

Elektronski fakultet, Univerzitet u Nišu



*Sistemi za upravljanje bazama podataka*

*Seminarski rad*

# **Obrada transakcija, planovi izvršenja transakcija, izolacija i zaključavanje u MySQL-u**

Mentor:

Aleksandar Stanimirović

Student:

Mila Mirović 1525

Niš, maj 2023.

## Sadržaj

Uvod.....	3
1. Transakcije u MySQL-u .....	5
1.1. Obrada transakcija.....	6
1.1.1. Primer COMMIT naredbe.....	8
1.1.2. Primer ROLLBACK naredbe .....	9
1.1.3. Primer SAVEPOINT i ROLLBACK TO naredbi .....	10
1.1.4. Primer RELEASE SAVEPOINT naredbe .....	11
2. Izolacija .....	13
2.1. Anomalije koje se javljaju u transakcijama.....	13
2.2. Nivoi izolacije .....	14
2.2.1. READ UNCOMMITTED (Nepotvrđeno čitanje) .....	14
2.2.2. READ COMMITTED (Potvrđeno čitanje).....	16
2.2.3. REPEATABLE READ (Ponovljeno čitanje).....	17
2.2.4. SERIALIZABLE .....	19
3. Zaključavanje u MySQL-u .....	20
3.1. Table Locking .....	21
3.1.1. READ LOCK.....	22
3.1.2. WRITE LOCK.....	23
3.2. Row Locking.....	24
3.2.1. Primer Shared Row Locking-a.....	25
3.2.2. Primer Exclusive Row Locking-a.....	26
4. SET TRANSACTION naredba .....	27
5. Zaključak .....	28
6. Literatura .....	29

# Uvod

Baze podataka su ključni element u različitim aplikacijama i sistemima, a kako se sve više podataka generiše i manipuliše njima, važno je da baze podataka pruže pouzdanu i doslednu obradu transakcija. U domenu sistema za upravljanje bazama podataka, obrada transakcija igra ključnu ulogu u osiguravanju *integriteta* i *pouzdanosti* operacija nad podacima.

**Transakcija** predstavlja logičku jedinicu rada koja se sastoji od jedne ili više SQL operacija nad bazom podataka koje se izvršavaju u potpunosti sve ili nijedna, a njihovo pravilno rukovanje je od vitalnog značaja za održavanje doslednosti podataka. Te operacije mogu uključivati selekciju (**SELECT**), umetanje (**INSERT**), modifikovanje (**UPDATE**) ili brisanje (**DELETE**) podataka. One su zapravo jedna od ključnih osobina koje bazu podataka razlikuju od datotečnog sistema, jer on može dovesti do oštećenja datoteke u slučaju pada operativnog sistema, a transakcije obezbeđuju konzistentnost baze podataka. Transakcije omogućavaju oporavak baze podataka čak i u slučaju sistemskih grešaka. Kada više korisnika pokušava pristupiti i menjati podatke istovremeno u istoj tabeli, dolazi do preklapanja akcija i mogućnosti nekonzistentne promene podataka. ***Glavni zadatak jedne transakcije je da prevede bazu podataka iz jednog konzistentnog stanja u drugo konzistentno stanje.***

**MySQL** pruža snažne mogućnosti za obradu transakcija, planiranje izvršenja, izolaciju i mehanizme zaključivanja. Ključni cilj obrade transakcija je da se sve operacije u transakciji smatraju uspešnim ili neuspešnim, kao jedna celina. To znači da ako bilo koja operacija u transakciji ne uspe, sve promene koje su prethodno izvršene u okviru te transakcije moraju biti poništene (**ROLLBACK**), čime se održava doslednost podataka.

Planiranje izvršenja transakcija je proces kojim se određuje najefikasniji redosled izvršavanja SQL naredbi unutar transakcije radi postizanja maksimalnih performansi i efikasnosti. Kada se transakcija pokrene, MySQL koristi optimizator upita koji analizira SQL upite iz transakcije i generiše optimalne planove izvršenja. Optimizator uzima u obzir različite faktore, kao što su raspoloživi indeksi, statistika tabela i konfiguracijski parametri, kako bi odabrao najbrži put za izvršavanje transakcija. Cilj je minimizirati vreme izvršenja transakcije i optimizirati resurse baze podataka.

Važan aspekt obrade transakcija je **izolacija transakcija** koja se odnosi na način na koji se transakcije izvršavaju paralelno, a da ne dolazi do neslaganja i konflikta između njih. Sprečava nepravilne rezultate među programima koji pristupaju bazi podataka istovremeno. Kada više korisnika ili procesa pristupa i menja podatke istovremeno, postoji mogućnost preplitanja akcija i nekonzistentnih promena podataka. Cilj izolacije transakcija je sprečiti takve neželjene interakcije i osigurati da svaka transakcija radi sa konzistentnim podacima, bez obzira

na druge transakcije koje se odvijaju paralelno. MySQL pruža različite nivoe izolacije i svaki od njih ima svoje karakteristike i određuje koliko strogo se podaci iz jedne transakcije izoluju od drugih transakcija. Neki od najčešće korišćenih nivoa izolacije u MySQL-u su: **READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ i SERIALIZABLE.**

***Zaključavanje podataka*** je ključni mehanizam koji osigurava doslednost i integritet podataka kada više transakcija istovremeno pristupa i menja iste podatke. MySQL koristi mehanizme zaključavanja kako bi regulisao pristup resursima baze podataka i sprečio situacije u kojima bi moglo doći do neslaganja podataka. Postoje 2 različite vrste zaključavanja, a to su zaključavanje čitanja (Read Lock) i zaključavanje pisanja (Write Lock), koje se primenjuju u zavisnosti od vrste operacija i nivoa izolacije.

Kroz pravilnu obradu transakcija, planiranje izvršenja, izolaciju i zaključavanje, MySQL pruža moćne alate za upravljanje i osiguranje integriteta podataka u bazi podataka. Razumevanje ovih koncepata omogućava programerima, administratorima baza podataka i inženjerima da razviju efikasne, pouzdane i skalabilne aplikacije. Cilj rada je da pruži detaljan uvid u sve navedeno.

# 1. Transakcije u MySQL-u

Transakcija je sekvencijalna grupa SQL upita kao što su select, insert, update i delete koje se izvršavaju kao jedna celina ili radna jedinica. Transakcija može biti potvrđena (committed) ili poništena (roll back). Tokom obrade transakcije baza podataka se nalazi u nekonzistentnom stanju jer postoje aktivne operacije koje vrše promene nad bazom. Da bi transakcija bila uspešna, svaka izvršena operacija mora biti committed, tj. transakcija ne može biti uspešna bez uspešnog izvršenja svake operacije u okviru nje. Ako bilo koja naredba transakcije bude neuspešna, celokupna transakcija se poništava, a promene se ne primenjuju na bazu podataka.



Slika 1. Ilustracija ACID svojstava MySQL-a

Transakcije u MySQL-u imaju ACID svojstva[2]:

- **Atomičnost (Atomicity)**

Transakcija se smatra atomičnom jedinicom, što znači da se sve operacije unutar transakcije izvršavaju u potpunosti ili se nijedna ne primenjuje. Ako jedna operacija u transakciji ne uspe, sve promene izvršene do tada se poništavaju (roll back), a baza podataka ostaje nepromenjena. Ako su sve operacije uspešne, transakcija se potvrđuje (commit) i promene postaju trajne. Ukoliko transakcija treba da ažurira 100 redova, a sistem iznenada otkáže nakon 55 ažuriranja, baza podataka poništava promene svih 55 redova i tako vraća prethodno stanje.

- **Konzistentnost (Consistency)**

Odnosi se na održavanje konzistentnog stanja baze podataka tokom izvršavanja transakcija. To znači da transakcije moraju poštovati sva definisana ograničenja i pravila baze podataka, a ako transakcija narušava bilo koje ograničenje ili pravilo, ona se poništava i baza ostaje u prethodnom konzistentnom stanju. Na primer, ako postoji pravilo da neka kolona tabele baze može imati samo Integer vrednosti, konzistentnost omogućava da se ovo pravilo uvek prati i da vrednosti koje se upisuju u ovu kolonu mogu biti isključivo Integer tipa i ništa drugo.

- **Izolacija (Isolation)**

Izolacija se odnosi na sposobnosti izvršavanja transakcija nezavisno i neometano od drugih istovremenih transakcija koje se izvršavaju u sistemu. Dakle, učinak transakcije nije vidljiv drugim transakcijama dok se transakcija ne izvrši. Obezbeđuje da se rezultati jedne transakcije ne mešaju sa rezultatima drugih transakcija koje rade sa istim podacima. To se postiže primenom odgovarajućih mehanizama zaključavanja i upravljanja

konkurentonšću kako bi se sprečile nekonzistentne ili neželjene interakcije među transakcijama. MySQL podržava sledeće nivoe izolacije:

1. READ UNCOMMITTED
2. READ COMMITTED
3. REPEATABLE READ
4. SERIALIZABLE

- **Trajnost (Durability)**

Trajnost se odnosi na postojanost promena izvršenih u transakciji, čak i u slučaju sistemskih kvarova, padova ili oporavka sistema. Nakon što je transakcija potvrđena (committed), promene postaju trajne i ostaju sačuvane u bazi podataka čak i ako se dogodi neuspeh hardvera ili softvera. To se postiže pouzdanim mehanizmima zapisivanja i sinhronizacije podataka na disk.

## 1.1. Obrada transakcija

MySQL koristi određene naredbe za upravljanje transakcijama [1][7]:

1. **START TRANSACTION:** naredba koja označava početak transakcije. MySQL takođe nudi i "BEGIN" i "BEGIN WORK" kao sinonime za START TRANSACTION. Ova naredba definiše početak transakcije i postavlja početnu tačku za sve kasnije promene.
2. **COMMIT:** COMMIT naredba se koristi za potvrdu trenutne transakcije. Kada se COMMIT izvrši, sve promene izvršene tokom transakcije postaju trajne i reflektuju se u bazi podataka. To znači da se sve promene koje su napravljene tokom transakcije "zaključavaju" i ne mogu se poništiti.
3. **ROLLBACK:** ROLLBACK naredba se koristi za poništavanje trenutne transakcije. Kada se ROLLBACK izvrši, sve promene izvršene tokom transakcije se odbacuju i baza podataka se vraća u prethodno stanje pre početka transakcije. Ovo se koristi u slučajevima kada je potrebno odustati od transakcije zbog greške ili neželjenih rezultata.
4. **SET autocommit:** SET autocommit naredba se koristi za omogućavanje ili onemogućavanje automatskog potvrđivanja transakcija. Kada je autocommit uključen (podrazumevano), svaka SQL naredba se tretira kao zasebna transakcija i automatski se potvrđuje. Međutim, ako se autocommit isključi, transakcije se moraju eksplicitno potvrditi ili poništiti pomoću COMMIT ili ROLLBACK naredbe. Kada se koristi START TRANSACTION autocommit je disabled sve dok se ne završi transakcija sa COMMIT ili ROLLBACK. Moguće je eksplicitno zabraniti autocommit mod korišćenjem SET autocommit naredbe:

SET autocommit = 0; ili SET autocommit = OFF;
---

Da bi se nakon toga autocommit enable-ovao, koristi se ista naredba:

SET autocommit = 1; ili SET autocommit = ON;
--

5. **SAVEPOINT**: naredba koja kreira naznačenu tačku unutar transakcije i omogućava nam postavljanje određene tačke u transakciji sa koje se mogu ponišiti promene. Korišćenjem savepoint-a transakcija se može podeliti na manje segmente i selektivno se mogu poništiti promene do određenog savepoint-a, umesto da se poništi cela transakcija. Ovo je korisno kada želim da poništimo samo deo promena napravljenih u okviru transakcije. Sintaksa je sledeća [8]:

SAVEPOINT ime_savepointa;
---------------------------

6. **ROLLBACK TO**: naredba se koristi za vraćanje transakcije na prethodno definisani savepoint. Ona odbacuje sve promene napravljene u transakciji nakon naznačenog savepoint-a i vraća stanje baze podataka na ono koje je bilo kada je savepoint postavljen. Sintaksa je sledeća [8]:

ROLLBACK TO ime_savepointa;
-----------------------------

7. **RELEASE SAVEPOINT**: naredba za uklanjanje određenog savepoint-a unutar transakcije. Oslobađa resurse koji su vezani za savepoint, ali ne poništava promene napravljene nakon savepointa. Ova naredba se koristi kada neki savepoint više nije potreban i želimo da nastavimo sa transakcijom bez mogućnosti vraćanja na taj savepoint. Kada se savepoint jednom oslobodi, nije ga moguće vratiti. Sintaksa je sledeća [8]:

RELEASE SAVEPOINT ime_savepointa;
-----------------------------------

Ove naredbe omogućavaju programerima da upravljaju tokom transakcija, obezbeđujući da promene budu potvrđene samo kada su spremne za trajno čuvanje ili se mogu poništiti u slučaju potrebe. Pružaju nam detaljnu kontrolu nad transakcijom i omogućavaju nam da upravljamo procesom vraćanja unazad sa različitim tačkama unutar transakcije. Daju nam određenu fleksibilnost pri poništavanju promena i mogu biti posebno korisne u složenim scenarijima.

### 1.1.1. Primer COMMIT naredbe

Za primer je korišćena **film** tabela.

Table: **film**

Columns:	
<b>film_id</b>	smallint UN AI PK
<b>title</b>	varchar(128)
description	text
release_year	year
<b>language_id</b>	tinyint UN
<b>original_language_id</b>	tinyint UN
rental_duration	tinyint UN
rental_rate	decimal(4,2)
length	smallint UN
replacement_cost	decimal(5,2)
rating	enum('G','PG','PG-13','R','NC-17')
special_features	set('Trailers','Commentaries','Deleted Scenes','Behind the Scenes')
last_update	timestamp

Slika 2. Prikaz tabele film

Transakciju započinjemo START TRANSACTION naredbom, zatim vršimo SELECT upit i insert-uemo novi record u tabelu. Korišćenjem COMMIT naredbe transakcija je kompletirana.

```
MySQL 8.0 Command Line Client

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @rental_rate := MAX(rental_rate) FROM sakila.film;
+-----+
| @rental_rate := MAX(rental_rate) |
+-----+
|                                4.99 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> INSERT INTO sakila.film(title, language_id) VALUES ('MILAN', '1');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Slika 3. Prikaz COMMIT naredbe u MySQL-u



### 1.1.2. Primer ROLLBACK naredbe

Tabela koju ćemo koristiti u ovom primeru je tabela student:

Table: **student**

Columns:

<b>idstudent</b>	int PK
name	varchar(45)
gender	varchar(45)
grade	varchar(45)

Slika 4. Prikaz Student tabele

Transakciju započinjemo kao i u prethodnom primeru, START TRANSACTION naredbom. Izvršićemo brisanje redova tabele Student.

```
mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE from sakila.student;
Query OK, 0 rows affected (0.00 sec)
```

Slika 5. Primer ROLLBACK naredbe u MySQL-u

Ako otvorimo posebnu sesiju MySQL servera i izvršimo sledeću naredbu:

```
mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
|          1 | Milan | Male   | 1     |
|          2 | Mila  | Female | 2     |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

Slika 6. Izvršavanje SELECT naredbe

Iako su obrisani svi redovi tabele Student u prvoj sesiji i dalje možemo videti sve redove tabele u drugoj sesiji. Ovo se dešava zbog toga što izmene u prvoj sesiji nisu permanentne dok ne izvršimo COMMIT ili ROLLBACK naredbu. U ovom slučaju možemo izvršiti ROLLBACK naredbu u prvoj sesiji i dobiti sledeće rezultate:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
|          1 | Milan | Male   | 1     |
|          2 | Mila  | Female | 2     |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Slika 7. Prikaz rezultata SELECT naredbe nakon ROLLBACK-a

U MySQL-u nije moguće poništiti baš sve naredbe. Na primer, ove naredbe uključuju **Data Definition Language (DDL)** komande poput CREATE, ALTER ili DROP, kao i CREATE, UPDATE ili DROP tabele. Moramo se pobrinuti da prilikom dizajniranja naše transakcije izbegavamo ove naredbe. To znači da, ukoliko su takve naredbe deo transakcije, promene koje su izvršene neće biti poništene pri izvršavanju ROLLBACK naredbe. Stoga je važno da pravilno upravljamo transakcijama i pažljivo odaberemo naredbe koje su deo transakcije kako bismo osigurali očekivano ponašanje pri poništavanju transakcija.

### 1.1.3. Primer SAVEPOINT i ROLLBACK TO naredbi

**SAVEPOINT** je naredba u MySQL-u koja se koristi za označavanje tačke unutar transakcije na kojoj možemo postaviti privremeno sačuvano stanje. Ova tačka se koristi se kao referenca za eventualno vraćanje transakcije na to stanje. Možemo imati više savepointa unutar jedne transakcije.

Nakon što postavimo savepoint, možemo nastaviti izvršavanje transakcije i vršiti promene nad bazom podataka. Ukoliko se desi neki problem ili želimo da poništimo promene nakon određene tačke, možemo koristiti naredbu **ROLLBACK TO ime\_savepointa** kako bismo se vratili na taj savepoint. Ovo će poništiti sve promene izvršene nakon postavljanja tog savepointa i vratiti transakciju u stanje koje je bilo zabeleženo prilikom postavljanja savepointa.

U konkretnom primeru: započinjemo transakciju i prikazujemo trenutni sadržaj Student tabele. Vršimo insert record-a u tabelu i zatim kreiramo savepoint na koji ćemo se vratiti pomoću ROLLBACK TO naredbe. Na kraju vršimo COMMIT naredbu da bi izmene bile permanentne.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
| 1         | Milan | Male   | 1      |
| 2         | Mila  | Female | 2      |
| 3         | Ana   | Female | 1      |
| 4         | Milos | Male   | 1      |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('5', 'Igor',
-> 'Male', '4');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT a;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('6', 'Marija', 'Female', '2');
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT a;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('7', 'Aca', 'Male', '2');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
| 1         | Milan | Male   | 1      |
| 2         | Mila  | Female | 2      |
| 3         | Ana   | Female | 1      |
| 4         | Milos | Male   | 1      |
| 5         | Igor  | Male   | 4      |
| 7         | Aca   | Male   | 2      |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Slika 8. Primer korišćenja SAVEPOINT i ROLLBACK TO naredbi

Kao što se može videti, u tabeli se ne nalazi record sa id-em 6 jer je izvršeno vraćanje na savepoint kojim se taj insert poništava.

Ukoliko se javi ova greška: `ERROR 1305 (42000): SAVEPOINT identifier does not exist` to znači da ne postoji savepoint sa navedenim imenom.

#### 1.1.4. Primer RELEASE SAVEPOINT naredbe

**RELEASE SAVEPOINT** naredba uklanja prosleđeni savepoint iz trenutne transakcije, pri čemu se ne poništavaju efekti upita koji su izvršeni nakon postavljanja savepointa. Nakon ovih naredbi, neće se vršiti poništavanje transakcija. Ukoliko savepoint ne postoji u transakciji, doći će do greške.

```

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('8', 'Nina', 'Female', '2');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT b;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE sakila.student SET grade = '1' WHERE idstudent = 8;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> RELEASE SAVEPOINT b;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
| 1         | Milan | Male   | 1     |
| 2         | Mila  | Female | 2     |
| 3         | Ana   | Female | 1     |
| 4         | Milos | Male   | 1     |
| 5         | Igor  | Male   | 4     |
| 7         | Aca   | Male   | 2     |
| 8         | Nina  | Female | 1     |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)

```

Slika 9. Primer RELEASE SAVEPOINT naredbe

Ovde su i Insert i Update naredbe izvršene uspešno jer RELEASE SAVEPOINT naredba samo uklanja setovan savepoint.

## 2. Izolacija

*Izolacija transakcija* je ključni koncept u upravljanju bazama podataka koji omogućava konkurentno izvršavanje transakcija i održavanje konzistentnosti podataka. MySQL pruža podršku za različite nivoe izolacije transakcija kako bi se zadovoljile različite potrebe i zahtevi korisnika. Izolacija transakcija se odnosi na sposobnost baze podataka da omogući da se transakcije izvršavaju nezavisno jedna od druge, bez međusobnog mešanja i uticaja na rezultate. Cilj izolacije transakcija je sprečiti problem preplitanja akcija (engl. concurrency issues) i obezbediti da svaka transakcija vidi konzistentno stanje podataka, bez obzira na istovremene promene koje se odvijaju u drugim transakcijama.

### 2.1. Anomalije koje se javljaju u transakcijama

Pri izvršavanju transakcija u MySQL bazi podataka, određene anomalije mogu se javiti kada je nivo izolacije postavljen na manje strogi nivo. Ove anomalije mogu dovesti do nekonzistentnog stanja podataka i ometati doslednost transakcija. Evo nekoliko uobičajenih anomalija koje se mogu javiti:

#### 1. Prljava čitanja (Dirty reads)

Prljava čitanja se javljaju kada jedna transakcija čita nespotvrđene (nedovršene) promene podataka koje je druga transakcija izvršila. Ako ta druga transakcija kasnije odustane (rollback), prljava čitanja mogu dovesti do neispravnih ili netačnih rezultata.

#### 2. Neponavljajuća čitanja (Non-repeatable reads)

Javljaju se kada jedna transakcija čita isti podatak više puta tokom izvršavanja transakcije, ali vrednost tog podatka se menja između čitanja zbog izvršavanja drugih transakcija. To može dovesti do nelogičnosti ili netačnosti u rezultatima čitanja.

#### 3. Fantomska čitanja ili zapisi (Phantom reads)

Fantomska čitanja se javljaju kada jedna transakcija izvrši upit koji vrati skup rezultata, a zatim druga transakcija izvrši promene u podacima koje utiču na taj skup rezultata. Kada prva transakcija ponovno izvrši upit, može se pojaviti dodatni ili "fantomski" zapis koji nije bio prisutan prilikom prvog izvršavanja upita.

Da bi se sprečile ove anomalije, važno je pažljivo odabrati odgovarajući nivo izolacije transakcija i pravilno upravljati konkurentnošću prilikom izvršavanja transakcija. MySQL pruža različite nivoe izolacije kako bi se zadovoljile potrebe korisnika i sprečile ove anomalije.

## 2.2. Nivoi izolacije

Nivo izolacije transakcija je jedan od ključnih faktora prilikom upravljanja i kontrolisanja konzistentnosti podataka u bazi podataka. U MySQL-u, kao i u većini modernih baza podataka, pruža se podrška za različite nivoe izolacije. Nivo izolacije transakcija definiše kako se transakcije međusobno vide i kako se promene koje vrši jedna transakcija odražavaju na druge transakcije. Svaki nivo izolacije pruža određeni balans između konzistentnosti podataka, pouzdanosti i performansi.

Kao što je već ranije pomenuto, MySQL nudi nekoliko nivoa izolacije transakcija koji se mogu postaviti prilikom pokretanja transakcije i svaki nivo ima određene karakteristike. To su sledeća 4 nivoa:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

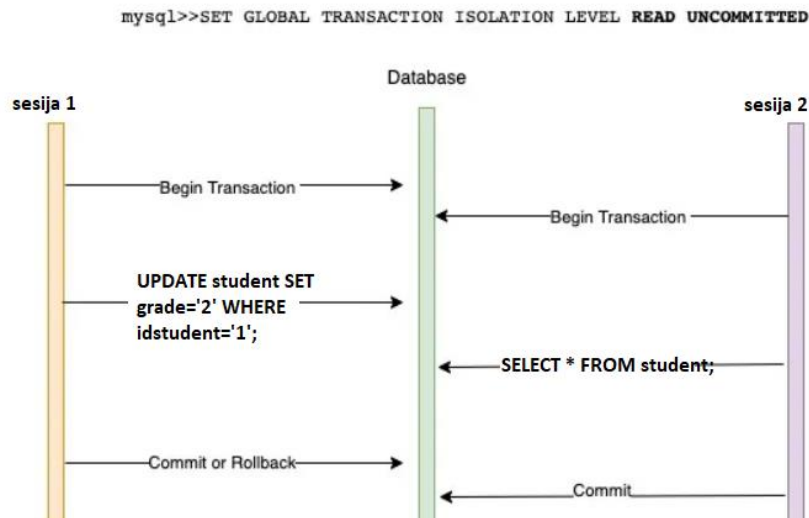
MySQL podrazumevano koristi **READ COMMITTED** nivo izolacije, ali je to moguće promeniti korišćenjem **SET TRANSACTION ISOLATION LEVEL** naredbe.

Nivoi izolacije su usko povezani sa pojavom određenih anomalija [3]:

Nivo izolacije	Dirty reads	Non-repeatable reads	Phantom reads
READ UNCOMMITTED	✓	✓	✓
READ COMMITTED	✗	✓	✓
REPEATABLE READ	✗	✗	✓
SERIALIZABLE	✗	✗	✗

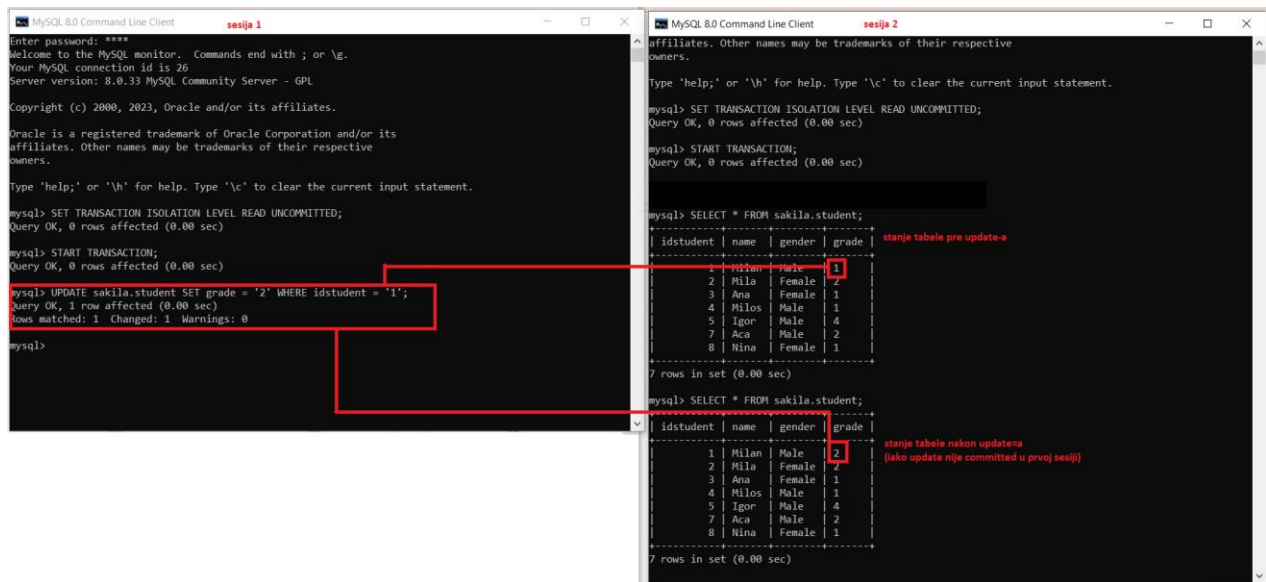
### 2.2.1. READ UNCOMMITTED (Nepotvrđeno čitanje)

Najniži nivo izolacije transakcija na kome transakcija može čitati podatke koji još nisu potvrđeni od strane drugih transakcija. To znači da druge transakcije mogu menjati podatke koje trenutna transakcija čita, ali te promene možda neće biti vidljive sve dok se operacija ne završi.



Slika 10. Grafički prikaz READ UNCOMMITTED izvršavanja

Ovaj nivo izolacije dozvoljava da jedna transakcija može da vidi nepotvrđene promene napravljene od strane neke druge transakcije, tj. ovaj nivo dozvoljava *prljava čitanja*.



Slika 11. Primer READ UNCOMMITTED naredbe

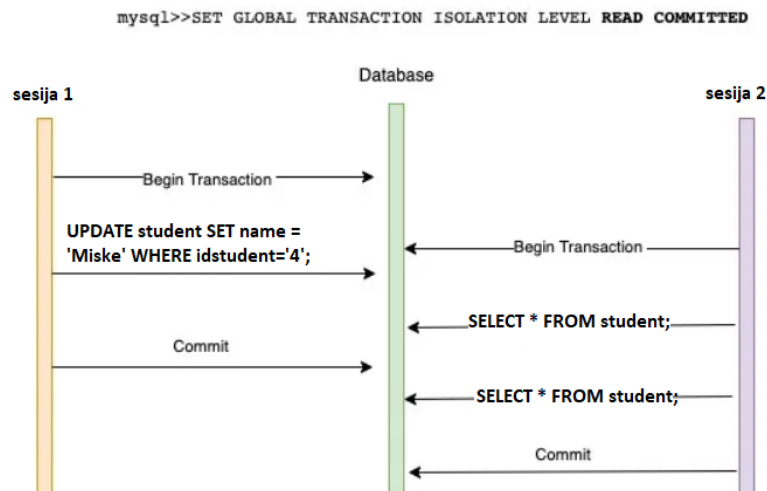
Važna karakteristika READ UNCOMMITTED nivoa izolacije je *nedostatak zaključavanja* (locking). Budući da transakcija čita nespremne promene iz drugih transakcija, ne postoji potreba za zaključavanjem redova ili tabela. To može poboljšati performanse jer neće biti blokiranja prilikom čitanja podataka. Međutim, upotreba READ UNCOMMITTED nivoa izolacije nosi sa sobom određene rizike. Budući da transakcija čita nespremne promene, te promene se mogu otkazati ili izmeniti pre nego što budu potvrđene. To znači da transakcija može videti privremene

promene koje nikada neće postati trajne. Ovo može dovesti do nekonzistentnih rezultata i pogrešnog tumačenja podataka. Zbog ovih rizika, READ UNCOMMITTED nivo izolacije se retko koristi u praksi, posebno u aplikacijama gde je konzistentnost podataka od velikog značaja.

### 2.2.2. READ COMMITTED (Potvrđeno čitanje)

READ COMMITTED je nivo izolacije transakcija u MySQL-u koji pruža veću konzistentnost od READ UNCOMMITTED nivoa, ali takođe i dalje dozvoljava neke od anomalija vezanih za izolaciju.

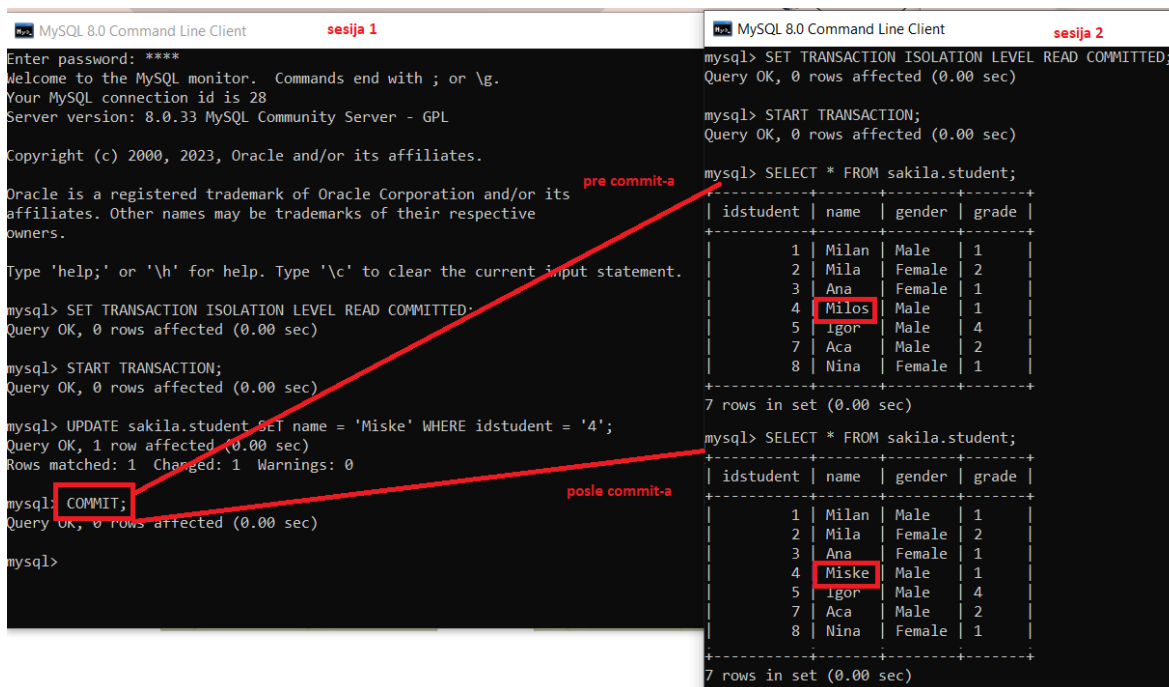
Na ovom nivou, transakcija može čitati samo podatke koji su već potvrđeni od strane drugih transakcija. To znači da druge transakcije mogu menjati podatke koje transakcija trenutno čita, ali te promene neće biti vidljive sve dok druga transakcija ne bude potvrđena. U ovom novou je prevaziđen problem prljavih čitanja, ali postoji problem *non-repeatable čitanja*.



Slika 12. Grafički prikaz primera READ UNCOMMITTED naredbe

U toku iste transakcije iste SELECT naredbe treba da vrate iste rezultate, ali zbog non-repeatable čitanja dobijaju se različiti rezultati jer se u međuvremenu vrše potvrđene izmene podataka u drugoj transakciji.





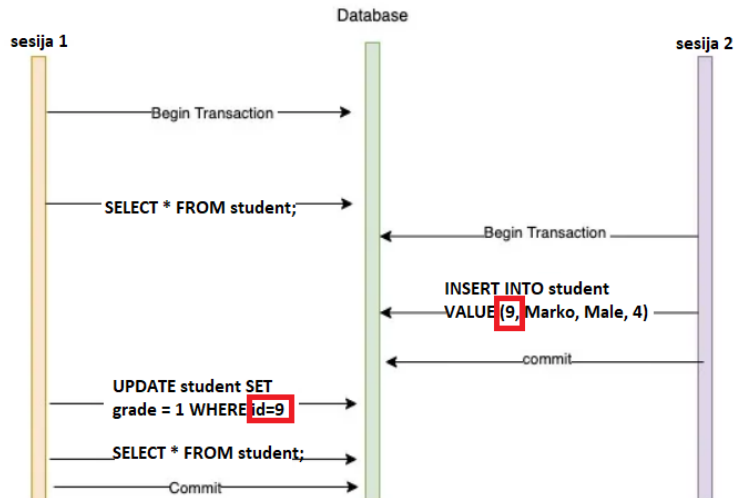
Slika 13. Primer READ COMMITTED naredbe

Uprkos prisutnim anomalijama, READ COMMITTED nivo izolacije se često koristi u praksi jer pruža dobar balans između konzistentnosti podataka i performansi sistema.

### 2.2.3. REPEATABLE READ (Ponovljeno čitanje)

REPEATABLE READ je jedan od viših nivoa izolacije transakcija u MySQL-u i transakcija može čitati samo podatke koji su već potvrđeni od strane drugih transakcija, a takođe sprečava druge transakcije da menjaju podatke koje trenutno čita. Dakle, U ovom nivou izolacije, transakcija garantuje da će ponovljena čitanja (SELECT naredbe) vratiti iste rezultate tokom izvršavanja transakcije, bez obzira na druge izvršene transakcije.

U REPEATABLE READ nivou izolacije, transakcija pravi snimak podataka na početku izvršavanja, uključujući broj redova koji zadovoljavaju uslov SELECT upita. Međutim, ako se tokom izvršavanja jedne transakcije pojave novi redovi (iz druge transakcije) u tabeli koji zadovoljavaju uslov SELECT upita, oni će biti vidljivi kao "fantomi" prilikom ponovljenog izvršavanja istog upita. Na primer, ako transakcija izvrši SELECT upit koji vraća sve redove u tabeli sa određenim uslovom, tokom izvršavanja te transakcije druga transakcija može dodati nove redove koji zadovoljavaju taj isti SELECT uslov. Kada prva transakcija ponovo izvrši SELECT upit, pojaviće se dodatni redovi koji nisu bili prisutni prilikom prvog izvršavanja, stvarajući efekat *fantomskog čitanja*.



Slika 14. Grafički prikaz primera REPEATABLE READ-a

U ovom primeru se jasno vidi pojava fantomskog čitanja. Dakle, u prvoj transakciji se čitaju svi podaci iz tabele Student, nakon čega se u drugoj transakciji vrši insert novog reda u istu tu tabelu i taj insert se potvrđuje. Kada se u prvoj transakciji ponovo izvrši pribavljanje svih podataka iz Student tabele, dobiće se rezultat različit od prvog, jer sada postoji dodatni red.

```

mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+----+-----+-----+-----+
| idstudent | name | gender | grade |
+----+-----+-----+-----+
| 1 | Milan | Male | 1 |
| 2 | Mila | Female | 2 |
| 3 | Ana | Female | 1 |
| 4 | Miske | Male | 1 |
| 5 | Igor | Male | 4 |
| 6 | Aca | Male | 2 |
| 7 | Nina | Female | 1 |
+----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> UPDATE sakila.student SET grade = '1' WHERE idstudent = '9';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM sakila.student;
+----+-----+-----+-----+
| idstudent | name | gender | grade |
+----+-----+-----+-----+
| 1 | Milan | Male | 1 |
| 2 | Mila | Female | 2 |
| 3 | Ana | Female | 1 |
| 4 | Miske | Male | 1 |
| 5 | Igor | Male | 4 |
| 6 | Aca | Male | 2 |
| 7 | Nina | Female | 1 |
| 9 | Marko | Male | 1 |
+----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('9', 'Marko', 'Male', '4');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
  
```

Slika 15. Primer REPEATABLE READ naredbe

#### 2.2.4. SERIALIZABLE

Najviši i najjači nivo izolacije u MySQL-u. U ovom nivou izolacije, transakcije su potpuno izolovane jedna od druge, što znači da transakcije neće videti promene koje su napravljene od strane drugih transakcija sve do završetka trenutne transakcije. Ovo osigurava potpunu doslednost podataka tokom izvršavanja transakcija. Takođe sprečava druge transakcije da menjaju podatke koje trenutna transakcija čita, kao i da dodaju nove redove koji bi bili vidljivi trenutnoj transakciji [4].

Ključna karakteristika SERIALIZABLE nivoa izolacije je da *onemogućava fantomsko čitanje* (phantom reads), što je problem koji se javlja u REPEATABLE READ nivou. Fantomsko čitanje se dešava kada jedna transakcija izvrši SELECT upit nad određenim setom redova, a tokom izvršavanja druge transakcije se unesu novi redovi koji zadovoljavaju taj uslov, što rezultira pojavom "fantom" u podacima koje prva transakcija čita. U SERIALIZABLE nivou izolacije, takve promene neće biti vidljive sve do završetka trenutne transakcije, čime se eliminiše problem fantomskog čitanja. Kako bi postigao SERIALIZABLE nivo izolacije, MySQL koristi mehanizme zaključavanja (locking) podataka. Zaključavanje sprečava druge transakcije da izvršavaju operacije koje bi mogle da utiču na podatke sa kojima trenutna transakcija radi. To znači da, dok jedna transakcija čita ili menja određene podatke, druge transakcije moraju čekati dok se trenutna transakcija ne završi. Ovo obezbeđuje da se podaci ne menjaju ili ne čitaju dok je transakcija u toku, čime se osigurava potpuna izolacija transakcija. Međutim, važno je imati na umu da SERIALIZABLE nivo izolacije može dovesti do većeg broja konflikata i čekanja između transakcija, što može negativno uticati na performanse sistema. Stoga se preporučuje korišćenje SERIALIZABLE nivoa izolacije samo kada je potrebna apsolutna doslednost podataka i kada se fantomsko čitanje mora potpuno eliminisati.

Dolazimo do pitanja: koji je najbolji nivo izolacije za korišćenje? Najbolji nivo izolacije transakcija u MySQL-u zavisi od konkretnih zahteva i potreba aplikacije. Ne postoji univerzalno "najbolji" nivo izolacije jer svaki nivo ima svoje prednosti i ograničenja. Važno je razmotriti sledeće faktore prilikom odabira nivoa izolacije:

- Ako je održavanje potpune konzistentnosti podataka prioritet, stroži nivoi izolacije poput REPEATABLE READ ili SERIALIZABLE mogu biti preporučljivi. Oni će sprečiti anomalije kao što su prljava čitanja ili fantomski zapisi.
- Performanse: Stroži nivoi izolacije mogu dovesti do veće upotrebe zaključavanja i blokiranja, što može uticati na performanse sistema. Ako aplikacija ima veliku količinu istovremenih transakcija i zahteva visoku propusnost, možda će biti bolje koristiti niže nivoe izolacije kao što je READ COMMITTED.

U većini slučajeva, preporuka je da se koristi podrazumevani nivo izolacije u MySQL-u, a to je READ COMMITTED. On pruža dobar balans između konzistentnosti i performansi.

### 3. Zaključavanje u MySQL-u

Zaključavanje (locking) je mehanizam koji se koristi u MySQL-u radi održavanja doslednosti podataka i sprečavanja konflikata između transakcija koje istovremeno pristupaju istim podacima. Kroz zaključavanje, MySQL omogućava kontrolu pristupa podacima i osigurava da transakcije bezbedno i dosledno izvršavaju svoje operacije.

Postoje dve vrste zaključavanja u MySQL-u:

- zaključavanje čitanja (READ LOCK)
- zaključavanje pisanja (WRITE LOCK).

**Zaključavanje čitanja (READ LOCK)** omogućava transakcijama da čitaju podatke bez ometanja drugih transakcija. Transakcija koja postavi zaključavanje čitanja na određene podatke sprečava druge transakcije da izvrše operacije koje bi mogle promeniti te podatke dok je zaključavanje aktivno. Više transakcija može imati zaključavanje čitanja na istim podacima u isto vreme, jer čitanje neće prouzrokovati konflikte.

**Zaključavanje pisanja (WRITE LOCK)** omogućava transakcijama da izvrše operacije pisanja nad podacima, dok istovremeno sprečava druge transakcije da pristupe ili menjaju te podatke. Kada transakcija postavi zaključavanje pisanja na određene podatke, druge transakcije neće biti u mogućnosti da čitaju ili pišu u te podatke dok zaključavanje nije oslobođeno. Zaključavanje pisanja je ekskluzivno i može biti postavljeno samo od strane jedne transakcije u isto vreme.

MySQL koristi različite nivoe zaključavanja u skladu sa nivoom izolacije transakcija. Na primer, u READ COMMITTED nivou izolacije, MySQL koristi zaključavanje na nivou reda (row-level locking), gde se zaključavanje postavlja na pojedinačne redove u tabelama. Ovo omogućava da više transakcija čitaju različite redove iste tabele paralelno. Međutim, kada jedna transakcija vrši operaciju pisanja nad određenim redom, zaključavanje će biti postavljeno na taj red kako bi se sprečilo čitanje ili pisanje drugim transakcijama dok je operacija pisanja u toku.

MySQL takođe podržava različite režime zaključavanja, kao što su **implicitna zaključavanja (implicit locking)** i **eksplicitna zaključavanja (explicit locking)**. Implicitna zaključavanja se automatski primenjuju na operacije nad podacima, dok eksplicitna zaključavanja zahtevaju da programer ručno postavi i oslobodi zaključavanja putem naredbi poput "LOCK TABLES" i "UNLOCK TABLES". Ovi režimi zaključavanja pružaju veću fleksibilnost i kontrolu nad upravljanjem zaključavanjima u transakcijama.

U MySQL-u postoje različiti nivoi zaključavanja koja se mogu koristiti za kontrolu pristupa podacima:

1. **Table Locking** – omogućava eksplicitno zaključavanje tabele. Kada se tabela zaključa, druge sesije ne mogu pristupiti tabeli dok se zaključavanje ne oslobodi. MySQL pruža naredbe poput LOCK TABLES i UNLOCK TABLES za upravljanje zaključavanjem tabele.

2. **Row Locking** – omogućava zaključavanje pojedinačnih vrsta/redova unutar tabele, što pruža paralelnost pristupa podacima tako što dozvoljava da više sesija pristupaju različitim redovima iste tabele istovremeno.
3. **Page Locking** – zaključavanje stranice podataka tj. jedinice podataka koja sadrži više redova tabele. Kada se jedna stranica zaključa, sve operacije nad redovima unutar te stranice će čekati dok se zaključavanje ne oslobodi. Page Locking je bio dosutan u starijim verzijama MySQL-a [6].

### 3.1. Table Locking

Naredba **LOCK TABLES** u MySQL-u se koristi za eksplicitno zaključavanje tabela tokom izvršavanja upita. Ova naredba omogućava jednoj sesiji da zaključa jednu ili više tabela za čitanje ili pisanje, sprečavajući druge sesije da pristupe tim tabelama sve dok zaključavanje nije oslobođeno [9].

Opšti oblik naredbe LOCK TABLES je:

```
LOCK TABLES ime_tabele {READ | [WRITE]} [, ime_tabele2 {READ | [WRITE]} ...];
```

- "ime\_tabele, ime\_tabele2, ..." su imena tabela koje se zaključavaju.
- "READ" označava zaključavanje za čitanje, dok "WRITE" označava zaključavanje za pisanje.

Dakle, ovom naredbom može se izvršiti zaključavanje više tabela istovremeno.

Kada se izvrši naredba LOCK TABLES, MySQL će zaključati navedene tabele u okviru trenutne sesije. Ova zaključavanja mogu biti tabela za čitanje ili tabela za pisanje, u zavisnosti od navedenog režima zaključavanja. Kada se koristiti LOCK TABLES naredba, mora postojati doza opreznosti jer može doći do potencijalnih problema istovremenosti. Na primer, ako jedna sesija zaključa tabelu za pisanje, to može sprečiti druge sesije da pristupe toj tabeli, što može dovesti do zastoja u radu ili čekanja na oslobađanje zaključavanja.

Da bi se oslobodile zaključane tabele i omogućio pristup drugim sesijama, koristi se naredba **UNLOCK TABLES**:

```
UNLOCK TABLES;
```

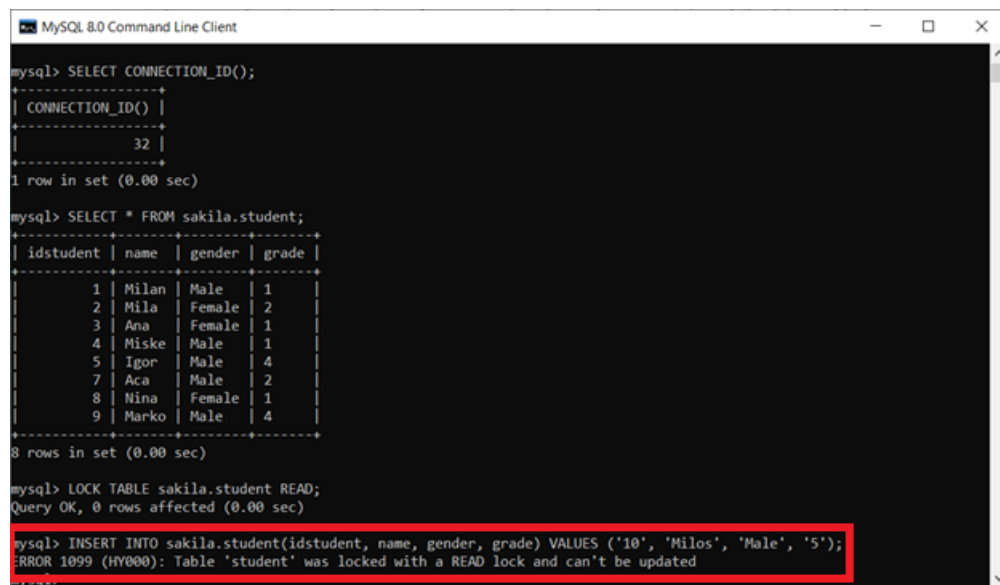
Kada se izvrši UNLOCK TABLES, MySQL će osloboditi sva zaključavanja koja su bila aktivna u okviru trenutne sesije. Treba imati na umu da se "UNLOCK TABLES" automatski izvršava pri završetku sesije, tako da u većini slučajeva nije potrebno eksplicitno pozivati ovu naredbu.

### 3.1.1. READ LOCK

READ lock omogućava sesijama da čitaju podatke iz tabele, dok istovremeno sprečava druge sesije da izvrše operacije pisanja na tu tabelu. Evo nekoliko ključnih karakteristika READ zaključavanja:

- Više sesija može izvršiti READ zaključavanje za istu tabelu istovremeno. To znači da se više sesija može čitati podatke iz tabele istovremeno, bez da se međusobno blokiraju.
- Sesija koja drži READ zaključavanje na tabeli može samo čitati podatke iz nje, ali ne može izvršiti operacije pisanja (INSERT, UPDATE, DELETE) nad tom tabelom. Ovo osigurava konzistentnost podataka, sprečavajući neovlašćene izmene dok se tabela čita.
- Druge sesije koje nisu izvršile READ zaključavanje mogu čitati podatke iz tabele bez eksplicitnog zaključavanja. To omogućava paralelno čitanje podataka iz tabele od strane različitih sesija bez međusobnih blokada.
- Kada sesija završi izvršavanje ili se prekine na normalan ili nenormalan način, MySQL implicitno oslobađa sva zaključavanja, uključujući i READ zaključavanje.

READ zaključavanje je korisno u situacijama gde više sesija želi samo čitati podatke iz tabele, ali ne i izvršavati operacije pisanja. Ovo omogućava efikasno i paralelno čitanje podataka bez međusobnih blokada, što može poboljšati performanse sistema.



```
mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          32      |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
|         1 | Milan | Male   | 1     |
|         2 | Mila  | Female | 2     |
|         3 | Ana   | Female | 1     |
|         4 | Miske | Male   | 1     |
|         5 | Igor  | Male   | 4     |
|         7 | Aca   | Male   | 2     |
|         8 | Nina  | Female | 1     |
|         9 | Marko | Male   | 4     |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> LOCK TABLE sakila.student READ;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('10', 'Milos', 'Male', '5');
ERROR 1099 (HY000): Table 'student' was locked with a READ lock and can't be updated
```

Slika 16. Primer READ LOCK-a

Kao što se i primeru vidi, nakon što je tabela zaključana za čitanje ne mogu se vršiti modifikacije nad njom u istoj sesiji. Možemo proveriti kako READ lock radi u drugoj sesiji:

```
MySQL 8.0 Command Line Client
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          35      |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name  | gender | grade |
+-----+-----+-----+-----+
| 1         | Milan | Male   | 1      |
| 2         | Mila  | Female | 2      |
| 3         | Ana   | Female | 1      |
| 4         | Miske | Male   | 1      |
| 5         | Igor  | Male   | 4      |
| 7         | Aca   | Male   | 2      |
| 8         | Nina  | Female | 1      |
| 9         | Marko | Male   | 4      |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('10', 'Milos', 'Male', '5');
```

Slika 17. Primer READ LOCK-a u drugoj sesiji

Dakle, ni ovde se ne može vršiti modifikacija, ne izlazi greška ali je naredba u procesu izvršenja.

Time	Action	Message	Duration / Fetch
14:15:26	INSERT INTO	Running...	?

Slika 18. Waiting status naredbe

Zapravo je naredba u Waiting statusu i to je zbog READ lock-a iz prethodne sesije. Onog momenta kada se izvrši naredba UNLOCK TABLES u prvoj sesiji, tada će se izvršiti i INSERT INTO naredba u drugoj sesiji.

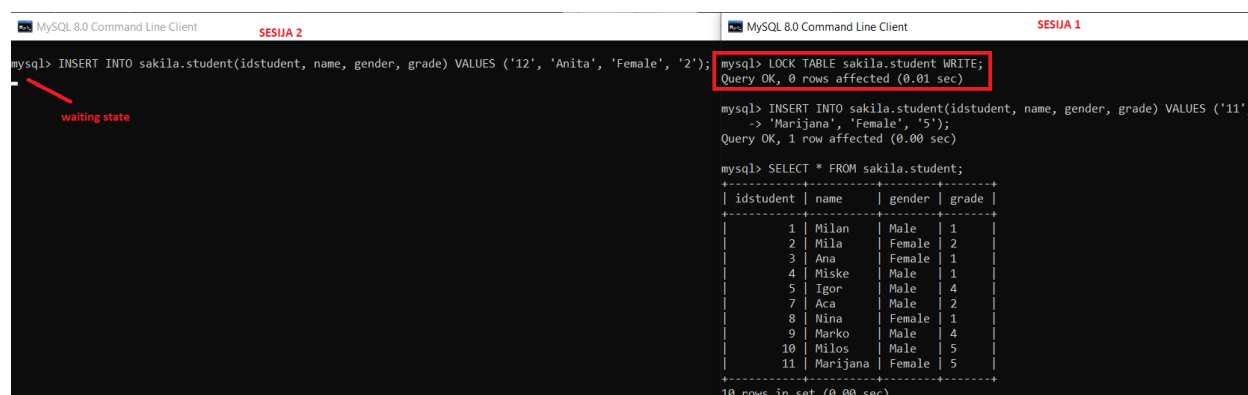
### 3.1.2. WRITE LOCK

WRITE zaključavanje omogućava sesiji da čita i piše podatke u tabeli, dok istovremeno sprečava druge sesije da pristupe toj tabeli dok se WRITE zaključavanje ne oslobodi. Samo jedna sesija može izvršiti WRITE zaključavanje za tabelu u datom trenutku. To znači da samo sesija koja drži WRITE zaključavanje može izvršiti operacije čitanja i pisanja nad tom tabelom, dok druge sesije moraju čekati da se zaključavanje oslobodi. WRITE zaključavanje blokira sve zahteve za zaključavanje tabele od strane drugih sesija dok je zaključavanje aktivno. Ovo osigurava da samo jedna sesija može menjati podatke u tabeli u datom trenutku, sprečavajući konflikte i očuvajući konzistentnost podataka. **WRITE zaključavanje ima veći prioritet od READ zaključavanja.** To znači da ako sesija izvrši READ zaključavanje za tabelu, a druga sesija zahteva WRITE zaključavanje, zahtev za READ zaključavanje će čekati dok sesija koja zahteva

WRITE zaključavanje ne završi svoje operacije. Kada sesija završi izvršavanje ili se prekine na normalan ili nenormalan način, MySQL implicitno oslobađa sva zaključavanja, uključujući i WRITE zaključavanje.

WRITE zaključavanje je korisno kada sesija treba da izvrši operacije čitanja i pisanja nad tabelom, a želi sprečiti druge sesije da pristupe toj tabeli dok je zaključavanje aktivno. Ovo se često koristi u transakcijama gde je potrebno očuvati integritet podataka i sprečiti konflikte prilikom upisa podataka. Međutim, korišćenje WRITE zaključavanja može dovesti do potencijalnih problema sa performansama, posebno u situacijama sa velikim brojem zahteva za zaključavanje.

U prvoj sesiji izvršićemo WRITE LOCK tabele Student i nakon toga izvršiti INSERT novog reda u tabelu. Nakon toga možemo pokušati pristup READ ili WRITE tabele Student iz druge sesije i to neće biti moguće, izvršenje naredbe će biti na čekanju dok se ne izvrši naredba UNLOCK TABLES u prvoj sesiji.



```
mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('12', 'Anita', 'Female', '2');
waiting state

mysql> LOCK TABLE sakila.student WRITE;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO sakila.student(idstudent, name, gender, grade) VALUES ('11',
-> 'Marijana', 'Female', '5');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM sakila.student;
+-----+-----+-----+-----+
| idstudent | name   | gender | grade |
+-----+-----+-----+-----+
| 1 | Milan | Male   | 1     |
| 2 | Mila  | Female | 2     |
| 3 | Ana   | Female | 1     |
| 4 | Miske | Male   | 1     |
| 5 | Igor  | Male   | 4     |
| 7 | Aca   | Male   | 2     |
| 8 | Nina  | Female | 1     |
| 9 | Marko | Male   | 4     |
| 10 | Milos | Male   | 5     |
| 11 | Marijana | Female | 5     |
+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

Slika 19. Primer WRITE LOCK-

## 3.2. Row Locking

Row locking, takođe poznat kao zaključavanje reda, je mehanizam zaključavanja koji omogućava kontrolu pristupa pojedinačnim redovima unutar tabele. U MySQL-u, row locking se koristi da omogući da više sesija istovremeno pristupaju i modifikuju različite redove unutar iste tabele, što povećava paralelnost i efikasnost rada. Kada sesija pristupa redovima u tabeli i izvršava operacije čitanja ili pisanja, može postaviti zaključavanje na nivou pojedinačnog reda. Ovo osigurava da nijedna druga sesija ne može izvršiti operaciju koja bi mogla narušiti konzistentnost ili integritet podataka u tom redu.

MySQL koristi različite vrste zaključavanja na nivou reda [5]:

1. **Shared Locking (S-lock):** Shared lock omogućava sesijama da istovremeno pristupaju redovima u read-only modu. *Sesija koja ima shared lock može čitati podatke, ali ne može vršiti modifikacije.* Više sesija može imati shared lock na istom redu istovremeno.



2. **Exclusive Locking (X-lock):** Exclusive lock omogućava samo jednoj sesiji da izvršava operacije čitanja i pisanja nad redom. Kada sesija ima exclusive lock na redu, nijedna druga sesija ne može pristupiti redu sve dok se zaključavanje ne oslobodi. Ovo se koristi kada je potrebno izvršiti modifikacije nad podacima kako bi se održala konzistentnost.

Redovi nisu automatski zaključani prilikom čitanja ili pisanja. Umesto toga, zaključavanje se primenjuje samo kada postoji konflikt između sesija koje pristupaju istom redu. Na taj način se smanjuje broj blokada i povećava efikasnost pristupa podacima.

### 3.2.1. Primer Shared Row Locking-a

Sintaksa je sledeća:

```
SELECT ... FROM table_name WHERE ... LOCK IN SHARE MODE;
```

Dakle, bitno je navesti sufiks: LOCK IN SHARE MODE.

- U sledećem primeru, sesija 1 i sesija 2 imaju shared lock nad vrstom čiji je film\_id jednak 1. Obe sesije mogu čitati podatke iz te vrste, ali ne mogu izvršiti modifikacije. Sesija 3 pokušava izvršiti UPDATE operaciju nad istom tom vrstom, ali je blokirana jer su shared lockovi već prisutni:

**Sesija 1** ima shared lock nad vrstom sa film\_id = 1 i može čitati podatke:

```
START TRANSACTION;  
SELECT * FROM film WHERE film_id = 1 LOCK IN SHARE MODE;
```

**Sesija 2** ima shared lock nad vrstom sa film\_id = 1 i takođe može čitati podatke

```
START TRANSACTION;  
SELECT * FROM film WHERE film_id = 1 LOCK IN SHARE MODE;    Works
```

**Sesija 3** ne može izvršiti UPDATE operaciju jer sesija 1 i sesija 2 već drže shared lock na vrsti

```
START TRANSACTION;  
UPDATE film SET title = 'novi naslov' WHERE film_id = 1;    Not working...
```

- Drugi primer:

**Sesija 1:**

```
START TRANSACTION;  
SELECT * FROM actor WHERE actor_id = 1 LOCK IN SHARE MODE;
```

## Sesija 2:

```
UPDATE actor SET first_name = 'Adam' WHERE actor_id = 2;
```

*Radi jer vrsta 2 nije locked.*

```
UPDATE actor SET last_name = 'Smith' WHERE actor_id = 1;
```

*Čeka zbog zaključavanja vrste 1 u sesiji 1.*

### 3.2.2. Primer Exclusive Row Locking-a

Sintaksa je sledeća:

```
SELECT ... FROM table_name WHERE ... FOR UPDATE;
```

- Kada nema transakcije, lock automatski uklanja i moguće je vršiti modifikacije nad podacima vrste:

```
SELECT * FROM actor WHERE actor_id = 2 FOR UPDATE;
```

```
UPDATE actor SET first_name = 'Adam' WHERE actor_id = 2;      Works
```

- U sledećem primeru, sesija 1 ima exclusive lock nad vrstom čiji je film\_id jednak 1 i može izvršiti i čitanje i pisanje nad tom vrstom. Sesija 2 pokušava dobiti exclusive lock nad istom vrstom, ali čeka jer je sesija 1 već drži. Sesija 3 pokušava izvršiti UPDATE operaciju, ali je blokirana jer je exclusive lock prisutan.

**Sesija 1** ima exclusive lock nad vrstom sa film\_id = 1 i može čitati i pisati podatke:

```
START TRANSACTION;
```

```
SELECT * FROM film WHERE film_id = 1 FOR UPDATE;
```

**Sesija 2** pokušava dobiti exclusive lock nad vrstom sa film\_id = 1, ali čeka zbog sesije 1:

```
START TRANSACTION;
```

```
SELECT * FROM film WHERE film_id = 1 FOR UPDATE;      Waiting...
```

**Sesija 3** ne može izvršiti UPDATE operaciju jer sesija 1 drži exclusive lock nad vrstom:

```
START TRANSACTION;
```

```
UPDATE film SET description = 'novi opis filma' WHERE film_id = 1;
```

*Not working...*

## 4.SET TRANSACTION naredba

Naredba SET TRANSACTION se koristi u MySQL-u za podešavanje karakteristika transakcije. Ova naredba omogućava podešavanje izolacionog nivoa transakcije, automatskog potvrđivanja transakcije i drugih opcija koje utiču na ponašanje transakcija u bazi podataka. Sintaksa je sledeća [10]:

**SET [scope] TRANSACTION transaction\_characteristic, [, transaction\_characteristic...];**

Gde *transaction\_characteristics* predstavlja opcione karakteristike transakcije koje želimo da postavimo. Ove karakteristike se zadaju kao niz ključnih reči sa odgovarajućim vrednostima i neke od njih su:

- **ISOLATION LEVEL** – služi za predstavljanje nivoa izolacije transakcije, koji određuje kako će se transakcije ponašati u odnosu na zaključavanje podataka. Sintaksa je sledeća:

**ISOLATION LEVEL *level***

Moguće vrednosti za *level* su:

- READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- **Access mode** – READ WRITE ili READ ONLY – određuje da li transakcija ima pravo na čitanje i pisanje (READ WRITE) ili samo čitanje podataka (READ ONLY).
  - **AUTOCOMMIT = 1** ili **AUTOCOMMIT = 0** – uključivanje ili isključivanje automatskog potvrđivanja transakcije. Kada je AUTOCOMMIT = 1 (podrazumevana vrednost), svaka SQL naredba se automatski smatra zasebnom transakcijom i potvrđuje se odmah nakon izvršenja. Ako se postavi AUTOCOMMIT = 0, naredbe se grupišu u jednu transakciju i potrebno je eksplicitno potvrđivanje ili poništavanje.
  - **Scope** (opseg) – specificiranje opsega transakcije pomoću ključnih reči:
    - **SESSION** – označava da će se karakteristike transakcije biti primenjene samo na trenutnu sesiju. Postavljene karakteristike transakcije važiće samo za tekuću sesiju, a druge sesije neće biti pogođene. Kada se sesija završi, karakteristike transakcije će biti resetovane na podrazumevane vrednosti. Primer:  
SET **SESSION** TRANSACTION ISOLATION LEVEL READ COMMITTED;  
SET **SESSION** TRANSACTION READ WRITE;
    - **GLOBAL** – karakteristike transakcije će biti primenjene na sve sesije u MySQL serveru. To znači da će sve nove transakcije u svim sesijama imati karakteristike postavljene ovom naredbom. Međutim, već pokrenute transakcije neće biti pogođene. Primer:  
SET **GLOBAL** TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
SET **GLOBAL** TRANSACTION READ ONLY;

## 5. Zaključak

U MySQL-u, transakcije predstavljaju grupu radnji koje se izvršavaju kao jedna celina, čime se osigurava doslednost podataka. Da bi se označio početak transakcije koristi se naredba `START TRANSACTION`. Sve što se desi u okviru neke transakcije moguće je kontrolisati `COMMIT` i `ROLLBACK` naredbama. Dakle, ako želimo da potvrdimo izmene nekih naredbi izvršenih u transakciji, koristi se naredba `COMMIT`, dok ukoliko ne želimo da se izmene napravljene u transakciji trajno primene koristimo `ROLLBACK` naredbu koja je neka vrsta vraćanja unazad. Za podelu transakcije u manje celine koristimo `SAVEPOINT`-e i nad njima je takođe moguće vršiti potvrđivanje ili vraćanje unazad.

Izolacija transakcija odnosi se na nivo izolacije podataka koji se primenjuje tokom izvršenja transakcija i videli smo da MySQL pruža četiri nivoa izolacije: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` i `SERIALIZABLE`. Svaki nivo izolacije ima svoje karakteristike i utiče na vidljivost podataka i konkurentnost transakcija.

Zaključavanje je mehanizam koji se koristi u MySQL-u kako bi se regulisao pristup podacima i sprečilo sukobljavanje transakcija. MySQL podržava različite nivoe zaključavanja, kao što su `table locking`, `row locking` i `page locking`. Pravilno upravljanje zaključavanjem je od suštinske važnosti kako bi se izbegle anomalije i osigurala doslednost podataka.

Za podešavanje svih karakteristika neke transakcije koristi se naredba `SET TRANSACTION`.

## 6. Literatura

- [1] Kako koristiti MySQL Transakcije, <https://www.freecodecamp.org/news/how-to-use-mysql-transactions/> (10. Februar 2023)
- [2] MySQL Transakcije, <https://www.javatpoint.com/mysql-transaction>
- [3] Razumevanje MySQL Transakcija i Izolacije, <https://medium.com/analytics-vidhya/understanding-mysql-transaction-isolation-levels-by-example-1d56fce66b3d> (18.jun 2021)
- [4] Nivoi izolacije u MySQL-u, <https://buildatscale.tech/transaction-isolation-level-in-innodb/> (11.jul 2021)
- [5] Shared i Exclusive Locking u MySQL-u, <http://geekdirt.com/blog/shared-and-exclusive-locks/> (17.avgust 2015)
- [6] Razumevanje granularnosti zaključavanja u MySQL-u, <https://severalnines.com/blog/understanding-lock-granularity-mysql/> (13.april 2021)
- [7] MySQL dokumentacija, START TRANSACTION, COMMIT, ROLLBACK, <https://dev.mysql.com/doc/refman/8.0/en/commit.html>
- [8] MySQL dokumentacija, SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT, <https://dev.mysql.com/doc/refman/8.0/en/savepoint.html>
- [9] MySQL dokumentacija, LOCK TABLES i UNLOCK TABLES naredbe, <https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html>
- [10] MySQL dokumentacija, SET TRANSACTION naredba, <https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html>