Transakcije in izolacija v MariaDB

Povezava tup · podatkovna baza sandbox · primeri SQL, naloge in rešitve

> Okolje: Za povezavo uporabite že pripravljeno povezavo tup (geslo je shranjeno). Vse poskuse izvajajte v bazi sandbox. Zaradi skupne baze ima vsak študent svoje tabele z enolično oznako. Kjer je v primerih 1234, številke zamenjajte z zadnjimi štirimi številkami vpisne številke.

V naslednjih primerih se uporabljajo ukazi in pravila za MariaDB. Podrobnosti se lahko razlikujejo od PostgreSQL, MS SQL Server ali SQLite. Poglavje se konča pred protokolom 2PL.

1. Povezava in preverjanje okolja

V SQL-odjemalcu odprite shranjeno povezavo z imenom tup in izvedite:

```
USE sandbox;
SELECT DATABASE() AS baza, CONNECTION_ID() AS seja;
SELECT @@autocommit AS autocommit;
SHOW VARIABLES LIKE '%isolation%';
```

Za poskuse z izolacijo potrebujete dve neodvisni povezavi v isti podatkovni bazi. Dve okni ali zavihka, ki delita isto strežniško sejo, nista dovolj. Različna rezultata funkcije `CONNECTION_ID()` pomenita različni strežniški seji.

2. Transakcija in ACID

Transakcija je logična enota dela, sestavljena iz ene ali več operacij. S COMMIT spremembe potrdimo, z ROLLBACK pa razveljavimo še nepotrjene spremembe, kadar uporabljena tabela podpira transakcije.

Lastnost	Pomen
Atomicity – atomarnost	Vse spremembe transakcije ali nobena.
Consistency – konsistentnost	Ohranijo se podatkovne omejitve in poslovna pravila.
Isolation – izolacija	Sočasne transakcije ne smejo nedovoljeno vplivati druga na drugo.
Durability – trajnost	Potrjene spremembe ostanejo trajno shranjene.

Primer: Ana ima 100 EUR, Bor 50 EUR. Prenos 20 EUR mora dati stanji 80 in 70 EUR. Če drugi korak ne uspe, mora transakcija ostati brez učinka: 100 in 50 EUR.

3. Vsaka tabela ima svoj podatkovni pogon

MariaDB uporablja podatkovne pogone (storage engines). Posamezne tabele iste baze lahko uporabljajo različne pogone. Iz običajnega ukaza SELECT ali UPDATE ne ugotovimo, ali tabela podpira transakcije; to moramo preveriti. Isti ROLLBACK lahko vpliva na tabelo InnoDB, na tabelo MyISAM pa ne.

3.1 Pregled glavnih pogonov

Pogon	Transakcije	Opomba
InnoDB	DA	Privzeti in priporočeni pogon za večstavčne ACID-transakcije.
MyRocks / RocksDB	DA (če je nameščen)	Transakcijski pogon; podpira le nekatere izolacijske stopnje.
XtraDB	zgodovinsko DA	Starejši pogon, v novejšem MariaDB ni na voljo.
MyISAM	NE	Sprememb ne razveljavi ROLLBACK.
Aria	NE	TRANSACTIONAL=1 pomeni zaščito ob sesutju stavka, ne polnega ACID-ROLLBACK.
MEMORY	NE	Podatki so v pomnilniku, brez transakcijskega ROLLBACK.
CSV, ARCHIVE, MERGE, BLACKHOLE	NE	Specializirani netransakcijski pogoni; lahko niso nameščeni.

Nekateri posebni pogoni (npr. ColumnStore, Spider, FederatedX) imajo različne omejitve glede izvedbe in različice. Najzanesljivejši podatek za konkretni strežnik je stolpec `TRANSACTIONS` v `information_schema.ENGINES`.

3.2 Kako preverimo pogon

```
SHOW ENGINES;
SELECT ENGINE, SUPPORT, TRANSACTIONS, SAVEPOINTS
FROM information_schema.ENGINES
ORDER BY ENGINE;

SELECT TABLE_NAME, ENGINE
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'sandbox'
  AND TABLE_NAME IN ('tx_1234_racun', 'tx_1234_narocilo');
```

Pogon določimo ob ustvarjanju posamezne tabele:

```
CREATE TABLE tx_1234_racun (
  id INT PRIMARY KEY,
  ime VARCHAR(30) NOT NULL,
  stanje DECIMAL(10,2) NOT NULL
) ENGINE=InnoDB;
```

> Pomembno: Če je znotraj transakcije posodobljenih več tabel, lahko del sprememb ostane, če katera od njih ne podpira transakcij. ROLLBACK ni univerzalno zagotovilo za vsako tabelo v MariaDB.

4. AUTOCOMMIT: TRUE/FALSE

V MariaDB je `autocommit=1` praviloma privzeta nastavitev. Posamezen stavek DML se takoj potrdi, razen kadar je del izrecno začete transakcije. Nastavitev velja za trenutno povezavo.

```
SELECT @@autocommit;              -- 1 = ON, 0 = OFF
SET SESSION autocommit = 0;      -- OFF / FALSE
SELECT @@autocommit;
UPDATE tx_1234_racun SET stanje = 110 WHERE id = 1;
ROLLBACK;                        -- vrne prejšnjo vrednost (InnoDB)
SET SESSION autocommit = 1;      -- ON / TRUE
```

> Prehod `SET autocommit = 0` → `SET autocommit = 1` lahko implicitno potrdi že odprto transakcijo. Za jasne poskuse raje uporabite `START TRANSACTION` / `COMMIT` ali `ROLLBACK`.

4.1 Kratek samostojen poskus

```
SET SESSION autocommit = 1;
START TRANSACTION;
UPDATE tx_1234_racun SET stanje = stanje - 20 WHERE id=1;
UPDATE tx_1234_racun SET stanje = stanje + 20 WHERE id=2;
ROLLBACK;                        -- obe spremembi izginejo
```

Način, kot je nastavljen v aplikaciji, ni nujno enak privzeti nastavitvi strežnika. Pri Pythonovem pyodbc obstaja še lastnost `connection.autocommit`; uporabo SQL-ukazov `START TRANSACTION` je najlažje demonstrirati ob ustrezno izbranem načinu povezave.

5. Priprava testnih tabel v sandbox

V naslednji skripti je oznaka 1234 samo primer. Pred izvajanjem jo v celotnem dokumentu zamenjajte s svojo oznako. Skripta ustvari in pobriše le tabeli z vašo oznako. Ukazi `CREATE TABLE`, `DROP TABLE`, `TRUNCATE` in nekateri drugi ukazi DDL v MariaDB sprožijo implicitni COMMIT, zato jih izvajamo pred transakcijskimi eksperimenti.

```
USE sandbox;
DROP TABLE IF EXISTS tx_1234_narocilo;
DROP TABLE IF EXISTS tx_1234_racun;
CREATE TABLE tx_1234_racun (
  id INT PRIMARY KEY, ime VARCHAR(30),
  stanje DECIMAL(10,2) NOT NULL
) ENGINE=InnoDB;
CREATE TABLE tx_1234_narocilo (
  id INT PRIMARY KEY,
  znesek DECIMAL(10,2) NOT NULL
) ENGINE=InnoDB;
INSERT INTO tx_1234_racun VALUES (1,'Ana',100),(2,'Bor',50);
INSERT INTO tx_1234_narocilo VALUES (1,100),(2,120);
SELECT * FROM tx_1234_racun;
SELECT * FROM tx_1234_narocilo;
```

Za obnovitev začetnega stanja pred posameznim poskusom najprej zaključite odprte transakcije v obeh sejah in nato izvedite:

```
SET SESSION autocommit = 1;
UPDATE tx_1234_racun SET stanje=100 WHERE id=1;
UPDATE tx_1234_racun SET stanje=50 WHERE id=2;
DELETE FROM tx_1234_narocilo WHERE id=3;
```

6. Anomalije pri sočasnem dostopu

6.1 Izgubljena sprememba (lost update)

T1 in T2 prebereta stanje 100 EUR. T2 prišteje 100 in shrani 200; T1 pa na osnovi prej prebrane vrednosti odšteje 10 in shrani 90. Pravilno končno stanje bi bilo 190. Gre za napako pri ločenem postopku branje–računanje–pisanje, ne za posledico enega atomarnega stavka `UPDATE stanje = stanje - 10`.

6.2 Branje nepotrjenih podatkov (dirty read)

T1 spremeni stanje 100 na 200, vendar ne potrdi spremembe. T2 prebere 200; T1 nato razveljavi spremembo. T2 je uporabila podatek, ki ni bil nikoli potrjen.

6.3 Neponovljivo branje (non-repeatable read)

T1 dvakrat prebere isto vrstico. Vmes T2 to vrstico spremeni in potrdi. T1 dobi dve različni vrednosti iste vrstice.

6.4 Fantomsko branje (phantom read)

T1 dvakrat izvede poizvedbo po množici vrstic (npr. naročila nad 100 EUR). Vmes T2 doda ustrezno vrstico in potrdi, zato T1 ob drugem branju dobi drugačno število vrstic.

7. Stopnje izolacije

Stopnja izolacije določa, katere nepravilnosti so dovoljene. To je učna primerjava po SQL-standardu; dejanski rezultati so odvisni od pogona in vrste branja.

Stopnja	Dirty read	Non-repeatable	Phantom
READ UNCOMMITTED	mogoče	mogoče	mogoče
READ COMMITTED	ne	mogoče	mogoče
REPEATABLE READ	ne	ne	standard dopušča
SERIALIZABLE	ne	ne	ne

Pri InnoDB v MariaDB navadni SELECT na ravni REPEATABLE READ običajno bere konsistenten posnetek, zato tudi navidezno fantomska nova vrstica pri ponovljenem navadnem SELECT ne postane vidna. Zaklepno branje in UPDATE lahko delujeta drugače.

```
-- Nastavitev za naslednje transakcije v trenutni seji:
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT stanje FROM tx_1234_racun WHERE id=1;
COMMIT;

-- Preverjanje, odvisno od različice strežnika:
SHOW VARIABLES LIKE '%isolation%';
```

8. Dve povezavi: konkretni poskusi

V istem odjemalcu uporabite dve ločeni strežniški seji A in B (preverite `CONNECTION_ID()`). V obeh najprej izvedite `USE sandbox;`. Kadar je treba ukaz izvesti pozneje, je posebej označeno.

8.1 READ COMMITTED: neponovljivo branje

Seja A (začetek):

```
SET SESSION autocommit = 1;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT stanje FROM tx_1234_racun WHERE id=1; -- 100
```

Seja B:

```
SET SESSION autocommit = 1;
UPDATE tx_1234_racun SET stanje=120 WHERE id=1;
```

Seja A (nadaljevanje):

```
SELECT stanje FROM tx_1234_racun WHERE id=1; -- 120
COMMIT;
```

8.2 REPEATABLE READ: ponovljivo branje

Po ponastavitvi stanja na 100 na strani B:

```
-- A:
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT stanje FROM tx_1234_racun WHERE id=1; -- 100
-- B medtem izvede UPDATE na 120 in ga potrdi
SELECT stanje FROM tx_1234_racun WHERE id=1; -- 100
COMMIT;
START TRANSACTION;
SELECT stanje FROM tx_1234_racun WHERE id=1; -- 120
COMMIT;
```

8.3 READ COMMITTED: fantomska vrstica

```
-- A:
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT COUNT(*) FROM tx_1234_narocilo WHERE znesek>=100; -- 2
-- B vmes izvede: INSERT INTO tx_1234_narocilo VALUES (3,160);
SELECT COUNT(*) FROM tx_1234_narocilo WHERE znesek>=100; -- 3
COMMIT;
```

Po poskusu naj seja B izbriše vrstico `id=3`, ko je seja A že zaključena.

9. Razlika: InnoDB proti MyISAM

Ta poskus izvedite samo, če rezultat `SHOW ENGINES` potrdi, da je na strežniku na voljo MyISAM in če imate pravico ustvariti tabelo. Ustvarite lastni tabeli z enoličnim imenom.

```
CREATE TABLE tx_1234_innodb (id INT PRIMARY KEY, x INT)
  ENGINE=InnoDB;
CREATE TABLE tx_1234_myisam (id INT PRIMARY KEY, x INT)
  ENGINE=MyISAM;
INSERT INTO tx_1234_innodb VALUES (1,10);
INSERT INTO tx_1234_myisam VALUES (1,10);
START TRANSACTION;
UPDATE tx_1234_innodb SET x=15 WHERE id=1;
UPDATE tx_1234_myisam SET x=15 WHERE id=1;
ROLLBACK;
SELECT x FROM tx_1234_innodb; -- 10
SELECT x FROM tx_1234_myisam; -- 15
```

> Učinek je odvisen od pogona: ROLLBACK vrne InnoDB na 10, MyISAM pa ostane na 15. Na takšen mešan način ni varno izvajati denarnih prenosov. Ob netransakcijskih spremembah MariaDB lahko izpiše opozorilo.

10. Naloge – transakcije

T1 Ana ima 100 EUR, Bor 50 EUR. Prenesi 30 EUR. Določi novi stanji, skupno vsoto in pojasni, kaj se mora zgoditi ob napaki pri drugem UPDATE.

T2 Napiši dve različici SQL-prenosa 30 EUR iz računa 1 na račun 2: s COMMIT in z ROLLBACK. Vsakič začni iz začetnega stanja.

T3 T1 odšteje 10 EUR in T2 prišteje 100 EUR, obe izhajata iz prebranih 100 EUR. T2 shrani 200, T1 nato 90. Kolikšna je napaka? Predlagaj en sam varen UPDATE za odštevanje.

T4 Poimenuj anomalije: (a) branje pozneje razveljavljenega zapisa, (b) druga vrednost iste vrstice, (c) drugačno število vrstic v ponovljeni poizvedbi.

T5 Z dvema sejama ponovi primer READ COMMITTED pri računu 1. Zapiši obe vrednosti in ime anomalije.

T6 Ponovi T5 z REPEATABLE READ. Zapiši vrednost ob obeh branjih ter po novem START TRANSACTION.

T7 Pri READ COMMITTED preveri število naročil z zneskom vsaj 100 EUR pred dodatnim INSERT in po njem. Zakaj se število spremeni?

T8 Katera je najnižja stopnja izolacije po učni shemi SQL, ki odpravlja (a) dirty read, (b) non-repeatable read in (c) phantom read?

T9 Popravi zaporedna UPDATE, ki preneseta 15 EUR brez transakcije. Zakaj vmesni CREATE TABLE ni primeren?

T10 Z ukazi SQL nastavi autocommit na 0, spremeni stanje, razveljavi spremembo in ga vrni na 1. Zakaj je nevarno preklopiti z 0 na 1 pred ROLLBACK?

T11 Kako ugotoviš pogon in podporo transakcijam za dve svoji tabeli? Napovej rezultat ROLLBACK pri InnoDB in pri MyISAM.

11. Kratka ponovitev nerelacijskih baz

N1 – MongoDB V sample_mflix.movies najdi filme, ki vsebujejo hkrati žanra Comedy in Drama, nato filme, ki vsebujejo enega ali drugega. Razloži razliko.

N2 – MongoDB Z agregacijo preštej število filmov po letu izida. Upoštevaj tip movie in številsko polje year; prikaži prvih pet let z največ filmi.

N3 – Neo4j in SQL V grafu Jadralec–Čoln–Rezervacija izpiši ime čolna in število rezervacij, padajoče. Zapiši enakovreden SQL JOIN/GROUP BY.

12. Rešitve – transakcije

T1. Ana = 70 EUR, Bor = 80 EUR, vsota 150 EUR. Ob neuspehu drugega UPDATE mora ROLLBACK obnoviti 100 EUR in 50 EUR; to zahteva transakcijski pogon.

T2. Pred posameznim poskusom obnovimo začetne vrednosti. Po COMMIT dobimo 70/80, po ROLLBACK 100/50.

```
START TRANSACTION;
UPDATE tx_1234_racun SET stanje=stanje-30 WHERE id=1;
UPDATE tx_1234_racun SET stanje=stanje+30 WHERE id=2;
COMMIT;          -- pri InnoDB: 70, 80
-- Obnovi začetno stanje, nato ponovi s ROLLBACK.
```

T3. Pravilno je 190 EUR, napačno 90 EUR: izgubljenih je 100 EUR. Operacijo lahko izvedemo z neposrednim atomarnim UPDATE:

```
UPDATE tx_1234_racun SET stanje=stanje-10 WHERE id=1;
```

T4. (a) Dirty read, (b) non-repeatable read, (c) phantom read.

T5. V seji A najprej 100, po COMMIT seje B še v isti transakciji 120: non-repeatable read pri READ COMMITTED.

T6. V seji A obakrat 100; po COMMIT in začetku nove transakcije 120. Pri navadnih SELECT v InnoDB REPEATABLE READ uporabljamo konsistenten posnetek.

T7. Najprej 2, nato 3 ustrezna naročila: phantom read. Vmesna transakcija je uspešno potrdila nov INSERT.

T8. (a) READ COMMITTED; (b) REPEATABLE READ; (c) SERIALIZABLE – po učni shemi standarda. Pri navadnem branju v InnoDB lahko REPEATABLE READ prepreči tudi opisani fantom na istem posnetku.

T9. Uporabimo START TRANSACTION, oba UPDATE in COMMIT; ob napaki ROLLBACK. CREATE TABLE lahko sproži implicitni COMMIT pred koncem prenosa.

```
START TRANSACTION;
UPDATE tx_1234_racun SET stanje=stanje-15 WHERE id=1;
UPDATE tx_1234_racun SET stanje=stanje+15 WHERE id=2;
COMMIT;  -- ob napaki namesto tega: ROLLBACK;
```

T10. Na začetku `SELECT @@autocommit`; nato `SET SESSION autocommit=0`, UPDATE, ROLLBACK in `SET SESSION autocommit=1`. Prehod 0→1 potrdi odprte spremembe, če so še v teku.

T11. Za pogon posamezne tabele preverimo `information_schema.TABLES.ENGINE`; za sposobnosti pogona `information_schema.ENGINES.TRANSACTIONS`. ROLLBACK povrne InnoDB; sprememb MyISAM ne povrne.

```
SELECT TABLE_NAME, ENGINE FROM information_schema.TABLES
WHERE TABLE_SCHEMA='sandbox' AND
 TABLE_NAME IN ('tx_1234_racun','tx_1234_narocilo');
SELECT ENGINE, TRANSACTIONS FROM information_schema.ENGINES;
```

13. Rešitve – nerelacijske baze

N1. `$all` zahteva oba žanra, `$in` pa vsaj enega.

```
db.movies.find({genres: {$all: ['Comedy', 'Drama']}});
db.movies.find({genres: {$in: ['Comedy', 'Drama']}});
```

N2. `$match` filtrira, `$group` združuje po letu, `$sort` ureja, `$limit` omeji.

```
db.movies.aggregate([
 {$match: {type:'movie', year:{$type:'number'}}},
 {$group: {_id:'$year', st:{$sum:1}}},
 {$sort: {st:-1, _id:1}},
 {$limit:5}
]);
```

N3. Grafna povezava REZERVIRAL predstavlja eno rezervacijo; v SQL jo predstavlja vrstica v tabeli Rezervacija. Prikazani sta samo entiteti z vsaj eno rezervacijo.

```
// Cypher
MATCH (:Jadralec)-[r:REZERVIRAL]->(c:Coln)
RETURN c.ime AS coln, COUNT(r) AS st
ORDER BY st DESC, coln;

-- SQL
SELECT c.ime AS coln, COUNT(*) AS st
FROM Coln c JOIN Rezervacija r ON r.cid=c.cid
GROUP BY c.cid, c.ime ORDER BY st DESC, coln;
```

14. Povezava s Pythonom in Jupyterjem

Če je tup registriran kot ODBC DSN, se Jupyter poveže z gonilnikom `pyodbc`. Gesla ni treba zapisati v zvezek, če ga zna uporabljena ODBC povezava pridobiti iz shranjene konfiguracije. V nekaterih odjemalcih pa je »tup« le shranjena povezava, ne sistemski DSN; v tem primeru Python potrebuje ustrezno nastavljeno ODBC povezavo.

```
# V Jupyter okolju: %pip install pyodbc
import pyodbc
cn = pyodbc.connect('DSN=tup;DATABASE=sandbox;',
                    autocommit=True, timeout=8)
print(cn.cursor().execute(
    'SELECT DATABASE(), CONNECTION_ID()').fetchone())
cn.close()
```

Za spreminjanje Pythonove samodejne potrditve se uporabi `cn.autocommit = False` ali `cn.autocommit = True`. Pri `False` se potrdi z `cn.commit()`, razveljavi z `cn.rollback()`. Za demonstracije z `START TRANSACTION` v priloženem zvezku je uporabljena možnost `autocommit=True`, krmiljenje pa poteka eksplicitno s SQL.

15. Dokumentacija

MariaDB: Storage Engines

MariaDB: INFORMATION_SCHEMA.ENGINES

MariaDB: Aria (crash safety ni ACID)

MariaDB: START TRANSACTION in AUTOCOMMIT

MariaDB: Implicitni COMMIT

MariaDB: MyRocks izolacija

Podlaga za transakcije, ACID, anomalije in izolacijo: predavanja TUP 2026/27, »Transakcijska varnost«, poglavje do uvoda v 2PL. Informacije o MariaDB, pogonih, implicitnih potrditvah in ODBC primeru so dopolnitve iz dokumentacije MariaDB.