Poglavje X

Varnost v SUPB

Trije vidiki varnosti v SUPB

1. Dostopna
   - kdo sme dostopati do podatkovne baze
   - kdo sme kaj delati s katerimi podatki
2. Transakcijska
   - kaj so transakcije
   - omogočanje istočasnega dela več uporabnikov nad istimi podatki
3. Podatkovna
   - celovita skrb za varnost podatkov v podatkovni bazi
   - mehanizmi za obnavljanje podatkov po nesrečah

Dostopna varnost

Nadzor dostopa...

- 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 izboljšave (Biba, Clark-Wilson)

Nadzor dostopa in SQL...

- Subjektivno določen nadzor dostopa
- 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 (PostgreSQL), implicitno (MySQL) ali avtomatsko (Oracle)

Nadzor dostopa in SQL...

- PostgreSQL (eksplicitno):
  - CREATE SCHEMA sandbox AUTHORIZATION pb;
  - CREATE SCHEMA AUTHORIZATION test; -- istoimenska shema
- MySQL (implicitno):
  - CREATE SCHEMA sandbox;
  - GRANT ALL PRIVILEGES ON sandbox.* TO pb;
- Oracle (avtomatsko):
  - Sheme vezane na uporabnike (uporabnik = shema)
  - Ob kreiranju uporabnika se avtomatsko ustvari istoimenska shema z vsemi privilegiji
  -

Nadzor dostopa in SQL...

- Dostopne pravice ali privilegiji določajo, kakšne operacije so uporabniku dovoljene nad določenim objektom podatkovne baze.
- SQL pozna naslednje standardne 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

Nadzor dostopa in SQL...

- Nekatere nad-standardne pravice:
  - TRUNCATE – hitro brisanje vsebine tabele
  - CREATE – kreiranje novih shem (nad bazo) ali objektov (nad shemo). Objekti so tabele, pogledi, indeksi, …
  - TRIGGER – kreiranje prožilcev nad tabelo
  - TEMPORARY – kreiranje začasnih tabel
  - EXECUTE – dovoljuje uporabo podprograma
  - CONNECT – dovoljuje uporabo baze (povezava na bazo pri prijavi v SUPB)

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...

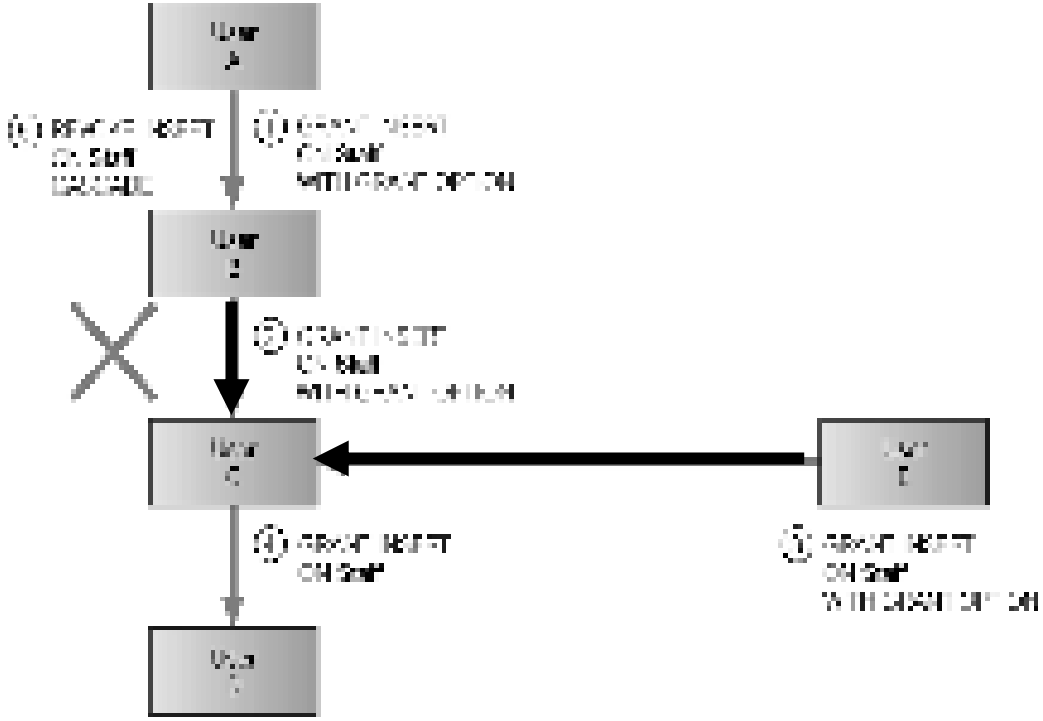
- ALL PRIVILEGES določa vse pravice, ki jih je uporabnik, ki REVOKE uporabi, dodelil uporabniku ali uporabnikom, na katere se REVOKE nanaša.
- GRANT OPTION FOR – omogoča, da se pravice, ki so bile dodeljene prek opcije WITH GRANT OPTION ukaza GRANT, odvzema posebej in ne kaskadno.
- RESTRICT, CASCADE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

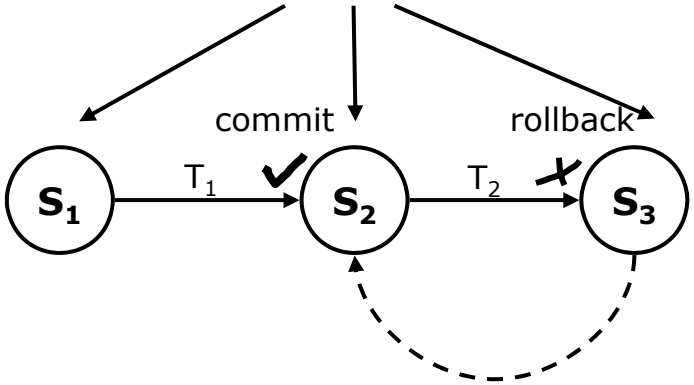
Transakcijska varnost

Transakcijska varnost

- 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 nedeljiva logična delovna enota – 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...

- Transakcija se lahko zaključi na dva načina: uspešno ali neuspešno
- Če končana uspešno, jo potrdimo (commited, operacija COMMIT), sicer razveljavimo (aborted, operacija ROLLBACK).
- Ob neuspešnem zaključku moramo podatkovno bazo vrniti v skladno stanje pred začetkom transakcije.

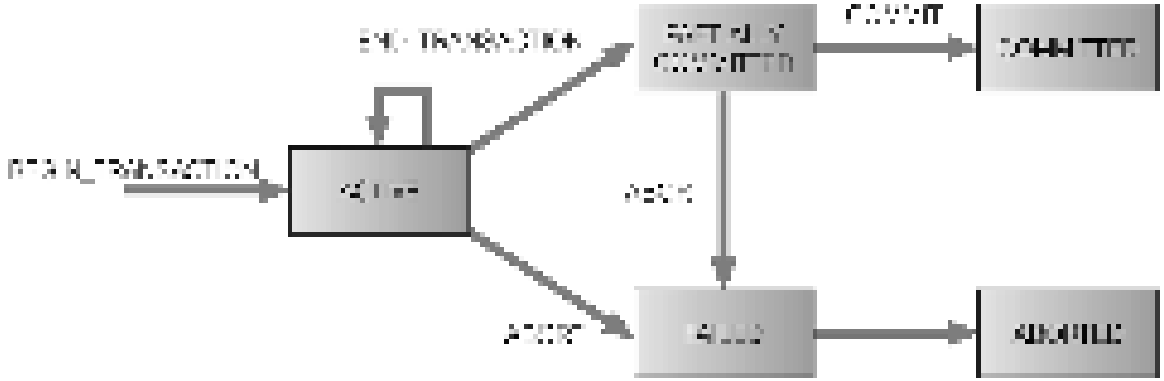


Opredelitev transakcije...

- 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…

- 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).

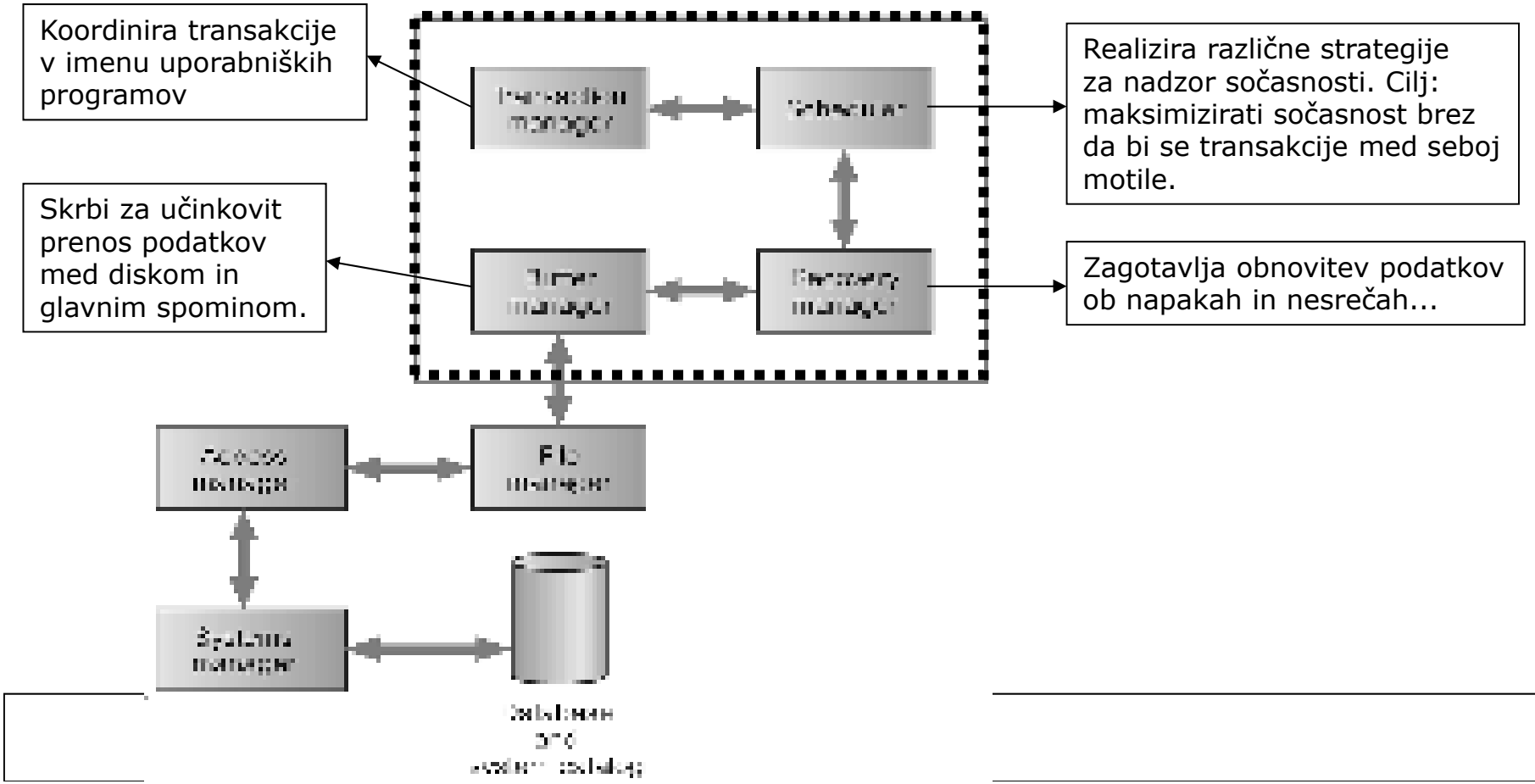
Lastnosti transakcij

- 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:



Nadzor sočasnosti

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

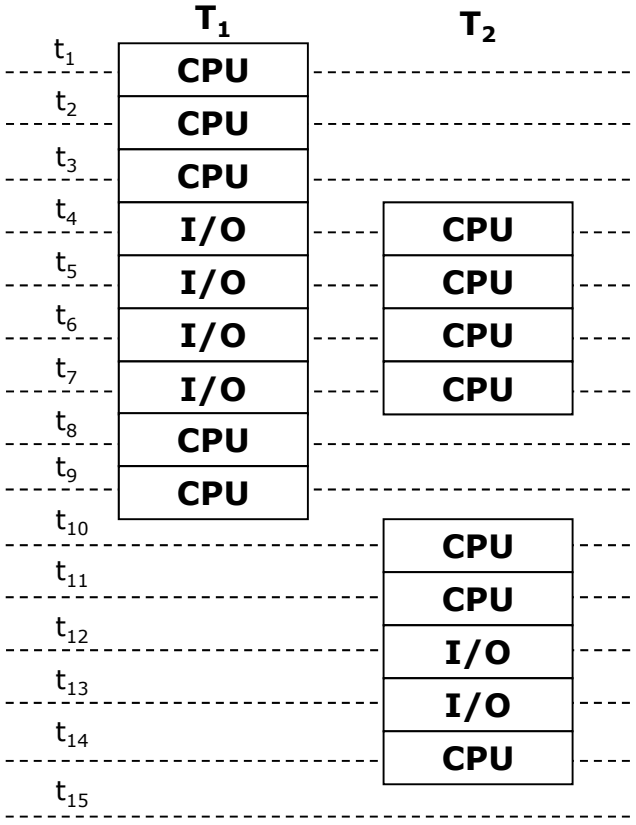
Nadzor sočasnosti

- 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...

T_1	T_2
CPU	CPU
CPU	CPU
CPU	CPU
I/O	CPU
I/O	CPU
I/O	CPU
I/O	I/O
CPU	I/O
CPU	CPU



Problemi v zvezi z nadzorom sočasnosti...

- V centraliziranem SUPB zaradi sočasnosti dostopa lahko pride do različnih problemov:
  - 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.

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...

- Neskladna 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	35	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 vzporednost izvajanja?

- Nekaj definicij:
- Serializacija:
  - način, kako identificirati vzporedne 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:
  - Če dve transakciji bereta isti podatek, nista v konfliktu. Vrstni red ni pomemben.
  - Če dve transakciji bereta ali pišeta popolnoma ločene podatke, nista v konfliktu. Vrstni red ni pomemben.
  - 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_5	T_6	T_7	T_8	T_9	T_{10}
t_1	begin_transaction		begin_transaction		begin_transaction	
t_2	read(bal_x)		read(bal_x)		read(bal_y)	
t_3	write(bal_x)		write(bal_x)		write(bal_y)	
t_4		begin_transaction		begin_transaction	read(bal_y)	
t_5		read(bal_x)		read(bal_y)	write(bal_y)	
t_6		write(bal_x)	read(bal_y)		commit	
t_7	read(bal_y)			write(bal_y)		begin_transaction
t_8	write(bal_y)		write(bal_y)			read(bal_y)
t_9	commit		commit			write(bal_z)
t_{10}		read(bal_y)		read(bal_y)		read(bal_z)
t_{11}		write(bal_y)		write(bal_y)		write(bal_z)
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.

Metode nadzora sočasnosti

- Metode za nadzor sočasnosti delimo na:
  - Pesimistične: v primeru, da bi lahko prišlo do konfliktov, se izvajanje ene ali več transakcij zadrži in
  - Optimistične: izhajamo iz predpostavke, da je konfliktov malo, zato dovolimo vzporedno izvajanje, za konflikte pa preverimo na kocu izvedbe.

- V nadaljevanju: zaklepanje in časovno žigosanje (pesimistični metodi) ter primer optimistične metode.

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) 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.

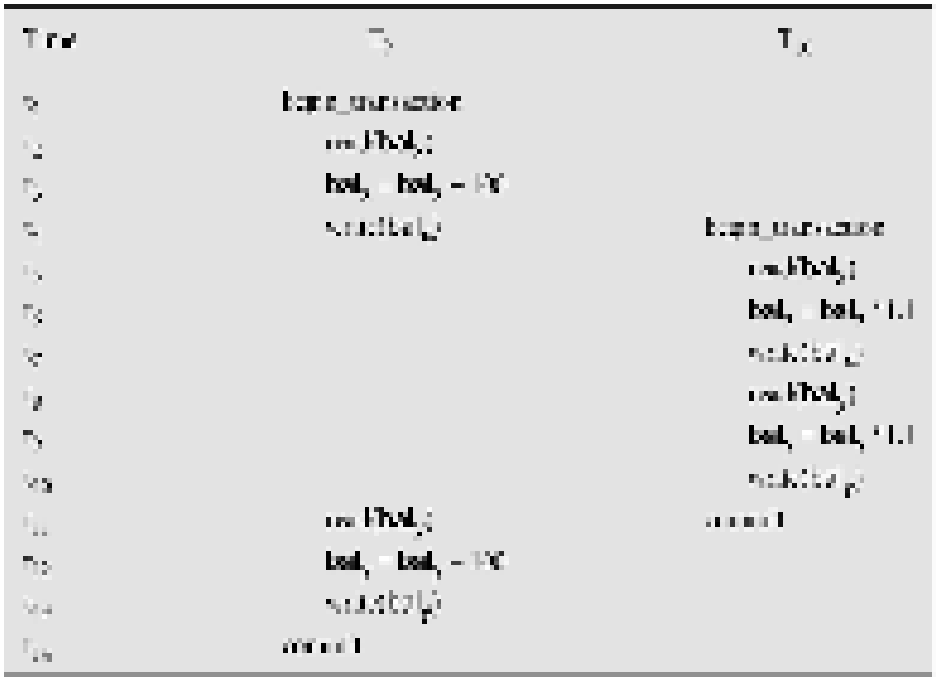
Zaklepanje...

- 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...

- Primer:



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) }

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

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 zasedanja: transakcija pridobija zaklepanja, vendar nobenega ne sprosti in
  - fazo sproščanja: transakcija sprošča zaklepanja, vendar ne more več pridobiti novega zaklepanja.
- 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

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)	$bal_y = bal_y + 10$
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…

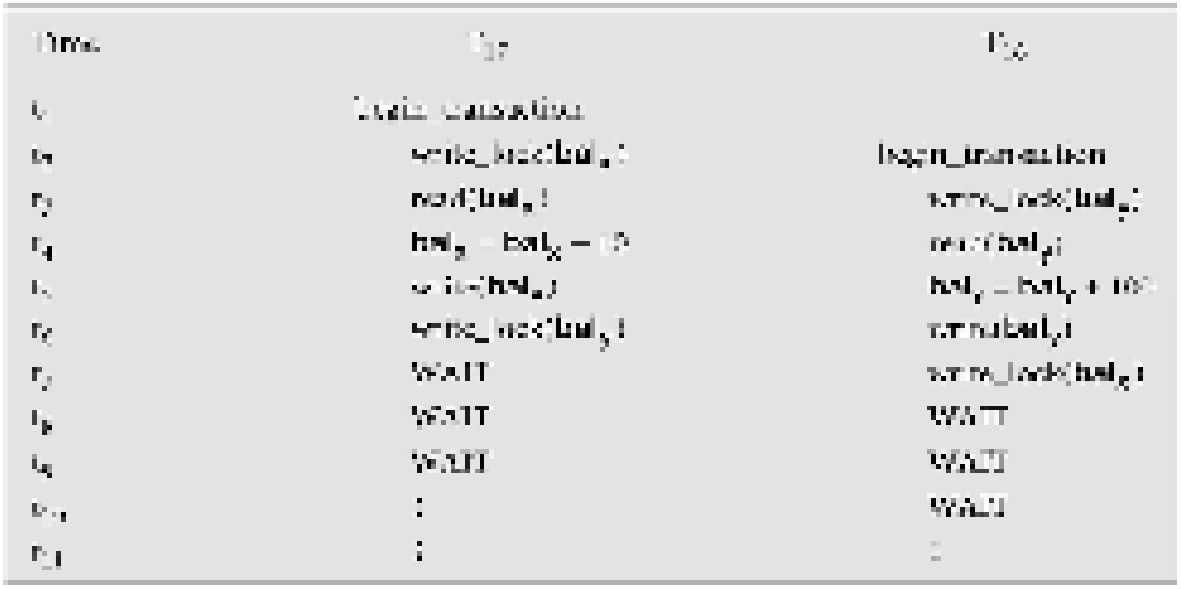
- Tehnike obravnave mrtvih zank:
  - Prekinitev: po poteku določenega časa SUPB transakcijo prekliče in ponovno zažene.
  - Preprečitev: uporabimo časovne žige; dva algoritma:
    - Wait-Die: samo starejše transakcije lahko čakajo na mlajše, sicer transakcija prekinjena (die) in ponovno pognana z istim 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.

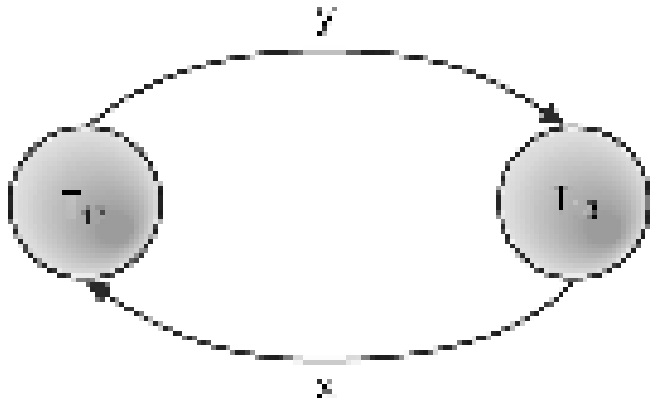
Mrtve zanke…

- WFG je usmerjen graf
- 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 periodično gradi graf in preverja obstoj mrtve zanke.

Mrtve zanke...





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.
  - 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)

Časovno žigosanje...

- Časovni žig
  - enolični identifikator, ki ga SUPB dodeli transakciji in pove relativni čas začetka transakcije.
- Časovni žig imajo tudi podatkovne enote
  - Read_timestamp: časovni žig transakcije, ki je podatkovno polje nazadnje prebrala,
  - Write_timestamp: časovni žig transakcije, ki je v podatkovno polje nazadnje pisala.

Časovno žigosanje...

- 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 restarta z novim žigom.
  - Ni zaklepanj → ni mrtvih zank
  - Ni čakanja → če transakcija v konfliktu, se restarta.

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 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.

Transakcije v SQL

- SUPB zagotavlja lastnosti ACID
- 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

- SQL definira transakcijski model z ukazoma COMMIT in ROLLBACK
- Nova transakcija se začne takoj po koncu prejšnje
- Autocommit način: vsak SQL ukaz je transakcija (privzeto pri MySQL)
- BEGIN ali START TRANSACTION – začne transakcijo (in do konca transakcije izklopi autocommit način).

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 COMMITa).
  - Program, znotraj katerega se izvaja transakcija, se ne konča uspešno. Transakcija se implicitno razveljavi (brez ROLLBACKa).

Transakcije v SQL

- Nova transakcija se začne z novim SQL stavkom, ki transakcijo začne.
- SQL transakcij ne moremo gnezditi.
- Transakcijo nastavimo 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.

Zakaj različne stopnje izolacije?

- Višja stopnja izolacije pomeni manjšo sočasnost (vzporednost) izvajanja transakcij
  1. READ UNCOMMITTED
  2. READ COMMITTED
  3. REPEATABLE READ
  4. SERIALIZABLE
- V praksi skušamo izbrati najnižjo stopnjo izolacije, ki nam zagotavlja pravilno delovanje
- Običajno je privzeta stopnja REPEATABLE READ

Transakcije v SQL

- Učinek SET TRANSACTION ISOLATION LEVEL

	Branje neobstoječega podatka	Nekonsistentna analiza	Fantomsko branje	Izgubljeno ažuriranje
Read Uncommitted	D	D	D	D
Read Committed	N	D	D	D
Repeatable Read	N	N	D	N
Serializable	N	N	N	N

(fantomsko branje je sorodno nekonsistentni analizi: ob kasnejši izvedbi neke poizvedbe znotraj transakcije se pojavijo **nove** vrstice, ki jih ob prejšnjih izvedbah iste poizvedbe znotraj iste transakcije ni bilo)

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**

Takojšnje in zakasnele 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 vsakem SQL ukazu;
  - INITIALLY DEFERRED – ob zaključku transakcije.
- Če izberemo INITIALLY IMMEDIATE (privzeta možnost), lahko določimo tudi, ali je zakasnitev moč določiti kasneje. Uporabimo [NOT] DEFERRABLE.
- MySQL ne implementira zakasnjenih omejitev

Takojšnje in zakasnele omejitve

- Način upoštevanja omejitev za trenutno transakcijo nastavimo z ukazom SET CONSTRAINTS.

```
SET CONSTRAINTS
{ALL | constraintName [, . . . ]}
{DEFERRED | IMMEDIATE}
```

- Omejitve, na katere SET CONSTRAINTS vpliva:
  - DEFERRABLE INITIALLY DEFERRED ali
  - DEFERRABLE INITIALLY IMMEDIATE