

Predavanja

Svibanj, 2021.



Upravljanje istodobnim pristupom (*Concurrency Control*)

- višekorisnički (*multiuser*) SUBP
 - ispravni i maksimalno dostupni podaci u uvjetima istodobnog pristupa velikog broja korisnika
 - komponenta SUBP-a zadužena za upravljanje istodobnoim pristupom

Jednostavan, ali neefikasan način korištenja višekorisničkog SUBP-a:

- korisnici obavljaju transakcije jednu za drugom (serijsko izvršavanje transakcija) - sustav počinje izvršavati sljedeću transakciju tek kad je u potpunosti završio prethodnu
 - slaba ukupna iskoristivost sustava (npr. CPU čeka završetak U/I operacije)
 - korisnik koji pokreće relativno kratku transakciju može (nepredvidivo) dugo čekati na završetak neke relativno duge transakcije

Zbog čega je istodobni pristup nužan?

- budući da transakcije obuhvaćaju U/I i CPU operacije, njihovo istodobno obavljanje omogućilo bi istodobno korištenje različitih resursa računala
 - uvećava se broj transakcija obavljen u jedinici vremena (*throughput*), čime se uvećava ukupna iskoristivost sustava (*utilization*)
 - prosječno vrijeme koje protekne između aktiviranja i završetka transakcije (*average response time*) se smanjuje
- današnji sustavi su (većinom) višekorisnički:
 - potrebno je omogućiti istodobno (ili prividno istodobno) izvršavanje transakcija

Istodobni pristup i transakcija

Upravljanje istodobnim pristupom (kao i postupak obnove baze podataka) usko su povezani s pojmom transakcije.

Transakcija je niz logički povezanih operacija koje se izvršavaju kao cjelina i prevode bazu podataka iz jednog u drugo **konzistentno stanje**.

Rezultat transakcije ne smije ovisiti o tome odvijaju li se istodobno i neke druge transakcije!

ACID svojstva transakcije:

- atomarnost, konzistentnost i izdržljivost
 - nisu ugroženi istodobnim pristupom
- izolacija - kada se istodobno obavljaju dvije ili više transakcija, njihov učinak mora biti jednak kao da su se obavljale jedna iza druge
 - problem kada više korisnika pristupa **istom** podatku/podacima njihove se aktivnosti (čitanje i/ili pisanje) **isprepliću**

Kako osigurati svojstvo izolacije?

Primjer: Istodobni pristup i transakcija

- dva **objekta** u bazi podataka ($x = 100$, $y = 100$)
- integritetsko ograničenje: $x = y$
- operacije čitanja ili pisanja (*database operations*)
 - **pročitaj** (x, p) u varijablu p učitaj vrijednost objekta x
 - **zapiši** (y, p) u objekt y upiši vrijednost varijable p

T₁

pročitaj (x, p)
 $p \leftarrow p + 100$
zapiši (x, p)
pročitaj (y, p)
 $p \leftarrow p + 100$
zapiši (y, p)

T₂

pročitaj(x, p)
 $p \leftarrow p * 2$
zapiši(x, p)
pročitaj(y, p)
 $p \leftarrow p * 2$
zapiši(y, p)

CILJ:

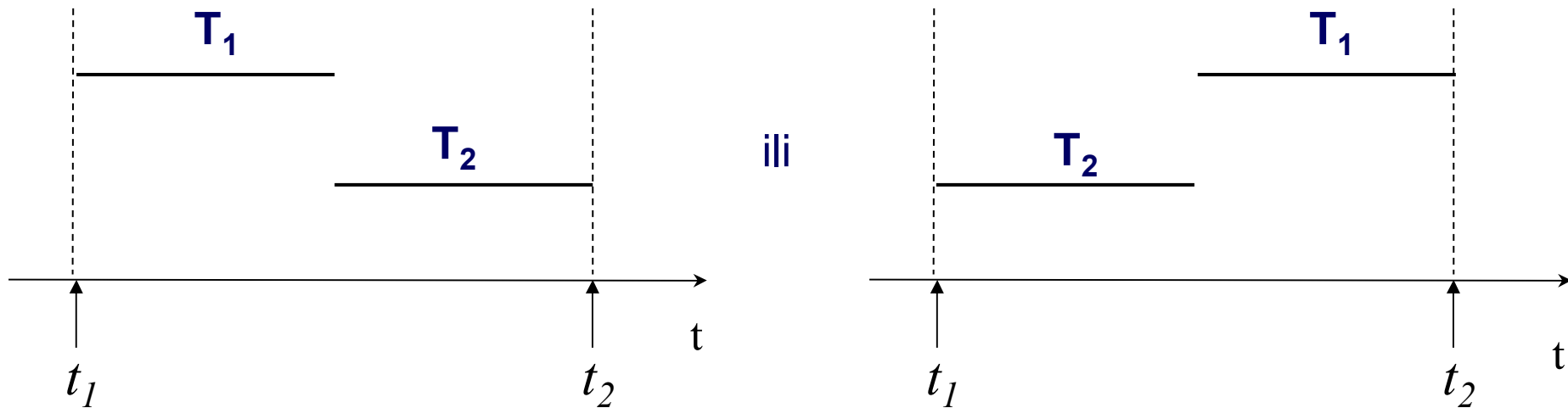
- transakcija prevodi bazu podataka iz jednog konzistentnog u drugo konzistentno stanje

Jednostavno „rješenje”:

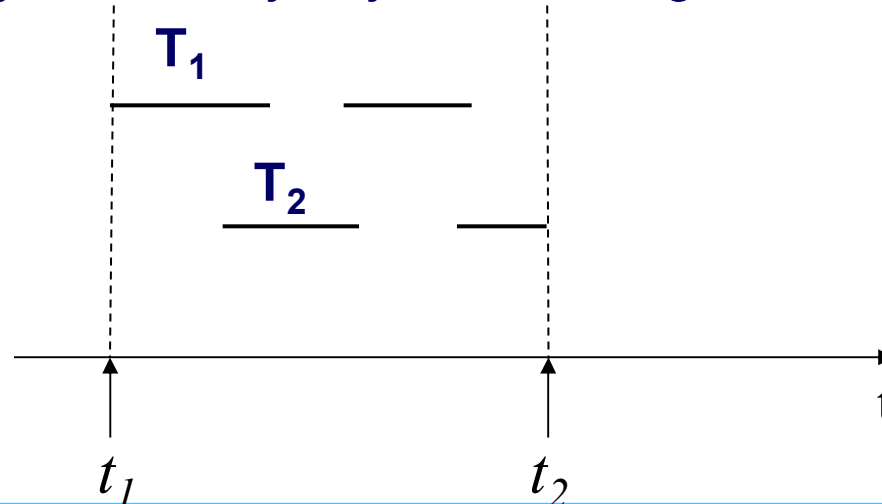
- Ako se transakcije izvršavaju međusobno izolirano (to znači jedna iza druge, serijski), neće narušiti konzistentnost baze podataka

Istodobno izvršavanje transakcija

serijsko izvršavanje transakcija



istodobno izvršavanje transakcija – jedan od mogućih redoslijeda

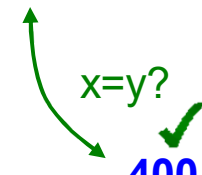


Serijsko izvršavanje transakcija

Primjer:

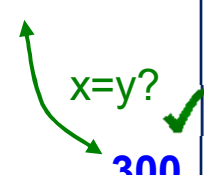
a) redoslijed T_1, T_2

T_1	T_2	x	y
		100	100
pročitaj(x, p)			
$p \leftarrow p + 100$			
zapiši (x, p)		200	
pročitaj(y, p)			
$p \leftarrow p + 100$			
zapiši (y, p)			200
	pročitaj(x, p)		
	$p \leftarrow p * 2$		
	zapiši (x, p)	400	
	pročitaj(y, p)		
	$p \leftarrow p * 2$		
	zapiši (y, p)		400



b) redoslijed T_2, T_1

T_1	T_2	x	y
		100	100
	pročitaj(x, p)		
	$p \leftarrow p * 2$		
	zapiši (x, p)	200	
	pročitaj(y, p)		
	$p \leftarrow p * 2$		
	zapiši (y, p)		200
pročitaj(x, p)			
$p \leftarrow p + 100$			
zapiši (x, p)			
pročitaj(y, p)			
$p \leftarrow p + 100$			
zapiši (y, p)			300



primjer iz [Garcia-Molina]

Istodobno izvršavanje transakcija

c) redoslijed izvršavanja koji narušava konzistentnost baze podataka

T_1	T_2	x	y
pročitaj(x, p)		100	100
$p \leftarrow p + 100$			
zapiši (x, p)		200	
	pročitaj(x, p)		
	$p \leftarrow p * 2$		
	zapiši (x, p)	400	
	pročitaj(y, p)		
	$p \leftarrow p * 2$		
	zapiši (y, p)		200
pročitaj(y, p)			
$p \leftarrow p + 100$			
pročitaj(y, p)			

Istodobno izvršavanje transakcija

d) redoslijed izvršavanja koji ne narušava konzistentnost baze podataka

T ₁	T ₂	x	y
pročitaj(x, p)		100	100
p ← p + 100			
zapiši (x, p)		200	
	pročitaj(x, p)		
	p ← p * 2		
	zapiši (x, p)	400	
pročitaj(y, p)			
p ← p + 100			
zapiši (y, p)			200
	pročitaj(y, p)		
	p ← p * 2		
	zapiši (y, p)		400

Redoslijed izvršavanja nije serijski ali je učinak izvršavanja jednak učinku serijskog izvršavanja.

Svaki takav redoslijed ne narušava konzistentnost baze podataka – za njega se kaže da je **serijalizabilan**.

Karakteristični problemi istodobnog pristupa

Prema SQL standardu neki od karakterističnih problema istodobnog pristupa su:

- P1 – prijava čitanje (*read uncommitted*)
- P2 – neponovljivo čitanje (*nonrepeatable read*)
- P3 – sablasne n-torke (*phantom rows*)
- P4 – izgubljena izmjena (*lost update*)

P1: Prljavo čitanje (*Dirty read*)

Prljavo čitanje je čitanje nepotvrđenih podataka druge transakcije (čije potvrđivanje može ali ne mora biti poništeno (ROLLBACK)).

osoba

sifOsoba	prez	ime
1	Car	Ana
2	Ban	Ivo

T1

```
BEGIN TRANSACTION;  
  
...  
  
UPDATE osoba  
  SET prez='Horvat'  
  WHERE sifOsoba = 1;  
  
...  
  
ROLLBACK TRANSACTION;
```

T2

```
SELECT *  
FROM osoba;
```

sifOsoba	prez	ime
1	Horvat	Ana
2	Ban	Ivo

n-torka koja s prikazanim vrijednostima atributa nikad nije stvarno postojala u bazi podataka

P2: Neponovljivo čitanje

- Ponovnim izvršavanjem **iste** SELECT naredbe (u istoj transakciji) dobije se drugačiji rezultat
 - posljedica drugih potvrđenih transakcija (potvrđenih nakon inicijalnog čitanja prve transakcije) koje su mijenjale (UPDATE) ntorke (ili više njih) zahvaćene SELECT naredbom

T1

```
SELECT brRacun, saldo  
FROM racun
```

```
WHERE brRacun = 2;
```

...

```
-- Ne mijenja saldo za
```

```
-- racun s brojem 2
```

...

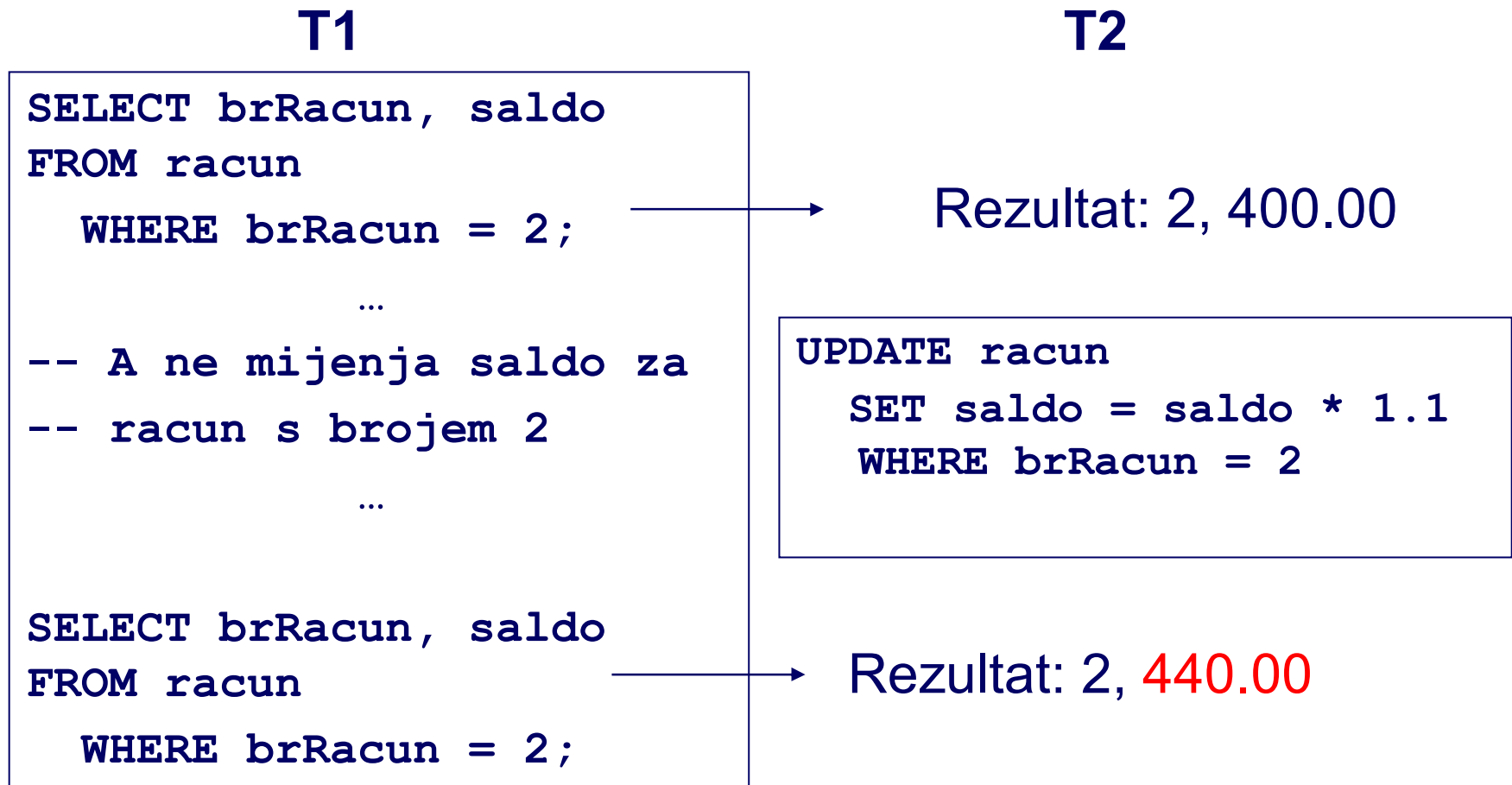
```
SELECT brRacun, saldo  
FROM racun
```

```
WHERE brRacun = 2;
```

Rezultat: 2, 400.00

Rezultat mora (opet) biti:
2, 400.00

Primjer: P2: Neponovljivo čitanje (*Nonrepeatable read*)



- **Ista transakcija** obavljanjem **istog upita** dobije „iste” ntorke, ali s drugačijom vrijednošću atributa.

Primjer: P3: Sablasne n-torke (*Phantom rows*)

- Ponovnim izvršavanjem **iste** SELECT naredbe (u istoj transakciji) dobije se drugačiji rezultat
- posljedica potvrđenih transakcija koje su unosile/mijenjale (INSERT/DELETE) ntorke zahvaćene SELECT naredbom

T1

T2

```
SELECT *  
  FROM racun  
 WHERE saldo > 100  
.  
-- Ne mijenja racun  
.  
SELECT *  
  FROM racun  
 WHERE saldo > 100
```

Rezultat: dvije ntorke

```
INSERT INTO racun  
VALUES (3, 400.00)
```

Rezultat: tri ntorke !!!

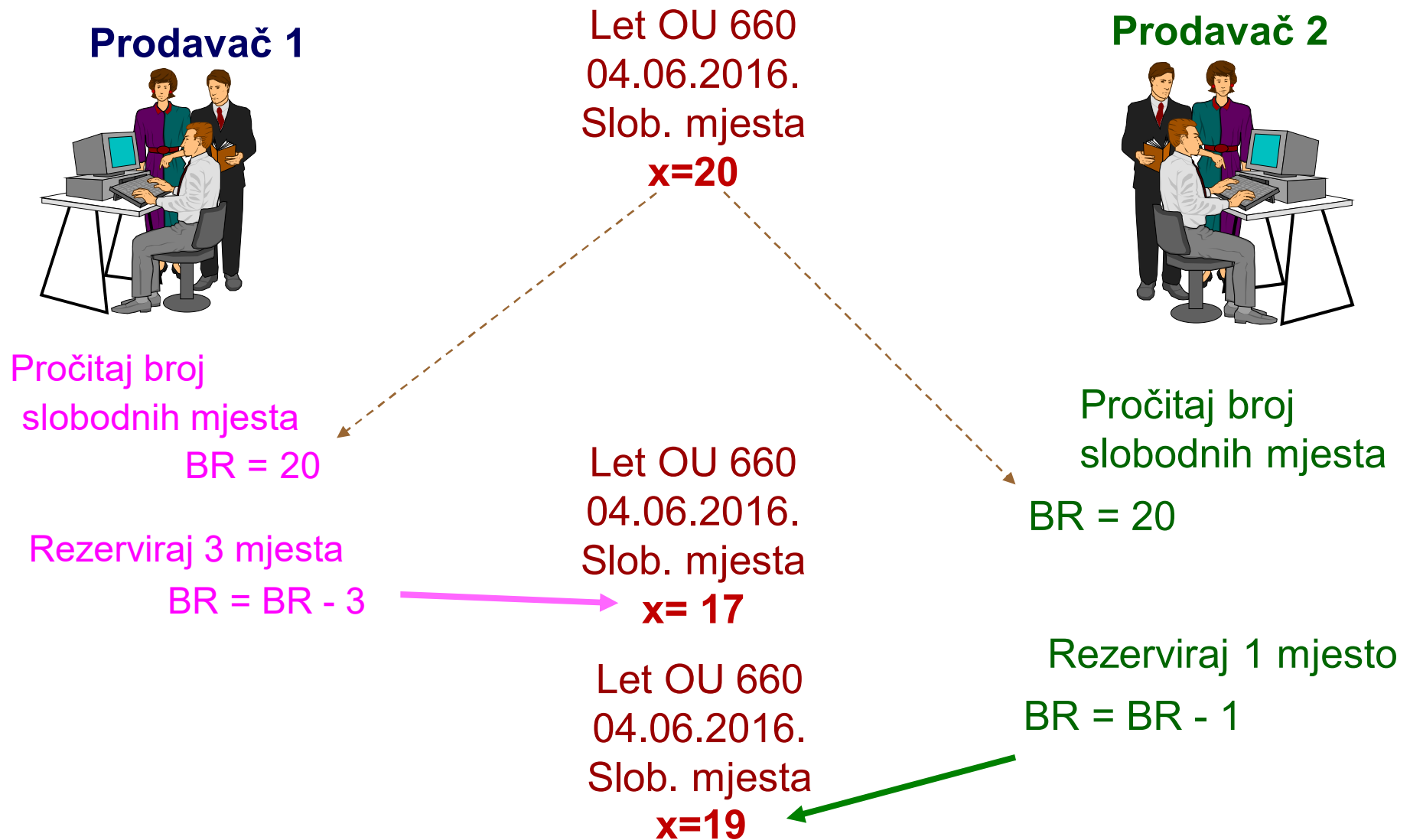
- **Ista transakcija** obavljanjem **istog upita** dobije drugačiji rezultat - zbog toga što je u međuvremenu transakcijom 2 unesena n-torka koja zadovoljava kriterij upita

Komentar: P1, P2, P3

- P2 (neponovljivo čitanje) i P3 (sablasne ntorke) se događaju kada transakcija čita podatke koje je **druga transakcija potvrdila** nakon njenog početka
 - P2 čita **potvrđene izmjene** (UPD)
-> iste ntorke, ali drugačijeg sadržaja
 - P3 čita **potvrđeni unos i/ili brisanja** (INS, DEL)
-> više ili manje ntorki
- P1 (prljavo čitanje) se događa kada transakcija **čita nepotvrđene podatke**, tj. čitanje „*work in progress*” podataka, koji u konačnici mogu i ne moraju biti potvrđeni
 - P1 čita nepotvrđene INS, UPD, DEL

P4: Izgubljena izmjena (*Lost update*)

Primjer: Rezervacija zrakoplovnih karata



Upravljanje istodobnim pristupom u SUBP - rješenja

1. protokol zasnovan na zaključavanju
2. protokol korištenja vremenskih oznaka (TO)
3. protokol zasnovan na validaciji
4. protokol zasnovan na grafovima
5. ...

1. Protokol zasnovan na zaključavanju

- transakcija može zaključati podatak (podatke)
 - sprječava druge transakcije da pristupe podatku dok ga ona ne otključa
- dio SUBP-a (***locking manager***) zaključava zapise i prosuđuje u slučajevima kad postoji više zahtjeva za zaključavanjem istog podatka

Prodavač 1



Zaključavanje

Prodavač 2



Zaključaj x

Pročitaj broj
slobodnih mjesta

BR = 20

Rezerviraj 3 mjesta

BR = BR - 3

Otključaj x

Let OU 660
04.06.2016.
Slob. mjesta
 $x=20$

Let OU 660
04.06.2016.
Slob. mjesta
 $x=17$

Let OU 660
04.06.2016.
Slob. mjesta

$x=16$

Zaključaj x

Pročitaj broj
slobodnih mjesta

BR = 17

Rezerviraj 1 mjesto

BR = BR - 1

Otključaj x

Vrste zaključavanja

- ključ za pisanje/izmjenu - **WRITE LOCK, EXCLUSIVE LOCK**
 - transakcija T_1 zaključa objekt za pisanje
 - niti jedna druga transakcija ga **ne može zaključati** (niti za čitanje niti za pisanje) **dok ga T_1 ne otključa**
 - svaka operacija **izmjene** (SQL naredbe INSERT, UPDATE, DELETE) postavlja ključ za pisanje
- ključ za čitanje - **READ LOCK, SHARED LOCK**
 - transakcija T_1 (SQL naredbom SELECT) zaključa objekt za čitanje
 - bilo koja druga transakcija ga također **može zaključati za čitanje**
 - niti jedna ga transakcija **ne može zaključati za pisanje**

Matrica kompatibilnosti ključeva

Proces 2
pokušava
postaviti na
isti objekt
ključ:

Proces 1 postavio je na objekt ključ:

	READ	WRITE	NO LOCK
READ	✓	✗	✓
WRITE	✗	✗	✓

Serijalizabilnost – je li zaključavanje dovoljno?

T ₁	T ₂	x	y
		100	100
zaključaj (x)			
pročitaj(x, p)			
p ← p + 100			
zapiši (x, p)		200	
otključaj (x)			
	zaključaj (x)		
	pročitaj(x, p)		
	p ← p * 2		
	zapiši (x, p)	400	
	otključaj (x)		
	zaključaj (y)		
	pročitaj(y, p)		
	p ← p * 2		
	zapiši (y, p)		200
	otključaj (y)		
zaključaj (y)			
pročitaj(y, p)			
p ← p + 100			
zapiši (y, p)			
otključaj (y)			

x=y?

x ≠ y

300

Protokol dvofaznog zaključavanja

Two-phase locking protocol (2PL)

Serijalizabilni redoslijed izvršavanja osigurava svojstvo izolacije (I iz ACID-a). Serijalizabilnost redoslijeda izvršavanja je osigurana ako sve transakcije poštuju protokol dvofaznog zaključavanja:

- ❶ prije obavljanja operacije nad objektom (npr. n-torkom iz baze), transakcija mora za taj objekt postaviti ključ
 - ❷ nakon otpuštanja ključa transakcija ne smije više zatražiti nikakav ključ
- transakcije koje poštuju 2PL protokol imaju 2 faze - fazu pribavljanja ključeva (faza rasta - **growing phase**) i fazu otpuštanja ključeva (fazu sužavanja - **shrinking phase**)



Protokol dvofaznog zaključavanja – otpuštanje ključeva

- prema protokolu dvofaznog zaključavanja ključevi se otpuštaju u fazi sužavanja
- faza otpuštanja ključeva najčešće je stiješnjena u jednu operaciju (COMMIT ili ROLLBACK na kraju transakcije)

osoba

sifOsoba	prez	ime
1	Car	Ana
2	Ban	Ivo

```
BEGIN TRANSACTION;  
INSERT INTO osoba  
VALUES (3, 'Jurić', 'Mia');  
...
```

→ Za n-torku se postavlja ključ za pisanje

```
UPDATE osoba  
SET ime = 'Anita'  
WHERE sifOsoba= 1;  
...
```

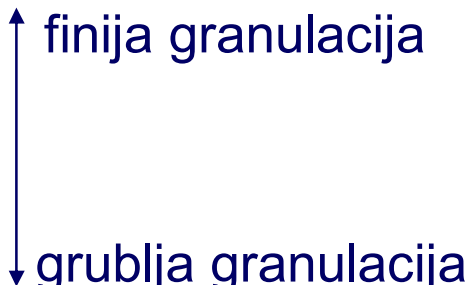
→ Za n-torku se postavlja ključ za pisanje

```
COMMIT TRANSACTION;
```

→ Otpuštaju se ključevi za obje n-torke

Granulacija zaključavanja

Granulacija zaključavanja je određena relativnom veličinom objekta koji će biti zaključan:

- n-torka
 - fizička stranica
 - relacija
 - baza podataka
- 
- finija granulacija
- grublja granulacija

- granulacija zaključavanja utječe na performanse sustava
 - odabirom finije granulacije uvećava se konkurentnost i troškovi postavljanja ključeva
 - odabirom grublje granulacije smanjuje se konkurentnost i troškovi postavljanja ključeva
- koja je granulacija "najbolja"?
 - ovisi o konkretnim operacijama transakcije

Primjer: Granulacija zaključavanja

osoba

sifOsoba	prez	ime
1	Car	Ana
2	Ban	Ivo
...

100 000 n-torki

T_1

```
SELECT ime, prez FROM osoba
WHERE sifOsoba = 1;
```

T_2

```
SELECT ime, prez FROM osoba;
```

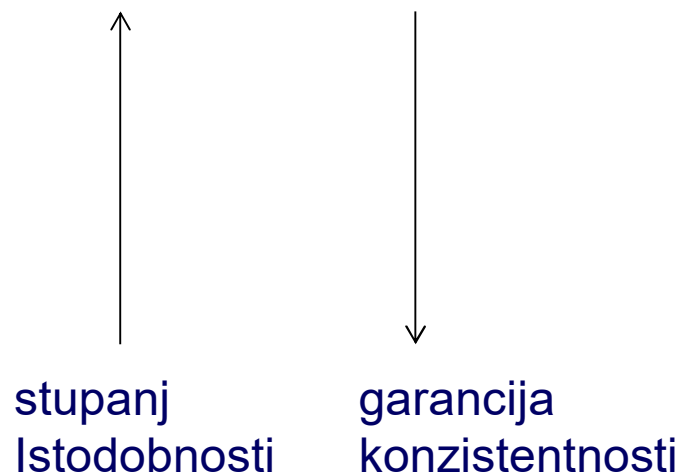
- Ako je granula zaključavanja relacija
⇒ slaba konkurentnost, nepotrebno se ograničava pristup svim n-torkama relacije
- Granula zaključavanja treba biti n-torka!
- Ako je granula zaključavanja n-torka
⇒ loše performance, pojedinačno se postavlja 100 000 ključeva
- Granula zaključavanja treba biti relacija!
- SUBP sustav mora podržavati zaključavanje na više razina granulacije

Razina izolacije - motiv

- serijalizabilnost se osigurava protokolom dvofaznog zaključavanja
 - ključevi se zadržavaju barem dok se ne postave svi transakciji potrebni ključevi
 - što se ključevi dulje zadržavaju povećava se vjerojatnost da će druga transakcija „zatražiti” ključ nad već zaključanim objektom
- za neke primjene serijalizabilnost nije nužna
 - npr. transakcija koja agregira velik broj n-torki će tolerirati nekonzistentnost u zamjenu za poboljšane performanse
- koncept **razina izolacije** je razvijen da bi se omogućilo transakcijama balansiranje između konzistentnosti i istovremenosti
- razine izolacije odnose se **isključivo na čitanje podataka** – kod pisanja/izmjena podaci se zaključavaju (u skladu s granulacijom) i ostaju zaključani do kraja transakcije.

Razina izolacije: SQL-92

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE



- definira se na razini transakcije
- za različite transakcije moguće je definirati različite razine izolacije
- promjenom razine izolacije mijenja se ponašanje transakcije pri postavljanju ključeva za čitanje

Problemi koji se javljaju kod različitih razina izolacije SQL-92

- SQL92 definira razine izolacije s obzirom na ove tri anomalije:

SQL92 standard	Prijava čitanje	Neponovljivo čitanje	Sablasne n-torke
READ UNCOMMITTED	Da	Da	Da
READ COMMITTED	Ne	Da	Da
REPEATABLE READ	Ne	Ne	Da
SERIALIZABLE	Ne	Ne	Ne

Razina izolacije: SQL-92

READ UNCOMMITTED

- podaci se čitaju bez zaključavanja i bez provjere da li su možda zaključani
 - mogu se pojaviti n-torke koje nikada nisu potvrđene u bazi podataka (zapravo nisu ni postojale u bazi podataka)

READ COMMITTED

- čitaju se isključivo potvrđene n-torke
- provjerava se da li je trenutno pročitani podatak zaključan za pisanje

REPEATABLE READ

- osigurava ponovljivo čitanje podataka u okviru transakcije
- podatak se zaključava i ostaje zaključan ključem za čitanje do kraja transakcije
- ne sprječava pojavu sablasnih n-torki

SERIALIZABLE

- čitanjem se podatak zaključava ključem za čitanje i ostaje zaključan do kraja transakcije
- sprječava probleme: prljavo čitanje, neponovljivo čitanje, sablasne n-torke i izgubljena izmjena

Potpuni zastoј (eng. *deadlock*)

- Važan problem kod 2PL protokola je mogućnost nastanka potpunog zastoja

T1

```
BEGIN TRANSACTION;  
SET TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;  
SELECT * FROM osoba  
WHERE sifOsoba = 1;
```

```
UPDATE osoba  
SET prezime = 'Car'  
WHERE sifOsoba = 2;
```

POGREŠKA: zapis zaključan!
(od strane T2)

T2

```
BEGIN TRANSACTION;  
SET TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;  
SELECT * FROM osoba  
WHERE sifOsoba = 2;
```

```
UPDATE osoba  
SET prez = 'Ban'  
WHERE sifOsoba = 1;
```

POGREŠKA: zapis zaključan!
(od strane T1)

- SUBP rješava problem potpunog zastoja
 - a) izbjegavanjem potpunih zastoja; 2PL protokol se dopunjava pravilima koja onemogućuju pojavu potpunih zastoja (npr. zabrana čekanja)
 - b) detekcijom te poništavanjem jedne ili više transakcija

Upravljanje istodobnim pristupom u SUBP - rješenja

- protokol zasnovan na zaključavanju
- protokol korištenja vremenskih oznaka (TO)
- protokol zasnovan na validaciji
- protokol temeljen na grafovima
- ...

Multiverzijski protokol upravljanja istodobnim pristupom - MVCC

- eng. *Multi-version Concurrency Control (MVCC)*
- Zasniva se na protokolu korištenja vremenskih oznaka
- Transakciji se dodjeljuje identifikator (T_{id}) (usporediv s *timestamp* tipom)
- SUBP održava višestruke fizičke verzije jednog logičkog objekta u bazi podataka
- Kada transakcija
 - čita iz objekta, SUBP čita najnoviju verziju zapisa koja je postojala (potvrđena) u trenutku određivanja snimke transakcije
 - piše u objekt, SUBP stvara novu fizičku verziju tog logičkog objekta
- Glavne prednosti:
 - Pisanje ne blokira čitanje
 - Čitanje ne blokira pisanje
 - Transakcije koje samo čitaju, čitaju konzistentne snimke **bez zaključavanja**
 - Pisanje blokira pisanje

MVCC - implementacijski detalji

- Pohrana različitih verzija istog objekta
 - istovremeno postoji više verzija istog objekta koje su međusobno povezane pokazivačima
- Uklanjanje nepotrebnih verzija objekta (eng. *Garbage Collection*)
 - verzije objekata koje niti jedna aktivna transakcija ne može vidjeti potrebno je ukloniti i osloboditi memoriju
 - kako otkriti koje verzije n-torke treba ukloniti?
 - a) zasebni procesi periodički pretražuju i pronalaze verzije koje je moguće ukloniti
 - b) po dovršetku transakcije SUBP donosi odluku o uklanjanju n-torki koje niti jedna aktivna ni buduća transakcija neće moći vidjeti
- Održavanje indeksa
 - indeks koji je posljedica osiguravanja primarnog ključa uvijek pokazuje na jednu verziju objekta
 - ovisno o implementaciji to može biti najmlađa ili najstarija verzija objekta

PostgreSQL: definiranje razine izolacije

```
SET TRANSACTION transaction_mode [, ...]
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode [, ...]
```

where transaction_mode is one of:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ |
                  READ COMMITTED | READ UNCOMMITTED }
```

- READ UNCOMMITTED se ponaša jednako kao READ COMMITTED (ne dozvoljava prljavo čitanje)
- REPEATABLE READ se ponaša jednako kao SERIALIZABLE osim što ne osigurava uvijek serijalizabilno obavljanje transakcija.

PostgreSQL	Prljavo čitanje	Neponovljivo čitanje	Sablasne n-torke
READ UNCOMMITTED	Ne	Da	Da
READ COMMITTED	Ne	Da	Da
REPEATABLE READ	Ne	Ne	Ne
SERIALIZABLE	Ne	Ne	Ne

- Predodređena razina: READ COMMITTED

MVCC u PostgreSQL-u

- Zaglavlje n-torke potrebno za implementaciju MVCC

xmin	xmax	...	korisnički podaci
------	------	-----	-------------------

xmin	identifikator transakcije koja je stvorila objekt
xmax	identifikator transakcije koja je obrisala objekt
...	dodatni meta podatci

Kako radi *xmin* i *xmax*? – unos, brisanje, izmjena

INSERT

- transakcija s id-om 4 stvara novi redak:

xmin	xmax	sifOsoba	prez	...
4	0	1	Car	...

UPDATE (transakcija s id-om 7)

- UPDATE = DELETE + INSERT !!

xmin	xmax	sifOsoba	prez	...
4	7	1	Car	...
7	0	1	Rac	...

DELETE

- transakcija s id-om 12 briše navedeni redak:

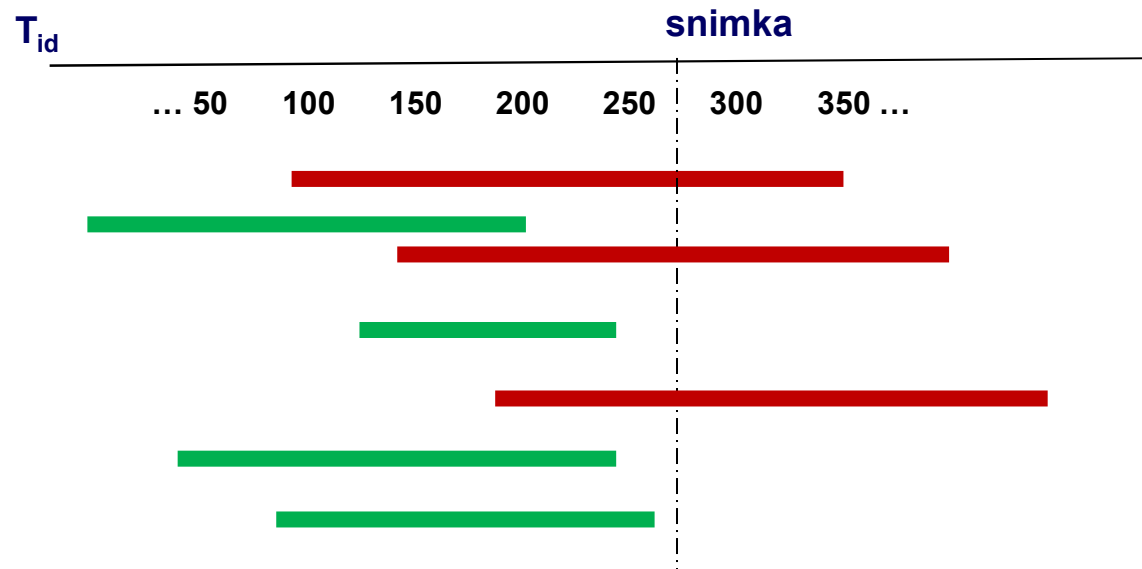
xmin	xmax	sifOsoba	prez	...
4	7	1	Car	...
7	12	1	Rac	...

U međuvremenu je vjerojatno obrisao (*garbage collector*), ili će biti, nije bitno.

PostgreSQL: snimka transakcija (eng. *transaction snapshot*)

- Snimka transakcija određuje koje će transakcije biti vidljive transakcijama koje ju koriste
- Trenutak određivanja snimke ovisi o odabranoj razini izolacije:
 - READ COMMITTED snimka se određuje na početku svake SQL naredbe
 - SERIALIZABLE snimka se određuje na početku transakcije
- Za snimku su potrebne sljedeće informacije:
 - Identifikator najranije još uvijek aktivne transakcije – za sve ranije transakcije smatra se da su potvrđene i njihovi su rezultati vidljivi, ili su poništene
 - Prvi sljedeći identifikator transakcije – koji još nije dodijeljen – svi identifikatori veći ili jednaki tom broju označavaju transakcije koje u trenutku izrade snimke nisu započele i samim time neće biti vidljive.
 - Identifikatori aktivnih transakcija (koje se trenutno obavljaju)

PostgreSQL: snimka (eng. *snapshot*)



- Efekti potvrđenih (zelenih) transakcija su vidljivi
- Efekti aktivnih (crvenih) transakcija nisu vidljivi

Vidljivost n-torki - primjer

U osoba postoje sljedeće verzije ntorki:

Koje n-torke će vidjeti SELECT naredba?

SELECT * FROM osoba;

ako vrijedi:

ID najranije aktivne tran.: 1311

ID koji još nije dodijeljen: 1315

Aktivne transakcije: 1311, 1312, 1314

✓ (1)

✗ (2)

✗ (3)

✓ (4)

✗ (5)

✓ (6)

✓ (7)

✗ (8)

xmin	xmax	sif Osoba	prez	...
1310	0	1	Ban	...
1311	0	2	Car	...
1309	1310	3	Horvat	...
1310	1312	4	Markuš	...
1308	1313	5	Kos	...
1313	0	5	Kosić	...
1309	1314	6	Balić	...
1314	0	6	Banić	

(1) **Da** - 1310 potvrđena (potvrđen unos)

(2) **Ne** - 1311 nepotvrđena (nepotvrđen unos)

(3) **Ne** - 1310 potvrđena (potvrđeno brisanje)

(4) **Da** - 1312 nepotvrđena (nepotvrđeno brisanje)

(5) **Ne** - 1313 potvrđena } (potvrđena izmjena)

(6) **Da** - 1313 potvrđena }

(7) **Da** - 1314 nepotvrđena } (nepotvrđena izmjena)

(8) **Ne** - 1314 nepotvrđena }

(1)

(4)

(6)

(7)

xmin	xmax	sif Osoba	prez	...
1310	0	1	Ban	...
1310	1312	4	Markuš	...
1313	0	5	Kosić	...
1309	1314	6	Balić	...

MVCC PostgreSQL: INSERT

T1 (Read Committed)

```
BEGIN TRANSACTION;  
SELECT txid_current() txId;  
INSERT INTO osoba  
VALUES (1, 'Car', 'Ana');  
SELECT xmin, xmax, osoba.*  
FROM osoba  
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1310	0	1	Car	Ana

```
COMMIT TRANSACTION;
```

Nije moguće
prijaviti čitanje

txid
1310

txid
1311

txid
1312

txid
1313

(Read Committed)

```
SELECT txid_current() txid;  
SELECT xmin, xmax, osoba.*  
FROM osoba  
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
------	------	----------	------	-----

```
SELECT xmin, xmax, osoba.*  
FROM osoba  
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1310	0	1	Car	Ana

PostgreSQL: UPDATE

T1 (Read Committed)

```
BEGIN TRANSACTION;
SELECT txid_current() txId;
UPDATE osoba
SET Ime = 'Iva'
WHERE sifOsoba = 1;
SELECT xmin, xmax, osoba.*
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1320	0	1	Car	Iva

```
COMMIT TRANSACTION;
```

xmin	xmax	sifOsoba	prez	ime
1310	0	1	Car	Ana

(Read Committed)

```
SELECT txid_current() txId;
SELECT xmin, xmax, osoba.*
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1310	1320	1	Car	Ana

```
SELECT xmin, xmax, osoba.*
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1320	0	1	Car	Iva

txid
1320

txid
1321

txid
1322

txid
1323

Nije moguće
prijaviti čitanje

MVCC PostgreSQL: DELETE

T1 (Read Committed)

xmin	xmax	sifOsoba	prez	ime
1320	0	1	Car	Iva

(Read Committed)

```
BEGIN TRANSACTION;
SELECT txid_current() txid;
DELETE FROM osoba
  WHERE sifOsoba = 1;
SELECT xmin, xmax, osoba.*
  FROM osoba
  WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
------	------	----------	------	-----

COMMIT TRANSACTION;

txid
1330

txid
1331

txid
1332

txid
1333

Nije moguće
prljavo čitanje

```
SET SESSION CHARACTERISTICS
AS TRANSACTION ISOLATION
LEVEL READ COMMITTED;
```

```
SELECT txid_current() txid;
SELECT xmin, xmax, osoba.*
  FROM osoba
  WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
1320	1330	1	Car	Iva

```
SELECT xmin, xmax, osoba.*
  FROM osoba
  WHERE sifOsoba = 1;
```

xmin	xmax	sifOsoba	prez	ime
------	------	----------	------	-----

PostgreSQL: neponovljivo čitanje

T1 (T_{id}=1340) (READ COMMITTED)

```
BEGIN TRANSACTION;
```

```
SELECT xmin, xmax, ime
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	ime
1335	0	Ana

```
SELECT xmin, xmax, ime
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	ime
1342	0	Ava

```
COMMIT TRANSACTION;
```

T2 (T_{id}=1341) (SERIALIZABLE)

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
SELECT xmin, xmax, ime
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	ime
1335	0	Ana

```
SELECT xmin, xmax, ime
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	ime
1335	1342	Ana

```
COMMIT TRANSACTION;
SELECT xmin, xmax, ime
FROM osoba
WHERE sifOsoba = 1;
```

xmin	xmax	ime
1342	0	Ava

T3 (T_{id}=1342) (READ COMMITTED)

```
BEGIN TRANSACTION;
```

```
UPDATE osoba SET ime='Ava'
WHERE sifOsoba = 1;
COMMIT TRANSACTION;
```

Čitanje ne blokira pisanje!

PostgreSQL: sablasne n-torke

T1 (T_{id}=1350) (READ COMMITTED)

```
BEGIN TRANSACTION;
```

```
INSERT INTO osoba  
VALUES (2, 'Ban', 'Ivo');  
COMMIT TRANSACTION;
```

T2 (T_{id}=1351) (READ COMMITTED)

```
BEGIN TRANSACTION;
```

```
SELECT xmin, xmax, ime  
FROM osoba;
```

xmin	xmax	ime
1342	0	Ava

...

```
SELECT xmin, xmax, ime  
FROM osoba;
```

xmin	xmax	ime
1342	0	Ava
1350	0	Ivo

```
COMMIT TRANSACTION;
```

T3 (T_{id}=1352) (SERIALIZABLE)

```
BEGIN TRANSACTION;
```

```
SET TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

```
SELECT xmin, xmax, ime  
FROM osoba;
```

xmin	xmax	ime
1342	0	Ava

...

```
SELECT xmin, xmax, ime  
FROM osoba;
```

xmin	xmax	ime
1342	0	Ava

```
COMMIT TRANSACTION;
```

```
SELECT xmin, xmax, ime  
FROM osoba;
```

xmin	xmax	ime
1342	0	Ava
1350	0	Ivo

Pisanje ne blokira čitanje!

PostgreSQL: pisanje blokira pisanje

T1 (T_{id} =1350) (READ COMMITTED)

```
BEGIN TRANSACTION;  
UPDATE osoba  
  SET ime='Ana'  
  WHERE sifOsoba = 1;  
--one row affected
```

```
COMMIT TRANSACTION;
```

```
SELECT xmin, xmax,  
       sifOsoba, ime  
FROM osoba;
```

xmin	xmax	sifOsoba	ime
1351	0	1	Iva

T2 (T_{id} =1351)(READ COMMITTED)

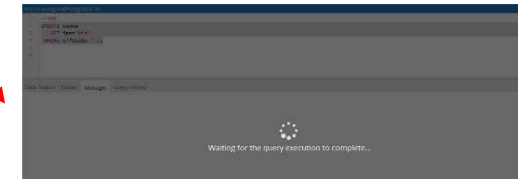
```
BEGIN TRANSACTION;  
  
UPDATE osoba  
  SET ime = 'Iva'  
  WHERE sifOsoba = 1;  
--Waiting for the query  
to complete  
--pokušat će ponovno kad  
T1 završi
```

```
--one row affected  
COMMIT TRANSACTION;
```

```
SELECT xmin, xmax,  
       sifOsoba, ime  
FROM osoba;
```

xmin	xmax	sifOsoba	ime
1351	0	1	Iva

- T1 postavlja ekskluzivni ključ na n-torku.
- T2 zbog nekompatibilnosti ključeva ne može postići to isto. Čeka do završetka T1. Nakon potvrđivanja T1, T2 će uspješno izmijeniti n-torku.



Pisanje blokira pisanje!

PostgreSQL: pisanje blokira pisanje

T1 (T_{id} =1360) (READ COMMITTED) **T2** (T_{id} =1361) (SERIALIZABLE)

```
BEGIN TRANSACTION;  
UPDATE osoba  
  SET ime='Ana'  
  WHERE sifOsoba = 1;  
--one row affected
```

```
COMMIT TRANSACTION;
```

```
SELECT xmin, xmax,  
       sifOsoba, ime  
FROM osoba  
WHERE sifOsoba = 1;
```

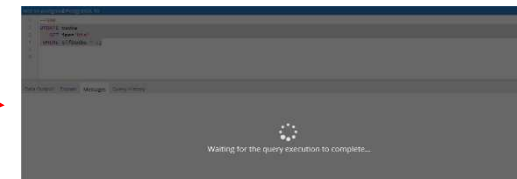
```
BEGIN TRANSACTION;  
SET TRANSACTION ISOLATION LEVEL  
SERIALIZABLE;
```

```
UPDATE osoba  
  SET ime='Mia'  
  WHERE sifOsoba = 1;  
--Waiting for the query to  
complete...
```

```
--ERROR: could not serialize  
access due to concurrent update  
COMMIT TRANSACTION;  
--PgAdmin: ROLLBACK
```

```
SELECT xmin, xmax,  
       sifOsoba, ime  
FROM osoba  
WHERE sifosoba = 1;
```

- SERIALIZABLE osigurava serijalizabilno izvođenje transakcija i neće dozvoliti UPDATE n-torke koja je izmijenjena nakon početka transakcije.



xmin	xmax	sifOsoba	ime
1360	0	1	Ana

xmin	xmax	sifOsoba	ime
1360	0	1	Ana

Pisanje blokira pisanje!

PostgreSQL: pisanje blokira pisanje

T1 (T_{id} =1360) (READ COMMITTED) **T2** (T_{id} =1361) (SERIALIZABLE)

```
BEGIN TRANSACTION;  
UPDATE osoba  
    SET ime='Ana'  
    WHERE sifOsoba = 1;  
--one row affected
```

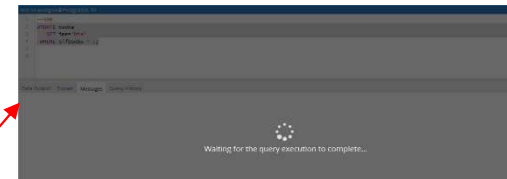
```
ROLLBACK TRANSACTION;
```

```
SELECT xmin, xmax,  
        sifOsoba, ime  
    FROM osoba  
    WHERE sifOsoba = 1;
```

```
BEGIN TRANSACTION;  
SET TRANSACTION ISOLATION  
LEVEL SERIALIZABLE;
```

```
UPDATE osoba  
    SET ime='Mia'  
    WHERE sifOsoba = 1;  
--Waiting for the query to  
complete...
```

```
-- one row affected  
COMMIT TRANSACTION;  
SELECT xmin, xmax,  
        sifOsoba, ime  
    FROM osoba  
    WHERE sifosoba = 1;
```



- Efekti T1 su poništeni, T1 nije mijenjala n-torku
- Serijalizabilnost nije upitna pa T2 mijenja n-torku.

xmin	xmax	sifOsoba	ime
1361	0	1	Mia

xmin	xmax	sifOsoba	ime
1361	0	1	Mia

Pravila istodobnog pisanja (PostgreSQL)

- Dvije istodobne transakcije mogu mijenjati (UPDATE, DELETE, SELECT FOR UPDATE) istodobno samo ako mijenjaju disjunktne skupove ntorki.
- Ako transakcija pokušava mijenjati ntorku koju istovremeno mijenja neka druga transakcija:
 - Ako druga transakcija traje, čekaj dok ne obavi COMMIT ili ROLLBACK:
 - **ROLLBACK**: nastavi s izmjenom (koristeći staru/nepromijenjenu verziju ntorka)
 - **COMMIT**, ovisno o razini izolacije:
 - a) **SERIALIZABLE**: odustani s „can't serialize” greškom
 - b) **READ COMMITTED**: nastavi izmjenu s novom verzijom ntorka, ali samo ako nova verzija ntorka i dalje zadovoljava WHERE uvjet

PostgreSQL: eksplicitno zaključavanje

- Kada SELECT naredba uključuje FOR UPDATE ili FOR SHARE SELECT

...

[FOR { UPDATE | SHARE }]

[OF table_name [, ...]] [NOWAIT] [...]

provodi se eksplicitno zaključavanje n-torki koje zadovoljavaju uvjet selekcije.

FOR SHARE postavlja ključ za čitanje

FOR UPDATE postavlja ključ za pisanje

- Ako je SELECT naredba obavljena unutar eksplicitno definiranih granica transakcije ključevi se otpuštaju tek po dovršetku transakcije (COMMIT ili ROLLBACK).

PostgreSQL: eksplicitno zaključavanje

T1 (T_{id}=1360) (READ COMMITTED)

```
BEGIN TRANSACTION;  
SELECT xmin, xmax, ime  
  FROM osoba  
 WHERE sifOsoba = 1  
  FOR SHARE;
```

xmin	xmax	ime
1332	1351	Ava

```
UPDATE osoba  
  SET ime = 'Ana'  
  WHERE sifOsoba = 1;  
  
--query is running (čeka se  
otključavanje zapisa kojeg je T3  
zaključao za čitanje)  
  
Query returned successfully: one row  
affected...  
  
COMMIT TRANSACTION;
```

T3 (T_{id}=1361) (READ COMMITTED)

```
BEGIN TRANSACTION;  
SELECT xmin, xmax, ime  
  FROM osoba  
 WHERE sifOsoba = 1  
  FOR SHARE;
```

xmin	xmax	ime
1332	1351	Ava

```
UPDATE osoba  
  SET ime = 'Ana'  
  WHERE sifOsoba = 1;  
  
ERROR:  deadlock detected  
DETAIL:  Process 13212 waits for  
ExclusiveLock on tuple (0,1) of  
relation 16417 of database 16393;  
blocked by process 8632.  
Process 8632 waits for ShareLock on  
transaction 569; blocked by process  
13212.
```

Čitanje blokira pisanje!