**Univerzitet u Nišu, Elektronski fakultet**

logo_1960_4**Katedra za računarstvo**

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

Mila Rančić 1370

**Predmet: Sistemi za upravljanje bazama podataka**

Sadržaj

[Uvod 3](#_Toc104986970)

[Transkacije 4](#_Toc104986971)

[Transakcije i ACID svojstva 4](#_Toc104986972)

[Transakcije i konkurentost 5](#_Toc104986973)

[Transkacije kod MariaDB 6](#_Toc104986974)

[Nivoi izolacije transakcija kod MariaDB 10](#_Toc104986975)

[Konzistentno čitanje 11](#_Toc104986976)

[Phantom Read 15](#_Toc104986977)

[Mehanizmi eksplicitnog zaključavanja 19](#_Toc104986978)

[Deljeno i isključivo zaključavanje 19](#_Toc104986979)

[Namerno zaključavanje 19](#_Toc104986980)

[Zaključavanje praznine 22](#_Toc104986981)

[Zaključavanje sledećeg ključa 22](#_Toc104986982)

[Zaključavanja namera umetanja 23](#_Toc104986983)

[AUTO-INC zaključavanje 23](#_Toc104986984)

[DEADLOCK 24](#_Toc104986985)

[Zaključak 28](#_Toc104986986)

[Reference 29](#_Toc104986987)

# Uvod

Da bi sistem za upravljanje bazom podataka funkcionisao na prihvatljiv i ispravan način potrebno je da zna kako treba da radi u višekorisničkom okruženju, u kom se upiti izvršavaju istovremeno, odnosno u kome je prisutna konkurentnost. Tu se pre svega mora voditi računa o tome u kakvom je stanju baza u svakom trenutku, tj stanje mora biti konzistentno. Ne sme se ugroziti ispravnost podataka ni u kom trenutku, što znači da se moraju zabraniti polovične, odnosno nepotpuno uspešne operacije, kao i one koje bi ugrozile tačnost ostalih operacija koje su u toku. Sa druge strane, tu su vremeneska ograničenja koja, zbog zahteva o konzistentosti, nije lako ispuniti. Zbog svega ovoga koncept transakcija je jako bitan u bilo kom sistemu za upravljanje bazama podataka. One obezbeđuju da se skup operacija izvršava kao celina, po principu sve ili ništa, čime direktno utiče na to da stanje u bazi bude konzistentno i da se tamo nađu samo ispravni podaci. Pomoću njih programer ima veću kontrolu nad onim što se dešava u bazi. Pored toga, konkurentnost bi bila nezamisliva da nije transakcija.

Kao primer možemo da uzmemo veb sajt za e-trgovinu. Ako poglebamo bazu podataka koju koristi, videćemo dve tabele: jednu u kojoj se skladište narudžbine, a drugu u kojoj se skladište dostupni proizvodi i njihova količinu. Kada kupac kupi proizvod, nova narudžbina mora biti upisana u odgovarajuću tabelu i količina tog proizvoda mora biti smanjena. Bitno je da se ove dve operacije odvijaju u okviru iste transakcije. Na taj način, ako dođe do greške na serveru nakon što je porudžbina izdata, ali pre smanjivanja količine proizvoda, transakcija se poništava i promene se ne upisuju u bazu. Samo ako se sve naredbe izvrše uspešno, transakcija će biti završena. Pored toga, transakcije imaju još jedno važno pomenuto svojstvo: izolovane su jedna od druge, odnosno operacije koje obavlja jedna transakcija nikada ne treba da se mešaju sa drugim. Ukoliko imamo više kupaca koji žele da kupe isti proizvod, prvi koji je izvršio porudžbinu, započinje transakciju, koja će zaključati red tabele koji odgovara proizvodu koji on/ona želi da kupi. Drugi kupac će morati da sačeka dok se prva transakcija ne završi. Na taj način, će se izbeći kupovina proizvoda koga možda nakon prve transakcije više nema.

Rad je organizovan tako da su u prvo poglavlju objašnjene transakcije i ACID svojstva. Zatim u drugom poglavlju obrađene su transakcije kod MariaDB. Nabrojane su neke od osnovnih komandi za rad sa transakcijama, čemu služe i kakva je njihova sintaksa. Kroz primere iz realnog sistema prikazano je kako se radi sa transakcijama u ovom sistemu.

Nakon toga sledi deo koji se bavi problemom konkurentnosti u vezi sa transakcijama. Sledeće poglavlje vezano je za izolovanost, tačnije nivoe izolovanosti kod transakcija. Dat je pregled svih nivoa izolovanosti i šta podrazumeva svaki od njih. Glavni deo tog poglavlja zapravo su efekti koje izazivaju zaključavanja nekom od opcija jer se tu na primerima vidi kako ceo taj sistem radi. Za svaki efekat dat je primer, objašnjeno je na kom nivou izolovanosti se može javiti i zašto se javlja baš na tom nivou.

Na kraju, tu je deo o mehanizmima ekspicitnog zaključavanja. Videćemo koje sve mehanizme zaključavanja podržava MariaDB, kao i problem deadlock-ova.

Za demonstraciju primera i rad sa bazom korišćen je Sequel-Ace tool.

# Transkacije

Transakcije su ključni koncept u sistemima za upravljanje bazama podataka. Relacioni modeli predstavljaju logičku jedinicu obrade podataka kao *transakciju.* Transakcije predstavljaju skup operacija izvedenih u nizu, kao jedna sekvenca. Relacione baze podataka obezbeđuju mehanizme za zaključavanje (*locking mechanism*) kako bi osigurale integritet transakcije. Zbog toga transkacije garantuju konzistentnost i trajnost podataka u bazi. Gotovo je nemoguće implementirati sistem za upravljanje bazama podataka bez odgovarajućeg dela koji bi se bavio upravljanjem transakcijama.

Transakcije se mogu sastojati od pojedinačnih operacija čitanja, pisanja, brisanja ili ažuriranja ili njihove kombinacije. Glavna ideja transakcija je da se, kada bilo koja od operacija koje su deo transkacije vrati grešku, celokupne modifikacije ponište (*rollback-uju)*  kako bi se obezbedio integritet podataka. S druge strane, ako se sve operacije uspešno završe, promene podataka će se primeniti i postaće trajne u bazi podataka.

## Transakcije i ACID svojstva

Sistem za upravljanje bazama podataka mora da obezbedi da operacije budu atomične, kao i da obezbedi konzistentost, izolaciju i trajnost podataka. Ovo se naziva ***ACID*** svojstvom baze podataka zbog reči: ***A****tomicity,* ***C****onsistency,* ***I****solation,* ***D****urability.*

Transakcija je logička izvršna jedinica i nedeljiva je, što znači sve ili ništa. Ovo svojstvo poznato je kao ***atomičnost***. To važi bez obzira na razlog zbog kog operacije nije uspela. Na primer, transakcija može da bude nespešna zbog matematičke greške, pogrešno napisano ime relacije ili čak pad operativnog sistema.

Nakon što se transakcija uspešno izvrši, efekat transakcije mora se održati čak i u slučaju hardverskih grešaka sistema (pada sistema). Ovo svojstvo se naziva ***trajnost***. Dalje, u višekorisničkom okruženju nekoliko korisnika može izvršiti nekoliko transakcija, a svaka transakcija sadrži skup radnji ili operacija. Svaka transakcija treba da se izvrši bez ometanja istovremeno izvršenih transakcija. Ovo svojstvo se naziva ***izolacija***.

Konačno, svojstvo ***konzistentnosti*** nije svojstvo same transakcije, već je poželjan efekat izolacije i atomičnosti transakcije. Konzistentnost baze podataka bavi se poslovnim zahtevima, koji su definisani pravilima, uključujući okidače, ograničenja i bilo koju njihovu kombinaciju. Ako je stanje baze podataka konzistentno neposredno pre izvršavanja transakcije, tada stanje baze podataka mora biti konzistentno i nakon izvršavanja transakcije. Za konzistentnost baze podataka odgovoran je programer koji kreira transakciju.

ACID svojstva nije lako ispuniti. Na primer, trajnost je jako zahtevna jer postoji mnogo faktora koji mogu prouzrokovati gubitak podataka, poput pada operativnog sistema, nestanka struje ili otkaza hard diska. Takođe, računarska arhitektura je prilično složena. Na primer, podaci mogu da prolaze kroz nekoliko slojeva memorije, kao što su glavna memorija, IO baferi i keš memorija diska, sve dok se ne zadrže na disku.

## Transakcije i konkurentost

Osnovni problem u vezi sa konkurentnošću je istovremeni pristup zajedničkom globalnom resursu. Do sukoba neće doći ako programeri ne pišu kod u konkurentnom maniru, već u serijskom, gde se naredbe izvršavaju jedna za drugom. Međutim, u stvarnosti to nije praktično i uzrokuje mnogo gubitaka i dovodi do znatno lošijih performansi.

Kontrola paralelnosti se sastoji u koordinisanju izvršavanja istovremenih transakcija u višekorisničkom okruženju i rešavanju potencijalnih problema koji nastaju kao posledica takvog načina izvršavanja, kao što su nekonzistentni podaci, izgubljena ažuriranja itd. MariaDB ima podršku za konkurentnost, čak i kod aktivnosti čitanja i pisanja i kobminacije. Kod MariaDB, aktivnost pisanja ne blokira aktivnost čitanja, ili obrnuto. MariaDB koristi tehniku koja se naziva ***Multi-Version Concurrenci Control (MVCC)***. Ovu tehniku takođe koristi nekoliko komercijalnih i nekomercijalnih baza podataka, kao što su Oracle, PostgreSQL, Informic, Firebird i CouchDB. MVCC nije jedini način za rukovanje paralelnog izvršenja u RDBMS-u.

# Transkacije kod MariaDB

Kod MariaDB, transakcije se mogu koristiti samo u okviru mehanizama za skladištenje koje ih podržavaju, kao što su InnoDB, TokuDB i SPIDER. U ovom radu fokusiraćemo se na InnoDB i svi primeri će se odnositi na njega.

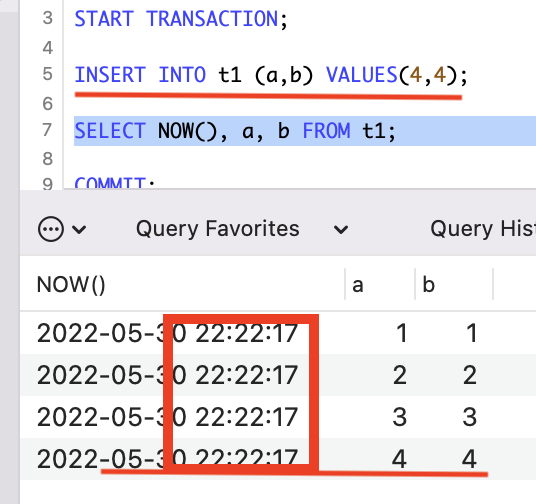
Sada ćemo videti neke od osnovnih komandi pri radu sa transakcijama kod MariaDB. Za eksplicitno zadavanje početka transakcije koristi se komanda ***START TRANSACTION*,** dok se za kraj transakcije koriste ***COMMIT*** ili ***ROLLBACK****.*

START TRANSACTION;

<one or more statements>

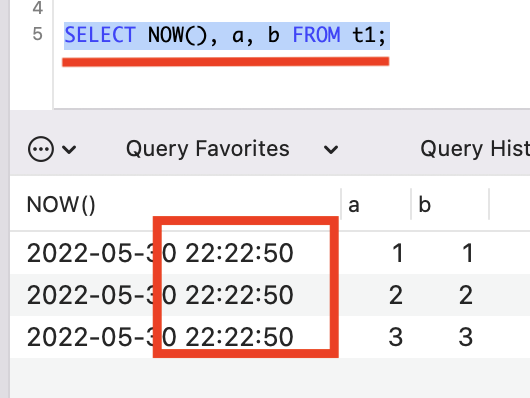
COMMIT;

Naredba ***COMMIT*** snima sve promene trenutne transakcije u bazu podataka. Da  bismo videli kako ***COMMIT***radi na delu, započećemo transakciju u kojoj dodajemo novi red u tabelu:



Slika 1: Prikaz rezultata SELECT naredbe nakon dodavanje novog reda u okviru transakcije

Vidimo da je u okviru te transakcije novi red vidljiv. Sada ćemo pokrenuti upit za čitanje izvan transakcije:

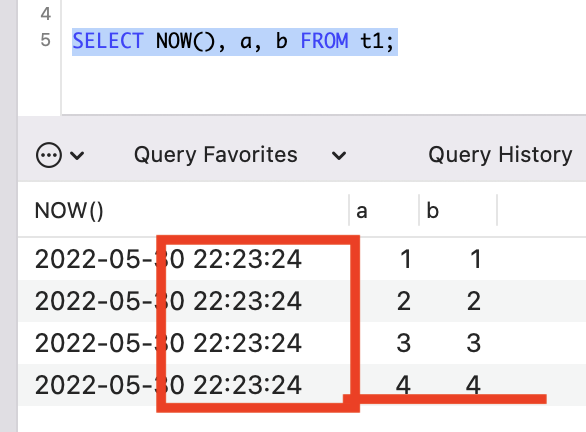


Slika 2: Prikaz rezultata SELECT naredbe VAN transakcije

Ako sada potvrdimo transakciju pomoću COMMIT:



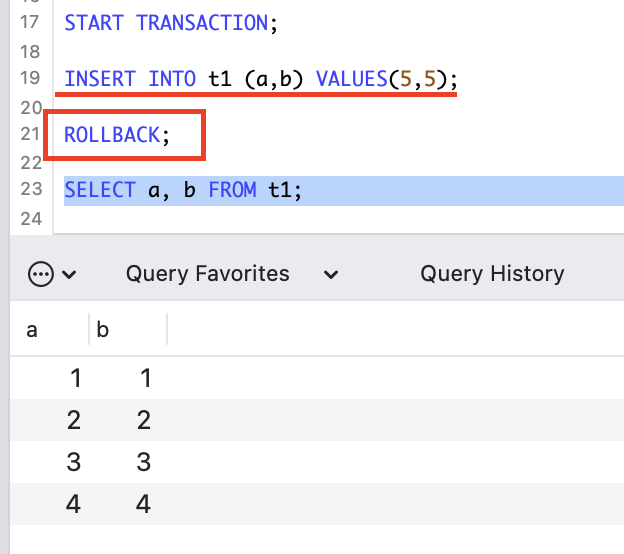
i opet proverimo sadržaj tabele:



Slika 3: : Prikaz rezultata SELECT naredbe nakon završetka transakcije

ovog puta vidimo novi red.

Naredba ***ROLLBACK*** završava transakciju,poništava sve modifikacije podataka koje su urađene tom transakcijom ili poništava sve promene do poslednje ***COMMIT*** naredbe ili savepoint-a. Za naredbu ***ROLLBACK*** ćemo kroz primer prikazati da zaista poništava efekte transakcije, tako što ćemo dodati red, poništiti transakciju i videti da li se taj red nalazi među rezultatima.



Slika 4: Poništavanje transakcije

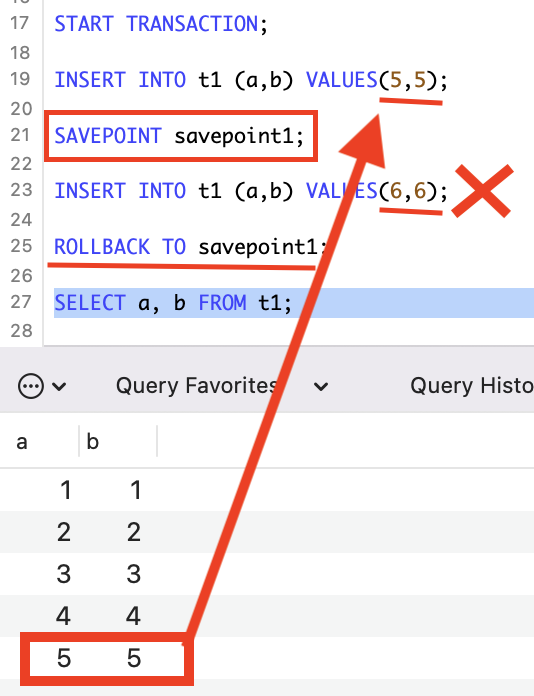
Vidimo da zaista nema reda sa vrednostima 5 za kolone *a* i *b*.

***SAVEPOINT*** naredba kreira tačke unutar grupa transakcija na koje se može vratiti pomoću ***ROLLBACK***. Ovim je obezbeđeno da možemo poništiti promene samo iz dela transkacije, umesto iz celokupne transkacije, čime se poboljšava kontrola nad podacima. Sintaksa za ovu naredbu je:

***SAVEPOINT SAVEPOINT\_NAME;***

,nakon čega bi sledilo:

***ROLLBACK TO SAVEPOINT\_NAME;***



Slika 5: Primer povratka na savepoint

***BEGIN / BEGIN WORK*** je sinonim za ***START TRANSACTION***, ali se ne može koristiti unutar uskladištenih programa, jer su ključne reči za zatvaranje kodnih blokova ***BEGIN*** i ***END.***

Ukoliko imamo više transakcija koje počinju jedna za drugom, moguće je iskoristiti komandu ***START TRANSACTION AND CHAIN*** i na taj način izbeći ponovno pisanje komande ***START TRANSACTION.***

Kod MarijaDB podrazumevano ponašanje je da se svi izrazi izvrsavaju u režimu automatskog upisivanja. To znači da, kada inače nije unutar transakcije, svaka naredba je atomična, kao da je okružena sa ***START TRANSACTION*** i ***COMMIT***. Ne može se koristiti ***ROLLBACK*** da se poništiti efekat. Međutim, ako dođe do greške tokom izvršavanja naredbe, naredba se vraća nazad.

Da bi se ovaj režim onemogućio koristi se naredba ***START TRANSACTION***. Ovom naredbom, režim automatskog upisivanja ostaje onemogućen sve dok se ne završi transakcija sa ***COMMIT*** ili ***ROLLBACK***. Režim automatskog upisivanja se zatim vraća u svoje prethodno stanje. Da bi eksplicitno onemogućili režim automatskog upisivanja, koristi se sledeća naredba:

***SET autocommit = 0;***

*autocommit - je promenljiva sesije i mora biti podešena za svaku sesiju.*

Nakon onemogućavanja režima, promene u tabelama koje su bezbedne za transakcije ne postaju odmah trajne. Potrebno je iskoristiti ***COMMIT*** za čuvanje promene na disku ili ***ROLLBACK*** za ignorisanje promena. Kada MariaDB završi izvršenje transakcije, sva zaključavanja postavljena transakcijom se oslobađaju.

# Nivoi izolacije transakcija kod MariaDB

Izolacija definiše način na koji MariaDB server odvaja svaku transakciju od drugih transakcija koje se istovremeno izvršavaju na serveru i takođe pruža dobru ravnotežu između performansi i pouzdanosti, konzistentnosti i ponovljivosti rezultata kada više transakcija istovremeno vrši promene i postavlja upite. Ako transakcije nisu izolovane, onda bi jedna transakcija mogla da izmeni podatke koje druga transakcija čita, stvarajući tako nedoslednost podataka.

Nivoi izolacije definišu stepen do kojeg transakcija treba da bude izolovana od modifikacija podataka izvršenih bilo kojom drugom transakcijom. Takođe, definišu na koji način će objekti baze podataka, kao što su tabele, biti zaključani. MariaDB podržava sva četiri nivoa izolacije koje definiše SQL-Standard, a to su: ***READ UNCOMMITTED***, ***READ COMMITTED***, ***REPEATABLE READ*** i ***SERIALIZABLE***. Podrazumevani nivo izolacije je ***REPEATABLE READ****.*

Nivo izolacije transakcije može se promeniti pozivanjem sledeće SQL naredbe:

***SET TRANSACTION ISOLATION LEVEL***

***{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }***

Izraz ***SET TRANSACTION ISOLATION LEVEL*** treba pozvati unutar bloka transakcije pre bilo kog upita, jer u suprotnom neće biti moguća promena nivoa. Alternativa je upotreba na sledeći način:

***BEGIN TRANSACTION ISOLATION LEVEL***

***{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }***

Poslednja opcija će promeniti podrazumevani nivo izolovanosti za čitavu bazu:

***ALTER DATABASE <DATABASE NAME>***

***SET DEFAULT\_TRANSACTION\_ISOLATION TO SERIALIZABLE ;***

Nivoi izolacije koje podržava MariaDB su:

* ***READ UNCOMMITTED***: Sa nivoom izolacije READ UNCOMMITTED, za svaku naredbu čitanja koja se izvršava u transakciji, kreira se poseban snimak podataka. Čitanja koriste taj snimak. On može da se sastoji od podataka iz transakcije koja još nije izvršena, pa je moguće da takvi podaci nikada neće postojati u tabeli. Zbog toga predstavlja model sa najslabijim nivoom izolacije, odnosno najmanje je restriktivan po pitanju zaključavanja.
* ***READ COMMITTED***: Kao READ UNCOMMITTED, nivo izolacije READ COMMITTED kreira drugačije snimake za svaku naredbu čitanja u transakciji, ali nikada ne koristi nepotvrđene podatke prilikom pravljenja snimka.
* ***REPEATABLE READ***: Ovo je podrazumevani MariaDB model. Drugi po snazi nivo izolacije transakcije. Sličan je ***READ COMMITTED*** po tome što omogućava samo čitanje podataka koji su potvrđeni, tj potvrđeni i definitivno upisani. Takođe garantuje da se bilo koji pročitani podatak ne može promeniti. Sva čitanja unutar transakcije koriste isti snimak.
* ***SERIALIZABLE***: najjači nivo što se konzistentnosti podataka tiče, najstriktniji po pitanju zaključavanja. Potpuno izoluje efekat jedne transakcije od drugih. Slično je ***REPEATABAL READ***-u sa dodatnim ograničenjem da red izabran od strane jedne transakcije ne može biti promenjen drugom dok se prva transakcija ne završi. Izbegava se fenomen fantomskog čitanja. Loša strana upotrebe ovog modela su performanse.

Izbor najboljeg nivoa izolacije ima veliki uticaj na bazu podataka. Međutim, ne može se reći da je neki nivo bolji od drugog, jer svaki nivo ima svoje prednosti i mane. Izbor zavisi od potreba samog sistema, da li je cilj postići bolje performanse, po cenu javljanja nekih nedoslednosti, ili bazu držati konzistentnom a performanse staviti u drugi plan.

Kako svaki od ovih nivoa funckioniše najlakše je objasniti ako vidimo koje neke od efekata, koji se u pojedinim nivoima javljaju a u drugima ne. Pomenućemo nekoliko i objasniti ih na primeru.

## Konzistentno čitanje

Konzistentno čitanje podrazumeva da MariaDB koristi strategiju sa više verzija sistema, tako da svaki upit ima “snimak” baze podataka u određenom trenutku. Upit vidi promene izvršene transakcijama koje su izvršene pre trenutka u kome je dobio svoj snimak, i nema promena učinjenih kasnijim transakcijama, ili onima koje jos uvek nisu potvrđene. Izuzetak od ovog pravila je da upit vidi promene napravljene ranijim naredbama u okviru ISTE transakcije. Ovaj izuzetak uzrokuje sledeću anomaliju: ako ažuriramo neke redove u tabeli, ***SELECT*** vidi najnoviju verziju ažuriranih redova, ali istovremeno može videti starije verzije bilo kojih drugih redova. Ako druge sesije istovremeno ažuriraju istu tabelu, to znači da možemo videti tabelu u stanju koje nikada nije postojalo u bazi podataka.

Ako je nivo izolacije transakcije ***REPEATABLE READ*** (podrazumevani nivo), sva konzistentna čitanja u okviru iste transakcije čitaju snimak koji je dobijen prvim takvim čitanjem transakciji. Noviji snimak može se dobiti potvrđivanjem trenutne transakcije i izvršavanjem novih upita nakon toga. Sa nivoom izolacije ***READ COMMITTED***, svako čitanje unutar transakcije postavlja i čita svoj sopstveni “sveži” snimak.

Dakle, Konzistentno čitanje je podrazumevani režim u kome MariaDB obrađuje ***SELECT*** naredbe na nivoima izolacije ***READ COMMITTED*** i ***REPEATABLE READ***. Konzistentno čitanje ne postavlja nikakva zaključavanja na tabele kojima pristupa, pa su stoga druge sesije slobodne da modifikuju te tabele u isto vreme kada se vrši dosledno čitanje na tabeli.

Pretpostavimo da koristimo podrazumevani nivo izolacije ***REPEATABLE READ***. Kada izdamo naredbu čitanja tj. ***SELECT***, MariaDB daje našoj transakciji vremensku tačku prema kojoj naš upit vidi bazu podataka. Ako druga transakcija izbriše red i izvrši *commit* nakon te vremenske tačke, nećemo viditi da je red izbrisan. Isto važi i za ***INSERT*** *i* ***UPDATE***naredbe.

Vremensku tačku vezanu sa transakciju  možemo “pomeriti” unapred samo naredbom ***COMMIT*** nakon koje bi sledio ***SELECT*** ili startovanjem transakcije pomoću ***WITH CONSISTENT******SNAPSHOT*** opcije. Takvo ponašanje se naziva multi-versioned concurrency.

Na sledećem primeru videćemo Consistent nonlocking reads kada je nivo izolacije ***REPEATABLE READ***. Oznaka *T (T1, T2, T3 …. Tn)* koristi se za vreme izvršenja gde veći indeks označava kasnije vreme izvršenja.

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |

|  |  |  |
| --- | --- | --- |
| T3 |  |  |

Table 1: Konzistentno čitanje na nivou izolacije REPEATABLE READ

Ovde možemo videti da postoje 2 transakcije - Transakcija 1 i Transakcija 2,  koje se izvršavaju paralelno, od kojih prva čita ukupan broj podataka u tabeli, dok druga dodaje red u istu tu tabelu. Prva transakcija vidi da je broj redova u tebeli jednak broju 4. Nakon toga, druga transakcija dodaje red i kao rezultat ***COUNT*** upita dobija vrednost 5. Transakcija 2 se zatim završava naredbom ***COMMIT***, posle čega Transakcija 1 ponovi svoju poslednju operaciju rezultat je opet 4. Razlog za to, kao što smo već objasnili, leži u činjenici da transakcija svoj snimak dobija pri izvršenju prve naredbe čitanja, tako da ne vidi sve promene posle toga, što nam i rezultat ovog primera dokazuje.

Nakon toga, pokrenućemo isti primer sa nivoom izolacije postavljenim na ***READ COMMITTED***.

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |
| T3 |  |  |

Table 2: Konzistentno čitanje na nivou izolacije READ COMITTED

Vidimo da je sada situacija drugačija. U trenutku T3, nakon što Transakcija 2 doda red, Transakcija 1 dobija da je broj redova 6. To znači vidi promene napravljene u Transakciji broj 2 nakon što su one potvrđene, čime smo dokazali da je snapshot koje je dobila ta operacija zaista “svež” što je odlika ***READ COMMITED*** nivoa izolacije.

## Phantom Read

Sledeći efekat koji ćemo videti na primeru je ***phantom read***. Takozvani problem fantomskog čitanja se javlja unutar transakcija kada jedan upit vraća različite skupove redova kada se opet izvrši. Na primer, ako se ***SELECT*** izvrši dva puta i drugi put vrati red koji nije vraćen prvi put, taj red nazivamo „fantomski“ red.

Za demonstraciju ove anomalije, kreiraćemo tabelu “t” sa kolonama *id* i *value* i u nju dodati vrednosti 1,3,5:

***CREATE TABLE t (a INT PRIMARY KEY, value INT) ENGINE = InnoDB;***

***INSERT INTO t VALUES (1,1), (2,3), (3,5);***

Nakon toga radimo sledeću sekvencu akcija:

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |
| T3 |  |  |

Table 3: Primer pojave phanom read na nivou READ COMITTED

Kod obe transakcije nivo izolovanosti je sada ***READ COMMITED*** zato što kod njega ne dolazi do zaključavanja opsega kombinacijom *gap i next-key* zaključavanja.  Transakcija 1 radi čitanje kolone *value* iz tabele uz uslov da vrednost kolone mora biti između 3 i 20. Ovde je bitno da je ***WHERE*** uslov urađen nad kolonom koja NEMA indeks. Nakon toga, Transakcija 2 dodaje red sa vrednošću 12, koja je izabrana zato što bi ušla u opseg uslova iz Transakcije 1. Kada potom ponovimo čitanje (trenutak T3) vidimo da se u vraćenim redovima pojavljuje i onaj dodat u Transakciji 2, odnosno “phantom row”, iako još uvek nije došlo do potvrđivanja, odnostno izvršenja ***COMMIT*** naredbe.

Sa druge strane, ako je nivo izolovanosti ***REPEATABLE READ***, MariaDB koristi *next-key* zaključavanje. Druga transakcija će i dalje moći da dodaje nove redove, ali ovoga puta oni neće biti vidljivi prvoj transakciji. Ovo garantuje dobar nivo izolacije između različitih transakcija. To ćemo demonstrirati i na primeru.

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |
| T3 |  |  |
| T4 |  |

Table 4: Primer pojave phanom read na nivou REPEATABLE READ

Kao i kod prethodnog primera, imamo 2 transakcije. Prva počinje čitanjem vrednosti kolona *id* i *value* iz tabele uz uslov da je id veći od 5 (trenutak T1). Ovde bi trebalo da dođe do korišćenja *next-key* zaključavanja. Transakcija 2, nakon toga, radi dodavanje reda u tabelu (trenutak T2). Ono što je bitno je da pokušava da doda red sa id-em 9, odnosno id-em koji bi ušao u rezultat operacije iz Transakcije 1.

Sada se vraćamo na Transakciju 1 i ponavljamo operaciju čitanja (trenutak T3). Tada vidimo da nema reda dodatog u Transakciji 2 i time potvrđujemo da je *next-key* zaključavanje zaista korišćen. To dodatno potvrđujemo tako što završavamo tu transakciju i ponavljamo upit koji čita podatke. Ovog puta novi red se nalazi u rezultatu (trenutak T4).

# Mehanizmi eksplicitnog zaključavanja

Kada proces ili transakcija koristi neki resurs, kako bi se sprečilo da drugi procesi ili transakcije takođe koriste taj resurs, što bi dovelo do preuzimanja resursa i konflikta, resurs se zaključava.

Tipovi zaključavanja koje podržava MariaDB su:

* Deljeno i isključivo zaključavanje (Shared and exclusive locks)
* Namerno zaključavanje (Intentions locks)
* Zaključavanje zapisa (Record locks)
* Zaključavanje praznina (Gap locks)
* Zaključavanje sledećeg ključa (Next-Key locks)
* Zaključavanja namera umetanja (Insert intentions locks)
* AUTO-INC zaključavanje

## Deljeno i isključivo zaključavanje

MariaDB implementira standardno zaključavanje na nivou reda gde postoje dve mogućnosti: deljeno (S) ili isključivo (X) zaključavanje.

* **Deljeno (S)** zaključavanje dozvoljava transakciji koja je zaključala red, njegovo čitanje. Takođe, moguće je da više transakcija zaključaju isti red.
* **Isključivo (X)** zaključavanje dozvoljava transakciji koja drži zaključavanje da ažurira ili izbriše red. U jednom trenutku samo jedna transakcija može da zaključa red.

Primer:

1. Ako transakcija T1 drži deljeno (S) zaključavanje u redu r, onda se zahtevi neke druge transakcije T2 za zaključavanje u redu r obrađuju na sledeći način: Zahtev T2 za S zaključavanje može se odobriti odmah. Kao rezultat, i T1 i T2 drže S zaključavanje na r.
2. Zahtev T2 za zaključavanje X ne može se odmah odobriti. Ako transakcija T1 drži Isključivo (X) zaključavanje na redu r, zahtev neke različite transakcije T2 za zaključavanje bilo kog tipa na r ne može biti odmah odobren. Umesto toga, transakcija T2 mora da sačeka da transakcija T1 oslobodi zaključavanje u redu r.

## Namerno zaključavanje

MariaDB podržava višestruko zaključavanje granularnosti koje dozvoljava koegzistenciju zaključavanja redova i zaključavanja tabele. Da bi zaključavanje na više nivoa granularnosti bilo praktično, MariaDB koristi namerno zaključavanje. Namerna zaključavanja su zaključavanja na nivou tabele koja označavaju koji tip zaključavanja (deljeno ili isključivo) transakcija zahteva kasnije za red u tabeli. Postoje dve vrste namernog zaključavanja:

* **Zajedničko zaključavanje namere (IS)** označava da transakcija namerava da postavi deljeno zaključavanje pojedinačnih redova u tabeli.
* **Isključivo zaključavanje namere (IX)** označava da transakcija namerava da postavi Isključivo zaključavanje pojedinačnih redova u tabeli.

Na primer, ***SELECT ... FOR SHARE*** postavlja zaključavanje IS, a ***SELECT ... FOR UPDATE*** postavlja IX zaključavanje.

Protokol zaključavanja namere je sledeći:

* Pre nego što transakcija stekne deljeno zaključavanje na redu u tabeli, prvo mora da stekne IS ili jače zaključavanje na tabeli.
* Pre nego što transakcija stekne isključivo zaključavanje na redu u tabeli, prvo mora da stekne IX zaključavanje na tabeli.

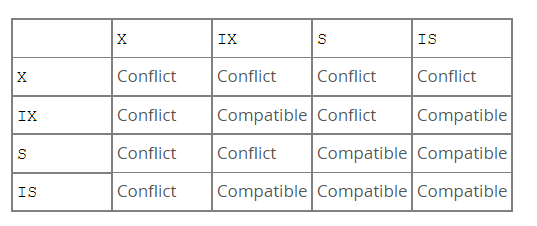


Tabela 5: Kompatibilnost tipa zaključavanja na nivou tabele.

Zaključavanje se dodeljuje transakciji koja je zahteva ako je kompatibilna sa postojećim zaključavanjima, ali ne i ako je u sukobu sa njima. Transakcija čeka dok se ne otpusti konfliktno postojeće zaključavanje. Ako je zahtev za zaključavanje u sukobu sa postojećim zaključavanjem, ne može se odobriti, jer bi to prouzrokovalo zastoj i dolašlo do greške.

Zaključavanja namere ne blokiraju ništa osim zahteva za punu tabelu ( LOCK TABLES ... WRITE). Glavna svrha zaključavanja namere je da pokaže da neko zaključava red ili namerava da zaključa red u tabeli.

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |

*Tabela 7: Primer IX-X conflikt-a*

*Zaključavanje zapisa*

Tabela 6: Primer IS-IS kompatibilnosti

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |

Zaključavanje zapisa je zaključavanje indeksnog zapisa, čak i ako je tabela definisana bez indeksa. U takvim slučajevima, MariaDB kreira skriveni klasterizovani indeks i koristi ovaj indeks za zaključavanje zapisa.

Na primer,  ***SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE,*** sprečava bilo koju drugu transakciju od umetanja, ažuriranja ili brisanja redova gde je vrednost t.c1 jednako 10.

## Zaključavanje praznine

Zaključavanje praznine je zaključavanje razmaka između indeksnih zapisa ili zaključavanje praznine pre prvog ili posle poslednjeg indeksnog zapisa. Razmak može da obuhvata jednu vrednost indeksa, više vrednosti indeksa ili čak da bude prazan. Zaključavanja praznina su deo kompromisa između performansi i konkurentnosti i koriste se na nekim nivoima izolacije transakcija.

Na primer, ***SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE***, sprečava druge transakcije da umetnu vrednost od 15 u kolonu t.c1, bez obzira da li je takva vrednost već postojala u koloni, jer su praznine između svih postojećih vrednosti u opsegu zaključane.

Zaključavanje praznina nije potrebno za naredbe koje zaključavaju redove koristeći jedinstveni indeks za traženje jedinstvenog reda. (Ovo ne uključuje slučaj da uslov pretrage uključuje samo neke kolone jedinstvenog indeksa sa više kolona; u tom slučaju dolazi do zaključavanja praznina.)

Na primer, ako kolona id ima jedinstveni indeks, sledeća naredba koristi samo zaključavanje zapisa indeksa za red koji ima id vrednost 100 i nije bitno da li druge sesije ubacuju redove u prethodni razmak:

***SELECT \* FROM child WHERE id = 100;***

Ako id nije indeksiran ili ima nejedinstveni indeks, izjava zaključava prethodnu prazninu.

Zaključavanje praznina se može eksplicitno onemogućiti. Ovo se dešava ako promenite nivo izolacije transakcije u ***READ COMMITTED***. U ovom slučaju, zaključavanje praznina je onemogućeno za pretrage i skeniranje indeksa koristi se samo za proveru ograničenja stranog ključa i proveru duplikata ključa.

## Zaključavanje sledećeg ključa

Zaključavanje sledećeg ključa je kombinacija zaključavanja zapisa na indeksnom zapisu i zaključavanja praznine na razmaku pre indeksnog zapisa. MariaDB izvodi zaključavanje na nivou reda na takav način da kada se pretražuje ili skenira indeks tabele, postavljaju se deljena ili isključiva zaključavanja na zapise indeksa na koja se naiđe. Ako jedna sesija ima deljeno ili isključivo zaključavanje zapisa R u indeksu, druga sesija ne može umetnuti novi indeksni zapis u prazninu neposredno ispred R u redosledu indeksa.

Primer:

Pretpostavimo da indeks sadrži vrednosti 10, 11, 13 i 20. Moguća zaključavanja sledećeg ključa za ovaj indeks pokrivaju sledeće intervale, gde okrugla zagrada označava isključivanje krajnje tačke intervala, a uglasta zagrada označava uključivanje krajnje tačke:

***(negative infinity, 10]***

***(10, 11]***

***(11, 13]***

***(13, 20]***

***(20, positive infinity)***

Za poslednji interval, zaključavanje sledećeg ključa zaključava prazninu iznad najveće vrednosti u indeksu i „supremum“ pseudo-zapisa koji ima vrednost veću od bilo koje vrednosti u stvarnom indeksu. Supremum nije pravi indeksni zapis, tako da, u stvari, ovo zaključavanje sledećeg ključa zaključava samo prazninu nakon najveće vrednosti indeksa.

Podrazumevano, MariaDB radi na nivou izolacije transakcije  ***REPEATABLE READ***. U ovom slučaju, MariaDB koristi zaključavanje sledećeg ključa za pretrage i skeniranje indeksa, što sprečava fantomske redove.

## Zaključavanja namera umetanja

Zaključavanja namera umetanja je tip zaključavanja praznina prilikom umetanja reda. Ovo zaključavanje signalizira nameru umetanja na takav način da više transakcija koje se ubacuju u istu indeksnu prazninu ne moraju da čekaju jedna drugu ako se ne ubacuju na istoj poziciji unutar praznine.

Primer:

Pretpostavimo da postoje indeksni zapisi sa vrednostima 4 i 7. Odvojene transakcije koje pokušavaju da ubace vrednosti 5 i 6, respektivno, svaka zaključava prazninu između 4 i 7 zaključavanjem namere umetanja pre dobijanja isključivog zaključavanja umetnutog reda. Redovi su nekonfliktni, pa nema potrebe da transakcije blokiraju jedna drugu.

## AUTO-INC zaključavanje

AUTO-INC zaključavanje je specijalno zaključavanje na nivou tabele koje je izazvano transakcijama koje upisuju u tabele sa AUTO\_INCREMENT kolonama. U najjednostavnijem slučaju, ako jedna transakcija ubacuje vrednosti u tabelu, sve druge transakcije moraju da sačekaju da bi urade svoje umetanje u tu tabelu, tako da redovi umetnuti prvom transakcijom dobiju uzastopne vrednosti primarnog ključa.

## DEADLOCK

U nekim situacijama, korišćenjem različitih tipova zaključavanja, može doći do toga da transakcije blokiraju jedna drugu. Pošto tada sve uključene transakcije čekaju da se otključa neki od lock-ova, one se nikada neće prirodno deblokirati. Takve situacije se nazivaju deadlock. MariaDB (kao i drugi mehanizmi za skladištenje) ima interni mehanizam za otkrivanje takvih problema. On rešava situaciju tako što prekida transakciju koja je dodala, izbrisala ili ažurirala najmanji broj redova. Takva transakcija se naziva žrtva (the victim).

U okruženju gde postoji konkurentnost, javljanje deadlock-ova je u nekoj meri očekivano i normalno. Kada dođe do zastoja, jedna ili više transakcija se završavaju sa greškom 1213. To se najčešće ne tretira kao veliki problem, i aplikacije bi trebalo jednostavno da reše ovu grešku ponovnim pokretanjem transakcije. Naravno, to ne znači da se očekuje da se zastoji često dešavaju. Ako je to slučaj, trebalo bi nešto učiniti da se problem reši.

Neka od rešenja za rešavanje problema sa deadlock-om su:

* Manje je verovatno da će male transakcije izazvati zastoje. Ako je moguće, dobra ideja je da transakcije budu male
* Smanjenje broja lock-ova
* Korišćenje restriktivnijih lock-ova na početku transakcije
* Različite transakcije koje pristupaju istim tabelama treba da pokušaju da to urade različitim redosledom. Na primer, ako transakcija pristupa tabelama A, B i C (ovim redosledom), druga transakcija treba da izbegava da im pristupa istim redosledom (ako je moguće); idealno bi bilo C, B, A
* Uglavnom nivo izolacije ne utiče direktno na pojavu deadlock-a, ali broj deadlock-ova koji se javi zbog locking read-a može biti smanjen korišćenjem  READ COMMITTED ili READ UNCOMMITTED nivoa

Za dobijanje informacija o deadlock-u možemo da koristimo komandu:

***SHOW ENGINE INNODB STATUS***

Moguće je i postaviti *innodb\_print\_all\_deadlocks* promenljivu servera na ON čime uključujemo upis svih zastoja u log grešaka (error log).

Sada ćemo na primeru demonstrirati kako dolazi do deadlock-a. U tu svrhu kreiraćemo dve tabele, t1 i t2, koje će imati jednu kolonu za primarni ključ i jedan jedinstveni indeks:

***CREATE TABLE t1 (a INT PRIMARY KEY, b INT UNIQUE)***

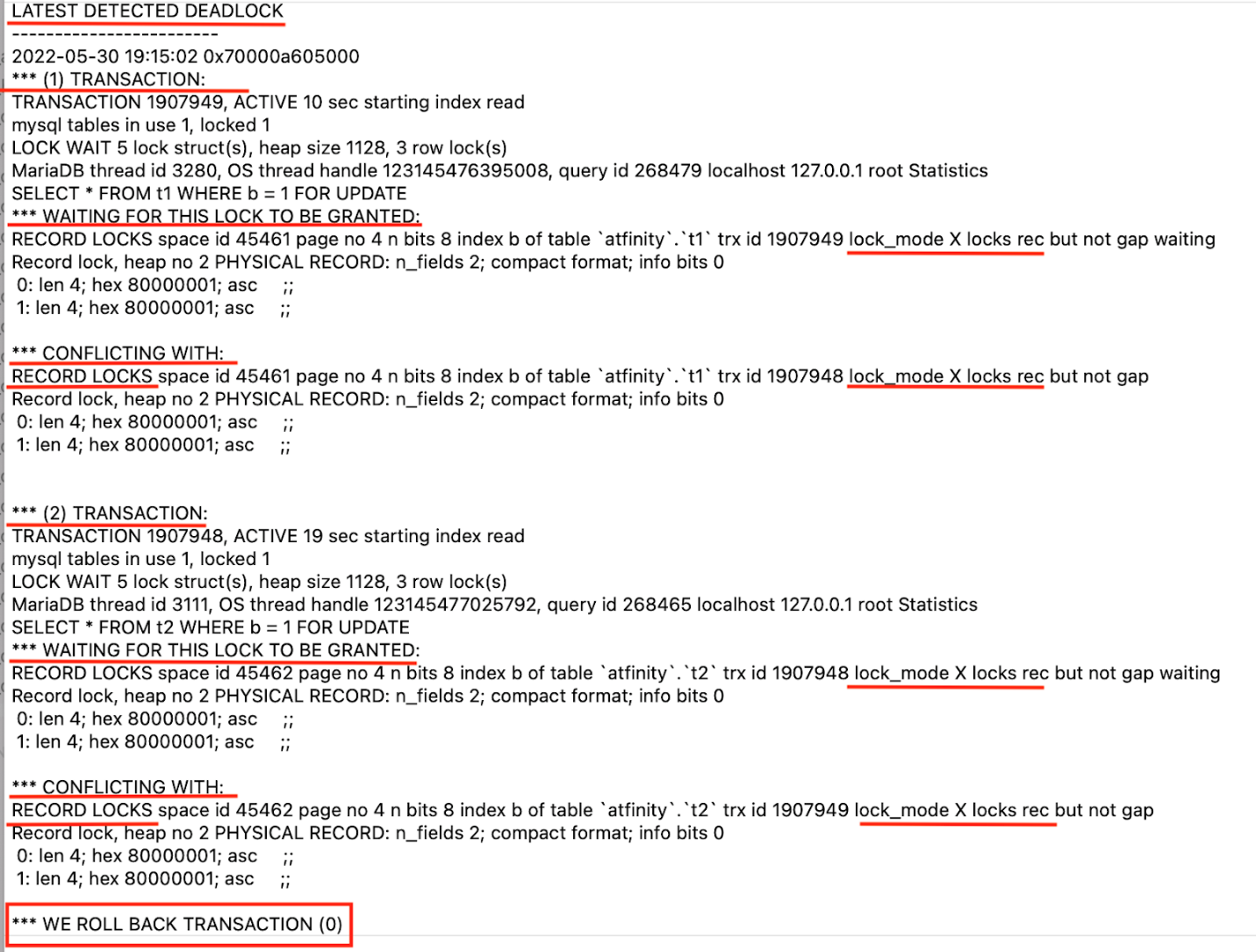
***CREATE TABLE t2 (a INT PRIMARY KEY, b INT UNIQUE)***

Takođe, otvarmo 2 instance MariaDB klijenata za pokretanje 2 paralelne transakcije. U okviru 1. transakcije koristićemo ***SELECT ... FOR UPDATE*** da bismo dobili *exclusive record lock* nad t1. Kod druge, uradićemo isto za t2. Nakon toga, Transakcija 1 će probati da pristupi redu zaključanom od strane Transakcije 2, a Transakcija 2 redu zaključanom od strane Transakcije 1.

|  |  |  |
| --- | --- | --- |
|  | Transakcija1 | Transakcija2 |
| T1 |  |  |
| T2 |  |  |
| T3 |  |  |
| T4 |  |  |

U trenutku T4, Transakcija 1 će čekati da se oslobodi lock koji drži Transakcija 2, ali on ne može biti oslobođen zato što ga “drži” Transakcija 2 koja čeka lock Transakcije 1 zbog čega dolazi do deadlock-a.  Iako znamo šta se desilo i zbog čega je došlo do zastoja, pretpostavimo da hoćemo da ispitamo problem. To možemo uraditi pomoću ***SHOW ENGINE INNODB*** status komande koja sadrži informacije o deadlock-u. Izvršićemo je i pogledati *LATEST DETECTED DEADLOCK* sekciju:

Table 8: Primer nastanka deadlock-a



Slika 6: Informacije o deadlock-u dobijene pomoću SHOW ENGINE INNODB komande

Ovde možemo videti koji tipovi zaključavanja se koriste od strane koje transakcije, kao i koji tip one “čekaju” ( WAITING FOR THIS LOCK TO BE GRANTED). Poslednja linija pruža informaciju o tome koje će konekcija biti prekinuta da bi se one odblokirale.

# Zaključak

Transakcije predstavljaju jedan od najvažnijih koncepata rada sa bazom podataka. Pomoću njih moguć je rad u višekorisničkom okruženju, gde se upiti izvršavaju konkurentno. Na primerima smo videli da postoji mnogo različitih podešavanja sistema kada se radi sa transakcijama. Od izabrane strategije zavisi kako će se sam sistem ponašati, počevši od toga koje će promene biti uzete u obzir, preko toga koliko je sistem otporan na greške, pa do samih performansi. Iako može biti moćno oružje, treba pažljivo rukovati podsistemom za transakcije jer se lako može ući u probleme.

Ukoliko ne bi koristili transakcije bilo bi primenjeno podrazumevano ponašanje sistema, a to je da se svaka naredba nakon izvršenja potvrđuje (automatsko upisivanje). Promene bi odmah bile upisane u bazu. Na osnovu ovog rada, videli smo na više primera da to nije dobro za održanje konzistentnosti baze podataka.

Takođe, imali smo primer gde se jasno vidi povezanost više naredba, odnosno zašto je bitna atomičnost transakcija i da naredbe ne bi trebalo da se izvrse jedna bez druge, jer ukoliko se prva izvrsi a druga ne, imaćemo bazu podataka u nekonzistentnom stanju.

Većina primera u ovom radu fokusirana je na konkurentno izvršavanje i probleme koji nastaju u vezi sa njim, izolaciju transakcija i zaključavanje. Videli smo na koje sve načine podaci mogu zaključati, šta to znači u bazi, šta za progremara i koje su posledice. Ukratko su objašnjene prednosti i mane svakog od pristupa i rečeno je kada bi mogli biti od koristi. Primerima smo pokazali na koje sve načine programer može da utiče na ponašanje sistema. Dokazali smo da je znanje o tome kako treba raditi sa transakcijama, kao i to na koji način funckcioniše zaključavanje i konkurentost  od krucijalnog značaja za rad sa bazom.

# Reference

[1] Federico Razzoli , “Mastering MariaDB”, Packt Publishing, 2014 [poslednji pristup 01.06.2022.]

[2] Pierre Mavro , “MariaDB High Performance”, Packt Publishing, 2014 [poslednji pristup 01.06.2022.]

[3] IRussell J.T. Dyer, “Learning MySQL and MariaDB\_ Heading in the Right Direction with MySQL and MariaDB”, O'Reilly Media, 2015 [poslednji pristup 01.06.2022.]

[4] MariaDB Handler\_icp\_% Counters: What They Are, and How To Use Them   -​​      <https://www.percona.com/blog/2017/05/09/mariadb-handler_icp_-counters-what-they-are-and-how-to-use-them/#comment-10968069> [poslednji pristup 26.05.2022.]

[5] InnoDB Transaction Model - <https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-model.html> [poslednji pristup 26.05.2022.]

[6] Transaction Isolation Levels - <https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html> [poslednji pristup 27.05.2022.]

[7] Transaction Life Cycle - <https://dev.mysql.com/doc/internals/en/transactions-life-cycle.html> [poslednji pristup 26.05.2022.]

[8] <https://mariadb.com/kb/en/start-transaction/> [poslednji pristup 28.05.2022.]

[9] Transaction isolation level of MariaDB - <https://developpaper.com/transaction-isolation-level-of-mariadb/> [poslednji pristup 28.05.2022.]

[10] Explicit use of table lock and row level lock in MariaDB- <https://developpaper.com/explicit-use-of-table-lock-and-row-level-lock-in-mariadb/> [poslednji pristup 30.05.2022.]

[11] Analysis of MySQL shared lock and exclusive lock usage - <https://developpaper.com/analysis-of-mysql-shared-lock-and-exclusive-lock-usage/> [poslednji pristup 31.05.2022.]