Poglavje 6

Varnost v SUPB

Trije vidiki varnosti v SUPB

1. Transakcijska
   - kaj so transakcije
   - omogočanje istočasnega dela več uporabnikov nad istimi podatki

2. Dostopna
   - kdo sme dostopati do podatkovne baze
   - kdo sme kaj delati s katerimi podatki

3. Podatkovna
   - celovita skrb za varnost podatkov v podatkovni bazi
   - obnavljanje PB

Transakcijska varnost v SUPB

- Definicija transakcije
- Lastnosti transakcij
- Gradniki SUPB povezani z nadzorom sočasnosti ter obnovljivostjo podatkov

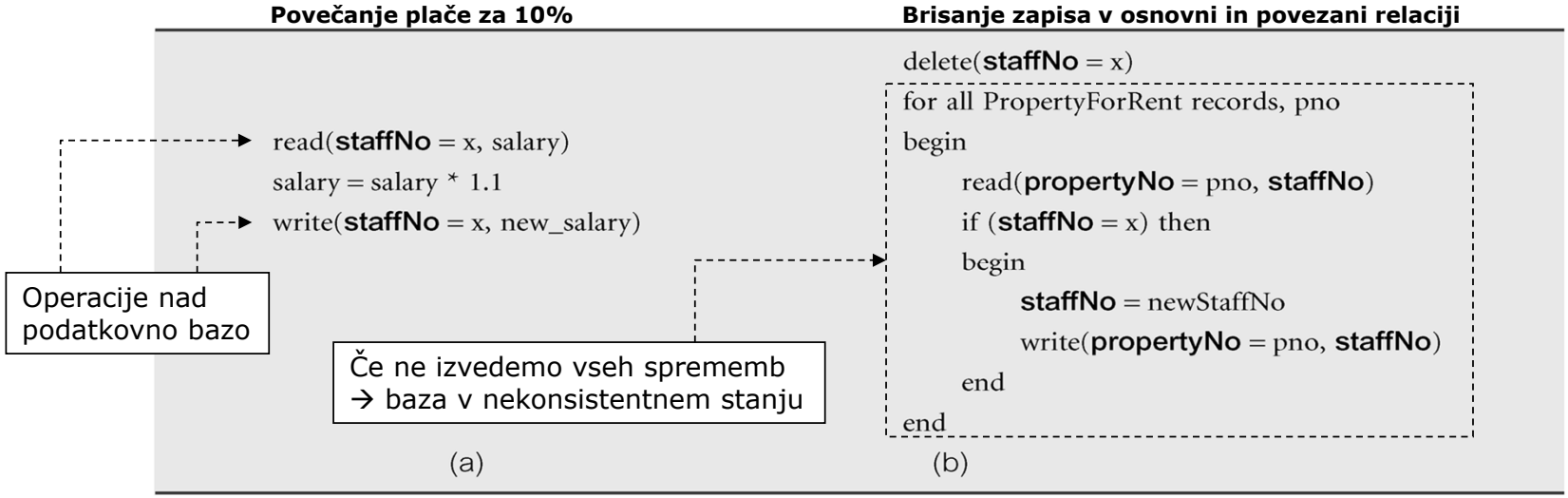
Transakcija - opredelitev oz. definicija

- Transakcija je operacija ali **niz operacij**, ki berejo ali pišejo v podatkovno bazo in so izvedene s strani enega uporabnika oziroma uporabniškega programa.
- Razvijalec določi, katere operacije tvorijo transakcijo (primer: bančni prenos sredstev med računi)
- Transakcija je logična enota dela – lahko je cel program ali samostojen ukaz (npr. INSERT ali UPDATE)
- Izvedba uporabniškega programa je s stališča podatkovne baze vidna kot ena ali več transakcij.

Opredelitev transakcije...

- Primeri transakcij

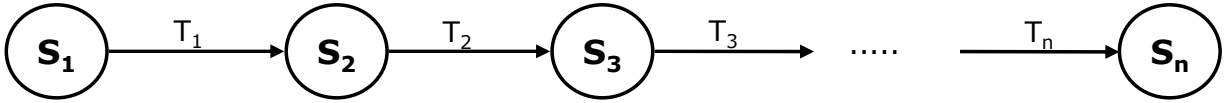
Staff(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent(propertyNo, street, city, postcode, type, rooms, rent,
ownerNo, staffNo, branchNo)



Opredelitev transakcije...

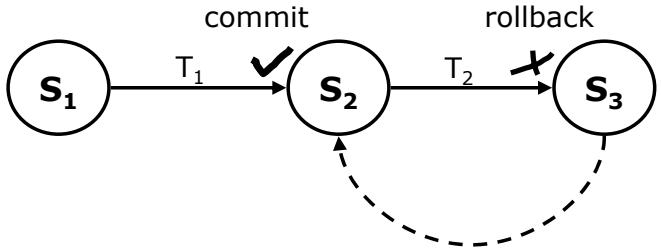
S_i; i=1 .. n ≈ konsistentna ali skladna stanja v podatkovni bazi

Med izvajanjem transakcije je lahko stanje v bazi neskladno!



Opredelitev transakcije…

- Transakcija se lahko zaključi na dva načina:
  - Uspešno ali
  - Neuspešno
- Če končana uspešno, jo potrdimo (commit), sicer razveljavimo (abort, rollback).
- Ob neuspešnem zaključku moramo podatkovno bazo vrniti v skladno stanje pred začetkom transakcije.

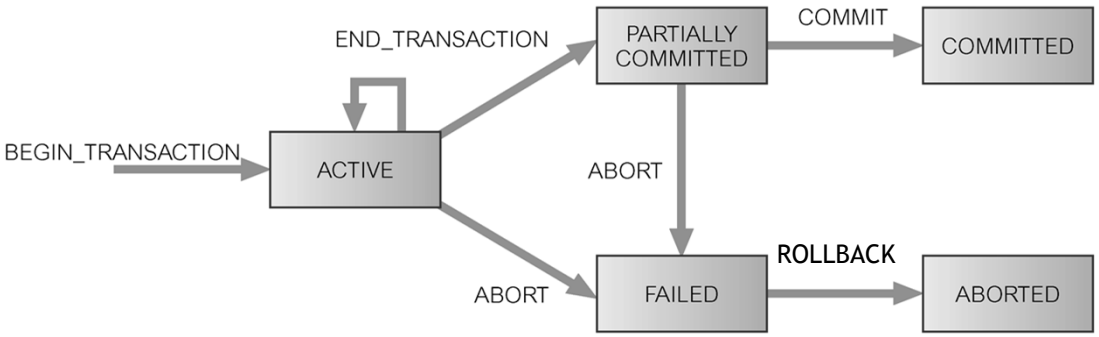


Opredelitev transakcije…

- Enkrat potrjene transakcije ni več moč razveljaviti.
  - Če smo s potrditvijo naredili napako, moramo za povrnitev v prejšnje stanje izvesti novo transakcijo, ki ima obraten učinek nad podatki v podatkovni bazi.
- Razveljavljene transakcije lahko ponovno poženemo.
- Enkrat zavrnjena transakcija je drugič lahko zaključena uspešno (odvisno od razloga za njeno prvotno neuspešnost).

Opredelitev transakcije

- SUPB se ne zaveda, kako so operacije logično grupirane. Uporabljamo eksplicitne ukaze, ki to povedo:
  - Po ISO standardu uporabljamo ukaz BEGIN TRANSACTION za začetek in COMMIT ali ROLLBACK za potrditev ali razveljavitev transakcije.
  - Če konstruktov za začetek in zaključek transakcije ne uporabimo, SUPB privzame cel uporabniški program kot eno transakcijo. Če se uspešno zaključi, izda implicitni COMMIT, sicer ROLLBACK.



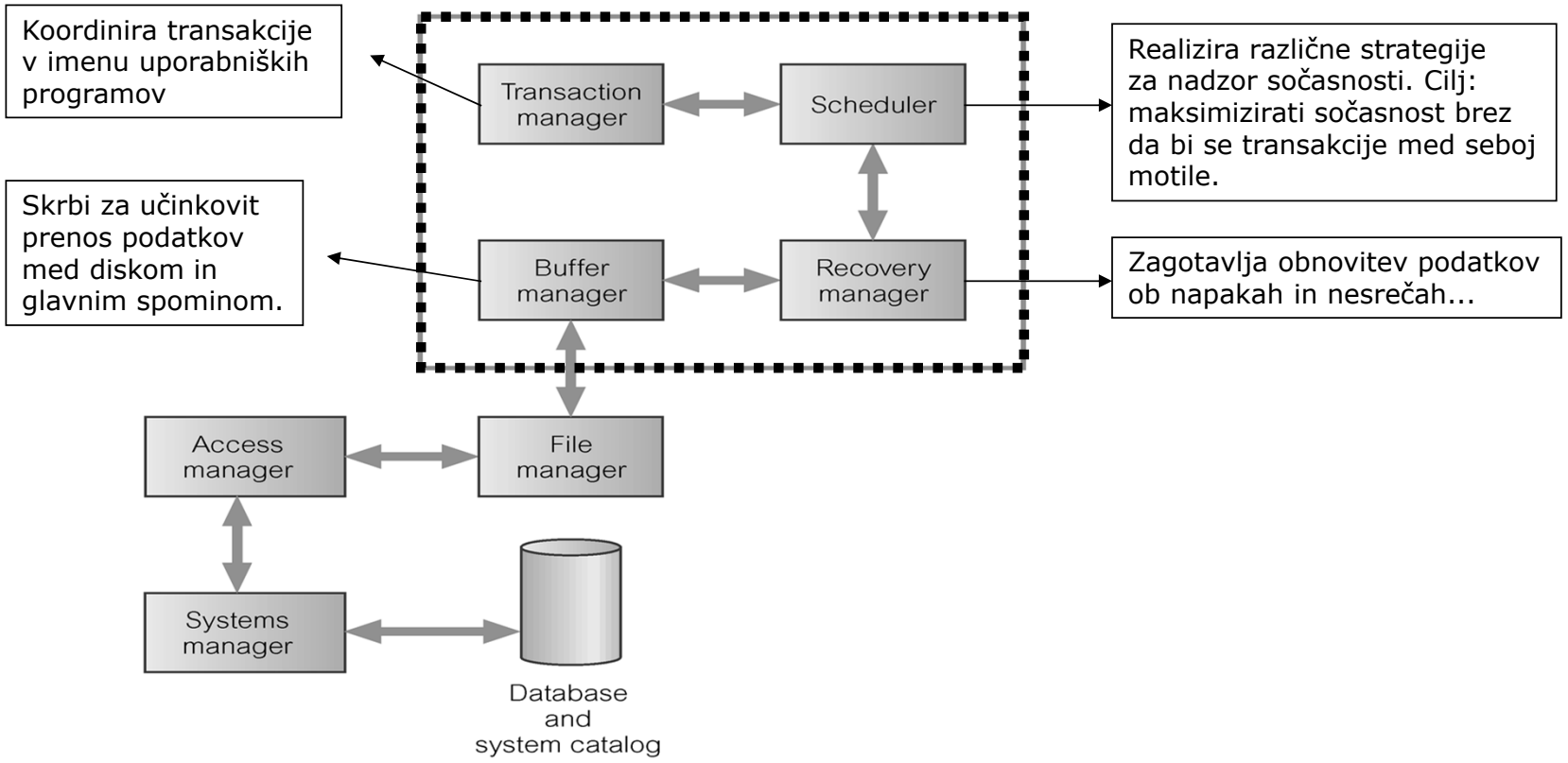
Lastnosti transakcij (ACID*)

- Vsaka transakcija naj bi zadoščala štirim osnovnim lastnostim:
  - Atomarnost: transakcija predstavlja atomaren sklop operacij. Ali se izvede vse ali nič. Atomarnost mora zagotavljati SUPB.
  - Konsistentnost: transakcija je sklop operacij, ki podatkovno bazo privede iz enega konsistentnega stanja v drugo. Zagotavljanje konsistentnosti je naloga SUPB (zagotavlja, da omejitve nad podatki niso kršene…) in programerjev (preprečuje vsebina neskladnosti).
  - Izolacija: transakcije se izvajajo neodvisno ena od druge → delni rezultati transakcije ne smejo biti vidni drugim transakcijam. Za izolacijo skrbi SUPB.
  - Trajnost: učinek potrjene transakcije je trajen – če želimo njen učinek razveljaviti, moramo to narediti z novo transakcijo, ki z obratnimi operacijami podatkovno bazo privede v prvotno stanje. Zagotavljanje trajnosti je naloga SUPB.

*__ACID__ – __A__tomicity, __C__onsistency, __I__solation and __D__urability

Obvladovanje transakcij – arhitektura

- Komponente SUPB za obvladovanje transakcij, nadzor sočasnosti in obnovitev podatkov:



Nadzor sočasnosti

- Namen nadzora sočasnosti
- Serializacija urnika transakcij
- Zaklepanje in časovno žigosanje
- Mrtve in žive zanke – detekcija in odprava
- Optimistične metode nadzora sočasnosti

Zakaj sočasnost?...

- Eden od ciljev in prednosti PB je možnost sočasnega dostopa s strani več uporabnikov do skupnih podatkov.

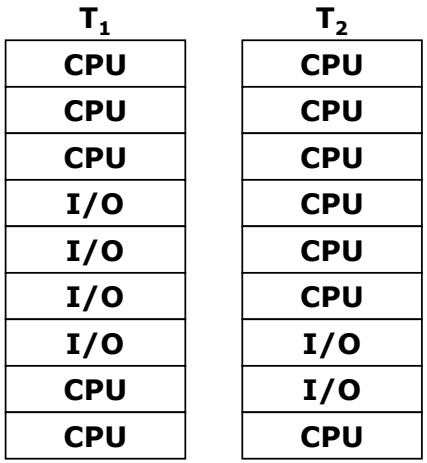
- Če vsi uporabniki podatke le berejo – nadzor sočasnosti trivialen;
- Če več uporabnikov sočasno dostopa do podatkov in vsaj eden podatke tudi zapisuje – možni konflikti.

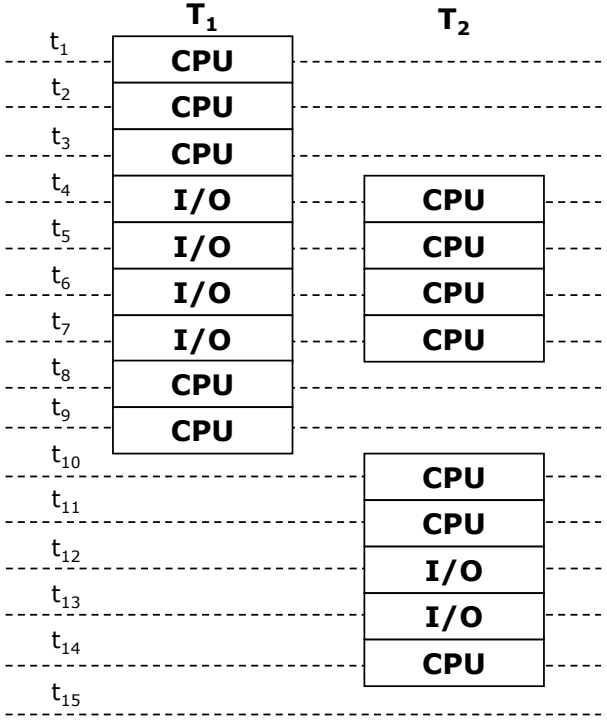
Zakaj sočasnost?...

- Za večino računalniških sistemov velja:
  - imajo vhodno izhodne enote, ki znajo samostojno izvajati I/O operacije.
  - V času I/O operacij centralna procesorska enota CPU izvaja druge operacije.
- Taki sistemi lahko izvajajo dve ali več transakcij sočasno.
- Primer:
  - Sistem začne z izvajanjem prve transakcije in jo izvaja vse do prve I/O operacije. Ko naleti na I/O operacijo, jo začne izvajati, CPU pa z izvajanjem operacij transakcije začasno prekine. V tem času se začne izvajati druga transakcija. Ko se I/O operacija prve zaključi, CPU začasno prekine z izvajanjem druge in se vrne k prvi.

Zakaj sočasnost?...

- Prepletanje operacij dveh transakcij...





Problemi v zvezi z nadzorom sočasnosti in konsistentnosti

- V *centraliziranem* SUPB zaradi sočasnosti dostopa različni problemi:
  - Izgubljene spremembe (lost update): uspešno izveden UPDATE se razveljavi zaradi istočasno izvajane operacije s strani drugega uporabnika.
  - Uporaba nepotrjenih podatkov (dirty read): transakciji je dovoljen vpogled v podatke druge transakcije, še preden je ta potrjena.
  - Neponovljivo branje (neskladnost analize) (nen-repeatable read): transakcija prebere več vrednosti iz podatkovne baze. Nekatere izmed njih se v (po navadi daljšem) času izvajanja transakcije zaradi operacij neke druge transakcije spremenijo.
  - Branje fantomskih vrstic (phantom read): transakcija dvakrat izvede poizvedbo in dobi drugič različen rezultat – dodatne, fantomske vrstice, zaradi uspešne operacije neke druge transakcije
- *Decentralizirani* (porazdeljeni) SUPB pa imajo še dodatne probleme

Primeri težav s sočasnostjo dostopa…

- Izgubljene spremembe (lost update)
  - T_1 dvig 10 € iz TRR, na katerem je začetno stanje 100 €.
  - T_2 depozit 100 € na isti TRR.
  - Po zaporedju T_1, T_2 končno stanje enako 190 €.

Time	T_1	T_2	$\mathbf{bal_x}$
t_1		begin_transaction	100
t_2	begin_transaction	read($\mathbf{bal_x}$)	100
t_3	read($\mathbf{bal_x}$)	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	100
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	write($\mathbf{bal_x}$)	200
t_5	write($\mathbf{bal_x}$)	commit	90
t_6	commit		90

Primeri težav s sočasnostjo dostopa…

- Uporaba nepotrjenih podatkov (dirty read)
  - T_3 dvig 10 € iz TRR.
  - T_4 depozit 100 € na isti TRR.
  - Po zaporedju T_3, T_4 končno stanje enako 190 €. Če T_4 preklicana, je pravilno končno stanje 90 €.

Time	T_3	T_4	$\mathbf{bal_x}$
t_1		begin_transaction	100
t_2		read($\mathbf{bal_x}$)	100
t_3		$\mathbf{bal_x} = \mathbf{bal_x} + 100$	100
t_4	begin_transaction	write($\mathbf{bal_x}$)	200
t_5	read($\mathbf{bal_x}$)	⋮	200
t_6	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	rollback	100
t_7	write($\mathbf{bal_x}$)		190
t_8	commit		190

Primeri težav s sočasnostjo dostopa…

- Nekonsistentna analiza (non-repeatable read)
  - Začetno stanje: bal_x=100 €, bal_y=50 €, bal_z=25 €;
  - Seštevek je 175 €
  - T_5 prenos 10 € iz TRR_x na TRR_z.
  - T_6 izračun skupnega stanja na računih TRR_x, TRR_y in TRR_z.

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Primeri težav s sočasnostjo dostopa…

- Fantomsko branje (phantom read)
  - podobno kot nekonsistentna analiza
  - primer: dvakratno izvajanje iste poizvedbe znotraj transakcije
    - **fantomsko branje**: pojavijo se **nove vrstice**, ki izpolnjujejo pogoje za vključitev v rezultat, a jih ob prvi izvedbi ni bilo (posledica INSERT)
  - razlika do nekonsistentne analize: ta se med izvedbo spremenijo vrednosti v **že prebranih vrsticah** (posledica UPDATE)

- Kaj pa posledice DELETE ukaza?

Transakcije v SQL

- SQL vsebuje mehanizme za uporabo in (delno) nadzor upravljanja s transakcijami
- **Kako** uporabljamo SQL mehanizme za podporo transakcijam
  - Začetek in konec transakcije
  - Stopnje izolacije
  - Dodatki ukazom
  - Preverjanje omejitev

Transakcije v SQL

- Standardni ISO SQL definira transakcijski model z ukazoma COMMIT in ROLLBACK
  - Transakcija se začne na začetku programa ali neposredno za COMMIT/ROLLBACK
- Razširitve z vpeljavo dodatnih parametrov izvajanja:
  - PostgreSQL, Oracle, MySQL: START TRANSACTION ali BEGIN
  - Microsoft Transact-SQL: BEGIN TRANSACTION
- Transakcija je logična enota dela z enim ali več SQL ukazi. S stališča zagotavljanja skladnega stanja je atomarna.
- Spremembe, ki so narejene znotraj poteka transakcije, niso vidne navzven drugim transakcijam, dokler transakcija ni končana.

Transakcije v SQL

- Transakcija se lahko zaključi na enega od štirih načinov:
  - Transakcija se uspešno zaključi s COMMIT; spremembe so permanentne.
  - Transakcija se prekine z ROLLBACK; spremembe, narejene s transakcijo, se razveljavijo.
  - Program, znotraj katerega se izvaja transakcija, se uspešno konča. Transakcija je potrjena implicitno (brez COMMIT).
  - Program, znotraj katerega se izvaja transakcija, se ne konča uspešno. Transakcija se implicitno razveljavi (brez ROLLBACK).

Transakcije v SQL

- Nova transakcija se začne z novim SQL stavkom, ki transakcijo začne (prvi stavek, za BEGIN/START TRANSACTION, za COMMIT ali ROLLBACK).
- SQL transakcij po standardu ne moremo gnezditi.
- Transakcijske nastavitve upravljamo s pomočjo ukaza SET TRANSACTION

```
SET TRANSACTION [READ ONLY | READ WRITE] |
    [ISOLATION LEVEL
        READ UNCOMMITTED    | READ COMMITTED |
        REPEATABLE READ     | SERIALIZABLE    ]
```

Transakcije v SQL

- READ ONLY – pove, da transakcija vključuje samo operacije, ki iz baze berejo.
  - SUPB bo dovolil INSERT, UPDATE in DELETE samo nad začasnimi tabelami.
- ISOLATION LEVEL – pove stopnjo interakcije, ki jo SUPB dovoli med to in drugimi transakcijami.
- MySQL (z ustreznimi pravicami):
  SET [GLOBAL | SESSION] TRANSACTION

  GLOBAL: globalno
  SESSION: znotraj iste povezave
  Brez: le za naslednjo transakcijo (lahko tudi pri START TRANSACTION)

Transakcije v SQL

- Učinek SET TRANSACTION ISOLATION LEVEL

	Branje neobsto-ječega podatka	**Neponovlji-vo branje**	**Fantomsko branje**	**Izgubljeno ažuriranje**
Read Uncommitted	Da	Da	Da	Da (eno- in dvodelni update)
Read Committed	Ne	Da	Da	Da (dvodelni update)
Repeatable Read	Ne	Ne	Da	Ne
Serializable	Ne	Ne	Ne	Ne

- Različne stopnje izolacije izbiramo zaradi različnega obsega želene sočasnosti (kompromis)

Takojšnje in zapoznele omejitve...

- Včasih želimo, da se omejitve ne bi upoštevale takoj, po vsakem SQL stavku, temveč ob zaključku transakcije.
- Omejitve lahko definiramo kot
  - INITIALLY IMMEDIATE – ob začetku transakcije;
  - INITIALLY DEFERRED – ob zaključku transakcije.
- Če izberemo INITIALLY IMMEDIATE (privzeta možnost), lahko določimo tudi, ali je zakasnitev moč določiti kasneje. Uporabimo [NOT] DEFERRABLE.

Takojšnje in zapoznele omejitve

- Način upoštevanja omejitev za trenutno transakcijo nastavimo z ukazom SET CONSTRAINTS.
- Zakaj? Ker smo znotraj transakcije krajši čas lahko v nekonsistentnem stanju (ni problema zaradi **A**CID)

SET CONSTRAINTS

{ALL | constraintName [, . . .]}

{DEFERRED | IMMEDIATE}

Transakcijski dodatki k SELECT stavku

- Pomagamo upravljalcu transakcij da pisalno ali bralno zaklene prebrani podatek, ne glede na nivo izolacije

- SELECT ... FOR UPDATE; -- na koncu SELECT stavka
  vse prebrane vrstice zaklene pisalno (ekskluzivno)
- SELECT ... LOCK IN SHARE MODE; -- na koncu SELECT
  vse prebrane vrstice zaklene bralno (deljeno)

- tovrstno zaklepanje **ni** odvisno od ISOLATION LEVEL, upoštevanje teh zaklepanj pa **je**

Serializacija in obnovljivost…

- Če transakcije izvajamo zaporedno, se izognemo vsem problemom. Problem: nizka učinkovitost.
- Vzporedno (nezaporedno) izvajanje: problem so interakcije s podatki (read/write).
- Kako v največji meri uporabiti vzporednost izvajanja?

Nekaj definicij

- Serializacija:
  - način, kako identificirati načine izvedbe transakcij, ki zagotovijo ohranitev skladnosti in celovitosti podatkov.

Serializacija in obnovljivost...

- Urnik
  - Zaporedje operacij iz množice sočasnih transakcij, ki ohranja vrstni red operacij posameznih transakcij.
- Zaporedni urnik
  - Urnik, v katerem so operacije posameznih transakcij izvedene zaporedoma, brez prepletanja z operacijami iz drugih transakcij.
- Nezaporedni urnik
  - Urnik, v katerem se operacije ene transakcija prepletajo z operacijami iz drugih transakcij.

Serializacija in obnovljivost…

- Namen serializacije:
  - Najti nezaporedne urnike, ki omogočajo vzporedno izvajanje transakcij brez konfliktov. Dajo rezultat, kot če bi transakcije izvedel zaporedno.
- S serializacijo v urnikih spreminjamo vrstni red bralno/pisalnih operacij med transakcijami (ne znotraj ene same). Vrstni red je pomemben:
  - Če dve transakciji bereta isti podatek, nista v konfliktu. Vrstni red nepomemben.
  - Če dve transakciji bereta ali pišeta popolnoma ločene podatke, nista v konfliktu. Vrstni red nepomemben.
  - Če neka transakcija podatek zapiše, druga pa ta isti podatek bere ali piše, je vrstni red pomemben.

Primer

	U_A Nezaporedni urnik		U_B Nezaporedni urnik		U_C Zaporedni urnik	
Time	T_7	T_8	T_7	T_8	T_7	T_8
t_1	begin_transaction		begin_transaction		begin_transaction	
t_2	read($\mathbf{bal_x}$)		read($\mathbf{bal_x}$)		read($\mathbf{bal_x}$)	
t_3	write($\mathbf{bal_x}$)		write($\mathbf{bal_x}$)		write($\mathbf{bal_x}$)	
t_4		begin_transaction		begin_transaction	read($\mathbf{bal_y}$)	
t_5		read($\mathbf{bal_x}$)		read($\mathbf{bal_x}$)	write($\mathbf{bal_y}$)	
t_6		write($\mathbf{bal_x}$)	read($\mathbf{bal_y}$)		commit	
t_7	read($\mathbf{bal_y}$)			write($\mathbf{bal_x}$)		begin_transaction
t_8	write($\mathbf{bal_y}$)		write($\mathbf{bal_y}$)			read($\mathbf{bal_x}$)
t_9	commit		commit			write($\mathbf{bal_x}$)
t_{10}		read($\mathbf{bal_y}$)		read($\mathbf{bal_y}$)		read($\mathbf{bal_y}$)
t_{11}		write($\mathbf{bal_y}$)		write($\mathbf{bal_y}$)		write($\mathbf{bal_y}$)
t_{12}		commit		commit		commit
	(a)		(b)		(c)	

Serializabilnost

- Razpored ukazov transakcij je serializabilen oziroma zaporedniški kadar velja
  - Razpored je izmeničen
  - Rezultat izvajanja razporeda vedno ustreza nekemu zaporednemu razporedu ukazov

Transakcije, ki jih ni moč serializirati…

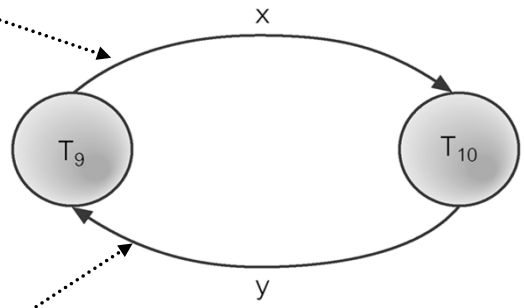
- Preverjamo s pomočjo usmerjenega grafa zaporedja
  G = (N, E); N → vozlišča, E → povezave
- Gradnja grafa:
  - Kreiraj vozlišče za vsako transakcijo
  - Kreiraj usmerjeno povezavo $T_i \rightarrow T_j$, če T_j bere vrednost, predhodno zapisano s T_i
  - Kreiraj usmerjeno povezavo $T_i \rightarrow T_j$, če T_j piše vrednost, ki je bila predhodno prebrana s T_i (tudi, če je vmes COMMIT)
  - Kreiraj usmerjeno povezavo $T_i \rightarrow T_j$, če T_j piše vrednost, ki je bila predhodno zapisana s T_i (tudi, če je vmes COMMIT)

Če graf vsebuje cikel, potem serializacija urnika ni možna!

Primer

- Imamo naslednjo situacijo:
  - T_9 prenese \$100 iz TRR_x na TRR_y.
  - T_{10} stanje na obeh računih poveča za 10%.
  - Graf zaporedja vsebuje cikel, zato transakcij ni moč serializirati.

Time	T_9	T_{10}
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 100$	
t_4	write(bal_x)	begin_transaction
t_5		read(bal_x)
t_6		$bal_x = bal_x * 1.1$
t_7		write(bal_x)
t_8		read(bal_y)
t_9		$bal_y = bal_y * 1.1$
t_{10}		write(bal_y)
t_{11}	read(bal_y)	commit
t_{12}	$bal_y = bal_y - 100$	
t_{13}	write(bal_y)	
t_{14}	commit	



Vrste serializacij

- Predmet serializacije, ki smo jo obravnavali, so bile konflikte operacije.
  - Serializacija konfliktnih operacij (Conflict Serializibility) zagotavlja, da so konfliktne operacije izvedene tako, kot bi bile v zaporednem urniku.

- Obstajajo tudi druge vrste serializacije.
  - Primer: serializacija vpogledov (View serializibility)

Metode nadzora sočasnosti...

- Osnovne metode za nadzor sočasnosti temeljijo na dveh principih:
  - Zaklepanje: zagotavlja, da je sočasno izvajanje enakovredno zaporednemu izvajanju, pri čemer zaporedje ni določeno.
  - Časovno žigosanje: zagotavlja, da je sočasno izvajanje enakovredno zaporednemu izvajanju, pri čemer je zaporedje določeno s časovnimi žigi.

- Dva problema:
  - Čim hitrejše izvajane nadzora (v konstantnem času, ne glede na dolžino transakcije)
  - Potèka izvajanja transakcijskih ukazov ne poznamo vedno vnaprej
    - DA – transakcijski program sestavlja zaporedje ukazov
    - NE – transakcijski program vsebuje ne-konstantne pogojne stavke (odvisne od podatkov)
    - NE – transakcijski program delno teče zunaj SUPB (npr. Python-SQL)

Metode nadzora sočasnosti

- Pesimistične: v primeru, da bi lahko prišlo do konfliktov, se izvajanje ene ali več transakcij zadrži.

- Optimistične: izhajamo iz predpostavke, da je konfliktov malo, zato dovolimo vzporedno izvajanje, za konflikte pa preverimo na koncu izvedbe.

Zaklepanje...

- Zaklepanje je postopek, ki ga uporabljamo za nadzor sočasnega dostopa do podatkov.
  - Ko ena transakcija dostopa do nekega podatka, zaklepanje onemogoči, da bi ga istočasno uporabljale tudi druge, kar bi lahko pripeljalo do napačnih rezultatov.

- Obstaja več načinov izvedbe. Vsem je skupno naslednje:
  - Transakcija mora preden podatek prebere zahtevati deljeno zaklepanje (shared lock, read lock)
  - Transakcija mora pred pisanjem podatka zahtevati ekskluzivno zaklepanje (exclusive lock, write lock).

Zaklepanje...

- Zrnatost zaklepanja:
  - Zaklepanje se lahko nanaša na poljuben del podatkovne baze (od polja do cele podatkovne baze). Imenovali bomo "podatkovna enota".
  - Transakcije enote zaklepajo pred uporabo in jih odklenejo (sprostijo), ko jih več ne potrebujejo.

- Pomen deljenega in ekskluzivnega zaklepanja:
  - Če ima transakcija deljeno zaklepanje nad neko podatkovno enoto, lahko enoto prebere, ne sme pa vanjo pisati.
  - Če ima transakcija ekskluzivno zaklepanje nad neko podatkovno enoto, lahko enoto prebere in vanjo piše.
  - Deljeno zaklepanje nad neko podatkovno enoto ima lahko več transakcij, ekskluzivno pa samo ena.

Kompatibilnost zaklepanja

- T1: ima zaklepanje. T2: skuša pridobiti zaklepanje.
- T1: lahko nadgradi vsa svoja zaklepanja, če to ni v neskladju z drugimi
- Deljeno ali bralno:
  - Shared lock
  - Read lock
- Ekskluzivno ali pisalno:
  - Exclusive lock
  - Write lock

T1/T2	**Deljeno (bralno)**	**Ekskluzivno (pisalno)**
Deljeno (bralno)	DA	NE
Ekskluzivno (pisalno)	NE	NE

Datotečni pogoni in zaklepanje v MySQL (MariaDB)

- Primer: MySQL nudi več različnih tipov datotečne organizacije (podobno MariaDB)
  - Aria, MyISAM (samo ročno zaklepanje)
  - MERGE
  - ISAM
  - HEAP
  - **InnoDB** (privzeto, popolna podpora transakcijam)
  - BDB - BerkeleyDB Tables
- Kriteriji izbire: podpora transakcijam, zrnatost zaklepanja vrstice/tabele/zapisi), hitrost, varnost
  - https://www.developer.com/db/article.php/2235521/Pros-and-Cons-of-MySQL-Table-Types.htm
  - https://mariadb.com/kb/en/mariadb/show-engines/
  - http://dev.mysql.com/doc/refman/8.0/en/show-engines.html

Zaklepanje...

- Postopek zaklepanja:
  - Če transakcija želi dostopati do neke podatkovne enote, mora pridobiti deljeno (samo za branje) ali ekskluzivno zaklepanje (za branje in pisanje).
  - Če enota še ni zaklenjena, se transakciji zaklepanje odobri.
  - Če je enota že zaklenjena:
    - če je obstoječe zaklepanje deljeno, se odobri, če je kompatibilno
    - če je obstoječe zaklepanje ekskluzivno, mora transakcija počakati, da se sprosti.
  - Ko transakcija enkrat pridobi zaklepanje, le-to velja, dokler ga ne sprosti. To se lahko zgodi eksplicitno (ko transakcija enote ne potrebuje več) ali implicitno (ob prekinitvi ali potrditvi transakcije).

Nekateri sistemi omogočajo prehajanje iz deljenega v ekskluzivno zaklepanje in obratno.

Zaklepanje...

- Opisan postopek zaklepanja sam po sebi še ne zagotavlja serializacije urnikov.
- Primer:

X=(x+100)*1.1
Y=(y*1.1) - 100

Serializacija:
X=(x+100)*1.1
Y=(y-100)*1.1

... ali

X=(x*1.1) + 100
Y=(y*1.1) - 100

Time	T_9	T_{10}
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 100$	
t_4	write(bal_x)	begin_transaction
t_5		read(bal_x)
t_6		$bal_x = bal_x * 1.1$
t_7		write(bal_x)
t_8		read(bal_y)
t_9		$bal_y = bal_y * 1.1$
t_{10}		write(bal_y)
t_{11}	read(bal_y)	commit
t_{12}	$bal_y = bal_y - 100$	
t_{13}	write(bal_y)	
t_{14}	commit	

S =
{write_lock(T9, balx), read(T9, balx),
write(T9, balx), unlock(T9, balx),
write_lock(T10, balx), read(T10, balx),
write(T10, balx), unlock(T10, balx),
write_lock(T10, baly), read(T10, baly),
write(T10, baly), unlock(T10, baly),
commit(T10), write_lock(T9, baly),
read(T9, baly), write(T9, baly),
unlock(T9, baly), commit(T9) }

Dvofazno zaklepanje – 2PL...

- Da zagotovimo serializacijo, moramo upoštevati dodaten protokol, ki natančno definira, kje v transakcijah so postavljena zaklepanja in kje se sprostijo.

- Eden najbolj znanih protokolov je dvofazno zaklepanje (2PL – Two-phase locking).
- Transakcija sledi 2PL protokolu, če se vsa zaklepanja v transakciji izvedejo pred prvim odklepanjem.

Dvofazno zaklepanje – 2PL...

- Po 2PL lahko vsako transakcijo razdelimo na
  - fazo zaseganja: transakcija pridobiva zaklepanja, vendar nobenega ne sprosti
  - fazo sproščanja: transakcija sprošča zaklepanja, vendar ne more več pridobiti novih zaklepanj
- Protokol 2PL zahteva:
  - Transakcija mora pred delom z podatkovno enoto pridobiti zaklepanje
  - Ko enkrat sprosti neko zaklepanje, ne more več pridobiti novega.
  - Če je dovoljeno nadgrajevanje zaklepanja (iz deljenega v ekskluzivno, je to lahko izvedeno le v fazi zasedanja..

Reševanje izgubljenih sprememb z 2PL

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x + 100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

Reševanje nepotrjenih podatkov z 2PL

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		write_lock(bal_x)	100
t_3		read(bal_x)	100
t_4	begin_transaction	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write(bal_x)	200
t_6	WAIT	rollback/unlock(bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write(bal_x)		90
t_{10}	commit/unlock(bal_x)		90

Reševanje nekons

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	write_lock(bal_x)		100	50	25	0
t_4	read(bal_x)	read_lock(bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write(bal_x)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read(bal_z)	WAIT	90	50	25	0
t_9	$bal_z = bal_z + 10$	WAIT	90	50	25	0
t_{10}	write(bal_z)	WAIT	90	50	35	0
t_{11}	commit/unlock(bal_x, bal_z)	WAIT	90	50	35	0
t_{12}		read(bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read_lock(bal_y)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read(bal_z)	90	50	35	140
t_{19}		sum = sum + bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, bal_y, bal_z)	90	50	35	175

Kaskadni preklic…

- Če vse transakcije v urniku sledijo 2PL protokolu, je urnik moč serializirati.

- Pojavijo se lahko težave zaradi nepravilno izvedenih preklicev zaklepanj.
  - Ali lahko preklic zaklepanja neke podatkovne enote naredimo takoj, ko je končana zadnja operacija, ki do te enote dostopa?

Kaskadni preklic...

Time	T_{14}	T_{15}	T_{16}
t_1	begin_transaction		
t_2	write_lock($\mathbf{bal_x}$)		
t_3	read($\mathbf{bal_x}$)		
t_4	read_lock($\mathbf{bal_y}$)		
t_5	read($\mathbf{bal_y}$)		
t_6	$\mathbf{bal_x} = \mathbf{bal_y} + \mathbf{bal_x}$		
t_7	write($\mathbf{bal_x}$)		
t_8	unlock($\mathbf{bal_x}$)	begin_transaction	
t_9	⋮	write_lock($\mathbf{bal_x}$)	
t_{10}	⋮	read($\mathbf{bal_x}$)	
t_{11}	⋮	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	
t_{12}	⋮	write($\mathbf{bal_x}$)	
t_{13}	⋮	unlock($\mathbf{bal_x}$)	
t_{14}	⋮	⋮	
t_{15}	**rollback**	⋮	
t_{16}		⋮	begin_transaction
t_{17}		⋮	read_lock($\mathbf{bal_x}$)
t_{18}		**rollback**	⋮
t_{19}			**rollback**

Kaskadni preklic

- Kaskadni preklici so nezaželeni.
- 2PL, ki onemogoča kaskadne preklice, zahteva, da se sprostitev preklicev izvede šele na koncu transakcije.
  - Rigorozni 2PL (Rigorous 2PL): do konca transakcij zadržujemo vse sprostitve.
  - Striktni 2PL (Strict 2PL): zadržujemo le ekskluzivna zaklepanja.
- Večina DBMS-jev realizira rigorozni ali striktni 2PL.
- Večina primerov bo z rigoroznim 2PL (lažja sledljivost)

Urnike, ki sledijo rigoroznemu 2PL protokolu, je vedno moč serializirati.

Mrtve zanke…

- Mrtva zanka (dead lock): brezizhoden položaj, do katerega pride, ko dve ali več transakcij čakajo ena na drugo, da bodo sprostile zaklepanja.

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

Mrtve zanke…

- Samo ena možnost, da razbijemo mrtvo zanko: preklic ene ali več transakcij.

- Mrtva zanka oziroma njena detekcija in odprava mora biti za uporabnika transparentna.
  - SUPB sam razveljavi operacije, ki so bile narejene do točke preklica transakcije in transakcijo ponovno starta.

Mrtve zanke…

- Tehnike obravnave mrtvih zank:
  - Prekinitev: po poteku določenega časa SUPB transakcijo prekliče in ponovno zažene.
  - Preprečitev: uporabimo časovne žige; dva algoritma:
    - Wait-Die: samo starejše transakcije lahko čakajo na mlajše, sicer je transakcija prekinjena (die) in ponovno pognana z istim časovnim žigom. Sčasoma postane starejša…
    - Wound-Wait: simetrični pristop: samo mlajša transakcija lahko čaka starejšo. Če starejša zahteva zaklepanje, ki ga drži mlajša, se mlajša prekine (wounded).
  - Detekcija in odprava: sestavimo graf WFG (wait-for graph), ki nakazuje odvisnosti med transakcijami in omogoča detekcijo mrtvih zank.

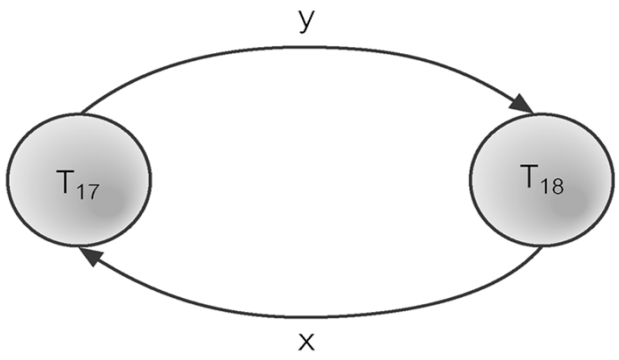
Mrtve zanke…

- WFG je usmerjen graf G = (N, E), kjer N vozlišča, E povezave.
- Postopek risanja WFG:
  - Kreiraj vozlišče za vsako transakcijo
  - Kreiraj direktno povezavo $T_i \rightarrow T_j$, če T_i čaka na zaklepanje podatkovne enote, ki je zaklenjena s strani T_j.

- Pojav mrtve zanke označuje cikel v grafu.
- SUPB gradi graf in periodično preverja obstoj mrtve zanke (iskanje ciklov).

Mrtve zanke...

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮



Mrtve zanke

- Ko je mrtva zanka detektirana, je potrebno eno ali več transakcij prekiniti.
- Pomembno:
  - Izbira transakcije za prekinitev: možni kriteriji: ‘starost’ transakcije, število sprememb, ki jih je transakcija naredila, število sprememb, ki jih transakcija še mora opraviti.
  - Kolikšen del transakcije preklicati: namesto preklica cele transakcije včasih mrtvo zanko moč rešiti s preklicem le dela transakcije.
  - Izogibanje stalno istim žrtvam: potrebno preprečiti, da ni vedno izbrana ista transakcija. Podobno živi zanki (live lock)

Časovno žigosanje kot alternativa zaklepanju

- Časovni žig: enolični identifikator, ki ga SUPB dodeli transakciji in pove relativni čas začetka transakcije.
- Časovno žigosanje: protokol nadzora sočasnosti, ki razvrsti transakcije tako, da so prve tiste, ki so starejše.
  - Alternativa zaklepanju pri reševanju sočasnega dostopa
  - Če transakcija želi brati/pisati neko podatkovno enoto, se ji to dovoli, če je bila zadnja sprememba nad to enoto narejena s starejšo transakcijo. Sicer se ponovno zažene z novim žigom.
  - Ni zaklepanj → ni mrtvih zank
  - Ni čakanja → če je transakcija v konfliktu, se ponovno zažene.
- Procesiranje časovnih žigov je za CPU mnogo zahtevnejše od zaklepanja!

Optimistične tehnike…

- Optimistične metode za nadzor sočasnosti
  - temeljijo na predpostavki, da je konfliktov malo, zato je vzporedno izvajanje dovoljeno brez kontrole, morebitne konflikte pa preverimo na kocu izvedbe.
  - Ob zaključku transakcije (commit) se preveri morebitne konflikte. Če obstaja konflikt, se transakcija razveljavi.
  - Omogočajo večjo stopnjo sočasnosti (pri predpostavki, da je konfliktov malo)

Optimistične tehnike…

- Protokoli, ki temeljijo na optimističnem pristopu, imajo tipično tri faze:
  - Faza branja: traja vse od začetka transakcije do tik pred njeno potrditvijo (commit). Preberejo se vsi podatki, ki jih transakcija potrebuje ter zapišejo v lokalne spremenljivke. Vse spremembe se izvajajo nad lokalnimi podatki.
  - Faza preverjanja: začne za fazo branja. Preveri se, ali je moč spremembe, ki so vidne lokalno, aplicirati tudi v podatkovno bazo.
    - Za transakcije, ki zgolj berejo, še enkrat preverimo, če so prebrane vrednosti še vedno iste. Če konfliktov ni, sledi potrditev, sicer zavrnitev ter ponovni zagon transakcije.
    - Za transakcije, ki podatke spreminjajo, moramo preveriti, če spremembe ohranijo konsistentnost podatkovne baze.
  - Faza pisanja: sledi fazi preverjanja. Če slednja uspešna, se podatki zapišejo v podatkovno bazo.

Optimistične tehnike

- Izvedba faze preverjanja:
  - Vsaka transakcija T ima dodeljene tri časovne žige: ob začetku – start(T), ob preverjanju – validation(T) in ob zaključku – finish(T).
  - Preverjanje je uspešno, če velja vsaj eden od pogojev:
    - Vse transakcije S s starejšim žigom se morajo končati pred začetkom T: finish(S) < start(T)
    - Če T začne preden se starejša transakcija S konča, potem:
      (a) množica podatkov, zapisanih s starejšo transakcijo, ne vključuje tistih, ki jih je trenutna transakcija prebrala.
      (b) starejša transakcija zaključi fazo pisanja preden mlajša začne s fazo preverjanja: start(T) < finish(S) < validation(T).