Univerzitet u Nišu

Elektronski fakultet



Sistemi za upravljanje bazama podataka

Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje

- Seminarski rad -

Mentor: Prof. Aleksandar Stanimirović Kandidat: Pavle Stojanović 1148

Maj 2021.

Sadržaj

[Šta je MySQL transakcija 3](#_Toc73269193)

[ACID 4](#_Toc73269194)

[MySQL transakcione naredbe 4](#_Toc73269195)

[AUTOCOMMIT Mod 7](#_Toc73269196)

[Implicitni Commit 8](#_Toc73269197)

[Nivoi izolacije 8](#_Toc73269198)

[Zaključavanje 11](#_Toc73269199)

[Deadlock-ovi 13](#_Toc73269200)

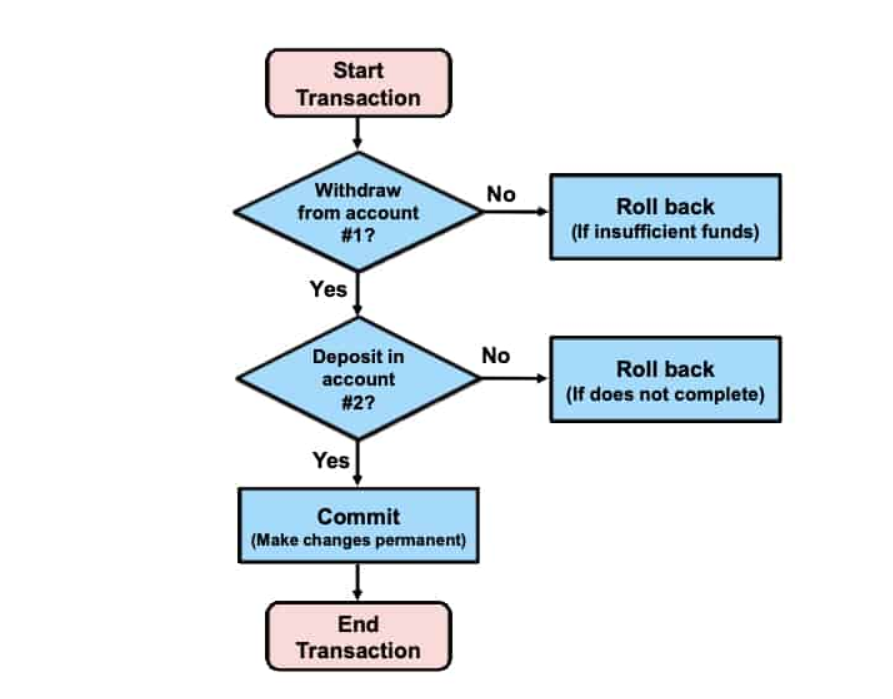
[Implicitni lock-ovi 13](#_Toc73269201)

[Literatura 15](#_Toc73269202)

# Šta je MySQL transakcija

Transakcija je način na koji možete izvršiti jedan ili više SQL izraza kao jednu atomičnu radnju, tako da uspeju ili sve naredbe ili nijedna. Ovo se događa izolovano od posla koji se obavlja u drugim transakcijama. Ako se svi izrazi uspešno izvrše, transakcija se potvrđuje (commit) i izmene iz iste se pamte u bazi podataka. Ako se tokom transakcije dogodi greška, otkazujete celu transakciju (rollback). Sve naredbe izvršene do tog trenutka u transakciji su poništene, a baza podataka ostaje u stanju u kojem je bila pre trenutka kada je transakcija započeta.

U MySQL-u su transakcije podržane samo za one tabele koje koriste mehanizam za skladištenje transakcija (poput InnoDB).



Gornja slika je primer pokušaja prenosa $ 1,000 sa vašeg štednog računa na vaš tekući račun. Ne biste bili srećni da je novac uspešno skinut sa vašeg štednog računa, ali nikada nije prebačen na vaš tekući račun.

Da bi se zaštitio od ove vrste greške, program koji obrađuje vaš zahtev za prenos prvo započinje transakciju, a zatim izdaje SQL naredbe potrebne za prebacivanje novca sa vašeg štednog na vaš tekući račun. Završava transakciju samo ako sve uspe. Ako dođe do problema, program će narediti serveru da poništi sve promene napravljene u toku transakcije.

## ACID

Transakcija sadrži četiri svojstva, koja se nazivaju ACID svojstva. ACID označava:

* Atomičnost - Sve naredbe unutar transakcije se izvršavaju uspešno ili se sve poništavaju.
* Konzistentnost – Transakcija prebacuje bazu podataka iz jednog u drugo konzistentno stanje.
* Izolacija – Transakcije su međusobne izolovane tj promene u jednoj transakciji nisu vidljive u drugoj dok se iste ne commit-uju.
* Trajnost - Sve promene izvršene u transakciji koja se uspešno završila se pamte u bazi podataka. Promene se ne gube usled otkaza sistema.

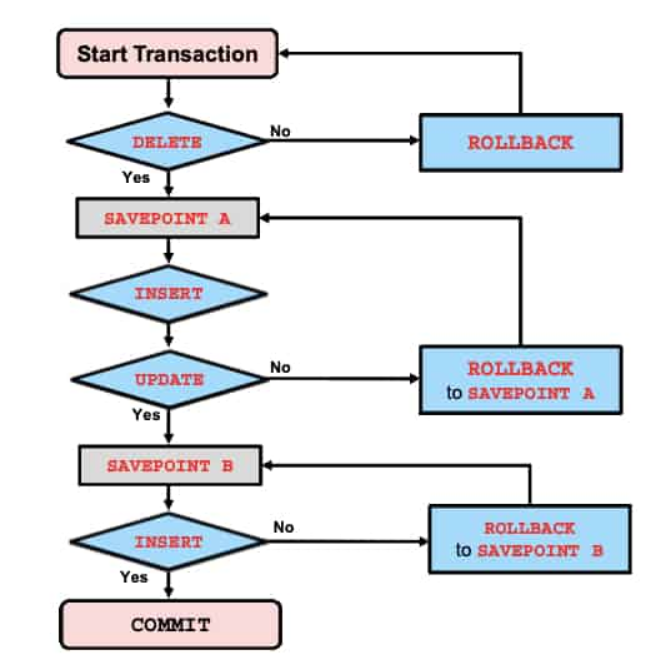
Transakciona obrada pruža jače garancije o ishodu operacija nad bazom podataka, ali takođe zahteva i više troškova u ciklusima procesora, memoriji i prostoru na disku. Transakciona svojstva su od suštinskog značaja za neke aplikacije, ali ne i za sve.

Finansijskim operacijama su obično potrebne transakcije, i u ovom slučaju je dodatni trošak opravdan. S druge strane, za aplikaciju koja evidentira pristupe web stranici u tabelu baze podataka, gubitak nekoliko zapisa, usled pada servera, može biti podnošljiv.

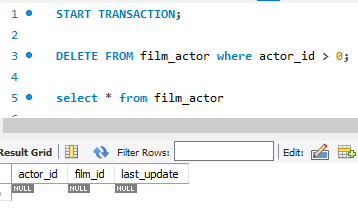
# MySQL transakcione naredbe

* START TRANSACTION (or BEGIN): Započinje transakciju
* SAVEPOINT: Definiše lokaciju u transakciji
* COMMIT: Promene načinjene u transakciji se trajno pamte
* ROLLBACK: Poništava efekte tekuće transakcije
* ROLLBACK TO SAVEPOINT: Poništava efekte transakcije od zadatog savepoint-a
* RELEASE SAVEPOINT: Uklanja savepoint
* SET AUTOCOMMIT: Ažurira autocommit mod za tekuću konekciju.

Primer u nastavku ilistruje primenu ovih naredbi. Naime, u svakom trenutku je moguće vratiti se do određenog koraka unutar transakcije (savepoint) i poništiti izmene načinjene nakon tog savepoint-a. Takođe, moguće je u bilo kom trenutku poništiti sve promene koje je transakcije učinila korišćenjem Rollback naredbe. Da bi se izmene sačuvale neophodno je izvršiti commit naredbu.

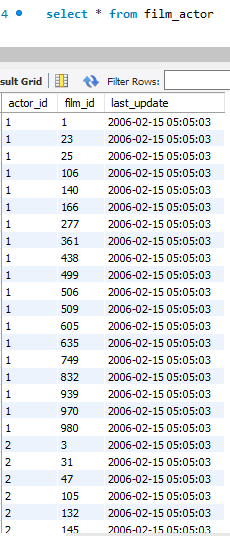


Primer u nastavku pokazuje efekte transakcionih naredbi prilikom konkurentog rada sa jednom bazom podataka. Naime u sesiji broj 1 izdate su sledeće SQL naredbe:

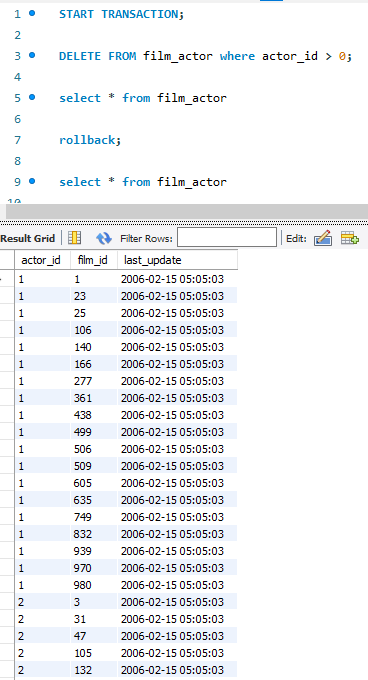


Rezultat poslednje select naredbe u sesiji broj 1 prikazan je na slici.

Otvorićemo drugu konekciju ka istoj bazi I probati indentičnu select naredbu.



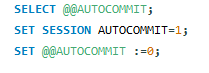
U ovom slučaju select naredba prikazuje celu tabelu kao da nismo izdali delete naredbu u prethodnoj sesiji. Razlog za ovo jeste korišćenje transakcija. Naime u sesiji broj 1 nismo izdali commit naredbu tako da sesija broj 2 ne vidi izmene. Izvršićemo rollback u sesiji broj 1 I ponovo pokrenuti select naredbu. Nakon rollback-a podaci su vidljivi I u sesiji broj 1.



## AUTOCOMMIT Mod

AUTOCOMMIT određuje kako i kada se započinju nove transakcije. Kada je omogućen režim AUTOCOMMIT, svaka SQL naredba implicitno započinje novu transakciju. Kada se svaka naredba završi, transakcija se komituje. AUTOCOMMIT je podrazumevano omogućen.

Ovo ponašanje možete nadjačati eksplicitnim pokretanjem nove transakcije sa START TRANSACTION. Promene nisu vidljive dok transakciju ne završite sa eksplicitnim COMMIT.



## Implicitni Commit

Izraz COMMIT uvek uspešno završava trenutnu transakciju. Ostale naredbe o kontroli transakcija (poput onih dole navedenih) takođe imaju efekat implicitnog commit-ovanja trenutne transakcije.



Osim ovih naredbi o kontroli transakcija, druge vrste naredbi imaju efekat implicitnog commit-a (i time prekida) trenutne transakcije. Ove naredbe se ponašaju kao da ste izdali COMMIT pre izvršenja stvarne naredbe. Pored toga, ove naredbe same po sebi nisu transakcione, što znači da se ne mogu rollback-ovati.

Netransakcione naredbe koje podrazumevaju implicitni commit:

* **ALTER, CREATE, DROP**
* **GRANT, REVOKE, SET PASSWORD**
* **LOCK TABLES, UNLOCK TABLES**

# Nivoi izolacije

Na serveru može istovremeno postojati više transakcija na čekanju (najviše jedna transakcija po sesiji). Kada više klijenata istovremeno pristupa podacima iz iste tabele, mogu se pojaviti sledeći problemi

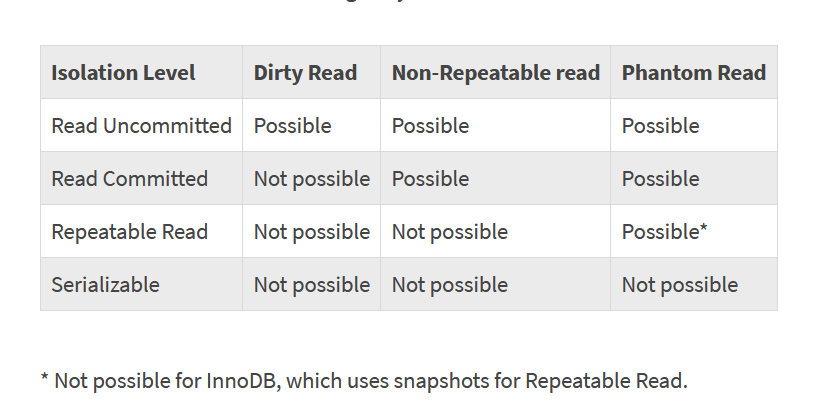
* Prljavo čitanje (Dirty read): Pretpostavimo da transakcija T1 modifikuje red. Ako transakcija T2 pročita red i vidi modifikaciju iako je T1 nije commit-ovao, to je prljavo čitanje. Jedan od razloga što je ovo problem je taj što ako T1 uradi rollback, promena je poništena, ali T2 to ne zna.
* Neponovljivo čitanje (Non-repeatable read): Ista operacija čitanja daje različite rezultate kada se kasnije ponovi u okviru iste transakcije.
* Fantomsko čitanje (Phanton read): Pretpostavimo da transakcije T1 i T2 započinju, a T1 čita neke redove. Ako T2 ubaci novi red i T1 vidi taj red kada ponavlja istu operaciju čitanja, došlo je do fantomskog čitanja (novi red je fantomski red).

Ako transakcija jednog klijenta promeni podatke, da li bi transakcije za druge klijente morale da vide te promene ili treba da budu izolovane od njih? Nivo izolacije transakcija određuje načine interakcije istovremenih transakcija prilikom pristupa istim podacima.

Izbori nivoa izolacije se razlikuju među serverima baza podataka, tako da nivoi koje je primenio InnoDB možda neće tačno odgovarati nivoima koji su primenjeni u drugim sistemima baza podataka.

InnoDB primenjuje četiri nivoa izolacije koji kontrolišu u kojoj meri su promene izvršene transakcijama primetne kod drugih transakcija koje se istovremeno javljaju:

* READ UNCOMMITTED: Omogućava pojavu prljavih čitanja, neponovljivih čitanja i fantomskih čitanja
* READ COMMITTED: Omogućava neponovljivo čitanje i fantomsko čitanje. Ne commit-ovane promene ostaju nevidljive.Ovo je default-ni pristup kod većine DBMS-ova ali ne i kod MySql-a.
* REPEATABLE READ: Default-ni pristup kod MySql-a, neponovljivo čitanje nije dozvoljeno, select uvek vraća isti rezultat tokom transakcije.
* SERIALIZABLE: : Kompletna izolacija, slično REPEATABLE READ sa dodatnim ograničenjem da redovi koje koristi jedna transakcija se ne mogu menjati u drugoj dok se prva transakcija ne završi.



Sintaksa za podešavanje nivoa izolacije:

SET GLOBAL TRANSACTION ISOLATION LEVEL [isolation\_level];

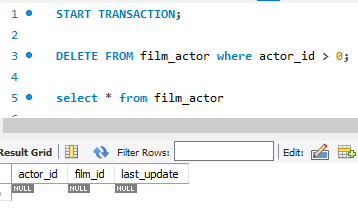
SET SESSION TRANSACTION ISOLATION LEVEL [isolation\_level];

SET TRANSACTION ISOLATION LEVEL [isolation\_level];

Nivo transakcije može se postaviti na globalnom ili na nivou sesije. Bez izričite specifikacije, nivo izolacije transakcije postavlja se na nivou sesije. Na primer, sledeća naredba postavlja nivo izolacije na READ COMITTED za trenutnu mysql sesiju:

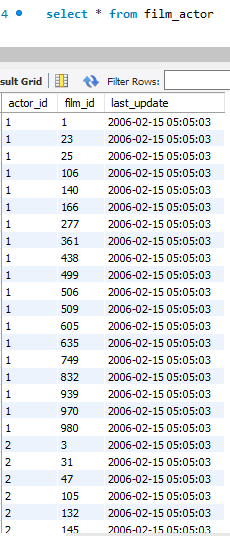
*SET TRANSACTION ISOLATION LEVEL READ COMMITTED;*

Primer u nastavku pokazuje efekat izolacije. Naime u sesiji 1 obrišemo određene redove i pokrenemo select naredbu. Prikazana je prazna tabela film\_actor.



Sesija 1

U sesiji 2 ponovo pokrenemo istu select naredbu i vidimo redove u tabeli film\_actor. Ovo je efekat default-nog nivoa izolacije u MySql-u (REPEATABLE READ) tj nekomitovane izmene u prvoj transakciji nisu vidljive u drugoj.



# Zaključavanje

Zaključavanje je mehanizam koji sprečava pojavu problema istovremenog pristupa podacima od strane više klijenata. Postavlja zaključavanje podataka u ime jednog klijenta kako bi drugim klijentima ograničio pristup podacima dok se zaključavanje ne oslobodi. Zaključavanje omogućava pristup podacima klijentu koji drži zaključavanje (lock), ali postavlja ograničenja na to koje radnje mogu da urade drugi klijenti koji zahtevaju pristup. Efekat mehanizma zaključavanja je serijski pristup podacima tako da kada više klijenata želi da izvršava neusaglašene operacije, svaki mora da sačeka svoj red. Ne uzrokuju sve vrste istovremenih pristupa konflikte, tako da vrsta zaključavanja koja je neophodna da bi se klijentu omogućio pristup podacima zavisi od toga da li klijent želi da čita ili piše:

* Ako klijent želi da čita podatke, drugi klijenti koji žele da čitaju iste podatke ne proizvode konflikt i svi mogu da čitaju istovremeno. Međutim, drugi klijent koji želi da modifikuje podatke mora da sačeka dok se čitanje ne završi.
* Ako klijent želi da upiše podatke, svi ostali klijenti moraju da sačekaju dok se pisanje ne završi, bez obzira da li ti klijenti žele da čitaju ili pišu.

Čitač mora blokirati pisce, ali ne sme blokirati druge čitaoce. Pisac mora blokirati i čitaoce i pisce. Brave za čitanje i brave za pisanje omogućavaju sprovođenje ovih ograničenja. Zaključavanje forsira klijente da čekaju pristup dok ne bude bezbedno da nastave dalje. Na ovaj način brave sprečavaju oštećenje podataka ne dozvoljavajući istovremeno konfliktne modifikovanja i čitanja podataka koji su u procesu modifikacije.

InnoDB podržava dva modifikatora zaključavanja koji se mogu dodati na kraj SELECT naredbe:

1. LOCK IN SHARE MODE klauzula: Zajednički lock, što znači da nijedna druga transakcija ne može uzeti ekskluzivni lock, ali druge transakcije takođe mogu koristiti zajednički lock. Sprečava druge transakcije da ažuriraju podatke koji se čitaju.
2. FOR UPDATE klauzula: Zaključava svaki izabrani red ekskluzivnim lock-om, sprečavajući druge transakcije da dobiju lock nad tim redovima, ali omogućava čitanje redova.

Naredni primer pokazuje efekat zaključavanja. Naime u sesiji broj 1 izvršimo FOR UPDATE naredbu.



Nakon toga je nemoguće iz neke druge sesije dobiti lock nad tabelom payment dok se u sesiji broj 1 ne izvrši naredba commit.

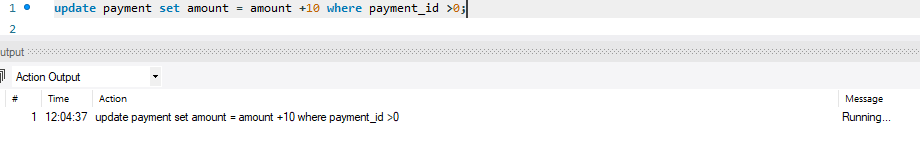
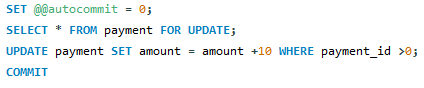


Tabela payment je slobodna nakon commit-a u prvoj sesiji.



U nivou izolacije REPEATABLE READ, LOCK IN SHARE MODE se može dodati operacijama SELECT da bi prisilio druge transakcije da čekaju transakciju ako žele modifikovati odabrane redove. Ovo je slično radu na nivou izolacije SERIALIZABLE, za koji InnoDB implicitno dodaje LOCK IN SHARE MODE u SELECT naredbe koje nemaju eksplicitni modifikator zaključavanja. Ako odabere redove koji su modifikovani u ne commit-ovanoj transakciji, zaključava SELECT dok se ta transakcija ne izvrši.

## Deadlock-ovi

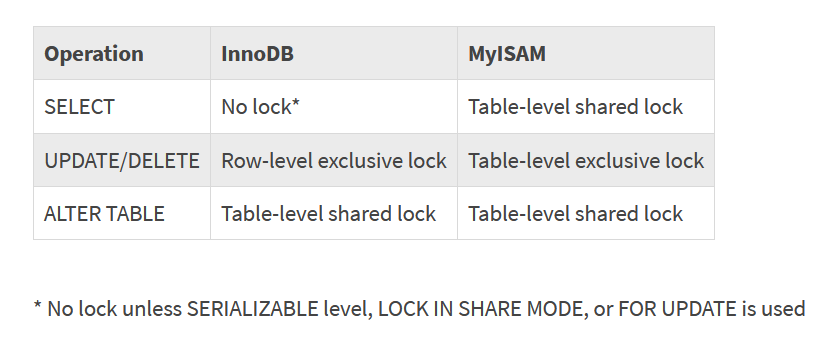
Deadlock je situacija u kojoj različite transakcije nisu u mogućnosti da se nastave jer svaka ima lock koji je potreban drugoj. Budući da obe transakcije čekaju da resurs postane dostupan, nijedna nikada ne oslobađa brave koje drži.

Deadlock-ovi su klasičan problem u transakcionim bazama podataka, ali nisu opasni ako nisu toliko česti da uopšte ne možete pokretati određene transakcije. Kada je omogućeno otkrivanje deadlock-a (podrazumevano podešavanje) i dogodi se deadlock, InnoDB detektuje ovo stanje i vraća unazad jednu od transakcija (žrtva). Do deadlock-a dolazi kada:

* Transakcije dobijaju lock-ove na više tabela, ali u suprotnom redosledu
* Postoji više transakcija, jedna čeka da se druga završi. Na primer, T1 čeka T2, T2 čeka T3, a T3 čeka T1.

### Implicitni lock-ovi

MySQL server zaključava tabelu (ili red) na osnovu izdatih naredbi i mehanizama za skladištenje koji se koristi:



Tabele InnoDB koriste zaključavanje na nivou reda tako da više sesija i aplikacija mogu istovremeno čitati iz iste tabele i pisati u nju, bez međusobnog čekanja i bez davanja neusklađenih rezultata. Za ovaj mehanizam za skladištenje, izbegavajte upotrebu izraza LOCK TABLES; ne nudi nikakvu dodatnu zaštitu, ali umesto toga smanjuje istovremenost.

Automatsko zaključavanje na nivou reda čini ove tabele pogodnim za vaše najprometnije baze podataka sa vašim najvažnijim podacima, istovremeno pojednostavljujući logiku aplikacije jer ne morate zaključavati i otključavati tabele. Shodno tome, InnoDB mehanizam za skladištenje je podrazumevani u MySQL 5.6.

# Literatura

* <https://www.mysqltutorial.org/mysql-transaction.aspx>
* https://dev.mysql.com/doc/refman/8.0/en/commit.html
* <https://www.tutorialspoint.com/mysql/mysql-transactions.html>

Korišćena baza podataka za primere: https://dev.mysql.com/doc/sakila/en/

