izvršavaju i druge transakcije i poslednja upisana vrednost će prepisate sve ostale, što prozokuje problem **izgubljenog ažuriranja** (*lost update*).

Ovakav problem bi se mogao prenebregnuti korišćenjem mehanizma zaključavanja kojim se ograničava pristup objektu sve dok se transakcija koja je zaključala objekat ne okonča.

3.4.2 Prljavo čitanje (WR)

Konflikti tipa upis-čitanje, odnosno ***write-read (WR)***, se dešavaju kada transakcija pročita vrednost objekta koji je već ažuriran od strane druge aktivne transakcije. Ta vrednost još uvek nije konačna, transakcija koja ju je upisala je može ponovo izmeniti pre potvrde, a može doći do obustavljanja same transakcije čime se vrednost vraća na prvobitnu. Ovo je posledica ograničenja da transakcija mora ostaviti bazu u konzistentom stanju nakon izvršenja, ali ne mora je nužno držati u konzistentnom stanju u toku samog izvršenja.

Ovaj konflikt bi se mogao izbeći ako bi transakcijama čije naredbe čitaju vrednost nekog objekta bio onemogućen pristup tom objektu dok se transakcija koja ga je inicijalno izmenila ne okonča.

3.4.3 Neponovljivo čitanje (RW)

Konflikti tipa čitanje-upis, odnosno ***read-write (RW)***, se dešavaju kada transakcija izmeni vrednost objekta koji je već bio pročitan od strane druge aktivne transakcije. Ako bi ta transakcija koja je pročitala objekat pokušala da ga pročita opet, dobila bi vrednost različitu od provbitne, iako se unutar transakcije ta vrednost nije menjala. Ovakav problem se ne može desiti u serijskom izvršenju i naziva se **neponovljivo čitanje** (*nonrepeatable read*).

Ovaj konflikt bi se mogao izbeći ako druge transakcije ne bi mogle da menjaju vrednost objekta dok transakcija koja je prva krenula čitanje ne završi krajnje čitanje te vrednosti unutar svog izvršenja.

3.4.4 Fantomsko čitanje (RW)

Još jedan tip ***RW*** konflikta slična neponovljivom čitanju je **fantomsko čitanje** (*phantom read*). Ovako se naziva anomalija kada se čitanje kolekcije objekata (npr. slogova tabele) iz baze izvrši više puta u toku transakcije ali se rezultati ne poklapaju. Ako

Ovaj konflikt bi se mogao izbeći ako druge transakcije ne bi mogle da menjaju vrednost objekta dok transakcija koja je prva krenula čitanje ne završi krajnje čitanje te vrednosti unutar svog izvršenja.

3.4.5 Promašeno i duplirano čitanje (WR)

Ovi konflikti su manje opšti od prethodnih, odnoso zavise od implementacionih detalja MS SQL Servera. Zavisno od specificiranog nivoa izolacija, može se desiti da druge transakcije imaju mogućnost izmene podataka koji se trenutno čitaju od strane druge transakcije. Tako se može desiti da se prilikom skeniranja indeksa desi da druga transakcija promeni vrednost indkesnog ključa sloga, i pomeri ga na poziciju koju je skeniranje već prošlo ili do koje još uvek nije stiglo, što izaziva nedostatak ili duplo čitanje tog sloga, respektivno.

Takođe se može dogoditi da se prilikom čitanja preskoči cela stranica sa više slogova. Ako se slogovi tabele čitaju korišćenjem IAM stranica (redosled alokacije stranica) a druga konkurentna transakcija izazove podelu stranice prilikom upisa.

3.5 Mehanizmi kontrole konkurentnosti u MS SQL Serveru

MS SQL Server implementira dve osnovne metode kontrole konkurentnosti, koje se zasnivaju na principima zaključavanja resursa i verzionisanja slogova.

Zbog nezgrapnosti prevoda „mehanizam zaključavanja“ za termin *lock* i nemogućnosti autora da pronađe konkretan standardni prevod u literaturi na srpskom jeziku, u daljem tekstu će se koristiti termin *brava*.

Oba tipa konkuretnosti će biti ilustrovani na istom primeru transfera novca sa tekućeg na štedni račun, koji je već opisan. Takođe će se podrazumevati da postoji konkuretna transakcija koja pokušava da povuče novac sa štednog računa.

3.5.1 Pesimistična kontrola konkuretnosti

Pesimistična kontrola konkurentnosti funkcioniše pod pretpostavkom da bi više korisnika koji istovremeno pristupaju podacima eventualno svi poželeli da te podatke i promene i to tako da prepišu promene ostalih korisnika. Sistem ovo sprečava korišćenjem skupa brava koje zaključavaju podatke tokom trajanja transakcije čim im ona prvi put pristupi i/ili ih izmeni. Ovakav pristup je pogodan za sisteme gde postoji konstantno nadmetanje za pristupom podacima i gde je zaštita podataka bravama jeftinija od poništavanja transakcija u slučaju nepovratnih konflikata.

Brave mogu biti definisane nad raznim vrstama resursa, kao što su tabele, slogovi ili same stranice. U narednom poglavlju će biti detaljno obrađeni različiti tipovi brava i njihova interakcija.

Na primeru transfera novca, transakcija koja je prva stigla da obezbedi bravu i koja upisuje, a vrlo verovatno i čita stanje štednog računa, će odmah zaključati taj slog u tabeli. Tek nakon što je transakcija završena će slog biti otključan, kada će druga transakcija dobiti mogućnost da preuzme bravu. Stanje koje će ona pročitati je konzistentno i izvršenje će se nastaviti bez problema.

3.5.2 Optimistična kontrola konkuretnosti

Optimistična kontrola konkurentnosti sa druge strane podrazumeva da i pored konkurentnog pristupa više korisnika ne postoje velike šanse za istovremenim upisom. Podaci se ne zaključavaju prilikom čitanja. U slučaju upisa sistem proverava da li je neki drugi korisnik takođe menjao iste podatke i u slučaju WW konflikta transakcije obustavljaju i poništavaju. Obično se koristi u okruženjima gde nema velikog nadmetanja za pristup podacima i jeftinije je poništiti transakciju u odnosu na korišćenje mehanizma zaključavanja.

Optimistična kontrola konkuretnosti nije naročita pogodna za isti primer transfera novca. S obzirom da nema zaključavanja a da obe transakcije pokušavaju da ažuriraju vrednost istog sloga, doći će do WW konflikta i jedna od transakcija bi morala da bude se poništi i prijavi grešku korisniku.

3.6 Nivoi izolacije

Nivo izolacije transakcije definiše stepen do koga jedna transakcija mora biti izolovana od izmena podataka izvršenih od strane drugih transakcija. Uglavnom su nazvani u odnosu na već pomenute anomalije, u odnosu da li su moguće ili ne za njihov konkretan slučaj.

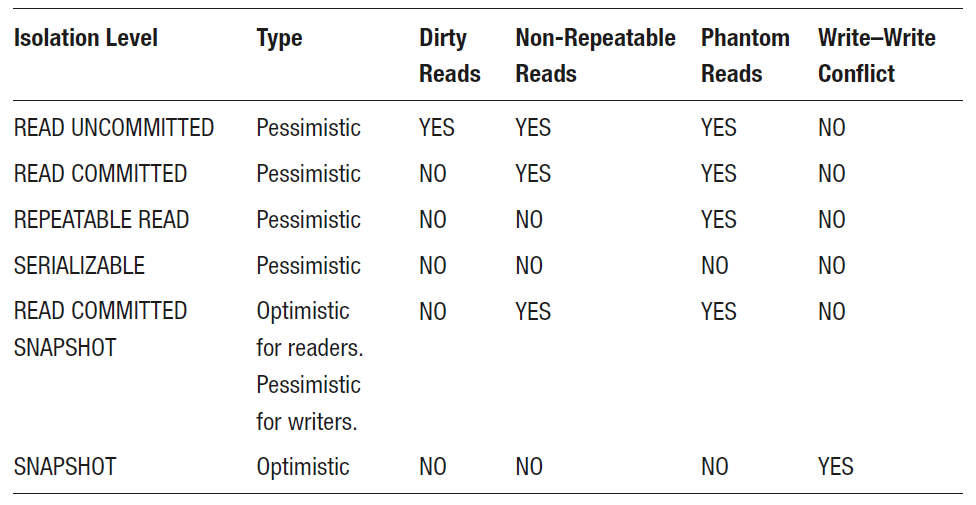
Za pesimističnu kontrolu konkurencije, nivoi izolacije kontrolišu:

* Da li i koje brave se zauzimaju kada se podaci čitaju
* Koliko dugačko se brave za čitanje okupiraju
* Da li operacija čitanja koja referencira slogove izmenjene od strane druge transakcije
  + Blokira dok se ne zauzme eksluzivna brava nad tim slogovima
  + Pribavlja snimljenu potvrđenu verziju reda koja je postojala u trenutku kada je transakcija započeta
  + Učitava nepotvrđene izmene

Nema svrhe pominjati slučajeve kada je u pitanju operacija pisanja jer se u tom slučaju za sve nivoe osim SNAPSHOT pribavlja eksluzivna brava.

SQL Server podržava sledeće nivoe izolacije:

* READ UNCOMMITED – Najniži nivo izolacije koji jedino osigurava da se ne pročitaju fizički netačni podaci. Prljavo čitanje je moguće pa transakcije mogu videti nepotvrđene promene od strane drugih transakcija.
* READ COMMITED – Dopušta transakciji da pročita vrednost koju je ranije pročitala (ali ne i izmenila) druga transakcija, ne čekajući da se ona prvo kompletira. SQL Server zadržava brave za pisanje do kraja transakcije ali za brave za čitanje oslobađaju odmah nakon čitanja (SELECT naredbe).
* REPEATABLE READ – Transakcija zadržava brave i za čitanje i za pisanje do kraja transakcije. Ne vodi se računa o rasponskim (*range*) bravama tako da je moguće fantomsko čitanje.
* SERIALIZABLE – Najviši nivo izolacije kod kojega su transakcije potpuno međusobno izolovane. Brave za čitanje i pisanje se zadržavaju do kraja transakcije, a rasponske brave se zauzimaju prilikom SELECT naredbi koje koriste rasponsku WHERE naredbu da bi se izbeglo fantomsko čitanje. Pojedine DDL operacije ne mogu biti izvršene unutar ovakvih transakcija jer mogu prouzrokovati promene nekompatibilne sa definicijom nivoa izolacije.
* READ COMMITED SNAPSHOT – Potrebno je aktivirati specijalnu opciju u sistemu da bi ovaj nivo bio podržan. Kada se on koristi, READ COMMITED izolacija koristi verzionisanje slogova da pruži konzistentost prilikom čitanja na nivou naredbi. Sistem koristi ovaj mehanizam da pruži svakoj naredbi transakciono konzistentnu sliku podataka za trenutak u kome je naredba pokrenuta.
* SNAPSHOT – Potrebno je aktiviratio specijalnu opciju u sistemu da bi ovaj nivo bio dozvoljen. Operacije čitanja ne zauzimaju brave nad stranicama i slogovima, već se zauzimaju samo SCH-S brave nad tabelama. Ovaj nivo izolacije je takođe nekompatibilan sa podskupom DDL operacija.



Slika 3 - Nivoi izolacije

Niži nivoi izolacije omogućavaju da veći broj korisnika istovremeno pristupa podacima ali takođe i povećavaju broj potencijalnih anomalija sa kojima se korisnici mogu susresti. Sa druge strane, striktniji nivoi izolacije će smanjiti broj negativnih sporednih efekata ali će trošiti više sistemskih resursa i znatno povećati šanse da će se transakcije međusobno blokirati.

SERIALIZABLE nivo garantuje da će svaka ponovljena operacija čitanja dobiti isti rezultat ali zato nivo zaključavanja koji podrazumeva može imati osetne posledice na višekorisničke sisteme.

READ UNCOMMITED ne podrazumeva ni brave ni verzionisanje slogova te je opterećenje resursa minimalno, ali se zato mogu dogoditi svi sporedni efekti osim WW konflikta.

U realnim situacijama je najčešće odgovarajući nivo neki između, zavisno od zahteva sistema. MS SQL SERVER podrazumevano koristi READ COMMITED ako drugačije nije naglašeno.

4. Zaključavanje

SQL Server koristi zaključavanje da bi ispunio zahtev transakcija za izolacijom. Mehanizmi zaključavanja odnosno brave se zauzimaju i drže nad određenim resursima kao što su slogovi, tabele, stranice, same baze itd. SQL Server podrazumevano koristi zaključavanje na nivou slogova, što minimizuje broj negativnih sporednih efekata konkurentnog izvršavanja.

4.1 Dijagnostika

Akutelno potraživanje zaključavanja u sistemu se može pribaviti upitom nad dinamičkim upravljačkim pogledom sys.dm\_tran\_locks, kao u sledećem primeru:

set transaction isolation level read uncommitted

begin tran

update Customers

set ShortBio = 'Example'

where CustomerId = 100

select resource\_type, resource\_description,

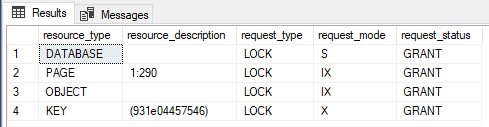
request\_type, request\_mode, request\_status

from sys.dm\_tran\_locks

where request\_session\_id = @@spid

commit

Može se primetiti da je zauzeta X brava na ključu, IX brave na stranici i objektu (tabeli), kao i S brava na samoj bazi. U daljem tekstu će biti objašnjeno značenje ovih tipova brava.



Slika 4 - Rezultati upita

4.2 Kategorije brava

SQL Server koristi preko 20 različitih tipova brava. One se, međutim, mogu grupisati u nekoliko glavnih kategorija, kao što će biti opisano u ostatku poglavlja.

4.2.1 Exclusive (X)

Eksluzivne brave se zauzimaju naredbama pisanja – INSERT, UPDATE, DELETE i MERGE, koje modifikuju podatke. Ovi upiti pribavljaju eksluzivnu (X) bravu nad slogovima koji se tiču upita i zadržavaju je do kraja transakcije. Kao što joj ime kaže, za određeni resurs eksluzivnu bravu može držati samo jedna sesija u bilo kom trenutku. Ovim se postiže sprečavanje WW tipova konflikta, jer samo jedna sesija može istovremeno modifikovati iste podatke. Nivo izolacije transakcije ne utiče na ponačanje X brava, one se zadržavaju do kraja transakcije čak i u READ UNCOMMITED modu izvršavanja.

4.2.2 Shared (S)

Deljene (*shared*) brave se zauzimaju operacijama čitanja – SELECT naredbama. Deljene brave su kompatibilne jedne sa drugima i više sesija ih istovremeno može držati. Ove brave se oslobađaju čim podaci budu pročitani, osim u slučaju REPEATABLE READ ili višeg nivoa izolacije, gde se zadržavaju do kraja transakcije.

4.2.3 Update (U)

Brave ažuriranja (*update*) se zauzimaju prilikom operacija izmene podataka, dok se traže slogovi koje zapravo treba ažurirati. Nakon što je U brava zauzeta, SQL Server čita slog i poređenjem sa predikatom upita određuje da li taj slog treba da bude ažuriran. U slučaju da da, U brava se konvertuje u X bravu i vrši se upis, dok se u suprotnom oslobađa U brava.

Ovim pristupom se sprečava čest slučaj uzajamne blokade (*deadlock*) kada dve transakcije obe imaju S bravu nad resursom i obe pokušavaju da je unaprede u X bravu da bi mogle da izvrše upis; budući da je X brave eksluzivna, S brava druge transakcije mora prvo biti oslobođena; obe transakcije čekaju jedna na drugu da oslobode S bravu i dolazi do zastoja. S obzirom da samo jedna transakcija može zauzeti U bravu, ona se bez problema može pretvoriti u X bravu u slučaju upisa.

4.2.4 Intent (I\*)

Zaključavanje na nivou slogova je najbolje rešenje za konzistentnost ali često ne i za performanse. Za slučajeve kada sesiji treba eksluzivni pristup celoj tabeli bi za zaključavanje na nivou slogova sesija morala da skenira celu tabelu i za svaki red proveri da li postoji brava čitanja. Jasno je da je ovakav pristup izuzetno neefikasan, i to sve više što je tabela veća.

SQL Serveru je uveo koncept brava *namere* (*intent*) (I\*), koje se pribavljaju na nivou tabela i stranica i saopštavaju postojanje brava nad objektima niže u hijerarhiji. Brave namere se pribavaljaju pre brava na nižim nivoima, te odatle i potiče i njihov naziv jer označavaju nameru da se konkretnija brava pribavi.

Pored veće efikasnosti u detektovanju konflikta zaključavanja na višim nivoima granularnosti, brave namere takođe sprečavaju druge transakcije da modifikuju resurse višeg nivoa na način koji bi invalidirao bravu na nižem nivou. Simbol zvezdice (\*) označava da ova kategorija obuhvata širi skup brava namere:

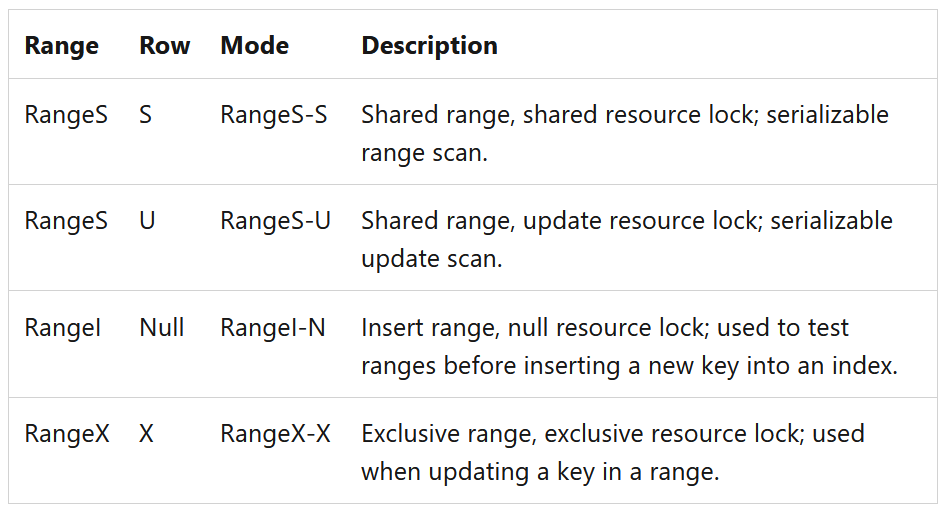
* IS – Štiti zatražene ili zauzete S brave nad pojedinim (ali ne svim) resursima niže u hijerarhiji
* IX – Nadskup IS brava, obavlja istu funkciju ali i za X brave
* SIX – Štiti zatražene ili zauzete S brave nad svim resursima niže u hijerarhiji i IX nad pojedinim (ali ne svim) resursima niže u hijerarhiji. Zauzimanje SIX brave na tabeli takođe zauzima IX bravu na stranicama koje se menjaju i X brave na slogovima koji se menjaju. Na resursu istovremeno može postojati samo jedna SIX brava što sprečava upis od strane drugih transakcija, mada druge transakcije mogu čitati resurse niše u hijerarhiji ako uspeju da pribave IS brave na nivou tabele.
* IU – Štiti zatražene ili zauzete U brave nad svim resursima niže u hijerarhiji. Koriste se samo nad stranicama. Pretvaraju se u IX brave ako se zapravo desi ažuriranje.
* SIU – Kombinacija S i IU brava, kao rezultat zasebnog istovoremenog pribavljanja oba tipa brava.
* UIX – Kombinacija U i IX brava, kao rezultat zasebnog istovoremenog pribavljanja oba tipa brava.

4.2.5 Range

Brave raspona, odnosno raspona ključa (*key-range*), štite raspon slogova koji se čitaju unutar neke naredbe kada se koristi SERIALIZABLE nivo izolacije.

Njhova glavna svrha je da spreče fantomsko čitanje. Bravom se zauzima deo indeksa od početka do kraja definisanog raspona. Time se izmene i brisanja postojećih slogova blokira i dodavanje novih slogova čija bi vrednost ključa bila unutar definisanog raspona.

Postoje sledeći tipovi ovih brava:



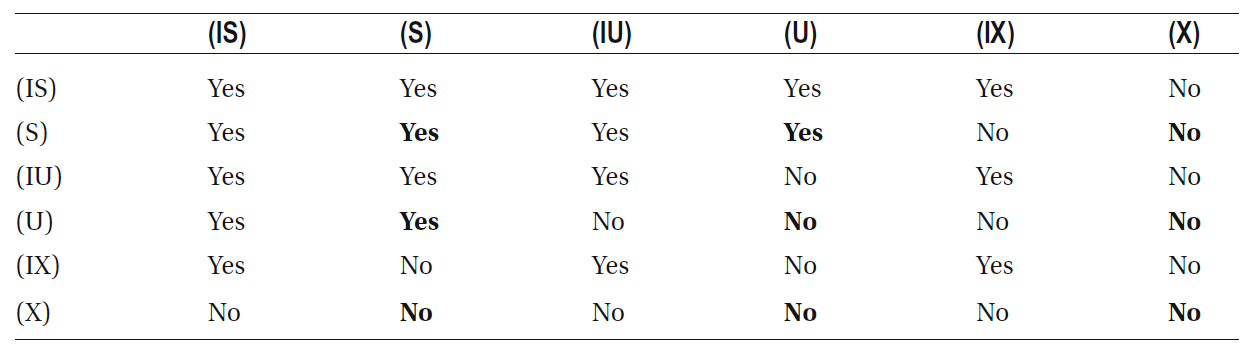
Slika 5 - Key-range

4.2.5 Schema

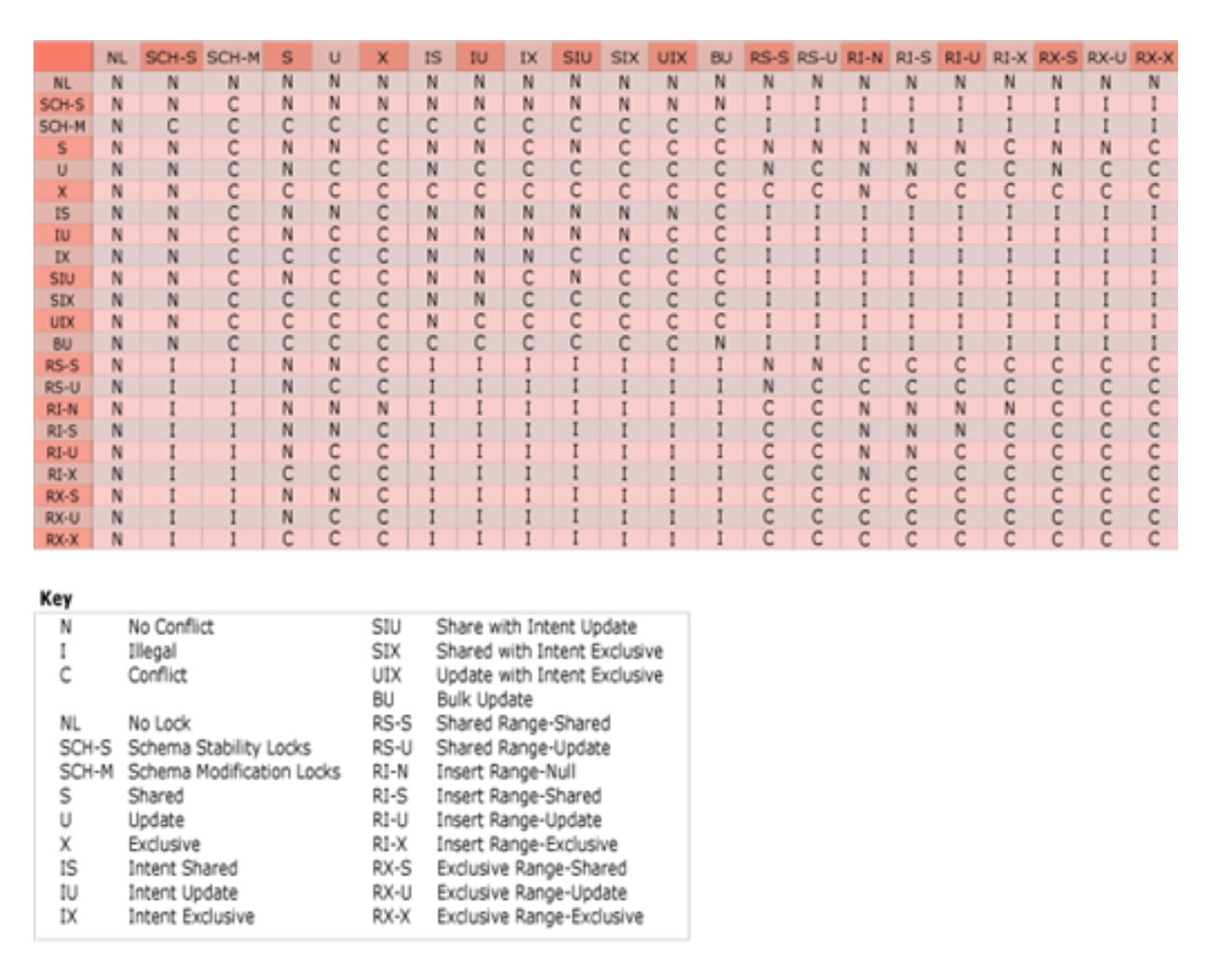
Zaključavanje na nivou šeme se koristi prilikom operacija koje modifikuju samu šemu baze podataka, kao što je dodavanje kolone u tabelu ili uklanjanje cele tabele. Dok je Sch-M brava zauzeta, druge sesije ne mogu pristupiti tabeli. Takođe postoji i brava Sch-S za stabilnost šeme, koja blokira samo transakcije koje bi zauzele Sch-M šemu, i ona se zauzima tokom kompilacije i izvršavanja upita, da bi se očuvala struktura tabele nad kojom je upit pripremeljen u toku njegovo izvršenja.

4.3 Kompatibilnost brava

* X brave su nekompatibilne sa svim tipovima brava, uključujući i druge X brave. Više sesija ne mogu istovremeno upisati u isti slog. Šta više, izmenjeni, još nepotvrđeni slogovi sa X bravama ne mogu čitati ove slogove.
* IS/IU/IX brave su međusobno kompatibilne je označavaju postojanje brava na objektima niže u stablu i više sesija može držati brave namere istovremeno
* U brave su nekompatibilne sa X bravama i međusobno. Sesije koje upisuju ne mogu da odrede da li se slog treba istovremeno ažurirati niti mogu pristupiti slogu nad kojim je zauzeta X brava.
* U brave su kompatibilne sa S bravama. Sesije koje upisuju mogu da odrede da li se slog treba ažurirati bez da blokiraju ili budu blokirane od strane drugih sesija koje čitaju. Kompatibilnost S i U brava je glavni razlog za postojanje U brava u SQL Serveru, jer se time znatno umanjujue blokiranje između čitanja i pisanja.



Slika 6 - Kompatibilnost tipova brava



Slika 7 - Detaljna matrica kompatibilnosti svih tipova brava u SQL Serveru

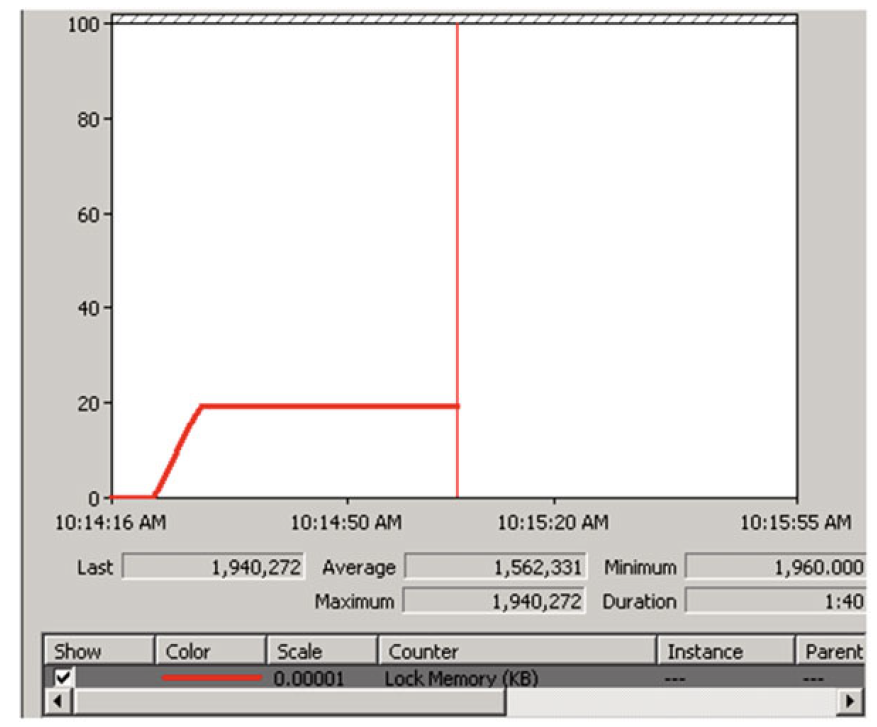
4.4 Eskalacija brava

Iako je zaključavanje na nivou slogova odlično iz perspektive konkurentnosti, skupo je iz prepsketive resursa, pogotovo memorije.

SQL Serveru bi bili potrebni gigabajti memorije samo da vodi računa o zaključavanju slogova i stranica. Zato se koristi tehnika eskalacije brava.

Kada neka naredba koja se izvršava zauzme bar 5.000 brava na slogovima i stranicama u okviru istog objekta, SQL Server pokušava da zameni ovu bravu jednom bravom nad celom tabelom, ili u nekim slučajevima, particijom. Ova operacija će uspeti pod uslvoom da nijedna druga sesija ne drži nekompatibilne brave na istom objektu ili particiji.

U slučaju uspeha, SQL Server oslobađa sve brave na nivou slogova i stranici trenutno zauzetih u okviru transakcije i zadržava samo bravu na objektu. U slučaju neuspeha, brave se zadržavaju i postupak se ponavlja na svakih 1.250 narednih brava koje se zauzmu. Takođe, pored ovih pragova, SQL Server koristi i opterećenost memorije ili specijalne konfiguracije kao okidač da se pokuša eskalacija brava.

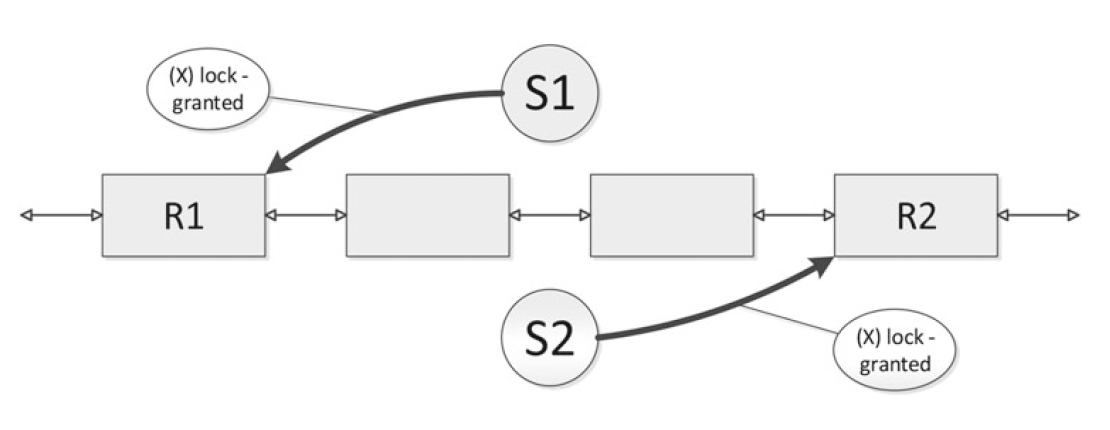


Slika 8 - Zauzeće memorije bravama

5. Deadlock

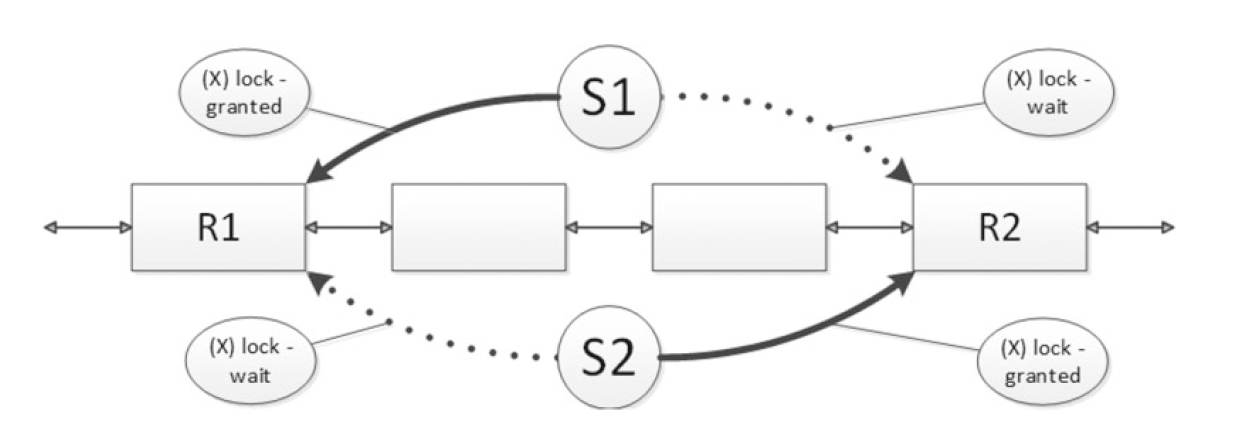
5.1 Klasično uzajamno blokiranje

Uzajamna blokada, odnosno zastoj, ili originalno na engleskom *deadlock*, je specijalni slučaj blokiranja kada se više sesija ili više niti izvršavanja međusobno blokira. SQL Server rešava zastoje nasilnim prekidom jedne od uključenih sesija, omogućavajući ostalima da nastave izvršavanje.



Slika 9 - Prvi korak ka uzajamnom blokiranju

Klasičan slučaj uzajamne blokade se dešava kada se dve ili više sesija nadmeće za isti skup resursa. Pretpostavimo da dve sesije pokušavaju da upišu vrednosti u slogove R1 i R2 ali obrnutim redosledom. Neka sesija 1 ažurira R1 a sesija 2 R2. U ovom trenutku su obe sesija pribavile X bravu nad respektivnim slogovima.



Slika 10 - Postignuta uzajamna blokada

U daljem izvršavanju je sesija 1 treba da ažurira R2, i biće joj potrebna X brava nad R2. Time će sesija 1 postati blokirana jer je sesija 2 već zauzela ekskluzivnu bravu nad R2. Sa druge strane, budući da sesija 2 sada pokušava da ažurira R1, i ona će biti ublokirana jer sesija 1 poseduje X bravu nad R1. U ovom trenutku obe sesija čekaju jedna na drugu i ne mogu nastaviti izvršenje.

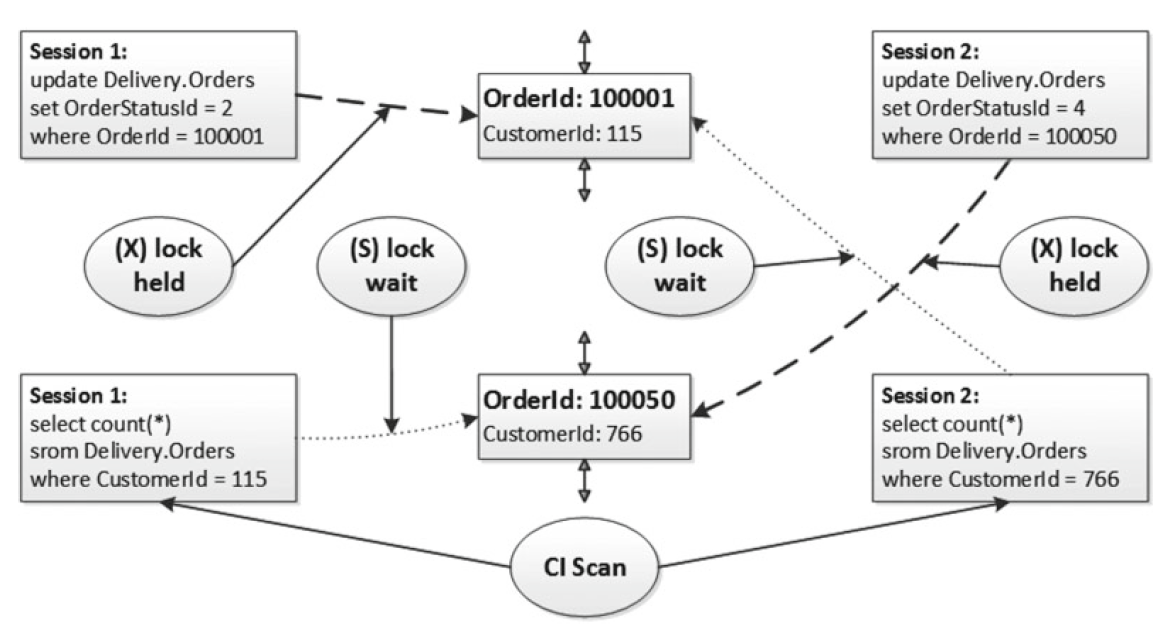
5.2. Izvori zastoja u SQL Serveru

Iako se *deadlock* često javlja kada se isti podaci često menjaju od strane većeg broja sesija, postoje i drugi neretki razlozi za ovu pojavu.

5.2.1 Neoptimalno skeniranje

Jedan od razloga je skeniranje izazvano neoptimalnim upitima.

Pretpostavimo da dve sesije treba da ažuriraju svaka po slog iz iste tabele, sa predikatom nad kolonom na kojoj nije definisan indeks, i da zatim izvrše upit koji računa agregaciju nad drugom kolonom nad kojom takođe nije definisan indeks. U oba slučaja (upis i čitanje) će biti potrebno izvršiti skeniranje klasterovanog indeksa. Ako imamo READ COMMITTED ili viši nivo izolacije, sesije će tokom upisa pribaviti X brave, a tokom čitanja će pokušati da pribave S brave, gde se dešava situacija analogna onoj iz primera za klasičan zastoj:

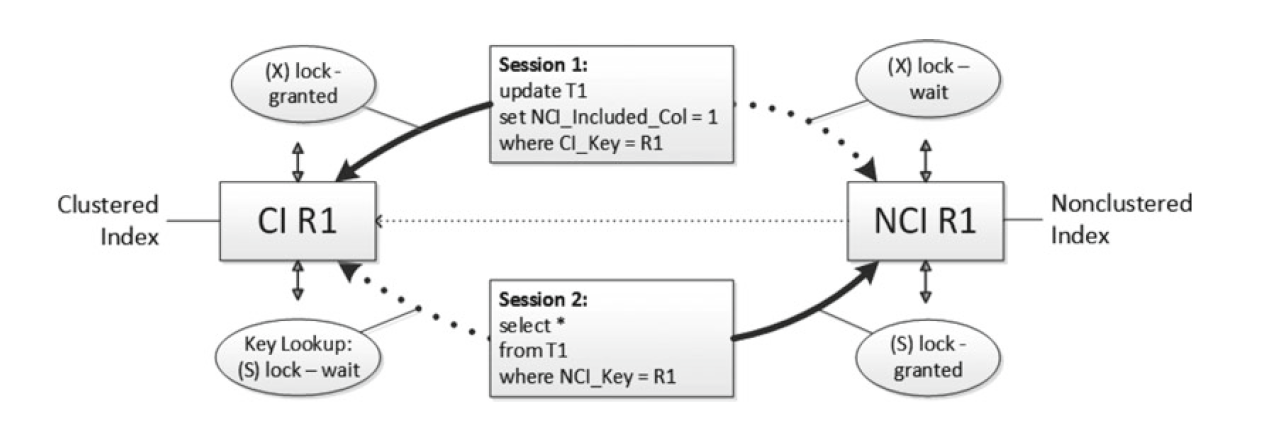


Slika 11 - Deadlock izazvan neefikasnim upitima

5.2.2 Istovremeno ažuriranje istog slog koriščnjem pretrage po ključu

Takođe može doći do zastoja kada više sesija pokušava da pročita i ažurira isti slog. Ako na tabeli postoji neklasterovani indeks i jedna sesija pokušava da pročita slog koristeći taj indeks. Ako indeks nije pokrivajući (*covering*), odnosno ako se svi parameteri predikata upita ne pokrivaju ovim indeksom, biće neophodno odraditi *index seek* operacija nad neklasterovanim indeksom a zatim pribaviti kroz klasterovani indeks pronaći traženu stranicu znajući vrednost njenog ključa (*key lookup*). Ova sesija bi time pribavila deljenu bravu nad slogom neklasterovanim indeksom a zatim bi pokušala da pribavi deljenu bravu i nad slogom klasterovanog indeksa.

Ako za to vreme druga sesija pokušava da ažurira vrednost kolone koja je deo neklasterovanog indeksa, ona će prvo zauzeti eksluzivnu bravu nad klasterovanim indeksom i onda pokušati da zauzme eksluzivnu bravu nad neklasterovanim indeksom, čime se dolazi do slećedeg zastoja:



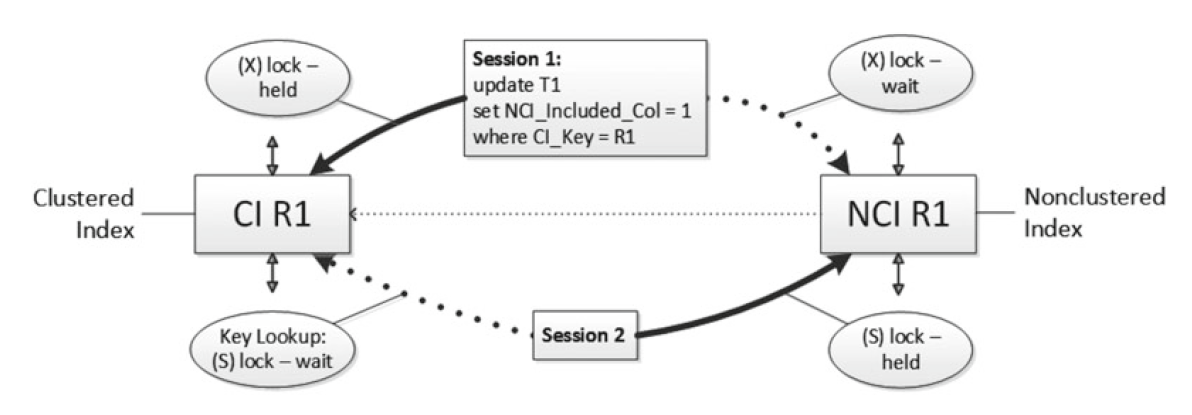
Slika 12 - Deadlock izazvan ažuriranjem istog sloga

5.2.2 Istovremeno ažuriranje istog slog koriščnjem pretrage po ključu

U poslednjem primeru imamo tabelu sa klasterovanim i neklasterovanim indeksom i trigerom na AFTER UPDATE. Ako sesija 1 ažurira kolonu koja ne pripada neklasterovanom indeksu ona će zauzeti bravu nad slogom u klasterovanom indeksu, i upisom se okida triger.

Ako se pretpostavi da postoji još jedna sesija koja pokuša da pročita isti taj slog korišćenjem neklasterovanog indeksa, ta sesija će uspešno pribaviti deljenu bravu pomoću *index seek* operacije.

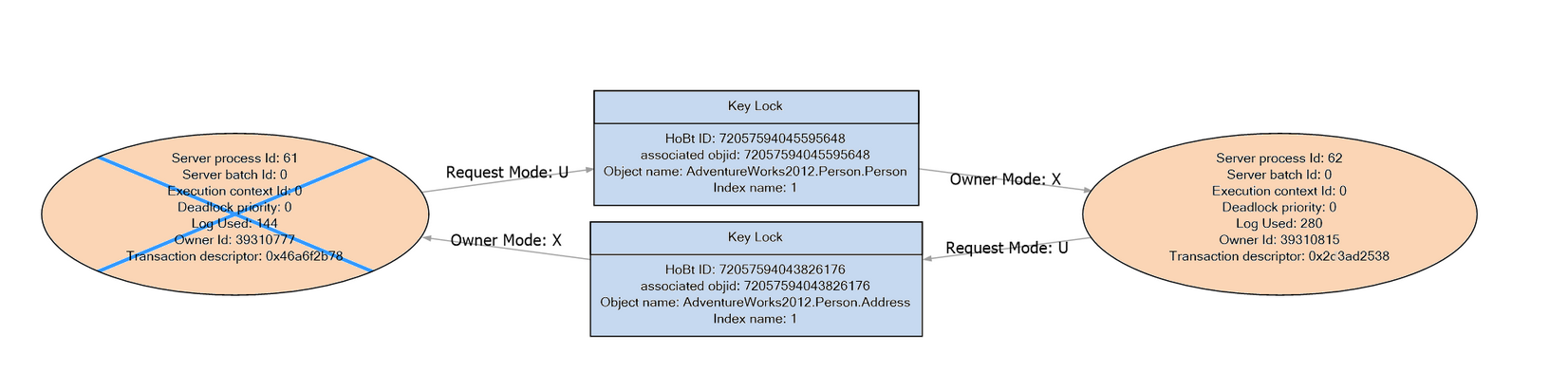
Međutim, biće blokirana u pribavljanu deljene brave nad klasterovanim indeksom. Sa druge strane, triger će pokušati da ažurira neklasterovani indeks i biće blokiran zauzetom deljenom bravom. Tako se dolazi do sledeće situacije:



Slika 13 - Deadlock kao posledica višestrukog ažuriranja

5.3 Rešavanje zastoja

SQL Server poseduje sistemski zadatak *Deadlock Monitor* koji se budi svakih minimum svakih 5 sekundi i proverava da li ima zastoja u sistemu kreiranjem grafa zavisnosti resursa. Na narednoj slici su plavom bojom označeni resursi, narandžastom procesi a potezima grafa su izkazani odnosi zauzetosti i potražnje između procesa i resursa.



Slika 14 - Primer grafa zavisnosti

Interval buđenja monitora se smanjuje u slučaju detekcije zastoja, i može se spustiti i do 100 ms. Nakon što je zastoj detektovan, bira se sesija koja će biti ugašena. SQL Server po osnovnim podešavanjima bira sesiju koja zauzima manje prostora u transakcionom logu. Komadom SET DEADLOCK\_PRIORITY se delimično može uticati na izbor sesije.

7. Literatura

1. Raghu Ramakrishnan and Johannes Gehrke, 2012, Database Management Systems (3rd. ed.), McGraw-Hill, Inc., USA.
2. Dmitri Korotkevich, 2016, Pro SQL Server Internals (2nd ed), Apress, USA
3. Dmitri Korotkevich, 2018, Expert SQL Server Transactions and Locking Concurrency Internals fir SQK Server Practicioners (1st ed), Apress, USA
4. <https://en.wikipedia.org/wiki/Microsoft_SQL_Server>
5. <https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide>
6. <https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15>