Poglavje IX

Varnost v SUPB

Trije vidiki varnosti v SUPB

1. Transakcijska varnost
   - omogočanje istočasnega dela več uporabnikov nad istimi podatki
   - transakcije – definicija in uporaba
2. Dostopna varnost (pravice)
   - kdo sme dostopati do podatkovne baze
   - kdo sme kaj delati s katerimi podatki
3. Podatkovna varnost (nesreče, obnavljanje)
   - celovita skrb za varnost podatkov v podatkovni bazi
   - mehanizmi za obnavljanje podatkov po nesrečah
   - tesna povezava z mehanizmi transakcijske varnosti

Transakcijska varnost

Bančna transakcija **stanje X = 1000** **stanje Y = 2000**
`s1=preberi_stanje_z_računa(X)`
`s1=s1-100 EUR`
`zapiši_stanje_na_račun(X, s1)`
`s2=preberi_stanje_z_računa(Y)`
`s2=s2+100 EUR`
`zapiši_stanje_na_račun(Y, s2)`

Simulacija transakcij (dva prostovoljca)

Scenarij 1: dva računa

	Bančna transakcija **stanje X = 1000** **stanje Y = 2000**
T1	`s1=preberi_stanje_z_računa(X)`
	`s1=s1-100 EUR`
	`zapiši_stanje_na_račun(X, s1)`
T2	`s2=preberi_stanje_z_računa(Y)`
	`s2=s2+100 EUR`
	`zapiši_stanje_na_račun(Y, s2)`

Scenarij 2: en račun

	Bančna transakcija **stanje X = 1000**
T1	`s1=preberi_stanje_z_računa(X)`
	`s1=s1-100 EUR`
	`zapiši_stanje_na_račun(X, s1)`
T2	`s2=preberi_stanje_z_računa(X)`
	`s2=s2+100 EUR`
	`zapiši_stanje_na_račun(X, s2)`

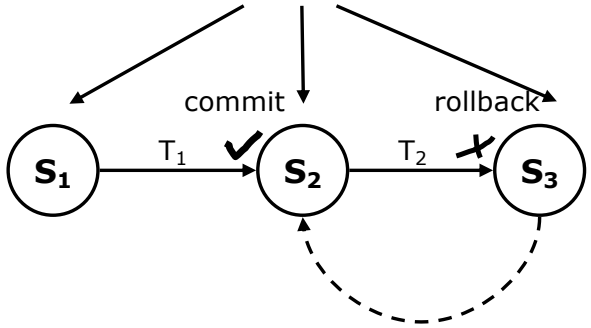
Za vsak scenarij: (a) zaporedno izvajanje (b) vzporedno izvajanje

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, zaporedje nekaj ukazov ali samostojen ukaz (npr. INSERT ali UPDATE)
- Izvedba uporabniškega programa je s stališča podatkovne baze vidna kot ena ali več transakcij.

Opredelitev transakcije…

- "Življenje" baze kot prehodi med skladnimi stanji podatkov v bazi
- 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.

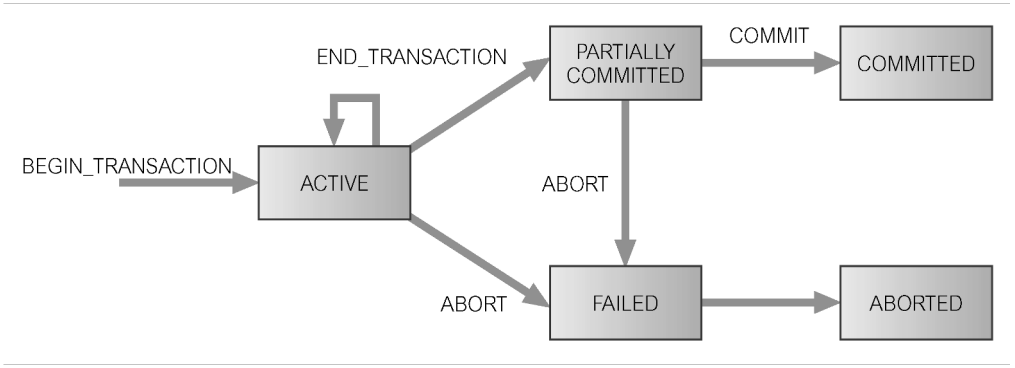


Opredelitev transakcije…

- 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.



ACID 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).

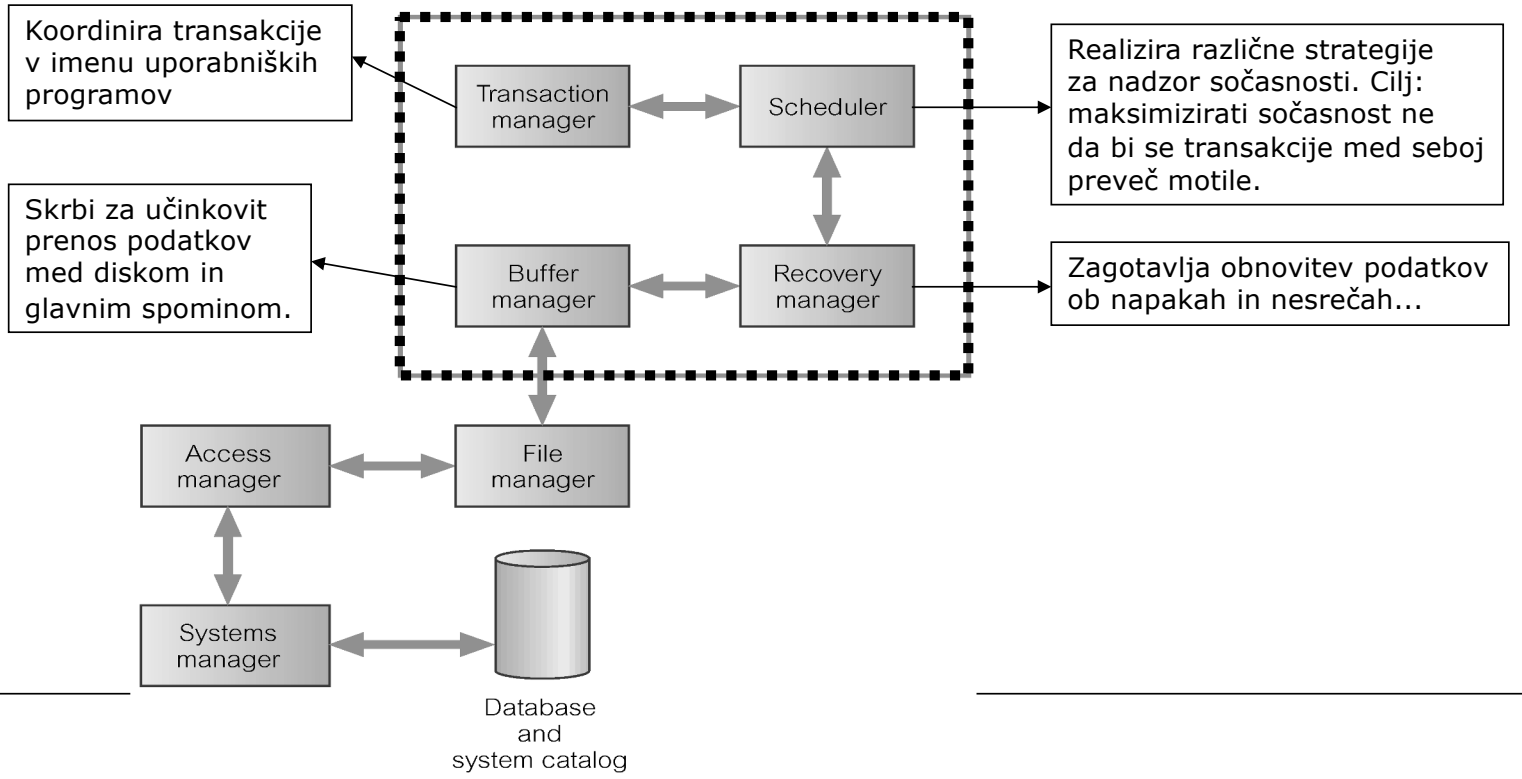
ACID lastnosti transakcij

- 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:



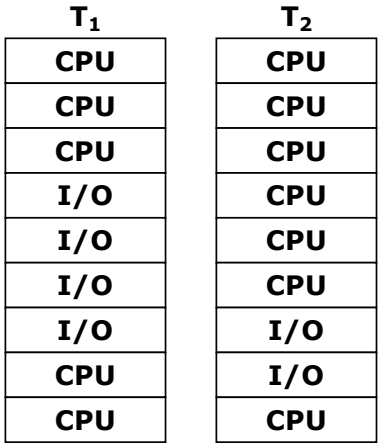
Nadzor sočasnosti

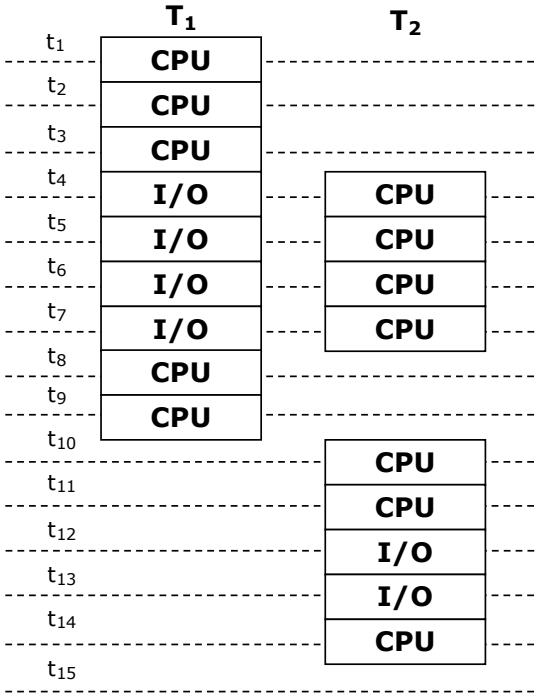
- Eden od ciljev in prednosti PB je možnost sočasnega dostopa s strani več uporabnikov do skupnih podatkov.
- To omogoča uporaba koncepta transakcij.
- Če vsi uporabniki podatke le berejo – nadzor sočasnosti trivialen;
- Če več uporabnikov sočasno dostopa do podatkov in vsaj eden podatke tudi zapisuje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

- Sočasnost dostopov naziramo s transakcijskimi protokoli
  - Najpopularnejši: protokol dvofaznega zaklepanja (2PL)

Zakaj sočasnost?

- Prepletanje operacij dveh transakcij...





Problemi v zvezi z nadzorom sočasnosti...

- V SUPB zaradi sočasnosti dostopa lahko pride do različnih problemov:
  - Izgubljene spremembe (lost update): uspešno izveden UPDATE se izgubi (prepiše) zaradi istočasno izvajane operacije s strani drugega uporabnika.
  - Uporaba nepotrjenih podatkov (dirty read): transakciji je dovoljen vpogled v spremenjene podatke druge transakcije, še preden je ta potrjena.
  - Ne-ponovljivo branje (non-repeatable read): transakcija prebere več vrednosti iz podatkovne baze. Nekatere izmed njih se na izvoru v času izvajanja prve transakcije zaradi drugih transakcij spremenijo: dvakratni SELECT nam vrne enaki množici vrstice z RAZLIČNIMI vrednostmi nekaterih atributov
  - Fantomsko branje (phantom read): podobno kot ne-ponovljivo branje, vendar za vstavljanje ali brisanje (pojavijo se nove vrstice, ali izginejo stare): dvakratni SELECT nam vrne različni množici vrstic.

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...

- Ne-ponovljivo branje
  - 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

Transakcije v SQL

- SUPB zagotavlja lastnosti ACID
  - Tipično z uporabo protokola dvofaznega zaklepanja (2PL)
- Transakcija je logična enota dela z enim ali več SQL ukazi. S stališča zagotavljanja skladnega stanja PB je atomarna.
- Spremembe, ki so narejene znotraj poteka transakcije, niso vidne navzven drugim transakcijam, dokler transakcija ni uspešno končana (impliciten ali ekspliciten COMMIT).

Transakcije v SQL

- SQL definira transakcijski model z ukazoma COMMIT in ROLLBACK
- Nova transakcija se začne takoj po koncu prejšnje
- Transakcije se vedno izvajajo. Ekstremni alternativi:
  - Autocommit način: vsak SQL ukaz zase je transakcija (privzeto pri MySQL);
    SET AUTOCOMMIT=1
  - Ena seja – ena transakcija (do zaključka)
- BEGIN ali START TRANSACTION
  - začne transakcijo (in do konca transakcije izklopi autocommit način);
    ekvivalent SET AUTOCOMMIT=0

Transakcije v SQL

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

Transakcije v SQL

- Nova transakcija se začne z novim SQL stavkom, ki transakcijo začne.
- SQL transakcij ne moremo gnezditi.
- Lastnosti transakcij nastavljamo s pomočjo ukaza SET TRANSACTION (pomoč upravljalcu transakcij)

  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 (ISO SQL)
  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 ali READ COMMITTED

Transakcije v SQL

- Učinek SET TRANSACTION ISOLATION LEVEL

	Branje neobstoječega podatka	Ne-ponovljivo branje	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

- Izgubljeno ažuriranje: dvodelno da, enodelno ne

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) in s tem pomaga preprečiti izgubljeno ažuriranje
- SELECT ... LOCK IN SHARE MODE; -- na koncu SELECT
  vse prebrane vrstice zaklene bralno (deljeno)

- tovrstno zaklepanje **ni** odvisno od ISOLATION LEVEL

Takojšnje in zakasnjene omejitve...

- Včasih želimo, da se omejitve (ponavadi referenčne - tuji ključi) 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 (privzeto za MySQL), lahko določimo tudi, ali je zakasnitev moč omogočiti kasneje. Uporabimo [NOT] DEFERRABLE.
- MySQL ne implementira zakasnjenih omejitev!!!

Delna rešitev za MySQL (tuji ključi):

```
SET FOREIGN_KEY_CHECKS=0;
SET FOREIGN_KEY_CHECKS=1;
-- ali
ALTER TABLE table_name DISABLE KEYS;
ALTER TABLE table_name ENABLE KEYS;
```

Takojšnje in zakasnjene 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

Primer transakcije

racun(<u>stev</u>, stanje)

```
START TRANSACTION;

UPDATE racun
      SET stanje=stanje-500
      WHERE stev='Racun1';

UPDATE racun
      SET stanje=stanje-500
      WHERE stev='Racun2';




ROLLBACK;
```

```
START TRANSACTION;

UPDATE racun
      SET stanje=stanje-100
      WHERE stev='Racun1';

UPDATE racun
      SET stanje=stanje+100
      WHERE stev='Racun2';

COMMIT;
```

- Kaj bi se moralo zgoditi po teh dveh transakcijah?
- Kaj bi se zgodilo brez uporabe transakcijskega protokola?
- Kje je problem?
- Vsaj kakšna bi morala biti stopnja izolacije?

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 (naknadno):
  - 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.
- Najpomembnejše 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)
- Mnogo specifičnih pravic (odvisno od sistema)

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...

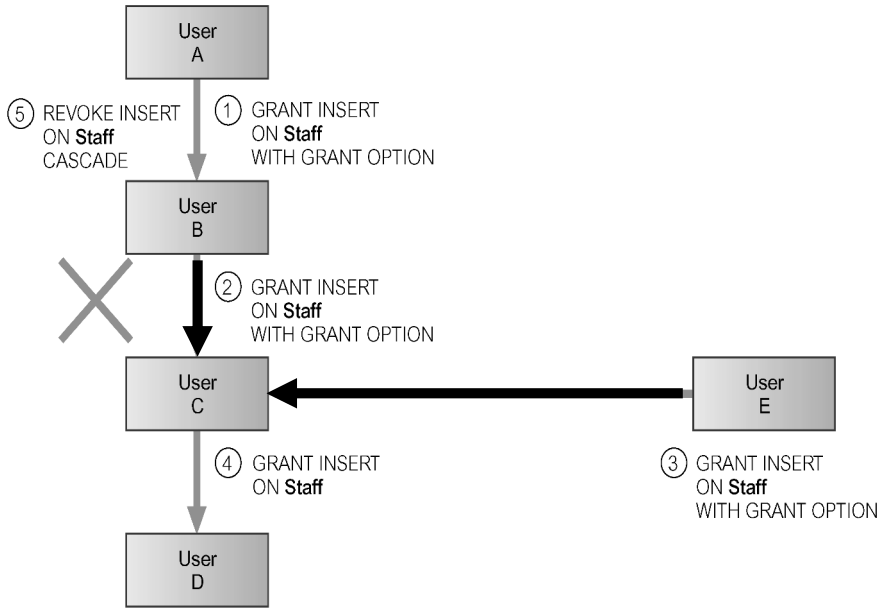
- ALL PRIVILEGES določa vse pravice, ki jih je uporabnik, ki REVOKE uporabi, dodelil uporabniku ali uporabnikom, na katere se REVOKE nanaša.
- GRANT OPTION FOR – omogoča, da se pravice, ki so bile dodeljene prek opcije WITH GRANT OPTION ukaza GRANT, odvzema posebej in ne kaskadno.
- RESTRICT, CASCADE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

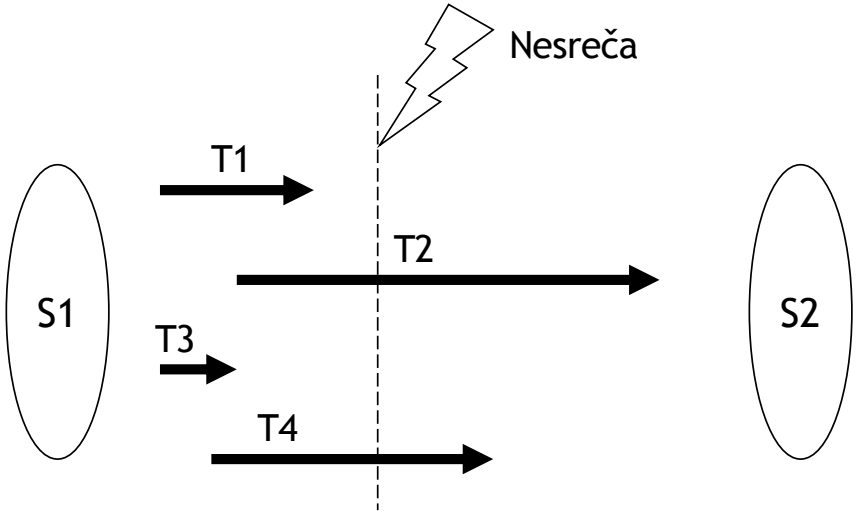
Podatkovna varnost

Podatkovna varnost

- Obnavljanje podatkov po nesrečah
- Transakcije in obnovljivost
- Komponente SUPB za obvladovanje obnovljivosti
- Tehnike obnovljivosti

Kaj je obnova 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 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,
  - optični disk: najzanesljivejši od vseh, še cenejši od traku, hitrejši od traku, omogoča neposredni dostop do podatkov.

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…

- 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 (Roll-forward 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).

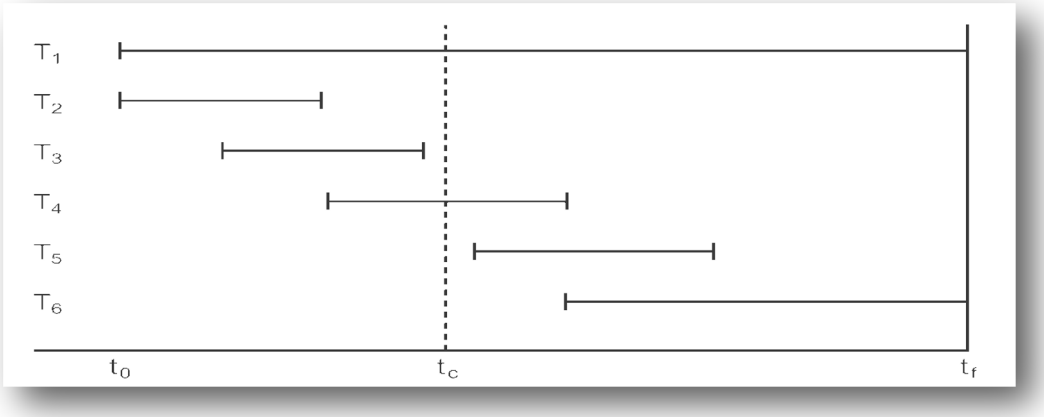
Kopije, dnevniki in kontrolne točke

- Običajen scenarij za zagotavljanje obnovljivosti
  - Kopija PB: periodično izvajanje na daljši časovni interval (npr. vsak dan opolnoči)
  - Dnevnik: sprotno zapisovanje v realnem času (običajno WAL – write-ahead log)

- Problem: večnivojska pomnilniška arhitektura
  - Kontrolna točka: točka sinhronizacije med PB in diskom (tudi kar se tiče dnevnika). Periodično (npr. na 15 minut) se izvede zahteva po izpisu vseh pomnilniških 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.

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.

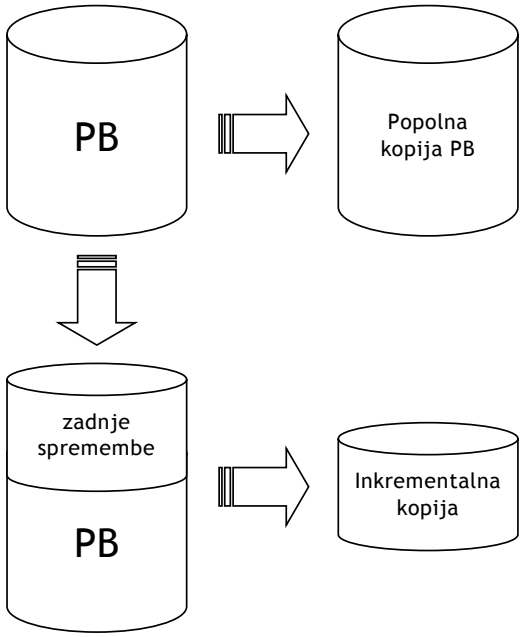
Mehanizem za izdelavo varnostnih kopij...

- 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.



Dnevnik

- V dnevnik se zapisujejo vse spremembe, ki jih transakcije izvedejo v PB.
- Najpogosteje: dnevnik vnaprejšnjih vpisov (write-ahead log oz. WAL)
- V dnevniku se hrani 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...

- 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 potrojen (princip 3-2-1).

- Včasih je bil dnevnik shranjen 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 del dnevnika hrani v pomnilniku, ostalo pa na disku (vsaj ena kopija).

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 disku, ki je ločen od podatkovnih datotek.

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...

- Tehnike obnovljivosti podatkov po nesrečah, ki privedejo PB v nekonsistentno stanje:
  - odloženo ažuriranje,
  - 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 (WAL). 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 (WAL) je teoretično učinkovitejše, če se v povprečju izvede več neuspešnih transakcij (ni potrebno spreminjati PB).
  - Sprotno ažuriranje (rollback journal) je teoretično učinkovitejše, če se v povprečju izvede več uspešnih transakcij (ni potrebno veliko popravljati podatkov v PB).

- Z vidika praktičnosti je odloženo ažuriranje veliko bolj praktično
  - WAL zagotavlja več sočasnosti (sočasno branje in pisanje)
  - Diskovne I/O operacije se lahko izvajajo bolj sekvenčno (optimizacija)
  - Manj težav pri sinhronizaciji medpomnilnikov (problematična funkcija fsync)
  - Posledično prevladujoča uporaba v praksi

Obnavljanje v MySQL ali MariaDB

- 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: 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)