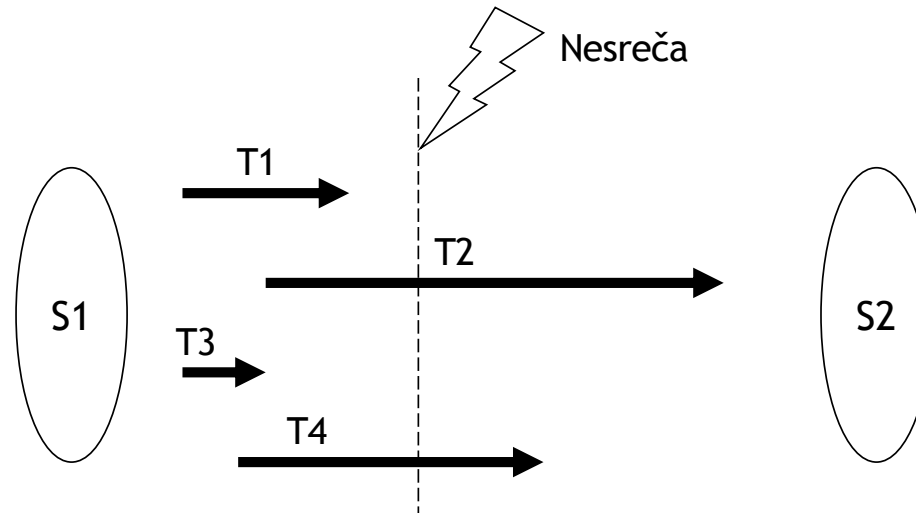


Podatkovna varnost in obnovljivost

- 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 nenamerne uničenja podatkov s strani administratorjev ali uporabnikov,
 - sabotaza (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...

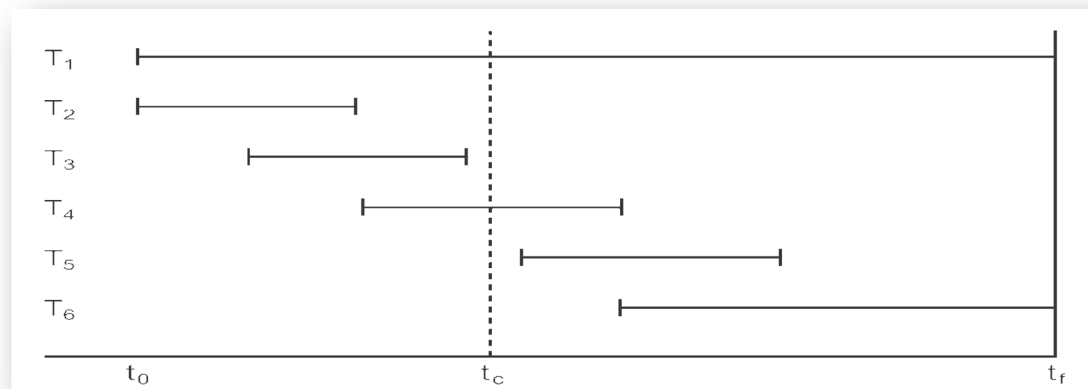
- Če se nesreča pripeti med pisanjem v pod. vmesnike ali med prenosom podatkov iz pod. vmesnikov v sek. pomnilnik, mora upravitelj za obnovljivost ugotoviti status transakcije, ki je izvajala pisanje v času nesreče:
 - če je transakcija izvedla ukaz COMMIT, mora upravitelj za obnovljivost zaradi zagotavljanja 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

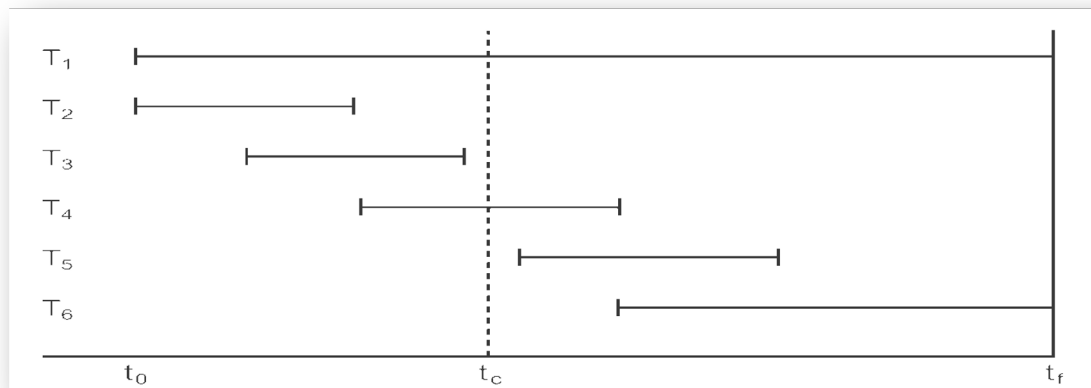
- Transakcije T_1 do T_6 se izvajajo sočasno, SUPB začne delovati ob t_0 , t_c je kontrolna točka, nesreča pa nastopi ob t_f :



- 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 .





Komponente SUPB za obnovljivost

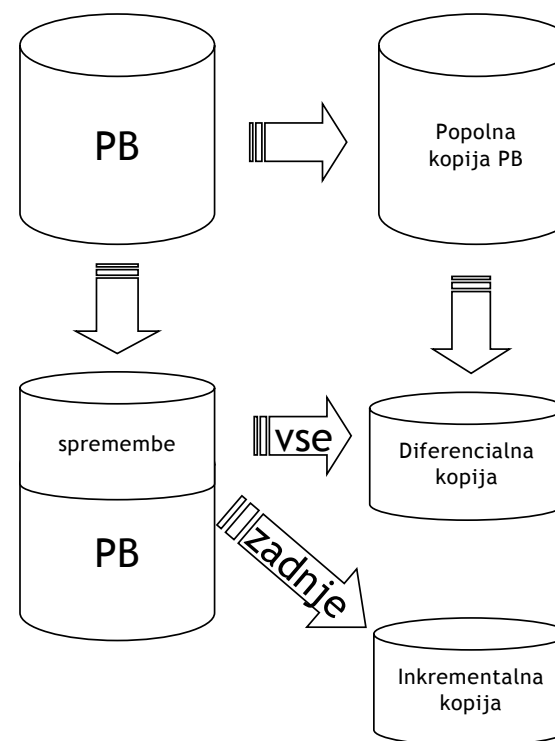
- V okviru SUPB so za obnavljanje podatkov po nesreči na voljo naslednje komponente:
 - mehanizem za izdelavo varnostnih kopij, ki periodično kreira kopije PB,
 - dnevnik, ki hrani podatke o trenutnem stanju transakcij in spremembah v PB,
 - mehanizem za izvajanje kontrolnih točk, ki omogoča da se posodobitve, ki jih izvajajo transakcije v PB, ohranijo (zahteva po zapisu vsega na disk),
 - upravljalca 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, delete, 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 dnevnik 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...

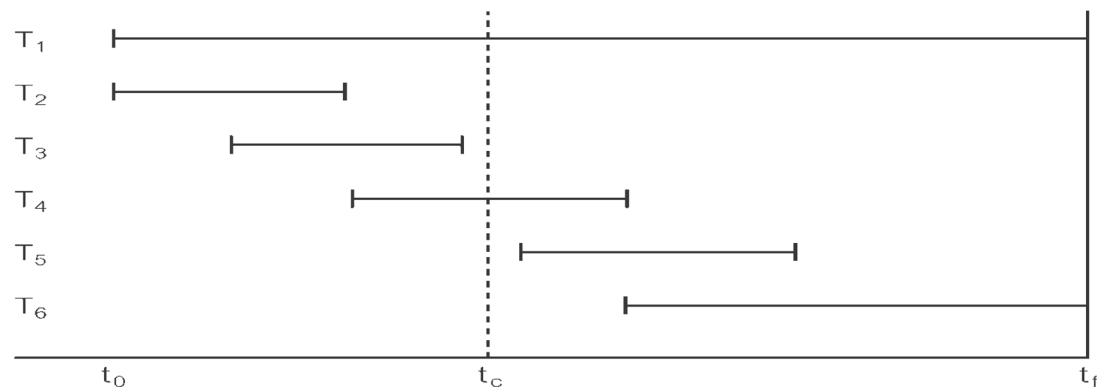
- Vse transakcije pred prej omenjeno transakcijo so že bile uveljavljene (committed) in njihove spremembe zapisane v PB v času izvedbe kontrolne točke.
- Zaradi tega je potrebno ponoviti samo transakcijo, ki je bila aktivna v času izvedbe kontrolne točke in vse transakcije, ki so ji sledile, katerih 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.

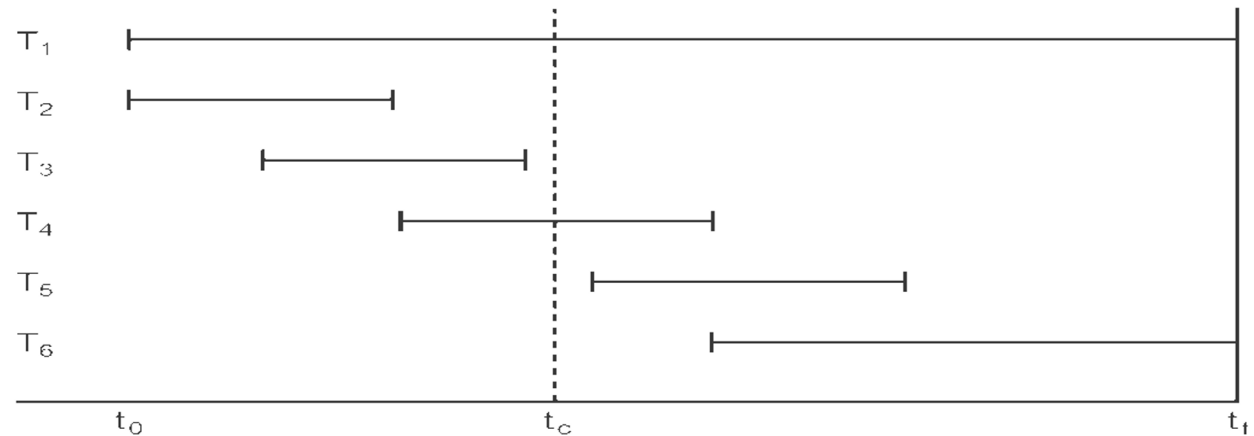
UNDO in REDO s kontrolno točko

- 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 ukazi 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

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;
```



Obnovljivost glede na obseg poškodb

- 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.

Obnovljivost glede na obseg poškodb

- 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 (ni potrebe po kopiji pod. baze):
 - odloženo ažuriranje (WRITE-AHEAD-LOG, WAL); sodobni SUPB
 - sprotno ažuriranje (DELETE-LOG); stari ali zelo preprosti SUPB
- 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.
- Po zaključku transakcije se dnevnik pobriše (DELETE) ali označi kot pobrisan/neveljaven
- Ta pristop se v "resnih" SUPB skoraj ne uporablja več

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)

Transakcije - povzetek

- Omogočajo transparentno sočasno delo na istih podatkih
 - ACID
 - Protokoli za zagotavljanje sočasnosti
 - 2PL+dodatki
 - Časovno označevanje,
 - Optimistični
 - MVCC
 - ...
- Služijo kot osnova za obnavljanje podatkovne baze
 - Kopija PB
 - Dnevnik (od zadnje kopije dalje)

Delo s transakcijami v Pythonu

- DB API v2
 - Definira standardni API za delo z (relacijskimi) SUPB
 - Definira standardne izjeme (neodvisno od SUPB)
 - Definira razrede/metode za delo s transakcijami
- Ukrepati je potrebno ob **neuspešnem zaključku** transakcije.
- Pazite na ustrezno sproščanje kurzorjev in povezav!
- Eksplicitno delo s transakcijami
 - Maksimalna kontrola in fleksibilnost, a več pisanja kode
- Kontekstni upravljalci in transakcije
 - Manj pisanja kode, bolj šablonska uporaba

DB API v2 (PEP 249) in transakcije

- Povezava (connection), metode:
 - cursor(), preko njega izvajamo transakcijske ukaze
 - commit()
 - rollback()
- Izjeme (exceptions), ne glede na SUPB
 - Razred DatabaseError ali njegovi podrazredi
 - Označujejo neuspešno izvedbo ukazov (uspešna izvedba ne "vrže/dvigne" izjem)
 - Koda napake se skriva v izjemi (e.args)
- Transakcije naj bodo čim krajše!
- Pozor: pazite na opcijo "autocommit" (gonilnik, povezava, SUPB, ...)

```
Exception
|__Warning
|__Error
    |__InterfaceError
    |__DatabaseError
        |__DataError
        |__OperationalError
        |__IntegrityError
        |__InternalError
        |__ProgrammingError
        |__NotSupportedError
```

Eksplícitno delo s transakcijami v Pythonu

```
connection = ...
c = connection.cursor()
try:
    c.execute("DELETE FROM rezervacija WHERE jid = 666")
    c.execute("DELETE FROM jadralec WHERE jid = 666")
    connection.commit()
except DatabaseError as e:
    print(e, e.args[0]) # Koda napake (odvisno od SUPB)
    connection.rollback()
```

Kontekstni upravljalci in transakcije v Pythonu

```
connection = ...
with connection.cursor() as c:
    try:
        c.execute("DELETE FROM rezervacija WHERE jid = 666")
        c.execute("DELETE FROM jadralec WHERE jid = 666")
        connection.commit()
    except DatabaseError as e:
        print(e, e.args[0]) # Koda napake (odvisno od SUPB)
        connection.rollback()
```

- Kurzor se sprosti takoj po izhodu iz konteksta (resursi)

Kontekstni upravljalci in transakcije v Pythonu

```
# try:
connection = ...
with connection:
    with connection.cursor() as c:
        c.execute("DELETE FROM rezervacija WHERE jid = 666")
        c.execute("DELETE FROM jadralec WHERE jid = 666")
# except ...
```

- Klic `commit()` ali `rollback()` se izvede implicitno ob izhodu iz konteksta povezave, odvisno od uspešnosti izvajanja.
- Povezava se ob izhodu iz konteksta zapre.
- Obravnava napake je potrebna zunaj konteksta povezave.