

**Univerzitet u Nišu**

**Elektronski fakultet**

**Katedra za računarstvo**

**Seminarski rad**

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

**Sistemi za upravljanje bazama podataka**

|  |  |  |
| --- | --- | --- |
| **Mentor**: |  | **Student**: |
| Dr Aleksandar Stanimirović |  | Miloš Veljanovski 1559 |

**Niš, maj 2023.**

**Sadržaj**

[1. Uvod u transakcije 4](#_Toc135852993)

[1.1. Atomičnost (Atomicity) 5](#_Toc135852994)

[1.2. Konzistentnost (Consistency) 5](#_Toc135852995)

[1.3. Izolacija (Isolation) 5](#_Toc135852996)

[1.4. Trajnost (Durability) 6](#_Toc135852997)

[2. Transakcije 7](#_Toc135852998)

[2.1. Lokalne transakcije 7](#_Toc135852999)

[2.2. Distribuirane transakcije 8](#_Toc135853000)

[2.3. Kontrola transakcija 9](#_Toc135853001)

[2.4. Stanje transakcije 9](#_Toc135853002)

[3. Primeri transakcija 11](#_Toc135853003)

[3.1. Primer COMMIT transakcije 11](#_Toc135853004)

[3.2. Primer ROLLBACK transakcije 13](#_Toc135853005)

[3.3. Upotreba globalne promenljive @@Error u transakcijama 13](#_Toc135853006)

[3.4. Transakcije sa automatskim poništavanjem 14](#_Toc135853007)

[3.5. Savepoint u transakcijama 15](#_Toc135853008)

[3.5.1. Kako osloboditi savepoint u transakciji? 15](#_Toc135853009)

[3.6. Implicitne transakcije 16](#_Toc135853010)

[3.7. Eksplicitne transakcije 17](#_Toc135853011)

[3.8. Označene transakcije 17](#_Toc135853012)

[3.9. Imenovane transakcije 18](#_Toc135853013)

[4. Izolacija transakcija 19](#_Toc135853014)

[4.1. Anomalije izazvane konkurentnošću 19](#_Toc135853015)

[4.1.1. Gubitak podataka 19](#_Toc135853016)

[4.1.2. Prljava čitanja 19](#_Toc135853017)

[4.1.3. Fantomsko čitanje 20](#_Toc135853018)

[4.1.4. Analiza nekonzistentnosti 20](#_Toc135853019)

[4.2. Nivoi izolacije 20](#_Toc135853020)

[4.2.1. Read Uncommitted (Čitanje bez potvrde) 21](#_Toc135853021)

[4.2.2. Read Committed (Čitanje sa potvrdom) 21](#_Toc135853022)

[4.2.3. Repeatable Read (Ponovljivo čitanje) 21](#_Toc135853023)

[4.2.4. Snapshot (Snimak) 21](#_Toc135853024)

[4.2.5. Serializable (Serijski) 22](#_Toc135853025)

[5. Zaključavanje (Locks) u SQL Serveru 22](#_Toc135853026)

[5.1. Uvod 22](#_Toc135853027)

[5.2. Režimi zaključavanja (Lock Modes) 23](#_Toc135853028)

[5.2.1. Isključivo zaključavanje 23](#_Toc135853029)

[5.2.2. Deljeno zaključavanje 23](#_Toc135853030)

[5.2.3. Ažurirajuće zaključavanje 23](#_Toc135853031)

[5.2.4. Zaključavanje nameravanja 24](#_Toc135853032)

[5.2.5. Šema zaključavanja 24](#_Toc135853033)

[5.2.6. Zaključavanje masovnog ažuriranja 24](#_Toc135853034)

[5.3. Primer 25](#_Toc135853035)

[6. Literatura 27](#_Toc135853036)

[7. Listing 28](#_Toc135853037)

# 1. Uvod u transakcije

A picture containing text, screenshot, font, design

Description automatically generatedTransakcija je logička radna jedinica koja obavlja jednu ili više aktivnosti u bazi podataka. Transakcije se mogu sastojati od jednog čitanja, pisanja, brisanja ili ažuriranja ili kombinacije tih operacija. Pretpostavimo da će, kada želimo da podignemo novac sa bankomata, aplikacija za bankomat ovu operaciju ostvariti u tri koraka. Kao prvi korak, aplikacija će proveriti stanje na računu, a zatim će oduzeti novac sa izvornog računa. Zajedno sa ova dva procesa, vodiće se evidencija o ovoj aktivnosti povlačenja novca. Sledeća slika u osnovi ilustruje princip rada transakcija u sistemima relacionih baza podataka.

Slika 1. Ilustracija transakcija

Glavna ideja transakcija je da kada bilo koja od naredbi vrati grešku, sve modifikacije se poništavaju radi obezbeđivanja integriteta podataka. S druge strane, ako su sve naredbe uspešno završene, modifikacije podataka postaju trajne u bazi podataka. Kao rezultat, ako dođe do prekida napajanja ili drugih problema prilikom podizanja novca sa bankomata, transakcije garantuju doslednost našeg stanja na računu. Bilo bi najbolje da sve ove korake obavimo putem transakcije, jer četiri glavna svojstva transakcija omogućavaju tačnije i doslednije operacije. Ova svojstva poznata su kao ACID (atomičnost, doslednost, izolacija, trajnost) u sistemima relacionih baza podataka, prema prvom slovu njihovih imena:

## **1.1. Atomičnost (Atomicity)**

Svojstvo atomičnosti identifikuje da je transakcija nedeljiva. Atomična transakcija je ili u potpunosti završena ili uopšte nije započeta. Sve promene koje transakcija može izazvati na sistemu se završavaju u celosti. Ako iz bilo kojeg razloga dođe do greške i transakcija nije u mogućnosti da završi sve korake, sistem se vraća u stanje u kojem je bio pre početka transakcije.

Primer atomične transakcije je transakcija transfera sredstava sa jednog na drugi račun. Novac se uklanja sa računa A i stavlja na račun B. Ako sistem otkaže nakon što se novac ukloni sa računa A, onda će sistem za obradu transakcija vratiti novac na račun A, čime se sistem vraća u prvobitno stanje. Ovo se naziva poništavanjem transakcije (rollback).

## **1.2. Konzistentnost (Consistency)**

Transakcija obezbeđuje doslednost u stanju sistema tako što garantuje da na kraju svake transakcije sistem bude u validnom stanju. Ako se transakcija uspešno završi, sve promene na sistemu će biti pravilno izvršene i sistem će biti u validnom stanju. Ako se u transakciji pojavi bilo kakva greška, sve već izvršene promene će automatski biti poništene. To će vratiti sistem u stanje pre početka transakcije. Pošto je sistem bio u doslednom stanju kada je transakcija započeta, ponovo će biti u doslednom stanju.

Ponovo posmatrajući sistem transfera računa, sistem je konzistentan ako je ukupan iznos svih računa konstantan. Ako se pojavi greška i novac bude uklonjen sa računa A, a ne bude dodat na račun B, ukupan iznos na svim računima bi se promenio. Sistem više ne bi bio konzistentan. Poništavanjem uklanjanja sredstava sa računa A, ukupan iznos će ponovo biti onakav kakav bi trebao da bude, a sistem će se vratiti u dosledno stanje.

## **1.3. Izolacija (Isolation)**

Kada se transakcija izvršava izolovano, čini se kao da je to jedina radnja koju sistem obavlja u tom trenutku. Ako postoje dve transakcije koje obavljaju istu funkciju i izvršavaju se istovremeno, izolacija transakcija će se pobrinuti da svaka transakcija misli da ima ekskluzivno korišćenje sistema. Ovo je važno jer tokom izvršavanja transakcije stanje sistema možda neće biti konzistentno. Transakcija obezbeđuje da sistem ostane dosledan nakon završetka transakcije, ali tokom pojedinačne transakcije to možda nije slučaj. Ako transakcija ne bi bila izvršena izolovano, mogla bi pristupiti podacima iz sistema koji možda nisu dosledni. Obezbeđivanjem izolacije transakcija, sprečava se da se to dogodi.

## **1.4. Trajnost (Durability)**

Transakcija je trajna u smislu da su sve promene koje je izvršila na sistemu trajne nakon uspešnog završetka. Postoje mehanizmi zaštite koji će sprečiti gubitak informacija, čak i u slučaju kvara sistema. Pomoću zapisivanja koraka koje transakcija obavlja, stanje sistema može biti rekreirano čak i ako je hardver sam po sebi otkazao. Koncept trajnosti omogućava programeru da zna da je završena transakcija trajni deo sistema, bez obzira na ono što se kasnije dogodi sa sistemom.

# 2. Transakcije

## **2.1. Lokalne transakcije**

Svi sistemi za baze podataka trebali bi pružati ugrađenu podršku za transakcije. Transakcije koje su ograničene samo na jedan resurs ili bazu podataka poznate su kao lokalne transakcije. Lokalne transakcije mogu biti u jednom od sledeća četiri režima transakcija:

1. **Autocommit transakcije** (transakcije sa automatskim potvrđivanjem): Autocommit režim je podrazumevani način upravljanja transakcijama u SQL Serveru. Svaka T-SQL naredba se potvrđuje (commit) ili poništava (rollback) kada se završi. Ako naredba uspešno završi, potvrđuje se; ako naiđe na grešku, vrši se poništavanje. SQL Server veza radi u autocommit režimu kada ovaj podrazumevani način nije pregažen nekim drugim vrstama transakcija.
2. **Eksplicitne transakcije**: Eksplicitne transakcije su one u kojima eksplicitno kontrolišete kada transakcija počinje i kada se završava. U prošlosti, eksplicitne transakcije su se, takođe, nazivale korisnički definisane ili korisnički određene transakcije.

T-SQL skripte za ovaj režim koriste naredbe BEGIN TRANSACTION, COMMIT TRANSACTION i ROLLBACK TRANSACTION. Režim eksplicitnih transakcija traje samo tokom trajanja transakcije. Kada se transakcija završi, veza se vraća u režim transakcije u kojem je bila pre nego što je eksplicitna transakcija započela.

1. **Implicitne transakcije**: Kada se povežete na bazu podataka koristeći SQL Server Management Studio i izvršite DML upit, promene se automatski čuvaju. Ovo se dešava jer je, podrazumevano veza u režimu transakcija sa automatskim potvrđivanjem. Ako želite da promene ne budu potvrđene osim ako to eksplicitno naznačite, morate postaviti vezu u režim implicitnih transakcija.

Možete postaviti vezu sa bazom podataka u režim implicitnih transakcija korišćenjem naredbe SET IMPLICIT TRANSACTIONS ON|OFF.

Nakon što je režim implicitnih transakcija postavljen na ON za vezu, SQL Server automatski započinje transakciju kada prvi put izvrši bilo koju od sledećih naredbi: ALTER TABLE, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE i UPDATE.

Transakcija ostaje na snazi sve dok se eksplicitno ne izda naredba COMMIT ili ROLLBACK. To znači da kada, na primer, bude izdata naredba UPDATE nad određenim zapisom u bazi podataka, SQL Server će održavati zaključavanje podataka namenjenih za izmenu sve do izdavanja naredbe COMMIT ili ROLLBACK. U slučaju da nijedna od ovih naredbi nije izdata, transakcija će se automatski poništiti kada se korisnik odjavi. Zbog toga nije preporučljivo koristiti režim implicitnih transakcija u bazi podataka sa visokom konkurentnošću.

1. **Transakcije obuhvaćene grupom naredbi** (batch-scoped transakcije): Veza može biti u režimu transakcija obuhvaćenih grupom naredbi, ako je transakcija koja se izvršava u njoj omogućena za višestruke aktivne skupove rezultata (Multiple Active Result Sets - MARS). U osnovi, MARS ima povezano okruženje izvršavanja grupa naredbi, jer omogućava ADO.NET-u da iskoristi mogućnost SQL Servera da ima više aktivnih komandi na jednom objektu veze.

Kada je MARS omogućen, možete imati više međusobno isprepletenih grupa naredbi koje se izvršavaju istovremeno, tako da su sve promene koje se vrše u okruženju izvršavanja ograničene na određenu grupu naredbi sve dok se izvršavanje grupe ne završi. Nakon završetka izvršavanja grupe, postavke izvršavanja se kopiraju u podrazumevano okruženje. Dakle, veza se smatra da koristi režim transakcija obuhvaćenih grupom naredbi ako izvršava transakciju, ima MARS omogućen i ima više grupa naredbi koje se izvršavaju istovremeno.

MARS omogućava izvršavanje više međusobno isprepletenih grupa naredbi. Međutim, MARS ne omogućava imati više transakcija na istoj vezi; samo omogućava postojanje višestrukih aktivnih skupova rezultata.

## **2.2. Distribuirane transakcije**

Za razliku od lokalnih transakcija koje su ograničene na jedan resurs ili bazu podataka, distribuirane transakcije obuhvataju dva ili više servera koji se nazivaju menadžerima (upravljačima) resursa. Upravljanje transakcijama mora biti usklađeno između upravljača resursa putem serverske komponente poznate kao menadžer transakcija ili koordinator transakcija. SQL Server može delovati kao menadžer resursa za distribuirane transakcije koje koordiniraju upravljači transakcija poput Microsoft Distributed Transaction Coordinator (Microsoft DTC).

Transakcija sa jednim SQL Serverom koja obuhvata dve ili više baza podataka zapravo je distribuirana transakcija. Međutim, SQL Server upravlja distribuiranom transakcijom interno.

Na nivou aplikacije, distribuiranom transakcijom se upravlja na sličan način kao i lokalnom transakcijom. Na kraju transakcije, aplikacija zahteva da se transakcija ili potvrdi ili poništi. Distribuiranim commit-om (potvrđivanjem) mora biti upravljano drugačije od strane menadžera transakcija kako bi se smanjio rizik da kvar u mreži dovede do situacije kada jedan od menadžera resursa potvrđuje umesto da poništi transakcije zbog greške izazvane različitim razlozima. Ovu kritičnu situaciju moguće je rešiti upravljanjem procesa potvrđivanja u dve faze, takođe poznatim kao dvofazno potvrđivanje:

* Faza pripreme: Kada menadžer transakcija primi zahtev za potvrdu (commit), šalje komandu za pripremu svim menadžerima resursa koji su uključeni u transakciju. Svaki upravljač resursa zatim obavlja sve potrebne radnje kako bi transakcija bila trajna, a svi baferi koji sadrže bilo kakve logičke slike za druge transakcije se ispuštaju na disk. Kada svaki menadžer resursa završi fazu pripreme, vraća uspeh ili neuspeh faze pripreme upravljaču transakcija.
* Faza potvrde: Ako upravljač transakcija dobije uspešne pripreme od svih menadžera resursa, šalje komandu za potvrdu (commit) svakom menadžeru resursa. Ako svi menadžeri resursa prijave uspešnu potvrdu, upravljač transakcija šalje obaveštenje o uspehu aplikaciji. Ako neki menadžer resursa prijavi neuspeh pri pripremi, upravljač transakcija šalje izjavu o poništavanju (rollback) svakom upravljaču resursa i obaveštava aplikaciju o neuspehu potvrde.

## **2.3. Kontrola transakcija**

Sledeće komande se koriste za kontrolu transakcija:

* **COMMIT**: Ključna reč COMMIT se koristi za trajno čuvanje promena napravljenih unutar transakcije. Kada se izvrši COMMIT naredba, sve izmene napravljene unutar transakcije se zapisuju u bazu podataka i transakcija se završava. Nakon potvrde, promene postaju vidljive drugim transakcijama.
* **ROLLBACK**: ROLLBACK se koristi za poništavanje promena napravljenih unutar transakcije. Ako nešto krene loše tokom transakcije i želite da odbacite sve promene koje su do tada napravljene, možete izdati ROLLBACK naredbu. Ona vraća bazu podataka u stanje u kojem je bila pre početka transakcije.
* **SAVEPOINT**: SAVEPOINT vam omogućava da definišete međutačke unutar transakcije na koje možete da se vratite. Pomaže u kreiranju manjih jedinica unutar transakcije, omogućavajući vam selektivno vraćanje na određenu tačku umesto da se vraćate na celu transakciju. SAVEPOINT se često koristi u složenim transakcijama gde želite finiju kontrolu nad procesom poništavanja.
* **RELEASE** **SAVEPOINT**: RELEASE SAVEPOINT se koristi za uklanjanje prethodno definisanog savepoint-a unutar transakcije. Kada se savepoint oslobodi, više se ne možete vratiti na tu određenu tačku pomoću rollback-a.
* **SET** **TRANSACTION**: Naredba SET TRANSACTION se koristi za specificiranje karakteristika povezanih sa transakcijom, kao što su nivo izolacije i režim pristupa transakcije. Omogućava vam da kontrolišete način interakcije transakcije sa drugim transakcijama i način sticanja zaključavanja nad objektima baze podataka.

## **2.4. Stanja transakcije**

Ovo označava kako se transakcije odvijaju tokom svog životnog ciklusa. Opisuje trenutno stanje transakcije kao i način na koji će se transakcija obraditi u budućnosti. Ova stanja definišu pravila koja određuju da li će se transakcija potvrditi ili poništiti.

A diagram of a state

Description automatically generated with low confidence

Slika 2. Stanja transakcije

Hajde da opišemo stanje svake transakcije u SQL Serveru:

* Aktivno stanje: Transakcija je u aktivnom stanju dok se instrukcije transakcije izvršavaju. Prelazi u stanje "delimično potvrđeno" ako su sve operacije čitanja i pisanja izvršene bez grešaka. Ako neka instrukcija ne uspe, prelazi u stanje "neuspele transakcije".
* Delimično potvrđeno stanje: Kada su sve operacije čitanja i pisanja završene, promena se čuva u glavnoj memoriji ili lokalnom baferu. Stanje će preći u "potvrđeno stanje" ako su promene trajno sačuvane u bazi podataka. U suprotnom, prelazi u "neuspele transakcije".
* Neuspelo stanje: Transakcija prelazi u stanje neuspele kada neka od instrukcija transakcije ne uspe ili trajna modifikacija u bazi podataka ne uspe.
* Poništeno stanje: Transakcija prelazi iz stanja neuspele transakcije u stanje poništeno kada se dogodi neka vrsta greške. Promene se uklanjaju ili poništavaju jer su ove promene napravljene samo u lokalnom baferu ili glavnoj memoriji u prethodnim stanjima.
* Potvrđeno stanje: Transakcija je završena i prelazi u ovo stanje kada su promene trajno sačuvane u bazi podataka i završene u "završenom stanju".
* Završeno stanje: Ako nema poništavanja i transakcija je u "potvrđenom stanju", sistem je konzistentan i spreman za novu transakciju dok je stara završena.

# 3. Primeri transakcija

Hajde da uzmemo neke primere kako bismo razumeli kako možemo implementirati transakcije u SQL Serveru. Ovde ćemo koristiti tabelu "Product" kako bismo demonstrirali sva stanja transakcije.

Sledeća SQL skripta kreira tabelu "Product" u izabranoj bazi podataka:

CREATE TABLE Product (

Product\_id INT PRIMARY KEY,

Product\_name VARCHAR(40),

Price INT,

Quantity INT)

Primer 1. Kreiranje Product tabele

Zatim, izvršenjem sledeće skripte upisujemo podatke sa kojima ćemo demonstrirati primer:

INSERT INTO Product VALUES(111, 'Mobile', 10000, 10),

(112, 'Laptop', 20000, 15),

(113, 'Mouse', 300, 20),

(114, 'Hard Disk', 4000, 25),

(115, 'Speaker', 3000, 20);

Primer 2. Dodavanje podataka u Product tabelu

Te na taj način dobijamo tabelu koja izgleda ovako:

A picture containing text, screenshot, font, number

Description automatically generated

Slika 3. Product tabela

## **3.1. Primer COMMIT transakcije**

Dobra je ideja da podelimo SQL naredbe korištene u transakciji na više logičkih delova. Zatim možemo odlučiti da li želimo da potvrdimo ili poništimo podatke. Sledeći koraci ilustruju kako kreirati transakciju:

* Započnite transakciju koristeći naredbu BEGIN TRANSACTION.
* Napišite SQL naredbe i podelite ih prema svojim potrebama.
* Koristite naredbu COMMIT da biste završili transakciju i trajno sačuvali promene.

Ispod su komande koje demonstriraju COMMIT transakciju u SQL Serveru:

-- Start a new transaction

BEGIN TRANSACTION

-- SQL Statements

INSERT INTO Product VALUES(116, 'Headphone', 2000, 30)

UPDATE Product SET Price = 450 WHERE Product\_id = 113

-- Commit changes

COMMIT TRANSACTION

Primer 3. Korišćeni upit za COMMIT transakciju

Ako ne bude prijavljena greška, videćemo sledeći rezultat u kojem se svaka SQL naredba transakcije izvršava nezavisno:

A screenshot of a computer

Description automatically generated with medium confidence

Slika 4. Primer COMMIT transakcije

INSERT i UPDATE naredbe ne mogu se poništiti nakon što je transakcija potvrđena. Kada proverimo tabelu nakon potvrde operacije, videćemo sledeće podatke:

A screenshot of a computer

Description automatically generated with low confidence

Slika 5. Product tabela nakon COMMIT transakcije

## **3.2. Primer ROLLBACK transakcije**

Koristićemo ROLLBACK naredbu da poništimo sve transakcije koje još uvek nisu sačuvane u bazi podataka i vratimo se na tačku gde je transakcija započela. Sledeći primer objašnjava ROLLBACK operaciju u SQL Serveru:

-- Start a new transaction

BEGIN TRANSACTION

-- SQL Statements

UPDATE Product SET Price = 5000 WHERE Product\_id = 114

DELETE FROM Product WHERE Product\_id = 116

Primer 4. Transakcija koja neće sačuvati izmene u bazi

Nakon izvršenja gornje transakcije, možemo videti da će se ona uspešno izvršiti. Međutim, neće uticati na promene u bazi podataka jer sve dok ne izvršimo COMMIT ili ROLLBACK naredbu, promene ne mogu postati trajne. Stoga imamo opciju da koristimo ROLLBACK naredbu transakcije da poništimo sve operacije nad bazom podataka. Evo celokupne transakcije:

-- Start a new transaction

BEGIN TRANSACTION

-- SQL Statements

UPDATE Product SET Price = 5000 WHERE Product\_id = 114

DELETE FROM Product WHERE Product\_id = 116

--Undo Changes

ROLLBACK TRANSACTION

Primer 5. ROLLBACK transakcija

## **3.3. Upotreba globalne promenljive @@Error u transakcijama**

Ova promenljiva se koristi za proveru da li postoji greška ili ne. Sledeći primer objašnjava njen koncept. Prvo ćemo započeti transakciju pomoću BEGIN naredbe, zatim ćemo napisati dve INSERT naredbe. Zatim ćemo koristiti globalnu sistemsku promenljivu @@ERROR u IF naredbi da proverimo da li postoji greška. Ako je vrednost veća od 0, to znači da postoji neka greška. Tada će se transakcija poništiti; u suprotnom, transakcija će biti potvrđena (committed).

BEGIN TRANSACTION

INSERT INTO Product VALUES(115,'Speaker', 3000, 25)

-- Check for error

IF(@@ERROR > 0)

BEGIN

ROLLBACK TRANSACTION

END

ELSE

BEGIN

COMMIT TRANSACTION

END

Primer 6. Provera postojanja grešaka u transakciji

Kada se izvrši gornja transakcija, primetićemo da je poništena. To je zbog našeg pokušaja da unesemo duplu vrednost u kolonu primarnog ključa.

## **3.4. Transakcije sa automatskim poništavanjem**

Većina transakcija sadrži više od jednog upita. Prilikom izvršavanja transakcije, ukoliko bilo koja SQL naredba proizvede grešku, neće biti izvršene nikakve promene u bazi podataka, a preostale naredbe neće biti izvršene. Ovaj koncept se naziva automatsko poništavanje transakcije u SQL Serveru. Koristimo jednostavan primer da bismo demonstrirali ovaj proces.

BEGIN TRANSACTION

INSERT INTO Product VALUES(118, 'Desktop', 25000, 15)

UPDATE Product SET Quantity = 'ten' WHERE Product\_id = 113

SELECT \* FROM Product

COMMIT TRANSACTION

Primer 7. Automatsko poništavanje transakcije

Ova transakcija proizvodi sledeći rezultat:

A screenshot of a computer

Description automatically generated with medium confidence

Slika 6. Rezultat pokretanja neispravne skripte

U ovom rezultatu možemo videti da je INSERT naredba uspešno izvršena. Međutim, izvršavanje UPDATE naredbe naišlo je na grešku zbog problema sa konverzijom tipova podataka. U ovom slučaju, SQL Server ne dozvoljava nikakve promene u bazi podataka, što znači da INSERT operacija ne dodaje nikakvu vrednost, a SELECT naredba se ne izvršava.

## **3.5. Savepoint u transakcijama**

Savepoint ubacuje poseban marker u transakciju koji nam omogućava da poništimo sve promene izvršene posle savepoint-a. Takođe se koristi za poništavanje određenog dela transakcije umesto cele transakcije. Možemo ga definisati koristeći SAVE TRANSACTION sp\_name naredbe. Sledeći primer će objasniti upotrebu savepoint-a u transakcijama koji čuva INSERT naredbu, ali poništava DELETE naredbu.

BEGIN TRANSACTION

INSERT INTO Product VALUES(117, 'USB Drive', 1500, 10)

SAVE TRANSACTION InsertStatement

DELETE FROM Product WHERE Product\_id = 116

SELECT \* FROM Product

ROLLBACK TRANSACTION InsertStatement

COMMIT

SELECT \* FROM Product;

Primer 8. Transakcija sa savepoint-om

Pogledajte rezultat ispod gde možemo videti da je proizvod sa ID-om 116 izbrisan, a 117 je umetnut u prvi rezultat. Međutim, u drugom rezultatu, operacija brisanja je poništena zbog savepoint-a.

A screenshot of a computer

Description automatically generated with medium confidence

Slika 7. Rezultat pokretanja transakcije sa savepoint-om

### 3.5.1. Kako osloboditi savepoint u transakciji?

Release savepoint se koristi da se ukloni imenovani savepoint iz trenutne transakcije, bez vraćanja rezultata upita izvršenih posle savepoint-a. SQL Server ne pruža posebnu komandu za oslobađanje savepoint-a. Umesto toga, oni se automatski oslobađaju na kraju COMMIT ili ROLLBACK transakcije, tako da ne moramo brinuti o njima u međuvremenu.

## **3.6. Implicitne transakcije**

Možemo definisati implicitnu transakciju omogućavanjem opcije IMPLICIT\_TRANSACTIONS. Sledeći primer će lako objasniti ovaj koncept:

SET IMPLICIT\_TRANSACTIONS ON

UPDATE Product SET Quantity = 10 WHERE Product\_id = 113

SELECT

IIF(@@OPTIONS & 2 = 2,

'Implicit Transaction Mode ON',

'Implicit Transaction Mode OFF'

) AS 'Transaction Mode'

SELECT @@TRANCOUNT AS OpenTrans

COMMIT TRANSACTION

SELECT @@TRANCOUNT AS OpenTrans

Primer 9. Implicitna transakcija

U ovoj transakciji smo koristili dve opcije: @@OPTION i @@TRANCOUNT. @@OPTION pruža informacije o trenutnim SET opcijama, a @@TRANCOUNT pruža broj BEGIN TRANSACTION instrukcija u trenutnoj sesiji.

Izvršavanje ove transakcije će vratiti sledeći rezultat:

A screenshot of a computer

Description automatically generated with medium confidence

Slika 8. Rezultat pokretanja date implicitne transakcije

## **3.7. Eksplicitne transakcije**

Eksplicitna transakcija se mora definisati kroz BEGIN TRANSACTION naredbu jer ona označava početak eksplicitne transakcije. Možemo definisati eksplicitnu transakciju u SQL Serveru na sledeći način:

BEGIN TRANSACTION [ {trans\_name | @trans\_name\_variable }

[WITH MARK ['description']]]

Primer 10. Sintaksa eksplicitne transakcije

U sintaksi, opcija trans\_name označava jedinstveno ime transakcije, dok @trans\_name\_variable označava korisnički definisanu promenljivu koja čuva ime transakcije. Konačno, opcija MARK nam omogućava da obeležimo određenu transakciju u log fajlu.

Eksplicitna transakcija kroz BEGIN TRANSACTION naredbu stiče zaključavanje u zavisnosti od nivoa izolacije resursa vezanih za transakciju. To pomaže u smanjenju problema sa zaključavanjem. Pogledajmo sledeći primer:

BEGIN TRANSACTION

UPDATE Product SET Quantity = 15 WHERE Product\_id = 114

SELECT @@TRANCOUNT AS OpenTrans

COMMIT TRANSACTION

SELECT @@TRANCOUNT AS OpenTrans

Primer 11. Eksplicitna transakcija

Evo rezultata:

A screenshot of a computer

Description automatically generated with low confidence

Slika 9. Rezultat pokretanja date eksplicitne transakcije

## **3.8. Označene transakcije**

Označena transakcija se koristi za dodavanje opisa određenoj transakciji u log fajlovima. Možemo je koristiti kao tačku oporavka umesto datuma i vremena prilikom vraćanja baze podataka na prethodno stanje. Moramo znati da se oznaka dodaje u log fajlove samo kada označena transakcija vrši izmene nad bazom podataka. Možemo razumeti njen koncept kroz sledeći primer.

Pretpostavimo da smo slučajno izmenili bazu podataka i ne znamo tačan trenutak izmene podataka; u tom slučaju, vraćanje podataka može potrajati dugo vremena. Međutim, ako koristimo označene transakcije, to može biti korisno sredstvo za određivanje tačnog vremena izmena podataka.

Sledeća sintaksa ilustruje označenu transakciju u SQL Serveru:

BEGIN TRANSACTION trans\_name WITH MARK 'description';

Ovde moramo definisati ime transakcije, a zatim dodati opciju WITH MARK. U sledećem primeru ćemo izbrisati zapise i dodati oznaku u log fajl:

BEGIN TRANSACTION DeleteProduct WITH MARK 'Deleted Product with id = 117'

DELETE Product WHERE Product\_id = 117

COMMIT TRANSACTION DeleteProduct

Primer 12. Označena transakcija

Tabela logmarkhistory je uključena u msdb bazu podataka i čuva informacije o svakoj označenoj transakciji koja je potvrđena. Izvršite sledeću naredbu da biste dobili detalje iz tabele logmarkhistory:

SELECT \* FROM msdb.dbo.logmarkhistory

## **3.9. Imenovane transakcije**

Ime transakcije u SQL Serveru omogućava nam da identifikujemo i upravljamo transakcijama pomoću imena. Ovo je posebno korisno kada radimo sa više transakcija u istom upitu ili kada želimo da imamo jasnu referencu na određenu transakciju.

Evo sintakse za definisanje imenovane transakcije u SQL Serveru:

BEGIN TRANSACTION AddProduct

INSERT INTO Product VALUES(118, 'Desktop', 25000, 15)

UPDATE Product SET Product\_name = 'Pen Drive'

WHERE Product\_id = 117

COMMIT TRANSACTION AddProduct

Primer 13. Korišćenje imenovanih transakcija

# 4. Izolacija transakcija

## **4.1. Anomalije izazvane konkurentnošću**

Izolacija je jedno od svojstava SQL transakcija. Odvajanje transakcija jedne od druge kako bi se očuvala integritet podataka u bazi podataka naziva se izolacija.

Pri razvoju velikih i javno dostupnih aplikacija mrežnog tipa, gde veliki broj korisnika pristupa istoj bazi podataka, istoj tabeli i to istovremeno, može se javiti situacija istovremenog pristupa podacima. Razmotrićemo ovu situaciju i mogućnosti pojave konflikata u 4 dela:

* Gubitak podataka
* Prljava čitanja
* Fantomsko čitanje
* Analiza nekonzistentnosti

### 4.1.1. Gubitak podataka

Gubitak podataka, takođe poznat kao pisanje koje se preklapa ili nerepetabilno pisanje, odnosi se na situaciju u kojoj jedna transakcija prepisuje podatke koji su izmenjeni od strane druge transakcije, čime se gube izmene. Ovo može da se dogodi kada dve ili više transakcija pokušaju da istovremeno ažuriraju iste podatke, pri čemu izmene jedne transakcije budu prepisane izmenama druge transakcije, što dovodi do nekonzistentnosti podataka.

Hajde da uzmemo za primer - Pretpostavimo da pristupaju dva korisnika istoj tabeli, u istom trenutku, kako bi ažurirali isti red. Svaka transakcija nije svesna druge transakcije. Korisnik A ažurira red, a zatim korisnik B ažurira isti red. Šta se dogodilo ovde je da je poslednja transakcija koju je napravio korisnik B pregazila ažurirani zapis korisnika A i korisnik A je izgubio svoje podatke u tabeli.

### 4.1.2. Prljava čitanja

Prljava čitanja se javljaju kada jedna transakcija čita podatke koji su izmenjeni od strane druge transakcije koja još uvek nije potvrđena ili poništena. Drugim rečima, transakcija čita nepotvrđene podatke iz druge transakcije. Ovo može dovesti do nekonzistentnosti ako transakcija koja vrši prljava čitanja zavisi od podataka koji kasnije budu poništeni.

Uzmimo drugi primer - Pretpostavimo da korisnik A i korisnik B pristupaju redu tabele istovremeno. Korisnik A želi da pročita, a korisnik B želi da ažurira red. Transakcije se izvršavaju u deliću vremenske razlike. Dakle, dok korisnik B još uvek nije ažurirao red, korisnik A čita taj red i dobija stari zapis koji možda nije tačan za njegovu operaciju. Ova situacija se naziva prljavim čitanjem.

### 4.1.3. Fantomsko čitanje

Fantomsko čitanje se javlja kada transakcija čita skup zapisa na osnovu određenog uslova, a tokom trajanja transakcije, druga paralelna transakcija unosi ili briše zapise koji odgovaraju istom uslovu. Kao rezultat toga, kada prva transakcija ponovo izvrši isti upit, primetiće dodatne ili nedostajuće zapise ("fantomskie zapise") koji nisu bili prisutni pri prvobitnom čitanju. Ova pojava može dovesti do nekonzistentnih rezultata ako transakcija zavisi od doslednog skupa podataka.

Uzmimo opet drugi primer - Pretpostavimo da je korisniku A dozvoljeno da unese red, ali istovremeno je korisnik B uneo taj red. Sada, kada korisnik A pokuša da unese red, ne može. Zbog toga, on će se naljutiti i reći: "Hej, ti si potvrdio da je ovo dostupno za mene za unos, ali si me prevario i dao nekom drugom da to uradi!". Ovaj problem se može javiti prilikom rezervacije vozne karte ili karte za film.

### 4.1.4. Analiza nekonzistentnosti

Analiza nekonzistentnosti odnosi se na proces evaluacije potencijalnih nekonzistentnosti koje mogu da se jave usled paralelnih transakcija i odabranog nivoa izolacije. Uključuje analizu efekata fenomena poput prljavih čitanja, fantomskih čitanja i drugih anomalija radi utvrđivanja nivoa doslednosti i ispravnosti podataka u bazi podataka. Ova analiza pomaže pri odabiru odgovarajućeg nivoa izolacije radi obezbeđivanja integriteta podataka uz istovremeno zadovoljavanje zahteva za konkurentnost.

Uzmimo isti primer korisnika A i korisnika B. Pretpostavimo da korisnik A izvršava transakciju koja ima tri upita - stored proceduru ili transakciju ili pojedinačni upit sa serijom. Prvi upit je za čitanje reda tabele, drugi upit je za ažuriranje tog reda, a treći upit je da ponovo pročita taj red. Time korisnik A želi da generiše izveštaj. Kao što znamo, korisnik B nije gospodin, uvek kvari nameru korisnika A, pristupio je redu tabele između dva upita za čitanje korisnika A i izvršio neku operaciju poput brisanja! Sada je korisnik A već izmenio podatke i kada poželi da ih ponovo pročita, iznenađen je! Dobio je nekonzistentnost u podacima.

## **4.2. Nivoi izolacije**

Iz prethodno navedenih tačaka, shvatili smo da - budući da je korisnik A slab, korisnik B se uvek poigrava sa korisnikom A i prisilno obavlja svoj posao preko reda. Ali kao administratori baze podataka ili programeri, moramo da pomognemo korisniku A. Sada imamo oružje koje se zove "nivo izolacije" koje možemo koristiti da pomognemo korisniku A da održi svoj integritet.

Nivo izolacije predstavlja zaključavanje reda prilikom izvršavanja određenog zadatka, tako da druge transakcije ne mogu pristupiti ili moraju čekati da se trenutna transakcija završi.

SQL Server pruža 5 nivoa izolacije koji se mogu primeniti u SQL transakcijama radi održavanja konkurentnosti podataka u bazi podataka.

### 4.2.1. Read Uncommitted (Čitanje bez potvrde)

Kada je ovaj nivo podešen, transakcija može čitati nepotvrđene podatke što može dovesti do problema prljavog čitanja. Sa ovim nivoom izolacije, dozvoljavamo transakciji da čita podatke koji su ažurirani od strane druge transakcije, a još nisu potvrđeni. Pretpostavimo da Korisnik A pokušava da pročita red koji ažurira Korisnik B. Ovde dozvoljavamo Korisniku A da čita nepotvrđene podatke, odnosno stare podatke.

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

### 4.2.2. Read Committed (Čitanje sa potvrdom)

Ovo sprečava prljavo čitanje. Kada je ovaj nivo podešen, transakcija ne može čitati podatke koji su u procesu izmene od strane druge transakcije. To će naterati korisnika da čeka završetak trenutno aktivne transakcije. Pretpostavimo da Korisnik A pokušava da pročita red koji ažurira Korisnik B. Ovde tražimo od Korisnika A da sačeka da Korisnik B završi svoj ažurirajući zadatak i dajemo ažurirane/tačne podatke Korisniku A. Međutim, problem sa ovim nivoom je da ne može rešiti problem fantomskog čitanja ili analize nekonzistentnosti, odnosno traži od Korisnika A da sačeka za čitanje, ali ne i za ažuriranje ili unos.

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

### 4.2.3. Repeatable Read (Ponovljivo čitanje)

Ovaj nivo obavlja sve što radi Read Committed, ali ima još jednu prednost. Korisnik A će čekati da se izvrši transakcija koju izvršava Korisnik B kako bi izvršio i svoj ažurirajući upit, slično kao kod čitanja. Međutim, unos ne čeka, što takođe stvara problem fantomskog čitanja.

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

### 4.2.4. Snapshot (Snimak)

Ovaj nivo pravi snimak trenutnih podataka. Svaka transakcija radi sa svojom kopijom podataka. Kada Korisnik A pokuša da ažurira, unese ili pročita bilo šta, tražimo od njega da ponovo proveri red tabele od početka izvršavanja, kako bi mogao raditi sa svežim podacima. Sa ovim nivoom, ne pridajemo potpuno poverenje Korisniku A da će raditi sa svežim podacima, ali dajemo mu visok nivo očuvanja doslednosti podataka.

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

### 4.2.5. Serializable (Serijski)

Ovo je najviši nivo izolacije koji pruža SQL Server transakcija. Možemo sprečiti problem fantomskog čitanja implementiranjem ovog nivoa izolacije. On traži od Korisnika A da sačeka trenutnu transakciju za bilo koju vrstu operacije koju želi da izvrši.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Nivo izolacije takođe ima problem koji se naziva "Deadlock" gde "Obe transakcije zaključavaju objekat i čekaju da druga završi posao". Deadlock je veoma opasan jer smanjuje konkurentnost i dostupnost baze podataka i objekata baze podataka.

# 5. Zaključavanje (Locks) u SQL Serveru

## **5.1. Uvod**

Zaključavanje je mehanizam povezan sa tabelom koji ograničava neovlašćeni pristup podacima. Koristi se pretežno za rešavanje problema konkurentnosti u transakcijama. Možemo primeniti zaključavanje na nivou reda, baze podataka, tabele i stranice. SQL Server podrazumevano koristi zaključavanje na nivou slogova, što minimizuje broj negativnih sporednih efekata konkurentnog izvršavanja.

Znamo da više korisnika može pristupiti bazama podataka istovremeno. Zbog toga je zaključavanje neophodno za uspešnu transakciju i štiti podatke od korupcije ili poništavanja kada više korisnika pokušava čitati, pisati ili ažurirati bazu podataka. Obično je zaključavanje (brava) struktura u memoriji sa vlasnicima, tipovima i hešom resursa koje treba da zaštiti. Kao struktura u memoriji, veličina jedne brave je 96 bajtova.

Važno je razumeti da je "zaključavanje dizajnirano da osigura integritet i doslednost podataka omogućavajući istovremeni pristup podacima, jer prisiljava svaku transakciju da prođe ACID test. Kada više korisnika pristupa bazi podataka da bi izmenili njene podatke u isto vreme, implementira se kontrola konkurentnosti."

## **5.2. Režimi zaključavanja (Lock Modes)**

Režim zaključavanja se koristi kako bi se sprečilo da drugi korisnici čitaju ili menjaju zaključani resurs. Može se kategorisati u sledećih šest tipova navedenih ispod:

* Isključivo zaključavanje (X)
* Deljeno zaključavanje (S)
* Ažurirajuće zaključavanje (U)
* Zaključavanje nameravanja (I)
* Šema zaključavanje (Sch)
* Zaključavanje masovnog ažuriranja (BU)

### 5.2.1. Isključivo zaključavanje

Isključivo zaključavanje je korisno u DML (Data Manipulation Language) operacijama kao što su INSERT, UPDATE ili DELETE naredbe. Ovo zaključavanje, kada se primeni na transakciju, sprečava druge osobe da pristupe zaključanim resursima. To znači da isključivo zaključavanje može držati samo jednu transakciju na resursu istovremeno. Korisnik ovog zaključavanja se naziva pisac. Ovo zaključavanje se primenjuje kada transakcija želi da izmeni podatke na stranici ili u redu. Može ga držati samo stranica ili red kada nema drugog deljenog ili isključivog zaključavanja na cilju.

### 5.2.2. Deljeno zaključavanje

Kada se primeni deljeno zaključavanje na stranici ili redu, ono će biti rezervisano samo u svrhu čitanja. To znači da nijedna druga transakcija ne može menjati zaključani resurs dok je zaključavanje aktivno. Kao što naziv implicira, nekoliko transakcija može držati ovo zaključavanje na istom resursu istovremeno. Korisnik ovog zaključavanja se naziva čitač. Ovo zaključavanje takođe dozvoljava i operacije pisanja, ali neće biti dozvoljene DDL (Data Definition Language) promene.

### 5.2.3. Ažurirajuće zaključavanje

Ažurirajuće zaključavanje je isto što i isključivo zaključavanje, ali je dizajnirano da bude prilagodljivije. Transakciji koja već drži deljeno zaključavanje može se dati ažurirajuće zaključavanje. U takvim slučajevima, ažurirajuće zaključavanje može držati još jedno deljeno zaključavanje na ciljanoj stranici ili redu. Ovo zaključavanje se može promeniti u isključivo zaključavanje kada transakcija koja drži ažurirajuće zaključavanje želi da izmeni podatke. Obično se koristi kada server filtrira zapise radi ažuriranja. Takođe, predstavlja rešenje za problem ciklične mrtve tačke.

### 5.2.4. Zaključavanje nameravanja

Zaključavanje nameravanja je način na koji transakcija obaveštava druge transakcije o svojoj nameri da dobije zaključavanje. Ovo zaključavanje ima za cilj da spreči drugu transakciju da dobije zaključavanje (bravu) na sledećem objektu u hijerarhiji, kako bi se osiguralo da se izmene podataka pravilno izvrše. Ukazuje na to da se ovo zaključavanje koristi za stvaranje hijerarhije zaključavanja. To je važan tip brave u aspektu performansi. Može se podeliti u tri tipa:

1. Deljeno zaključavanje nameravanja (IS): Ako stranica ili red drže ovo zaključavanje, tada transakcija namerava da čita resurse u nižoj hijerarhiji tako što nezavisno dobija deljena zaključavanja (S) na tim resursima.
2. Isključivo zaključavanje nameravanja (IX): Ako stranica ili red drže ovo zaključavanje, transakcija namerava da menja neke resurse niže hijerarhije tako što nezavisno dobija isključiva zaključavanja (X) na tim resursima.
3. Ažurirajuće zaključavanje nameravanja (IU): Ovo zaključavanje se može dobiti samo na nivou stranice i prelazi u isključivo zaključavanje nameravanja kada se završi operacija ažuriranja.

### 5.2.5. Šema zaključavanja

Šema zaključavanja se koristi u SQL Serveru kada se izvršava operacija koja zavisi od šeme tabele. Može se podeliti na dva tipa:

1. Modifikacija šeme (Sch-M): Koristi se kada se izvršava DDL naredba i sprečava pristup podacima zaključanog objekta dok se menja struktura objekta.
2. Stabilnost šeme (Sch-S): Koristi se kada se kompajlira i izvršava upit koji zavisi od šeme, kao i kada se generiše plan izvršenja.

### 5.2.6. Zaključavanje masovnog ažuriranja

Ovo zaključavanje se koristi za kopiranje masovnih podataka u tabelu korišćenjem TABLOCK sugestije. Korisnik ga obično koristi kada želi da unese veliku količinu podataka u bazu podataka.

## **5.3. Primer**

U SQL Serveru, dm\_tran\_locks predstavlja sistemski prikaz (view) koji pruža informacije o zaključavanjima (bravama) stečenim od strane aktivnih transakcija. Pripada porodici Dynamic Management Views (DMV), koja je skup virtuelnih tabela koje otkrivaju interne informacije o SQL Server instanci.

Prikaz dm\_tran\_locks sadrži jedan red za svako zaključavanje koje drži transakcija. Pruža detalje o transakciji, zaključanom resursu, režimu zaključavanja i drugim relevantnim informacijama. Ovaj prikaz je posebno koristan za praćenje i otklanjanje problema povezanih sa zaključavanjem, jer nam omogućava da pregledamo trenutno stanje zaključavanja unutar baze podataka.

Evo nekih od ključnih kolona dostupnih u prikazu dm\_tran\_locks:

* request\_session\_id: Identifikator sesije koja je zatražila zaključavanje (bravu).
* resource\_database\_id: Identifikator baze podataka u kojoj se nalazi zaključani resurs.
* resource\_type: Vrsta zaključanog resursa (npr. objekat, stranica, ključ).
* resource\_associated\_entity\_id: Identifikator povezanog entiteta unutar vrste resursa (npr. object\_id, page\_id).
* request\_mode: Režim zaključavanja koji je tražila transakcija (npr. SCH-S za stabilnost šeme, X za ekskluzivno).
* request\_status: Status zahteva za zaključavanje (npr. GRANT, WAIT).
* request\_owner\_type: Vrsta vlasnika zaključavanja (npr. transakcija, sesija).
* request\_owner\_id: Identifikator vlasnika zaključavanja (transakcija ili sesija).

Korišćenjem sledećeg upita nad svojom tabelom za demonstraciju korišćenoj u izradi ovog rada dobijam rezultat prikazan na Slici 10.

SELECT dm\_tran\_locks.request\_session\_id,

dm\_tran\_locks.resource\_database\_id,

DB\_NAME(dm\_tran\_locks.resource\_database\_id) AS dbname,

CASE

WHEN resource\_type = 'OBJECT'

THEN OBJECT\_NAME(dm\_tran\_locks.resource\_associated\_entity\_id)

ELSE OBJECT\_NAME(partitions.OBJECT\_ID)

END AS ObjectName,

partitions.index\_id,

indexes.name AS index\_name,

dm\_tran\_locks.resource\_type,

dm\_tran\_locks.resource\_description,

dm\_tran\_locks.resource\_associated\_entity\_id,

dm\_tran\_locks.request\_mode,

dm\_tran\_locks.request\_status

FROM sys.dm\_tran\_locks

LEFT JOIN sys.partitions ON partitions.hobt\_id = dm\_tran\_locks.resource\_associated\_entity\_id

LEFT JOIN sys.indexes ON indexes.OBJECT\_ID = partitions.OBJECT\_ID AND indexes.index\_id = partitions.index\_id

WHERE resource\_associated\_entity\_id > 0

AND resource\_database\_id = DB\_ID()

AND request\_session\_id=58

ORDER BY request\_session\_id, resource\_associated\_entity\_id

Primer 14. Korišćenje dm\_tran\_locks pogleda

A screenshot of a computer

Description automatically generated with medium confidence

Slika 10. Aktivna zaključavanja (brave)

# 6. Literatura

1. *Erkec, Esat (2021). "Transactions in SQL Server for beginners". SQLSHack, [*[*Online*](https://www.sqlshack.com/transactions-in-sql-server-for-beginners/)*], [Accessed: 13-May-2023]*
2. *Microsoft Learn (2006). "ACID properties". [*[*Online*](https://learn.microsoft.com/en-us/previous-versions/office/developer/server-technologies/aa480356(v=msdn.10)?redirectedfrom=MSDN)*], [Accessed: 13-May-2023]*
3. *Vidya Vrat, Agarwal (2019). "Understanding Transactions in SQL Server". C# Corner, [*[*Online*](https://www.c-sharpcorner.com/UploadFile/84c85b/understanding-transactions-in-sql-server/)*], [Accessed: 15-May-2023]*
4. *Javatpoint. "SQL Server Transaction". [*[*Online*](https://www.javatpoint.com/sql-server-transaction#:~:text=A%20transaction%20in%20SQL%20Server,combination%20of%20all%20these%20operations.)*], [Accessed: 16-May-2023]*
5. *Deepak Kumar, Jena (2019). "Using Isolation Level in SQL Transaction". C# Corner, [*[*Online*](https://www.c-sharpcorner.com/blogs/using-isolation-level-in-sql-transaction2)*], [Accessed: 17-May-2023]*
6. *Javatpoint. "Locks in SQL Server". [*[*Online*](https://www.javatpoint.com/locks-in-sql-server)*], [Accessed: 18-May-2023]*
7. *Dimitrijević, Nikola (2017). "All about locking in SQL Server". SQLShack, [*[*Online*](https://www.sqlshack.com/locking-sql-server/)*], [Accessed: 18-May-2023]*

# 7. Listing

Slika 1. Ilustracija transakcija 4

Slika 2. Stanja transakcije 10

Slika 3. Product tabela 11

Slika 4. Primer COMMIT transakcije 12

Slika 5. Product tabela nakon COMMIT transakcije 12

Slika 6. Rezultat pokretanja neispravne skripte 14

Slika 7. Rezultat pokretanja transakcije sa savepoint-om 15

Slika 8. Rezultat pokretanja date implicitne transakcije 16

Slika 9. Rezultat pokretanja date eksplicitne transakcije 17

Slika 10. Aktivna zaključavanja (brave) 26

Primer 1. Kreiranje Product tabele 11

Primer 2. Dodavanje podataka u Product tabelu 11

Primer 3. Korišćeni upit za primer COMMIT transakcije 12

Primer 4. Transakcija koja neće sačuvati izmene u bazi 13

Primer 5. ROLLBACK transakcija 13

Primer 6. Provera postojanja grešaka u transakciji 14

Primer 7. Automatsko poništavanje transakcije 14

Primer 8. Transakcija sa savepoint-om 15

Primer 9. Implicitna transakcija 16

Primer 10. Sintaksa eksplicitne transakcije 17

Primer 11. Eksplicitna transakcija 17

Primer 12. Označena transakcija 18

Primer 13. Korišćenje imenovanih transakcija 18

Primer 14. Korišćenje dm\_tran\_locks pogleda 26