Poglavje 2

Sočasen dostop do podatkovne baze

- Opredelitev transakcije
- Namen nadzora sočasnosti
- Serializacija urnika transakcij
- Zaklepanje podatkov
- Mrtva zanka in njeno preprečevanje

Sočasen dostop do podatkovne baze

Kaj si bomo pogledali?

- Opredelitev transakcije
- Namen nadzora sočasnosti
- Serializacija urnika transakcij
- Zaklepanje podatkov
- Mrtva zanka in njeno preprečevanje

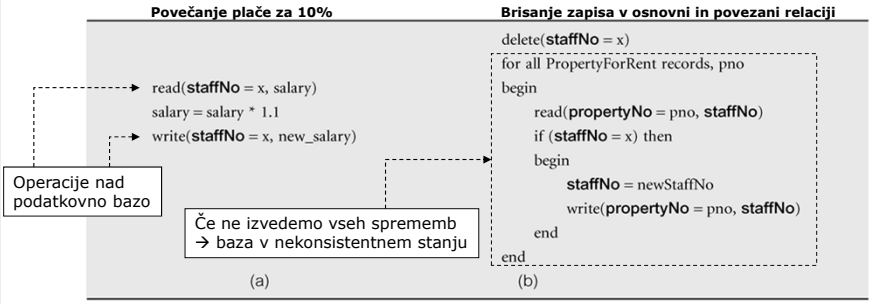
Opredelitev transakcije...

- Transakcija je operacija ali niz operacij, ki berejo ali pišejo v podatkovno bazo in so izvedene s strani enega uporabnika oziroma uporabniškega programa.
- 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...

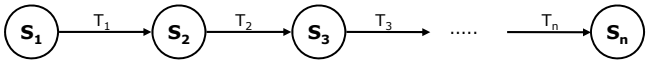
- Primeri transakcij

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



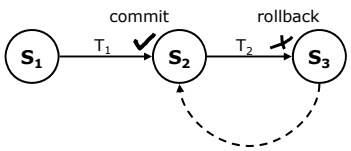
Opredelitev transakcije...

- S_i; i=1 .. n ≈ konsistentna ali skladna stanja v podatkovni bazi
- Za skladno stanje je značilno:
  - da so podatki v podatkovni bazi med seboj skladni,
  - za zadnje skladno stanje je značilno, da so podatki skladni z realnim svetom
- 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 (commited), sicer razveljavimo (aborted).
- Ob neuspešnem zaključku moramo podatkovno bazo vrniti v skladno stanje pred začetkom transakcije.

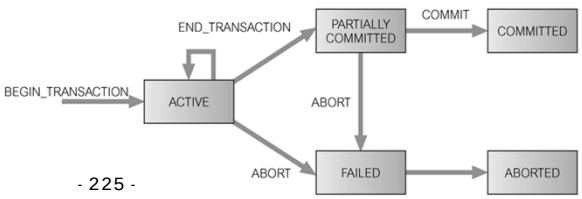


Opredelitev transakcije...

- 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.



Opredelitev transakcije...

- 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 vsebinske neskladnosti).

*__ACID__ – __A__tomicity, __C__onsistency, __I__solation and __D__urability

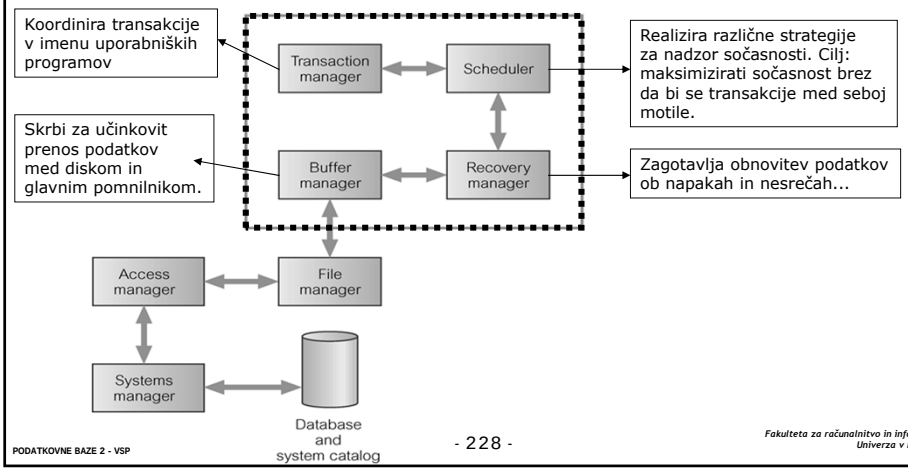
Opredelitev transakcije...

- Osnovne lastnosti transakcije (nadaljevanje)*:
  - 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:



Zakaj sočasnost?...

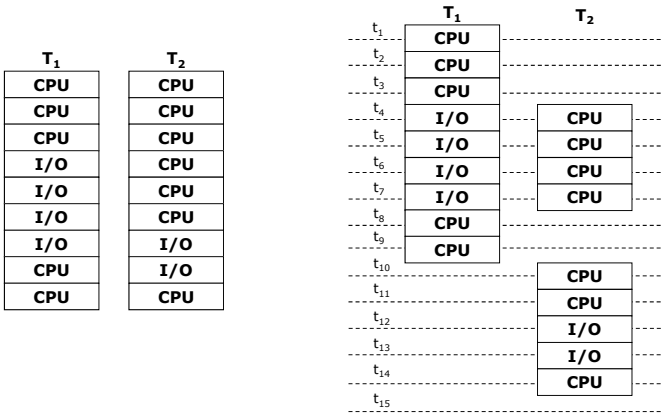
- Eden od ciljev in prednosti PB je možnost sočasnega dostopa s strani več uporabnikov do skupnih podatkov → večja učinkovitost
- Če vsi uporabniki podatke le berejo – nadzor sočasnosti trivialen;
- Če več uporabnikov sočasno dostopa do podatkov in vsaj eden podatke tudi zapisuje – možni konflikti.

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 izvajati prvo transakcijo 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

- V centraliziranem SUPB zaradi sočasnosti lahko nastopijo različni problemi:
  - Izgubljene spremembe: 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.
  - Neskladnost analize: transakcija prebere več vrednosti iz podatkovne baze. Nekatere izmed njih se v času izvajanja prve transakcije zaradi drugih transakcij spremenijo.

Primeri težav s sočasnostjo dostopa…

- Izgubljene spremembe
  - 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	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

Primeri težav s sočasnostjo dostopa…

- Uporaba nepotrjenih podatkov
  - 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, v resnici pa je 190!!!

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

Primeri težav s sočasnostjo dostopa…

- Nekonsistentna analiza
  - 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

Serializacija in obnovljivost…

- Če transakcije izvajamo zaporedno, se izognemo vsem problemom.
- Problem: nizka učinkovitost.
- Kako v največji meri uporabiti paralelnost?

- 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. (pojmi zaporedni, izmenični, zaporedniški razpored)
- S serializacijo v urnikih spreminjamo vrstni red bralno/pisalnih operacij. 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(bal_x)		read(bal_x)		read(bal_x)	
t_3	write(bal_x)		write(bal_x)		write(bal_x)	
t_4		begin_transaction		begin_transaction	read(bal_y)	
t_5		read(bal_x)		read(bal_x)	write(bal_y)	
t_6		write(bal_x)	read(bal_y)		commit	
t_7	read(bal_y)			write(bal_x)		begin_transaction
t_8	write(bal_y)		write(bal_y)			read(bal_x)
t_9	commit		commit			write(bal_x)
t_{10}		read(bal_y)		read(bal_y)		read(bal_y)
t_{11}		write(bal_y)		write(bal_y)		write(bal_y)
t_{12}		commit		commit		commit
	(a)		(b)		(c)	

Metode nadzora sočasnosti…

- Nadzor sočasnosti temelji na dveh osnovnih metodah:
  - 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.

Zaklepanje… (zaseganje podatkov)

- 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) pred pisanjem pa ekskluzivno zaklepanje (exclusive lock).

Zaklepanje…

- Zrnatost zaklepanja:
  - Zaklepanje se lahko nanaša na poljuben del podatkovne baze (od polja do cele podatkovne baze). Imenovali bomo "podatkovna enota".
- 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.

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 ni že zaklenjena, se transakciji zaklepanje odobri.
  - Če je enota že zaklenjena:
    - če je obstoječe zaklepanje deljeno, se odobri
    - 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 ali implicitno (ob prekinitvi ali potrditvi transakcije).

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

Zaklepanje…

- Kompatibilnostna matrika zaklepanja:

		Zahteva po zaklepanju		
		Ekskluzivno	**Deljeno**	**-**
Trenutno zaklepanje	**Ekskluzivno**	**NE**	**NE**	**DA**
	Deljeno	**NE**	**DA**	**DA**
	-	**DA**	**DA**	**DA**

Zaklepanje…

- Da zagotovimo serializacijo – zaporedni razpored, moramo upoštevati dodatna pravila, ki natančno opredeljujejo, kje v transakcijah so postavljena zaklepanja in kje se sprostijo.
- Protokol zagotavlja serializacijo urnika.
- 2 protokola:
  - **PXC** (Protocol eXclusive Commit): temelji na ekskluzivnem zaklepanju podatkov,
  - **PSC** (Protocol Shared Commit): temelji na ekskluzivnem in deljenem zaklepanju.

Zaklepanje…

- Pravila, ki jih uvaja PXC protokol:
  - transakcija, ki želi podatek ažurirati, ga mora najprej ekskluzivno zakleniti,
  - če zahteva po zaklepanju ne more biti takoj odobrena, preide transakcija v stanje čakanja, njeno izvajanje se nadaljuje po odobriti zaklepanja,
  - vsa zaklepanja se smejo sprostiti šele po zaključku transakcije (uspešnem ali neuspešnem),
  - Transakcija, ki želi le prebrati podatek in ji ni mar za sočasno ažuriranje tega podatka s strani kake druge transakcije, ga sme prebrati ne glede na to, ali je podatek zaklenjen ali ne.
- Dodatno vsebuje protokol PSC še naslednje pravilo:
  - Transakcija, ki želi ekskluzivno zakleniti podatek, mora imeti pred tem odobreno njegovo deljeno zaklepanje.

Zaklepanje…

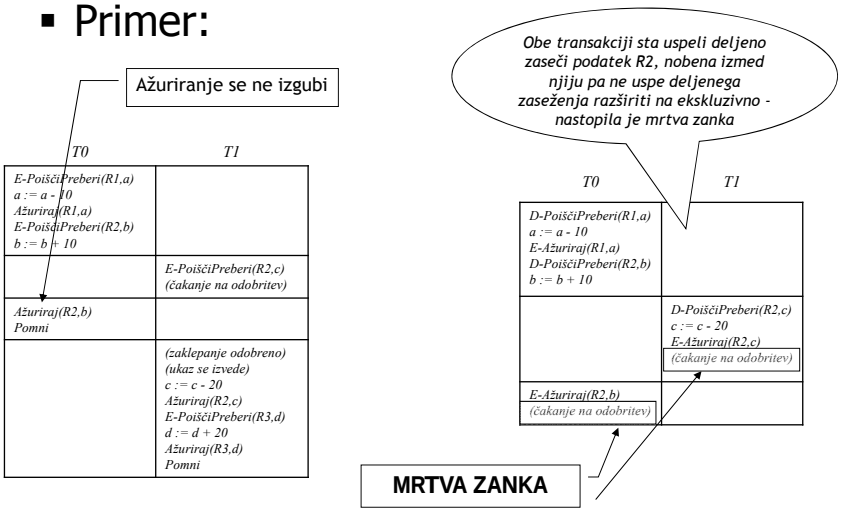
- RAZLIKE med protokoloma (obseg sočasnega izvajanja transakcij):
  - PSC dopušča sočasno izvajanje dveh zgolj povpraševalnih transakcij, PXC pa ne,
  - PSC veliko bolj dopušča nastop mrtve zanke (medsebojno blokiranje transakcij).

Zaklepanje…

- Da se protokol za zaklepanje podatkov lahko izvaja, mora SUPB za podatke voditi evidenco o tem:
  - ali podatek je ali ni zaklenjen,
  - kako je zaklenjen,
  - kdo vse ga je zaklenil,
  - kdo vse ga želi zakleniti in na kakšne način.
- Zato se za vsak podatek vzdržujeta 2 listi:
  - lista odobrenih zaklepanj,
  - lista zahtevanih zaklepanj.

Zaklepanje

- Primer:

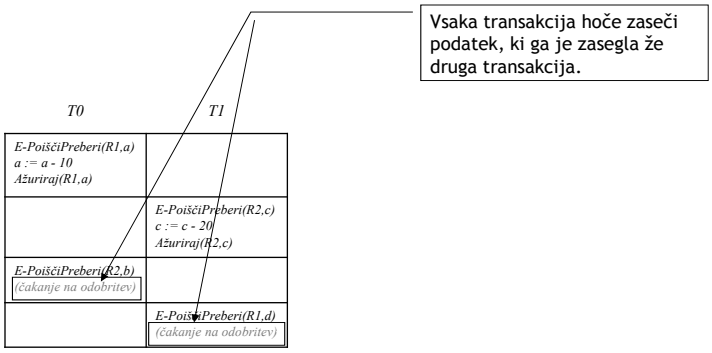


Mrtva zanka…

- Pri sočasnem izvajanju transakcij se pojavljata še naslednja problema:
  - livelock: Posamezno transakcijo lahko pri odobritvi zaklepanja določenega podatka prehitevajo vse ostale transakcije in se s tem njeno čakanje raztegne na nedoločen čas (problem je rešljiv z algoritmi za dodeljevanje zaklepanj),
  - mrtva zanka (deadlock): ko dve (ali več transakcij) zaklene vsaka svoj podatek, vsaka od njiju pa želi še podatek, ki ga je že zaklenila tekmica. Zaradi tega transakciji čakata druga na drugo, in ker se po protokolu zaklenjeni podatki sprostijo le ob zaključku transakcije, ni čakanja nikoli konec.

Mrtva zanka…

- Primer nastopa mrtve zanke:



Mrtva zanka...

- Tehnike obravnave mrtvih zank:
  - Prekinitev: po poteku določenega časa SUPB transakcijo prekliče in ponovno zažene.
  - Preprečitev: uporabimo časovne žige; dva algoritma:
    - Čakaj ali izdihni (Wait-Die): samo starejše transakcije lahko čakajo na mlajše, sicer (če je mlajša) je transakcija prekinjena (die) in ponovno pognana z istim časovnim žigom. Sčasoma postane starejša…
    - Rani ali čakaj (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.

Mrtva zanka...

Preprečevanje nastopa mrtve zanke – 2 protokola:

- Ko transakcija TA zahteva zaklepanje podatka, ki ga je že zaklenila transakcija TB, in se njeni zahtevi zaradi nekompatibilnosti zaklepanja ne da pri priči ugoditi, se zgodi naslednje:

- po protokolu **Čakaj ali izdihni (Wait-Die):**

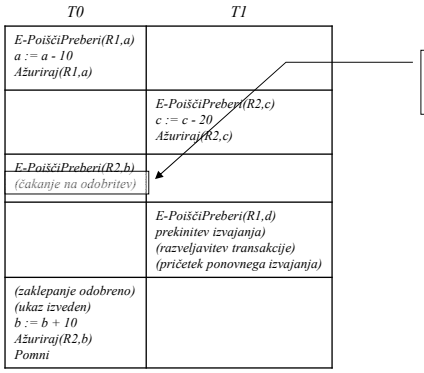
  če je transakcija TA starejša od TB, preide TA v stanje čakanja na odobritev, če je mlajša, pa se njeno izvajanje prekine, transakcija se razveljavi in posreduje transakcijskemu programu v ponovno izvajanje (z istim časovnim žigom, s čimer se zagotovi, da sčasoma postane starejša (da se je ne bi venomer prekinjalo));

- po protokolu **Rani ali čakaj (Wound-Wait)**:

  če je transakcija TA starejša od TB, se prekine, razveljavi in vrne v ponovno izvajanje transakcija TB (po njeni razveljavitvi se transakciji TA odobri zaklepanje podatka), če je TA mlajša, pa preide v stanje čakanja.

Mrtva zanka...

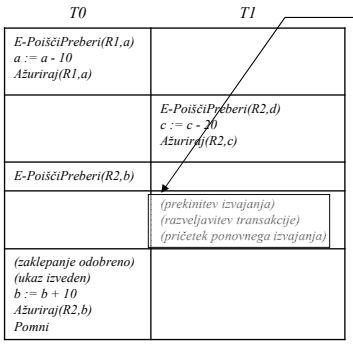
- Primer **Čakaj ali izdihni (Wait-Die):**



Protokol Čakaj ali izdihni prepreči nastop mrtve zanke. T0 je starejša, zato preide v stanje čakanja na odobritev.

Mrtva zanka...

- Primer **Rani ali čakaj (Wound-Wait)**:



Isti problem kot v prejšnjem primeru se s protokolom Rani ali čakaj izvede na naslednji način. Ker je T0 starejša od T1, se prekine, razveljavi in vrne v ponovno izvajanje transakcija T1.

Mrtva zanka…

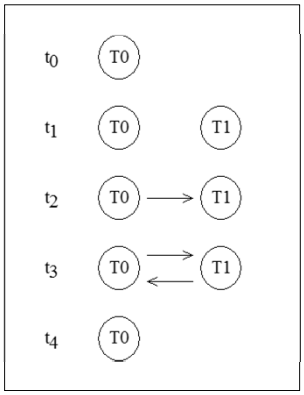
Konstrukcija čakalnega grafa:

- WFG je usmerjen graf G = (N, E), kjer N predstavlja vozlišča, E pa povezave.
- Postopek risanja WFG:
  - Kreiraj vozlišče za vsako transakcijo
  - Kreiraj direktno povezavo $T_i \rightarrow T_j$, če T_i čaka na zaklepanje podatkvne enote, ki je zaklenjena s strani T_j.
- Pojav mrtve zanke označuje cikel v grafu.
- SUPB periodično gradi graf in preverja obstoj mrtve zanke.

WFG…

- Primer:

T0	*čas*	*T1*
E-PoiščiPreberi(R1,a) *a := a - 10* *Ažuriraj(R1,a)*	*t0*	
	t1	*E-PoiščiPreberi(R2,c)* *c := c - 20* *Ažuriraj(R2,c)*
E-PoiščiPreberi(R2,b) *(čakanje na odobritev)*	*t2*	
	t3	*E-PoiščiPreberi(R1,d)* *(čakanje na odobritev)*
	(nastop mrtve zanke)	
		(prekinitev izvajanja) *(razveljavitev)* *(pričetek ponovnega izvajanja)*
(zaklepanje odobreno) *(ukaz je izveden)* *b := b + 10* *Ažuriraj(R2,b)* *Pomni*	*t4*	



WFG...

- 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.
  - Koliko transakcije preklicati: namesto preklica cele transakcije včasih mrtvo zanko moč rešiti s preklicom le dela transakcije.
  - Izogibanje stalno istim žrtvam: potrebno preprečiti, da ni vedno izbrana ista transakcija. Podobno živi zanki (live lock)