Poglavje 2

Varnost v podatkovnih bazah (predvsem relacijskih)

Trije vidiki varnosti v SUPB

Varnost spada na področje **fizičnega** načrtovanja podatkovnih baz

1. Transakcijska varnost
   - cilj: omogočanje sočasnega dela **več** uporabnikov nad **istimi** podatki
   - definicija transakcije, močna konsistentnost in ACID transakcijski model
   - implementacija ACID v relacijskih SUPB

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

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

Transakcijska varnost v SUPB

- Splošna definicija transakcije
- Lastnosti transakcij (ACID model močne konsistentnosti)
- Gradniki SUPB povezani z nadzorom sočasnosti ter obnovljivostjo podatkov
- Transakcije in SQL

Transakcija - opredelitev oz. definicija

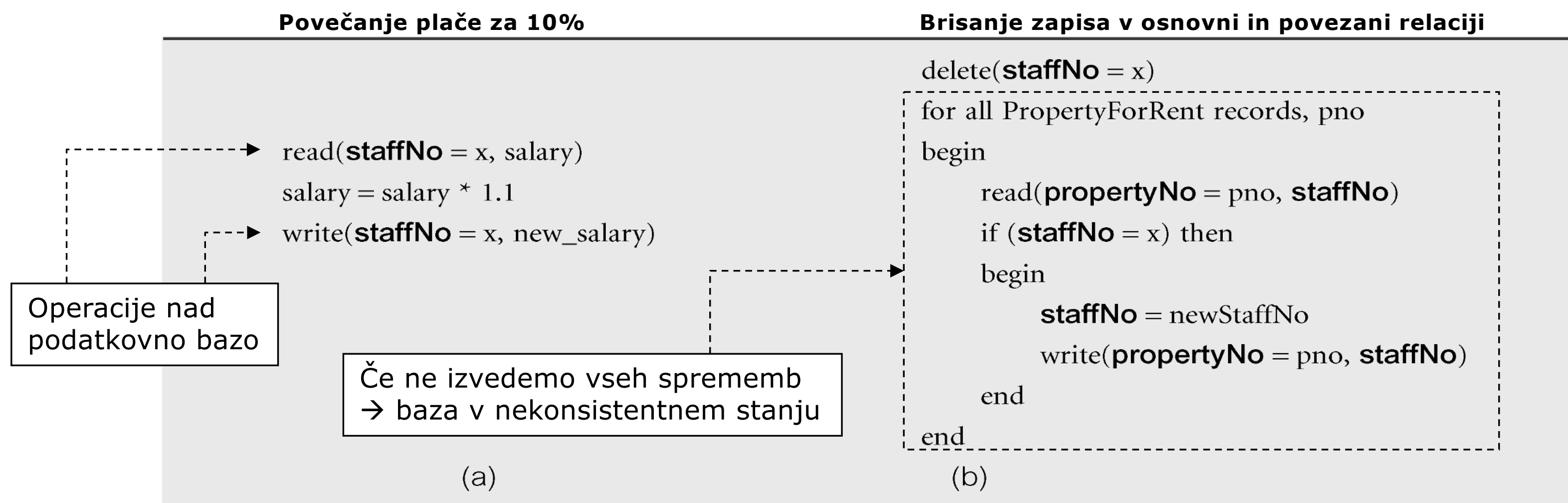
- Transakcija je operacija ali **niz operacij**, ki berejo ali pišejo v podatkovno bazo in so izvedene s strani enega uporabnika oziroma uporabniškega programa.
- Razvijalec določi, katere operacije tvorijo transakcijo (primer: bančni prenos sredstev med računi)
- Transakcija je logična enota dela – 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)

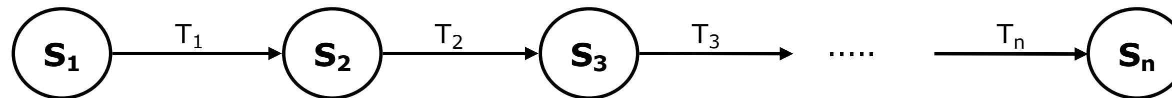
PropertyForRent(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)



Opredelitev transakcije…

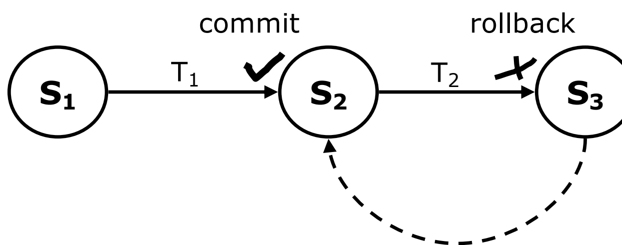
S_i; i=1 .. n ≈ konsistentna ali skladna stanja v podatkovni bazi

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 se mora podatkovna baza "vrniti" v zadnje skladno stanje pred začetkom transakcije.

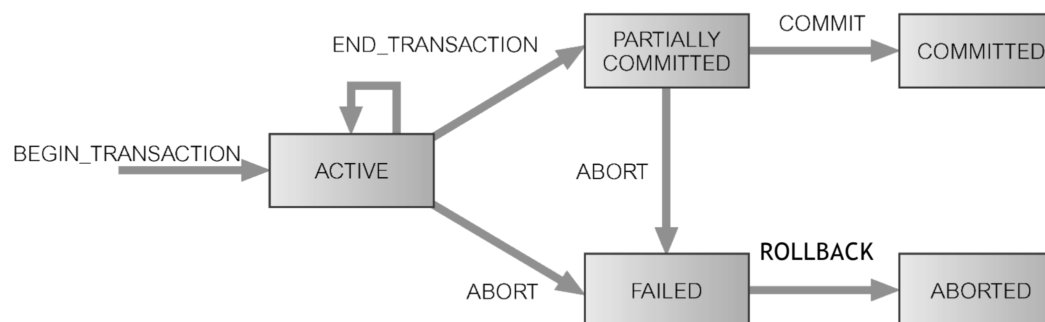


Opredelitev transakcije…

- 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 SQL 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.



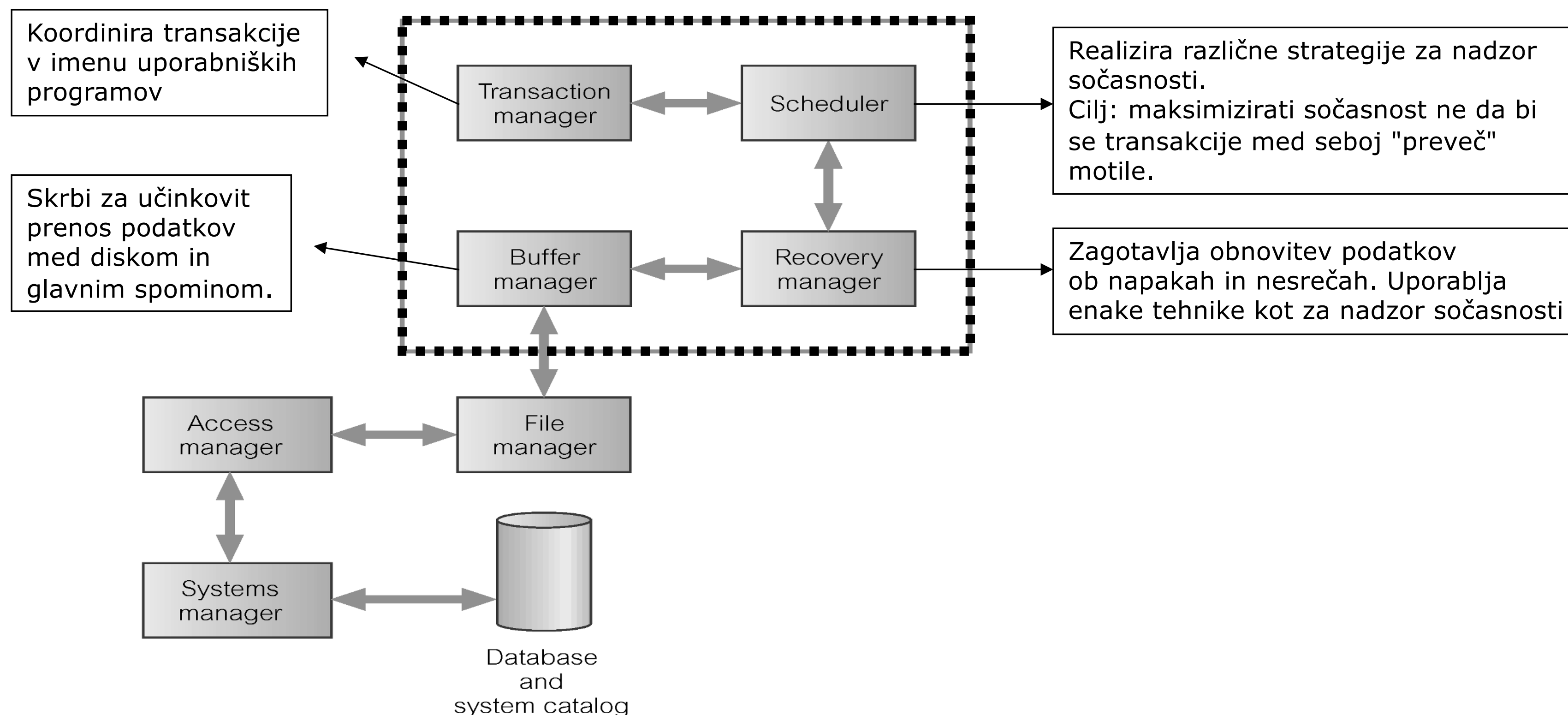
Lastnosti transakcij (ACID*)

- 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:



Zakaj sočasnost?…

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

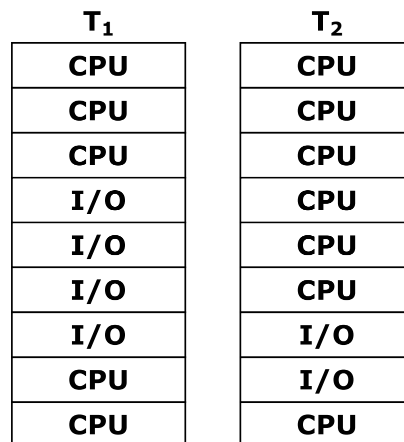
- Če vsi uporabniki podatke le berejo – nadzor sočasnosti trivialen;
- Če več uporabnikov sočasno dostopa do podatkov in vsaj eden podatke tudi zapisuje – možni konflikti, ki privedejo do *nekonsistentnosti* podatkov.

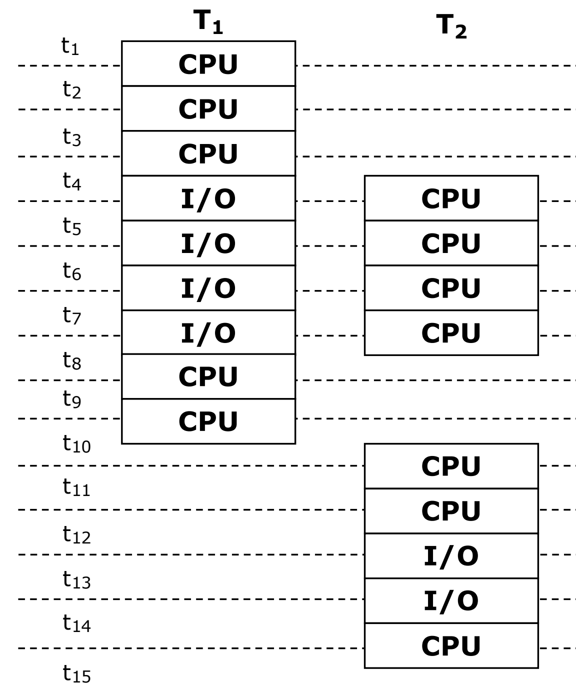
V resnici je prepletanja operacij lahko še bistveno več:

- Več procesov ali niti, ki tečejo "istočasno" (na enem jedru)
- Več procesorskih jeder istočasno

Zakaj sočasnost?

- Izraba časa med I/O operacijami
- Abstrakcija: transakcije na enem jedru
- 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 ali prepiše 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) (non-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 (latenca, particioniranje, replikacija in sinhronizacija, ...) - CAP

Izgubljene spremembe (lost update)

- Primer:
  - 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 €.
- Kaj je tu problem?

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

Uporaba nepotrjenih podatkov (dirty read)

- Primer:
  - 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	**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

Neponovljivo branje (non-repeatable read)

- Dvakrat izvedemo isti SELECT, dobimo različne vrednosti (UPDATE)
- Vrednost že prebranega podatka se na disku spremeni (UPDATE)
  - 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

Fantomsko branje (phantom read)

- Konceptualno podobno kot neponovljivo branje
- Dvakratno izvajanje iste poizvedbe znotraj transakcije
  - Lahko se pojavijo **nove vrstice**, ki izpolnjujejo pogoje za vključitev v rezultat, a jih ob prvi izvedbi ni bilo (posledica ukaza INSERT)
  - Lahko se izbrišejo **stare vrstice**, ki so izpolnjujevale pogoje za vključitev v rezultat, a jih ob drugi izvedbi ni več (posledica ukaza DELETE)
- Razlika med fantomskimi vrsticami in neponovljivim branjem: pri slednjem se med izvedbo spremenijo vrednosti v **že prebranih vrsticah** (posledica ukaza UPDATE)

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, MySQL: START TRANSACTION
  - Microsoft Transact-SQL: BEGIN TRANSACTION
  - Oracle: nima tega ukaza
  - START/BEGIN TRANSACTION implicitno izvede COMMIT predhodne transakcije
- 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 in zaključi sejo (session). Transakcija je potrjena implicitno (brez eksplicitnega COMMIT).
  - Program, znotraj katerega se izvaja transakcija, se ne konča uspešno. Transakcija se implicitno razveljavi (brez eksplicitnega 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.

- Ukaz velja za naslednjo transakcijo (MySQL) ali za tekočo transakcijo (PostgreSQL, Oracle) neposredno ob začetku

Nastavljanje lastnosti za več kot eno transakcijo

- Različno od sistema do sistema, ni po ISO SQL standardu
- Oracle (za tekočo sejo):
  ALTER SESSION SET TRANSACTION ...
- MySQL (za tekočo sejo ali globalno z ustreznimi pravicami):
  SET [GLOBAL | SESSION] TRANSACTION ...
- PostgreSQL:
  SET SESSION CHARACTERISTICS AS TRANSACTION ...
  - Nadaljnja sintaksa je enaka kot pri SET TRANSACTION ...
- V aplikaciji nam lahko pomaga knjižnica (npr. pri SQLAlchemy lahko pri `create_engine` navedemo parameter `isolation_level`)

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 s strani drugih transakcij 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(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)	

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

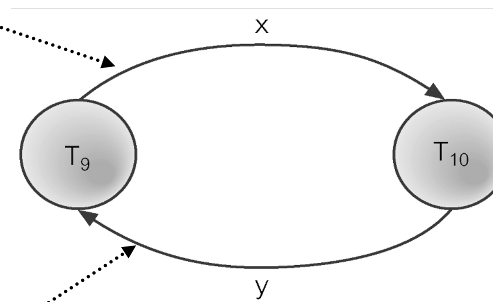
  G = (N, E); N → vozlišča, E → povezave
- Gradnja grafa:
  - Kreiraj vozlišče za vsako transakcijo
  - Kreiraj usmerjeno povezavo $T_i \rightarrow T_j$, če T_j bere vrednost, predhodno zapisano s T_i
  - Kreiraj usmerjeno povezavo $T_i \rightarrow T_j$, če T_j piše vrednost, ki je bila predhodno prebrana 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
  – Pogosto implementirano kot *Multiversion Concurrency Control* (MVCC)
  – Vsaka transakcija vzdržuje ločen pogled na podatke (kopijo)
  – Morebitne konflikte preverimo na koncu izvedbe in posodobimo podatke v bazi.

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/bralno zaklepanje (shared lock, read lock)
  - Transakcija mora pred pisanjem podatka zahtevati ekskluzivno/pisalno 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 (bralno pisalno), če to ni v neskladju z drugimi transakcijami
- 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: MariaDB/MySQL nudi več različnih tipov datotečne organizacije
  - 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 zaklepanja.

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	$\mathbf{bal_x}$
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

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

Reševanje nepono

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(bal_x)		
t_3	read(bal_x)		
t_4	read_lock(bal_y)		
t_5	read(bal_y)		
t_6	$bal_x = bal_y + bal_x$		
t_7	write(bal_x)		
t_8	unlock(bal_x)	begin_transaction	
t_9	⋮	write_lock(bal_x)	
t_{10}	⋮	read(bal_x)	
t_{11}	⋮	$bal_x = bal_x + 100$	
t_{12}	⋮	write(bal_x)	
t_{13}	⋮	unlock(bal_x)	
t_{14}	⋮	⋮	
t_{15}	rollback	⋮	
t_{16}		⋮	begin_transaction
t_{17}		⋮	read_lock(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(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(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.

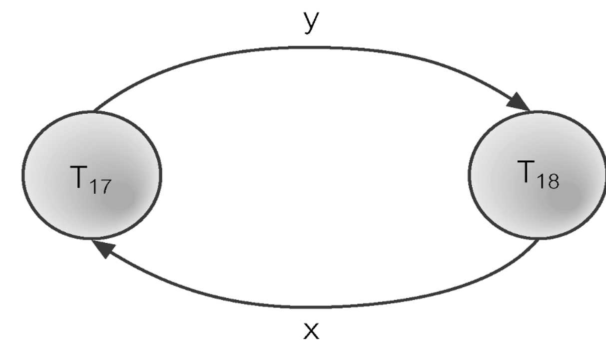
Mrtve zanke…

- 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…

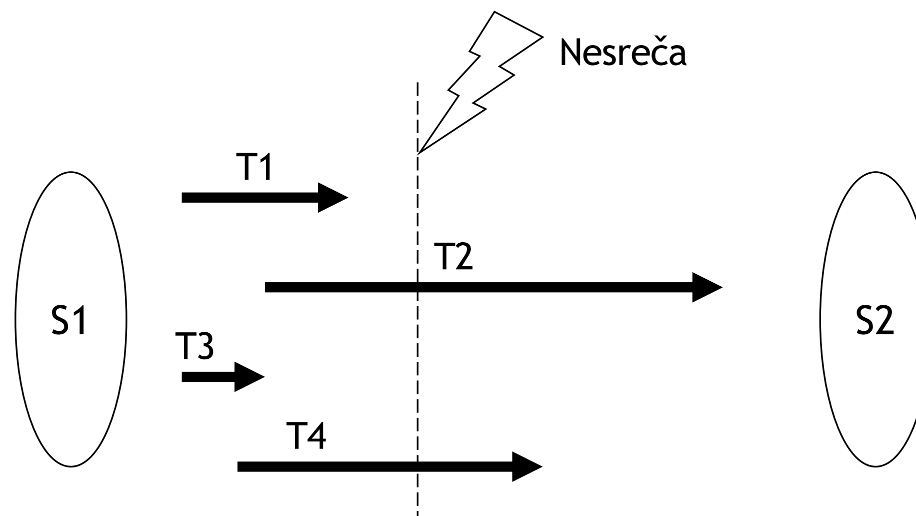
- Protokoli, ki temeljijo na optimističnem pristopu, imajo tipično tri faze:
  - Faza branja: traja vse od začetka transakcije do tik pred njeno potrditvijo (commit). Preberejo se vsi podatki, ki jih transakcija potrebuje ter zapišejo v lokalne spremenljivke. Vse spremembe se izvajajo nad lokalnimi podatki.
  - 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).

Podatkovna varnost in obnovljivost

- Obnovljivost – zmožnost obnove podatkov po nesreči.
- Proces vzpostavljanja podatkovne baze v zadnje veljavno stanje, ki je veljalo pred nastopom nesreče.



Potreba po obnovljivosti…

- Shranjevanje podatkov se običajno navezuje na štiri različne tipe medijev za shranjevanje podatkov, z naraščajočo stopnjo zanesljivosti:
  - glavni pomnilnik (neobstojni pomnilnik): podatki v njem ne preživijo sistemskih nesreč,
  - magnetni ali SSD disk ("online" obstojni pomnilnik): zanesljivejši in cenejši od glavnega pomnilnika, vendar tudi počasnejši,
  - magnetni trak ("offline" obstojni pomnilnik): še zanesljivejši in cenejši od diska, vendar tudi počasnejši, omogoča samo zaporedni dostop (tudi do 100 TB/trak)
  - optični disk: najzanesljivejši od vseh, še cenejši od traku, hitrejši od traku, omogoča neposredni dostop do podatkov (do 100 GB/disk).

Potreba po obnovljivosti…

- Obstaja več vrst nesreč, od katerih je potrebno vsako obravnavati na drugačen način.

- Nesreča lahko prizadane podatke tako v glavnem, kot v sekundarnem pomnilniku.

Potreba po obnovljivosti…

- Vzroki za nesreče:
  - odpoved sistema: zaradi napak v strojni ali programski opremi; posledica je izguba podatkov v glavnem pomnilniku,
  - poškodbe medija: zaradi trka glave diska ob magnetno površino postane medij neberljiv; posledica so neberljivi deli sekundarnega pomnilnika,
  - programska napaka v aplikaciji: zaradi logične napake v programu, ki dostopa do podatkov v PB, pride do napak v eni ali več transakcijah,
  - neprevidnost: zaradi nenamernega uničenja podatkov s strani administratorjev ali uporabnikov,
  - sabotaža (namerno oviranje dela): zaradi namernega popačenja ali uničenja podatkov, uničenja programske ali strojne opreme.

Potreba po obnovljivosti

- Ne glede na vrsto napake, vedno smo pri nesrečah soočeni z dvema bistvenima problemoma:
  - izguba podatkov v glavnem pomnilniku (vključno s podatki v medpomnilniku),
  - izguba podatkov na sekundarnem pomnilniku.

- V nadaljevanju:
  - pregled tehnik za lajšanje posledic nesreče in
  - tehnike za obnavljanje po nesreči.

Transakcije in obnovljivost…

- Transakcija predstavlja osnovno enoto obnovljivosti.
- Za obnovljivost skrbi upravljavec za obnovljivost (recovery manager), ki mora v primeru nesreče zagotavljati dve od štirih lastnosti transakcij (ACID):
  - atomarnost in
  - trajnost.

Transakcije in obnovljivost…

- Naloga upravitelja obnovljivosti je, da pri obnovitvi PB po nesreči zagotovi:
  - da se vse spremembe, ki so bile v PB izvedene v okviru posamične transakcije uveljavijo v celoti ali pa
  - da se ne uveljavi nobena sprememba.

- Problem je kompleksen, ker pisanje v PB ne predstavlja atomarne akcije → transakcija lahko izvede COMMIT (uveljavitev sprememb), vendar se spremembe v PB ne zabeležijo, ker enostavno ne "dosežejo" PB (nastop nesreče).

Transakcije in obnovljivost…

- Podatkovni vmesniki so področje v glavnem pomnilniku, v katerega se pri prenašanju podatkov iz/v sekundarnega pomnilnika podatki pišejo ali iz njega berejo.
- Prenos vsebine podatkovnih vmesnikov v sekundarni pomnilnik (trajne spremembe) se sproži samo v primeru izvedbe posebnih ukazov:
  - COMMIT ali
  - avtomatično, ko postanejo podatkovni vmesniki polno zasedeni (eksplicitno zapisovanje vsebine podatkovnih vmesnikov v sekundarni pomnilnik označujemo kot prisilno zapisovanje (force-writing)).

Transakcije in obnovljivost…

- Če se nesreča pripeti med pisanjem v pod. vmesnike ali med prenosom podatkov iz pod. vmesnikov v sek. pomnilnik, mora upravitelj za obnovljivost ugotoviti status transakcije, ki je izvajala pisanje v času nesreče:
  - če je transakcija izvedla ukaz COMMIT, mora upravitelj za obnovljivost zaradi zagotavljanja lastnosti trajnosti zagotoviti ponovno izvajanje transakcije (Rollforward ali Redo),
  - če transakcija ni izvedla ukaza COMMIT, mora upravitelj za obnovljivost zaradi zagotavljanja lastnosti atomarnosti izvesti razveljavljanje posodobitev, ki jih je do tedaj transakcija izvedla (Rollback ali Undo).

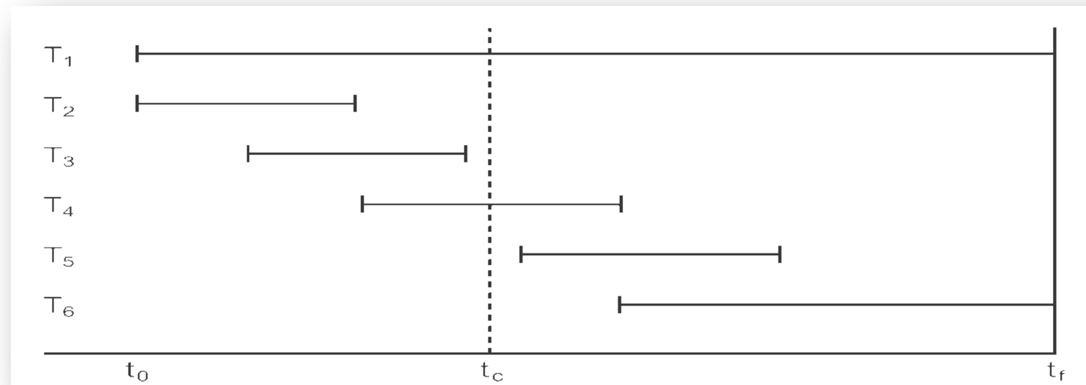
Transakcije in obnovljivost…

- Če je potrebno razveljaviti samo eno transakcijo govorimo o parcialnem razveljavljanju (partial undo). Ta se izvaja tudi pri sočasnem dostopanju do podatkov zaradi uporabe protokolov za nadzor sočasnosti.

- Če je potrebno razveljaviti vse v času nesreče aktivne transakcije, govorimo o globalni razveljavitvi (global undo).

PRIMER: uporaba UNDO/REDO

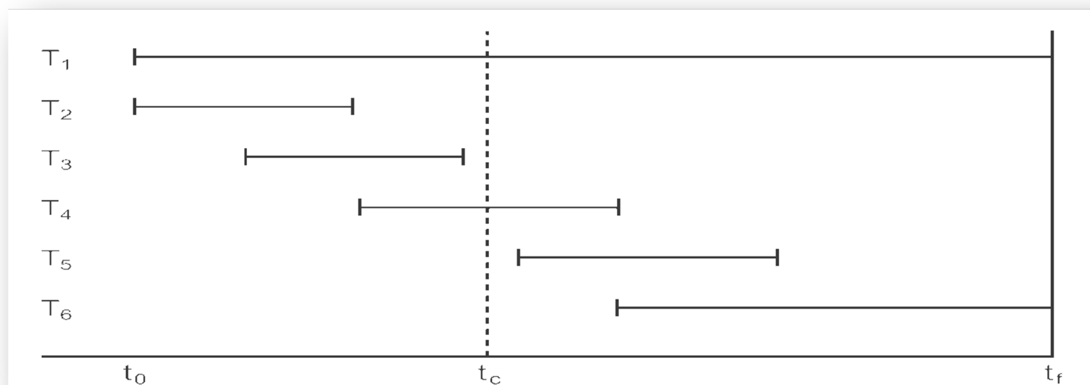
- Transakcije T_1 do T_6 se izvajajo sočasno, SUPB začne delovati ob t_0, t_c je kontrolna točka, nesreča pa nastopi ob t_f:



- T_2 in T_3 izvedeta COMMIT in spremembe se uveljavijo v PB.
- Kontrolna točka: točka sinhronizacije med PB in dnevnikom. Izvede se zahteva po zapisu vseh podatkovnih vmesnikov na disk.

PRIMER: uporaba UNDO/REDO

- T_1 in T_6 ne izvedeta ukaza COMMIT do trenutka nesreče, zato jih upravitelj za obnavljanje pri ponovnem zagonu razveljavi (UNDO).
- Za T_4 in T_5 ni jasno, do katere mere so se njune spremembe uveljavile v PB - ali je bila vsebina podatkovnih vmesnikov zapisana v sekundarni pomnilnik ali ne.
  - Ker nimamo na razpolago nobene dodatne informacije o stanju transakcij, je upravitelj za obnavljanje prisiljen ponoviti (REDO) transakcije T_4 in T_5.



- V okviru SUPB so za obnavljanje podatkov po nesreči na voljo naslednje komponente:
  - mehanizem za izdelavo varnostnih kopij, ki periodično kreira kopije PB,
  - dnevnik, ki hrani podatke o trenutnem stanju transakcij in spremembah v PB,
  - mehanizem za izvajanje kontrolnih točk, ki omogoča da se posodobitve, ki jih izvajajo transakcije v PB, ohranijo (zahteva po zapisu vsega na disk),
  - upravljalec za obnovljivost, komponenta SUPB, ki omogoča obnoviti podatkovno bazo v zadnje konsistentno stanje, ki je veljalo pred nastopom nesreče.

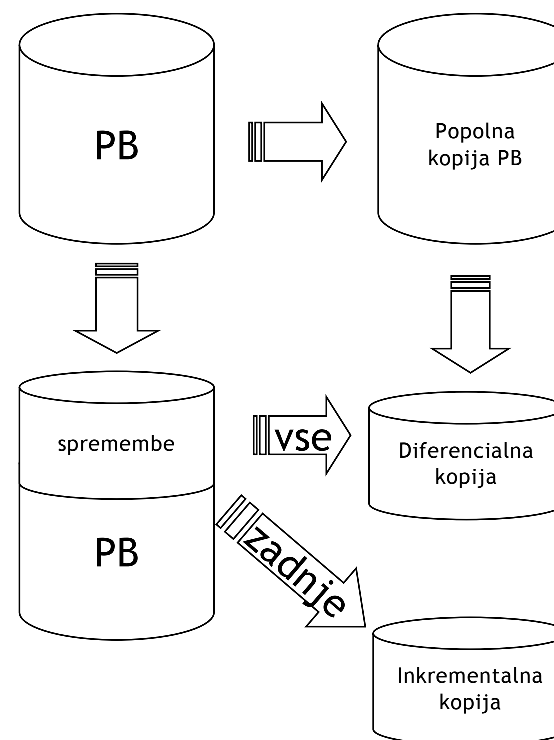
Mehanizem za izdelavo varnostnih kopij…

- Mehanizem mora omogočati izdelavo varnostnih kopij PB in dnevnika v določenih intervalih, ne da bi pred tem bilo potrebno prekiniti delovanje PB .

- Kopijo PB se uporabi v primeru poškodb PB ali njenega uničenja.
- Varnostna kopija se običajno hrani na magnetnem traku.

Mehanizem za izdelavo varnostnih kopij

- Varnostna kopija je lahko:
  - popolna kopija PB
    ali
  - inkrementalna kopija, ki vsebuje samo spremembe izvedene od zadnje (popolne ali inkrementalne) kopije PB
    ali
  - diferencialna kopija, ki vsebuje vse spremembe izvedene od zadnje popolne kopije PB.



Dnevnik…

- V dnevnik se zapisujejo vse spremembe, ki jih transakcije izvedejo v PB.
- Dnevnik lahko vsebuje naslednje podatke:
  - transakcijske zapise, kjer je dnevniški zapis sestavljen iz:
    - identifikatorja transakcije,
    - tipa dnevniškega vpisa (začetek tr., insert, update, detele, abort, commit),
    - identifikator podatka, na katerega se nanaša operacija (operacije: insert, delete, update) v okviru transakcije,
    - predhodna vrednost podatka: vrednost podatka pred ažuriranjem (samo za operacije update in delete),
    - vrednost podatka po ažuriranju (samo za operacije insert in update),
    - podatki potrebni za upravljanje dnevnika: kazalec na prejšnji in naslednji dnevniški zapis, ki pripada določeni transakciji.
  - zapise kontrolnih točk.

Dnevnik (semantičen prikaz)

- Primer segmenta dnevniške datoteke, ki prikazuje tri sočasne transakcije T_1, T_2 in T_3. Stolpca pPtr in nPtr predstavljata kazalce na predhodni in naslednji dnevniški vpis transakcije.

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

Dnevnik…

- Zaradi pomembne vloge dnevnika pri obnavljanju podatkov po nesrečah, je ta podvojen ali celo potrojen.
- Princip 3-2-1 (3 kopije dnevnika, 2 medija, 1 na drugi lokaciji)
- Včasih je bil dnevnik shranjen izključno na magnetnem traku (zanesljivejši in cenejši).
- Danes se pričakuje, da je SUPB pri manjših nesrečah sposoben hitro obnoviti PB v stanje pred nesrečo. To zahteva, da se vsaj zadnji del dnevnika hrani v pomnilniku in/ali disku.

Dnevnik…

- V okoljih, kjer se v dnevnike piše/spreminja velika količina podatkov (reda stotine GB dnevno), vseh podatkov dnevnika ni smiselno imeti ves čas na razpolago (v pomnilniku)
- V pomnilniku morajo biti na razpolago podatki za hitro obnavljanje:
  - manjše nesreče (npr. razveljavitev transakcije, ki bi lahko povzročila mrtvo zanko), možno je hitro obnavljanje iz pomnilnika.
  - večje nesreče (npr. udarec glave diska v magnetno površino) zahtevajo več časa za obnovitev in navadno večjo količino podatkov iz dnevnika. V takih primerih gre za prenos podatkov iz magnetnega traku.

Mehanizem za izvajanje kontrolnih točk…

- Podatki iz dnevnika se rabijo za obnovitev PB po nesreči → pri obnavljanju ne vemo, koliko dnevniških vpisov je potrebno prebrati, ne da bi ponavljali transakcije, ki so se v PB že uspešno uveljavile.

- Količino presežnega iskanja in procesiranja zaradi pregledovanja dnevniških vpisov lahko omejimo z uporabo kontrolnih točk.

Mehanizem za izvajanje kontrolnih točk…

- Kontrolna točka: točka sinhronizacije med PB in dnevnikom. Izvede se zahteva po izpisu vseh podatkovnih vmesnikov na disk.
  - Tako smo prepričani, da so bile transakcije, ki so bile zaključene pred izpisom vmesnikov, zanesljivo uveljavljene ali razveljavljene v PB na disku.

Mehanizem za izvajanje kontrolnih točk…

- Kontrolne točke se izvajajo po vnaprej določenem urniku in vključujejo naslednje operacije:
  - zapisovanje vseh dnevniških vpisov iz glavnega pomnilnika v sekundarni pomnilnik (neposredno iz pomnilnika na disk!!!),
  - zapisovanje posodobljenih blokov v podatkovnih vmesnikih v sekundarni pomnilnik,
  - zapisovanje zapisa kontrolne točke v dnevnik. Ta zapis vključuje identifikatorje vseh transakcij, ki so bile v času izvedbe kontrolne točke aktivne.
- Če se transakcije izvajajo zaporedno, potem je v dnevniku potrebno poiskati zadnjo transakcijo, ki se je pričela izvajati pred izvedbo zadnje kontrolne točke.

Mehanizem za izvajanje kontrolnih točk...

- Vse transakcije pred prej omenjeno transakcijo so že bile uveljavljene (commited) in njihove spremembe zapisane v PB v času izvedbe kontrolne točke.

- Zaradi tega je potrebno ponoviti samo transakcijo, ki je bila aktivna v času izvedbe kontrolne točke in vse transakcije, ki so ji sledile, katerih zapisi se nahajajo v dnevniku.

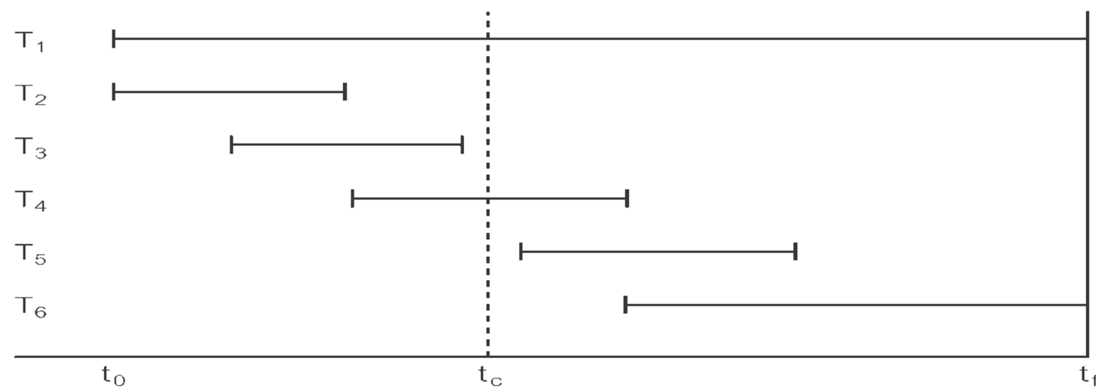
Mehanizem za izvajanje kontrolnih točk...

- Če je bila transakcija aktivna v trenutku nastopa nesreče, jo je potrebno razveljaviti.

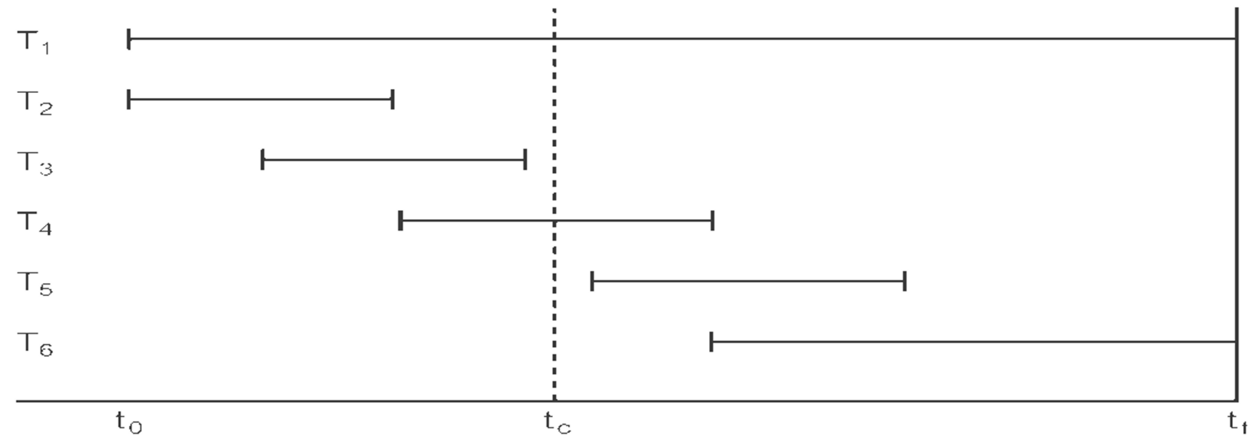
- Če se transakcije izvajajo sočasno, je potrebno ponoviti vse transakcije, ki so izdale ukaz COMMIT od zadnje kontrolne točke naprej in razveljaviti vse transakcije, ki so bile aktivne v času nastopa nesreče.

UNDO in REDO s kontrolno točko

- Predpostavimo, da se je v času t_c izvedla kontrolna točka:



UNDO in REDO s kontrolno točko



- Spremembe transakcij T_2 in T_3 se zapišejo v sekundarni pomnilnik.
- Upravitelju za obnavljanje ni potrebno ponoviti izvajanja transakcij T_2 in T_3.
- Transakciji T_4 in T_5 je potrebno ponoviti (REDO), ker sta ukaze COMMIT izdali po izvedbi kontrolne točke.
- Transakciji T_1 in T_6 pa je potrebno razveljaviti (UNDO), ker sta bili v času nastopa nesreče aktivni.

Mehanizem za izvajanje kontrolnih točk

- V splošnem je uporaba kontrolnih točk relativno poceni operacija.
- Ponavadi je mogoče izvesti najmanj 3 do 4 kontrolne točke na uro.
- Na ta način je možno zagotoviti, da bo potrebno obnoviti samo podatke iz zadnjih 15-20 minut obratovanja PB.
- Primer: PostgreSQL, privzeti interval 5 minut (torej 12 kontrolnih točk na uro)

Primer uporabe dnevnika

- UNDO nazaj po času
- REDO naprej po času

	Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
1	T1	10:12	START				0	2
2	T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
3	T2	10:14	START				0	4
4	T2	10:16	INSERT	STAFF SG37		(new value)	3	5
5	T2	10:17	DELETE	STAFF SA9	(old value)		4	6
6	T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
7	T3	10:18	START				0	11
8	T1	10:18	COMMIT				2	0
9		10:19	CHECKPOINT	T2, T3				
10	T2	10:19	COMMIT				6	0
11	T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T_f 12	T3	10:21	COMMIT				11	0

Tehnike obnovljivosti…

- Uporaba posamezne procedure za obnavljanje podatkov v PB po nesreči je odvisna od obsega nastale škode. Razlikujemo dva primera:
- Obsežne poškodbe PB:
  - vzrok: npr. diskovna nesreča.
  - posledica nesreče: uničena podatkovna baza.
  - podatke se obnovi z uporabo kopije PB in dnevnika; podatki iz dnevnika služijo za ponovitev (redo) uveljavljenih transakcij.
  - ta način obnavljanja predvideva, da dnevnik ni bil poškodovan; dnevnik naj se torej nahaja na drugem mediju, ločeno od podatkovnih datotek.

Kontrolna točka (CHECKPOINT) ni shranjena točka (SAVEPOINT)!

- SAVEPOINT: možnost za delno razveljavitev transakcije
- Imenovanih shranjenih točk je lahko več znotraj transakcije
- ISO SQL, Oracle, PostgreSQL, MySQL/MariaDB:
  - Kreiranje: SAVEPOINT ime_tocke;
  - Uporaba: ROLLBACK TO ime_tocke; --
  - Brisanje: RELEASE SAVEPOINT ime_tocke; -- Oracle ne podpira
- Uspešen COMMIT briše vse aktivne kontrolne točke
- Pozor: razlike med sistemi v detajlih izvajanja!

```
BEGIN;
    INSERT INTO t VALUES (1);
    SAVEPOINT my_savepoint1;
    INSERT INTO t VALUES (2);
    SAVEPOINT my_savepoint2;
    …
    ROLLBACK TO SAVEPOINT
              my_savepoint1;
    INSERT INTO t VALUES (3);
COMMIT;
```

Tehnike obnovljivosti…

- Manjše poškodbe; PB ni fizično poškodovana:
  - vzrok: odpoved sistema med izvajanjem transakcij.
  - posledica nesreče: PB preide v neveljavno – nekonsistentno stanje.
  - transakcije, ki so se prekinile je potrebno razveljaviti, ker so postavile PB v nekonsistentno stanje.
  - lahko se tudi zgodi, da je nekatere transakcije potrebno ponoviti, če njihove spremembe niso "dosegle" sekundarnega pomnilnika.
  - v tem primeru za obnavljanje ne potrebujemo kopije PB, ampak zadostujejo predhodne in posodobljene vrednosti podatkov, ki se nahajajo v dnevniških vpisih (glej primer izseka iz dnevnika).

Tehnike obnovljivosti…

- V nadaljevanju: predstavitev dveh tehnik za obnavljanje PB po nesrečah, ki ne povzročijo uničenja PB, ampak privedejo PB v nekonsistentno stanje:
  - odloženo ažuriranje in
  - sprotno ažuriranje.

Tehnike obnovljivosti…

- Tehnike obnovljivosti podatkov po nesrečah, ki privedejo PB v nekonsistentno stanje:
  - odloženo ažuriranje (WRITE-AHEAD-LOG, WAL),
  - sprotno ažuriranje.
- Odloženo in sprotno ažuriranje se ločita po načinu zapisovanja posodobljenih podatkov v PB, obe pa uporabljata dnevnik.
- Obnovitvene tehnike morajo biti za uporabnika transparentne!

Odloženo ažuriranje…

- Pri protokolu za odloženo ažuriranje se podatki (posodobljeni) ne zapisujejo neposredno v PB.
- Vsa ažuriranja v okviru transakcije se najprej shranijo v dnevnik. Pri uspešnem zaključku transakcije se izvede dejansko ažuriranje PB.
- V primeru nesreče:
  - če se transakcija prekine, v PB ni potrebno razveljaviti nobene spremembe, ker se te nahajajo samo v dnevniku,
  - pred nesrečo uspešno zaključene transakcije je potrebno ponoviti (redo), ker se njihova ažuriranja lahko še niso dejansko zapisala v PB. V tem primeru se uporabi zapise shranjene v dnevniku.

Sprotno ažuriranje…

- Pri uporabi protokola sprotnega ažuriranja transakcija izvaja neposredno spreminjanje podatkov v PB še preden se uspešno zaključi.
- V primeru nesreče je poleg ponavljanja (redo) uspešno zaključenih transakcij, potrebno razveljaviti (rollback) vse transakcije, ki so bile aktivne v času nesreče.

Primerjava odloženega in sprotnega ažuriranja

- Z vidika učinkovitosti:
  - Odloženo ažuriranje je učinkovitejše, če se v povprečju izvede več neuspešnih transakcij (ni potrebno spreminjati PB).
  - Sprotno ažuriranje je učinkovitejše, če se v povprečju izvede več uspešnih transakcij (ni potrebno veliko popravljati podatkov v PB).

Obnavljanje v MySQL

- Orodja temeljijo na ukazni vrstici in funkcionalnosti operacijskega sistema (npr. cron za periodične kopije)
- Varnostna kopija (kreiranje in obnova):
  - fizična: kopiranje v/iz datotečnega sistema; logična: program `mysqldump`
  - mysqlbackup (enterprise, plačljivo)
- Dnevnik (Point-in-Time Recovery)
  - Omogočeni morajo biti binarni dnevniki
    https://dev.mysql.com/doc/internals/en/binary-log.html
    `--log-bin =<log_prefix>` (mysqld) ali `log-bin=<log_prefix>` (my.ini)
  - Obnavljanje: `mysqlbinlog <log_name> | mysql -u root –p`
    (pretvori vsebino dnevnikov v SQL in ga posreduje mysql klientu)
  - Pozor: binarni dnevnik hitro raste, zato so potrebne pogoste kopije (npr. dnevno ali tedensko)

Dostopna varnost v SUPB

- Ena od pomembnih nalog SUPB je zagotoviti varnost dostopa do podatkovne baze.
- Večina današnjih SUPB omogoča eno ali obe od naslednjih možnosti:
  - Subjektivno določen nadzor dostopa (Discretionary access control)
  - Obvezen nadzor dostopa (Mandatory access control)

Nadzor dostopa...

- Subjektivno določen nadzor dostopa:
  - Vsak uporabnik ima določene dostopne pravice (privilegije) nad dostopom do objektov podatkovne baze.
  - Tipično uporabnik pravice dobi v povezavi z lastništvom, ko kreira objekt.
  - Pravice lahko posreduje drugim uporabnikom na osnovi lastne presoje.
  - Tak način nadzora je relativno tvegan.

Nadzor dostopa

- Obvezen nadzor dostopa:
  - vsak objekt podatkovne baze ima določeno stopnjo zaupnosti (npr. zaupno, strogo zaupno,...),
  - vsak subjekt (uporabnik, program) potrebuje za delo z objektom določeno raven zaupanja (clearance level).
  - Za različne operacije (branje, pisanje, kreiranje,...) nad objekti podatkovne baze lahko subjekti potrebujejo različne nivoje zaupanja
  - Ravni zaupanja so strogo urejene
  - Značilno za varovana okolja, npr. vojska
  - Eden znanih modelov takega nadzora v obliki končnega avtomata je Bell-LaPadula in nadgradnje (npr. IBM DB2, Oracle)

Nadzor dostopa in SQL...

- Vsak uporabnik podatkovne baze ima dodeljeno določeno pooblastilo - avtorizacijo (authorisation), ki mu ga dodeli skrbnik podatkovne baze (DBA).
- Pooblastilo je obenem tudi identifikator uporabnika.
- Navadno se za pooblastilo uporablja uporabniško ime ter geslo.
- SQL omogoča preverjanje pooblastila, s čimer identificira uporabnika.

Nadzor dostopa in SQL...

- Vsak SQL stavek, ki ga SUPB izvede, se izvede na zahtevo določenega uporabnika.

- Preden SUPB SQL stavek izvede, preveri dostopne pravice uporabnika nad objekti, na katere se SQL nanaša.

Nadzor dostopa in SQL...

- Vsak objekt, ki ga z SQL-om kreiramo, mora imeti lastnika.
- Vsak objekt se kreira v določeni shemi.
- Lastnika identificiramo na osnovi pooblastila, ki je določeno v shemi, kateri objekt pripada, in sicer v sklopu AUTHORIZATION
  - Oracle: ime sheme je enako uporabniškemu imenu
  - MySQL in PostgreSQL: isti uporabnik je lahko lastnik več shem

Nadzor dostopa in SQL...

- Dostopne pravice ali privilegiji določajo, kakšne operacije so uporabniku dovoljene nad določenim objektom podatkovne baze.
- SQL standard pozna naslednje osnovne pravice:
  - SELECT – pravica branja podatkov
  - INSERT – pravica dodajanja podatkov
  - UPDATE – pravica spreminjanja podatkov (ne pa tudi brisanja)
  - DELETE – pravica brisanja podatkov
  - REFERENCES – pravica sklicevanja na stolpce določene tabela v omejitvah (npr. tuji ključi)
  - USAGE – pravica uporabe domen, sinonimov, znakovnih nizov in drugih posebnih objektov podatkovne baze
- Oracle – več ko 200 pravic v 25 skupinah:

  http://psoug.org/definition/GRANT.htm

Nadzor dostopa in SQL...

- Pravice v zvezi z dodajanjem (INSERT) in spreminjanjem (UPDATE) tabel ali pogledov so lahko določene na ravni stolpcev tabele/pogleda.

- Enako velja za pravice sklicevanja (REFERENCES)

Nadzor dostopa in SQL...

- Ko uporabnik kreira tabelo s CREATE TABLE avtomatsko postane lastnik tabele z vsemi pravicami.

- Ostalim uporabnikom dodeli pravice z ukazom GRANT.

Nadzor dostopa in SQL...

- Ko uporabnik kreira pogled s CREATE VIEW avtomatsko postane njegov lastnik, ne dobi pa nujno vseh pravic nad njim.
- Za kreiranje pogleda potrebuje SELECT pravice nad tabelami, iz katerih sestavlja pogled, ter REFERENCES pravice nad tabelami, katerih stolpce uporablja v definiciji omejitev.
- Ob kreiranju pogleda dobi pravice INSERT, UPDATE in DELETE, če te pravice ima nad vsemi tabelami, ki jih pogled zajema.

Nadzor dostopa in SQL...

- Uporaba ukaza GRANT

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- PrivilegeList – je sestavljen iz ene ali več pravic, ločenih z vejico (INSERT, UPDATE,...)
- ALL PRIVILEGES – dodeli vse pravice.

Nadzor dostopa in SQL...

- PUBLIC – omogoča dodelitev pravic vsem trenutnim in bodočim uporabnikom.
- ObjectName – se nanaša na osnovno tabelo, pogled, domeno, znakovni niz, dodelitve in prevedbe.
- WITH GRANT OPTION – dovoljuje, da uporabnik naprej dodeljuje pravice.

Nadzor dostopa in SQL...

- Vloge: definiranje skupin privilegijev
- Nekatere definirane vnaprej (npr. dba)
- Uporabniško definirane vloge

```
-- Skupina privilegijev
CREATE ROLE Student;
GRANT priv1, priv2, … TO Student;
-- Podeljevanje skupine privilegijev uporabniku
GRANT Student TO PBB123456;
```

Primer dodeljevanja pravic...

- Uporabniku Janezu dodaj vse pravice nad tabelo rezervacija.

```
GRANT ALL PRIVILEGES
ON rezervacija
TO Janez WITH GRANT OPTION;
```

Primer dodeljevanja pravic

- Uporabnikoma Petru in Pavlu dodeli SELECT in UPDATE pravice nad stolpcem cid v tabeli rezervacija.

```
GRANT SELECT, UPDATE (cid)
ON rezervacija
TO Peter, Pavel;
```

Nadzor dostopa in SQL...

- Z ukazom REVOKE pravice odvzamemo

```
REVOKE [GRANT OPTION FOR]
{PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
[RESTRICT | CASCADE]
```

Nadzor dostopa in SQL...

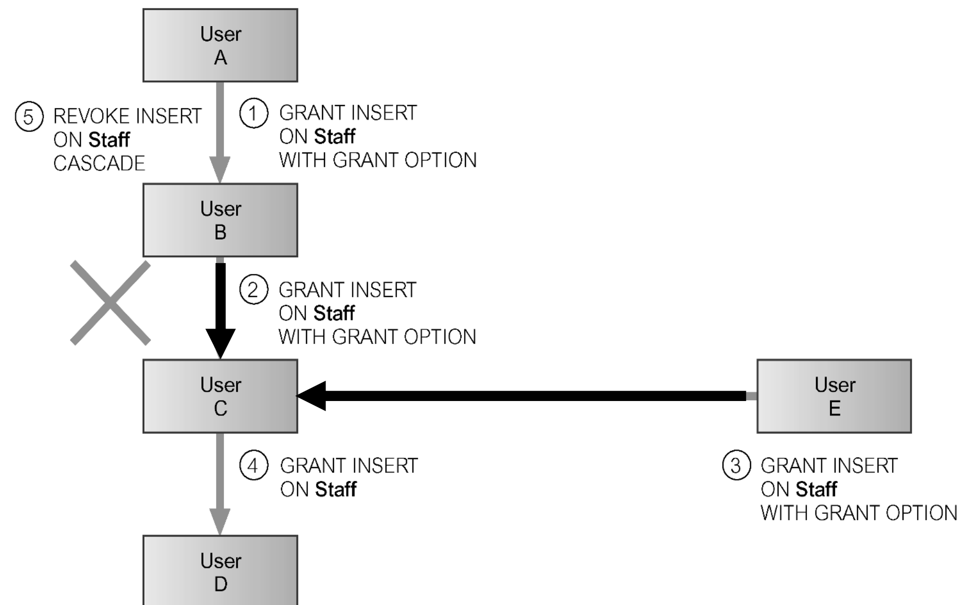
- ALL PRIVILEGES določa vse pravice, ki jih je uporabnik, ki REVOKE uporabi, dodelil uporabniku ali uporabnikom, na katere se REVOKE nanaša.
- GRANT OPTION FOR – omogoča, da se pravice, ki so bile dodeljene prek opcije WITH GRANT OPTION ukaza GRANT, odvzema posebej in ne kaskadno.
- RESTRICT, CASCADE – enako kot pri ukazu DROP

Nadzor dostopa in SQL...

- REVOKE ukaz ne uspe, kadar SUPB ugotovi, da bi njegova izvedba povzročila zapuščenost objektov:
  - Za kreiranje določenih objektov so lahko potrebne pravice. Če take pravice odstranimo, lahko dobimo zapuščene objekte.
  - Če uporabimo opcijo CASCADE, bo REVOKE ukaz uspel tudi v primeru, da privede do zapuščenih objektov. Kot posledica bodo ti ukinjeni.

Nadzor dostopa in SQL...

- Če uporabnik U_a odvzema pravice uporabniku U_b potem pravice, ki so bile uporabniku U_b dodeljene s strani drugih uporabnikov, ne bodo odvzete.



Primer odvzemanja pravic...

- Odvzemi DELETE pravice nad tabelo rezervacija vsem uporabnikom.

```
REVOKE DELETE
ON rezervacija
FROM PUBLIC;
```

Primer odvzemanja pravic

- Uporabniku Tinetu odvzemi vse pravice na tabelo rezervacija.

```
REVOKE ALL PRIVILEGES
ON rezervacija
FROM Tine;
```