



IT250 - BAZE PODATAKA

Upravljanje okruženjem baza podataka

Lekcija 11

PRIRUČNIK ZA STUDENTE

IT250 - BAZE PODATAKA

Lekcija 11

UPRAVLJANJE OKRUŽENJEM BAZA PODATAKA

- ✓ Upravljanje okruženjem baza podataka
- ✓ Poglavlje 1: Oporavak BP
- ✓ Poglavlje 2: Oporavak BP od pada sistema
- ✓ Poglavlje 3: Oporavak BP od pada transakcija
- ✓ Poglavlje 4: Kontrola konkurentnosti
- ✓ Poglavlje 5: Nekoizistentna analiza
- ✓ Poglavlje 6: Izgubljeno i poništeno ažuriranje
- ✓ Poglavlje 7: Rešenje problema konkurentnosti: zaključavanje
- ✓ Poglavlje 8: Pokazna vežba
- ✓ Poglavlje 9: Domaći zadatak
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Šta ćemo naučiti u ovoj lekciji?

Jedan od veoma važnih elemenata, vezanih za server baze podataka, jeste mogućnost oporavka baze podataka (eng. **recovery**), što predstavlja povratak baze podataka u stanje pre softverskog ili hardverskog otkaza sistema. Razlozi za otkaz sistema mogu biti greške u operativnom sistemu, greške u programiranju, greške u samom SUBP-u ili padanje glave diska, nestanak struje i dr.

Proces oporavka izvodi se globalno izvodi u tri koraka:

1. **periodično kopiranje (eng. **save**) baze podataka na eksternu memoriju**
2. **zapisivanje transakcija (promena) nad bazom podataka u žurnal, tzv. log file.**
3. **oporavak baze podataka koji se može vršiti na više načina**

Administracija baze podataka se prvenstveno odnosi na komponente sistema za upravljanje bazama podataka koje omogućuju istovremeno, korektno i bezbedno izvršavanje zahteva većeg broja korisnika. Cilj administracije baze podataka je obezbediti način da se izbalansiraju konfliktni ciljevi zaštite baze podataka i da se maksimizira njena raspoloživost i korist koje korisnici mogu imati od nje. Osnovni zadaci administracije baze podatka su:

1. **Obezbediti kontrolu konkurentnosti**
2. **Upravlјati pravima obrade i odgovornostima**
3. **Razviti zaštitu podataka**
4. **Obezbediti oporavak baze podataka**
5. **Upravlјati DBMS-om**
6. **Održavati repozitorijum podataka**

▼ Poglavlje 1

Oporavak BP

ŠTA ZNAČI OPORAVAK BAZE PODATAKA?

Povratak baze podataka u stanje pre softverskog ili hardverskog otkaza sistema zbog grešaka u operativnom sistemu, greške u programiranju, greške u samom SUBP-u i dr

Jedan od veoma važnih elemenata, vezanih za server baze podataka, jeste mogućnost oporavka baze podataka (engl. *data base recovery*), što predstavlja povratak baze podataka u stanje pre softverskog ili hardverskog otkaza sistema.

Razlozi za otkaz sistema mogu biti:

1. greške u operativnom sistemu,
2. greške u programiranju,
3. greške u samom SUBP-u ili
4. padanje glave diska, nestanak struje i dr.

Tehnika redundantnog pamćenja podataka i tehnika oporavka baze su veoma kompleksne. Poznato je da su do sada korišćene jednostavne procedure koje su se bazirale na periodičnom kopiranju baze podataka u neku arhivsku memoriju i svih transakcija koje su se u međuvremenu dogodile. Iako jednostavne, ove metode imaju čitav niz nedostataka.

Proces oporavka izvodi se globalno izvodi u tri koraka:

1. periodično kopiranje (save) baze podataka na eksternu memoriju
2. zapisivanje transakcija (promena) nad bazom podataka u žurnal, tzv. log file.
 - Transakcije se u log file mogu upisivati pre nego što se njima ažurira baza podataka, tako da ako sistem padne u trenutku između upisa transakcije u log file i ažuriranja baze podataka, postoji trag o neizvršenoj transakciji.
 - Transakcije se u log file mogu upisivati pošto se njima ažurira baza podataka, kada postoji mogućnost da se baza ažurira ali da nema traga o tom ažuriranju u log-u pa se može desiti da korisnik ponovo unese transakciju koja je već završena.
3. oporavak baze podataka koji se može vršiti na više načina:
 - oporavak kroz ponovnu obradu transakcija.
 - oporavak kroz rollback ili rollforward.

OPORAVAK BAZA PODATAKA KROZ PONOVNU OBRADU

Podrazumeva periodično pravljenje kopije baze podataka; Kada dođe do greške, vrši se restore baze i ponovo obrađuju sve transakcije od trenutka pravljenja save-a do trenutka pada sistema.

Jedan od načina za oporavak baze podataka je **oporavak kroz ponovnu obradu transakcija.**

Predstavlja najprostiji način oporavka koji podrazumeva periodično pravljenje kopije baze podataka (koja se zove *save baze podataka*) u koju su zabeležene sve transakcije koje su se obavile do trenutka pravljenja kopije. U ovom slučaju nema zapisivanja transakcija u log file.

Kada dođe do greške, vrši se restore baze iz urađenog save-a i ponovo obrađuju sve transakcije koju su se desile od trenutka pravljenja save-a do trenutka pada sistema. Ovakva strategija nije uvek primenljiva, jer ponovna obrada transakcija zahteva isto vreme kao i njihova prvobitna obrada, a ona se ne može primeniti u slučaju sistema sa konkurentnom obradom.

OPORAVAK BAZE PODATAKA KROZ ROLLBACK ILI ROLLFORWARD

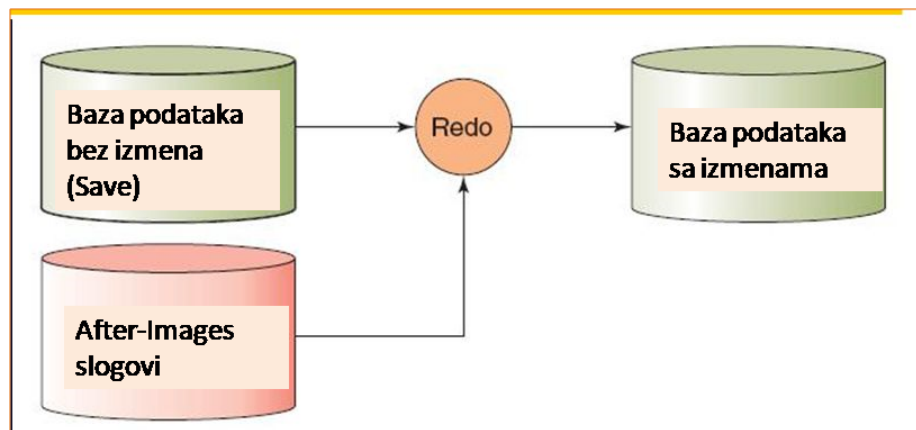
Pravi se save baze podataka i čuvaju sve promene u log file-u, od trenutka pravljenja save-a pa na dalje.

Oporavak kroz rollback ili rollforward podrazumeva pravljenje kopije baze podataka (*database save*) i čuvanja svih promena u log file-u, od trenutka pravljenja save-a. U slučaju pojave greške može se primeniti jedna od dve metode:

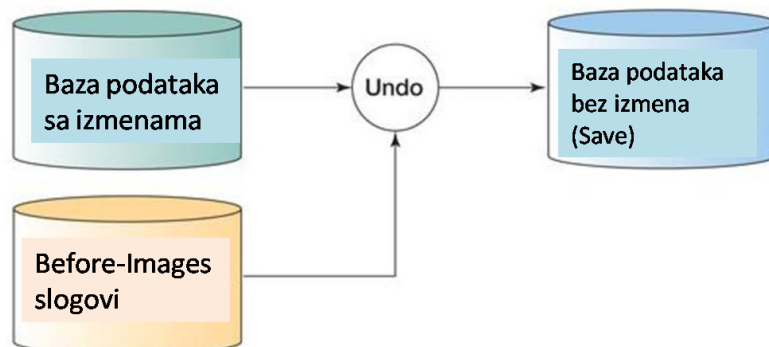
Rollforward – izvrši se restore baze podataka na osnovu napravljenog save-a i vrate sadržaji promenjenih slogova baze podataka, koji su od trenutka pravljenja save-a do trenutka otkaza sačuvani u log file-u.

Rollback – baza podataka se ostavlja u zatečenom stanju i vrate se nepromenjeni sadržaji slogova, koji su od trenutka pravljenja save-a do trenutka otkaza sačuvani u log file-u.

Na osnovu ovoga se može reći da se u slučaju greške, log može koristiti i za poništavanje (*undo*) i za vraćanje (*redo*) transakcija koje su se u bazi podataka dešavale od trenutka pravljenja save-a do trenutka nastanka greške što je prikazano na slikama 1.1 i 1.2.



Slika 1.1 Vraćanje transakcija [Izvor: Autor]



Slika 1.2 Poništavanje transakcija [Izvor: Autor]

BEFORE IMAGE I AFTER IMAGE SLOGOVI

Before image slogovi sadrže kopije slogova BP pre nego što su oni promenjeni; After image slogovi sadrže kopije slogova pošto su nad njima izvršene promene.

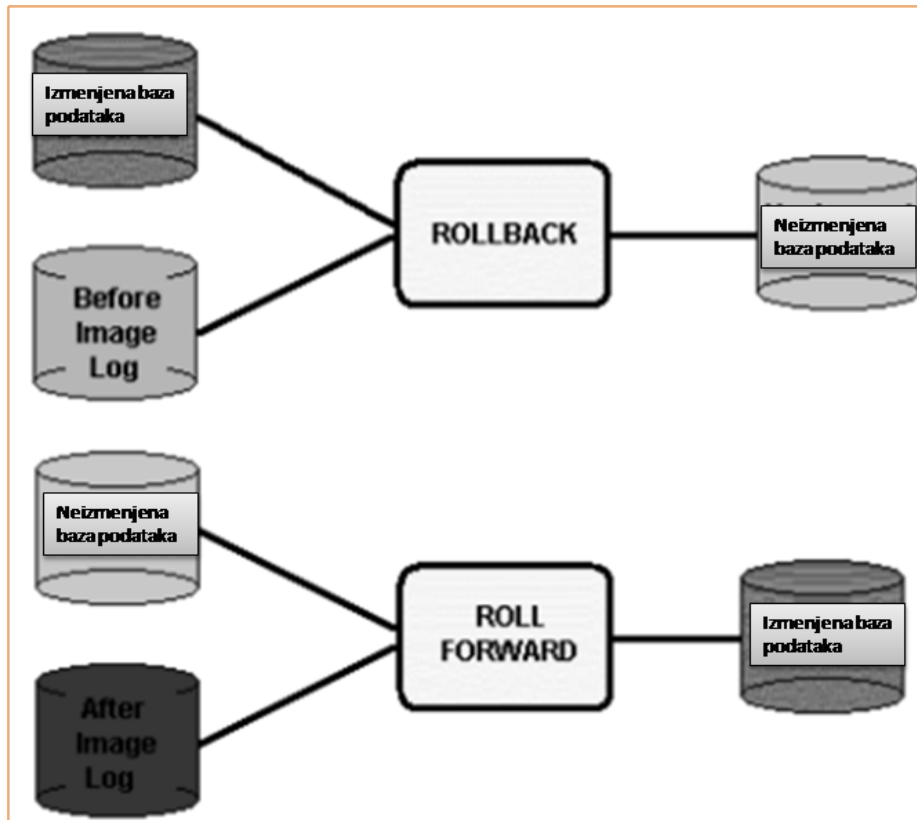
Da bi se transakcije poništile (undo), log file mora da sadrži kopiju svakog sloga baze podataka pre nego što je on promenjen. Takvi slogovi se zovu before image. Transakcije se poništavaju primenom before image-a svih promena nad bazom podataka.

Da bi se transakcije vratile (redo), log file mora da sadrži kopije svih slogova baze podataka pošto su nad njima izvršene promene. Ti slogovi se zovu after image. Transakcije se vraćaju primenom after image-a svih promena nad bazom podataka.

Da bi se transakcije koje se izvršavaju u okviru procesa tokom čijeg izvršenja je došlo do greške, prikazanog na slici 1.3 poništile (undo), u toku procesa oporavka (recovery), prikazanog na slici 1.3, treba jednostavno svaki promenjeni slog zameniti njegovim before image-om. Transakcije su poništene kada se vrate before image-i svih slogova.

Ako se proces oporavka (recovery) vrši vraćanjem (redo) transakcija, počinje se sa verzijom baze podataka koja je save-ovana u trenutku kada su transakcije počele i nad njom se

primenjuje after image, pa se tako dobija verzija baze koja je u odnosu na save-ovanu bazu novija.



Slika 1.3 Proces oporavka baze podataka [Izvor: Autor]

CHECKPOINT - TAČKA SINHRONIZACIJE

Tačka sinhronizacije između baze podataka i transakcija log file-a.

Restore baze podataka na neko prethodno stanje i ponovna obrada transakcija može zahtevati značajnu obradu, čiji se obim može smanjiti korišćenjem checkpoint-a.

Checkpoint je tačka sinhronizacije između baze podataka i transakcija log file-a. Prilikom izvršenja checkpoint-a, DBMS odbacuje novi zahtev, završava obradu zaostalih zahteva i njihove bafere upisuje na disk. DBMS tada čeka sve dok mu operativni sistem ne javi da su svi zahtevi za upisom u bazu podataka i u log file uspešno završeni.

U trenutku checkpoint-a, log file i baza podataka su sinhronizovani i checkpoint slog se upisuje u log file.

Kasnije se može izvršiti restore baza podataka od checkpoint-a i obraditi after image transakcija koje su se izvršile samo posle checkpoint-a.

Checkpoint je operacija koja ne košta ništa i pogodno ih je imati tri do četiri u toku jednog sata. Na taj način se može ponoviti samo proces obrade u trajanju od 15 do 20 minuta.

▼ Poglavlje 2

Oporavak BP od pada sistema

KADA SE VRŠI OPORAVAK OD PADA SISTEMA?

Kada u toku izvršenja neke obrade padne sistem zbog npr. prestanka električnog napajanja.

Oporavak se preduzima u slučaju da se, u toku izvršenja neke obrade, otkrije neki razlog koji onemogućava njeno uspešno kompletiranje.

Taj razlog može biti:

1. u samoj transakciji, kao što je prekoračenje neke od dozvoljenih vrednosti (pad transakcije)
2. u sistemu, npr. prestanak električnog napajanja (pad sistema), ili
3. u disku na kome je baza podataka, npr. oštećenje glava diska (pad medija).

Ovde će biti reči o oporavku od pada sistema.

U slučaju pada sistema, sadržaj unutrašnje memorije je izgubljen. Zato se, po ponovnom startovanju sistema, **za oporavak koriste podaci iz log datoteke da bi se poništili efekti transakcija koje su se obrađivale u trenutku pada sistema. Ove transakcije se mogu identifikovati čitanjem sistemskog loga unazad, kao transakcije za koje postoji BEGIN TRANSACTION slog ali ne postoji COMMIT slog.**

WAL (WRITE AHEAD LOG) PROTOKOL

Obezbeđuje da se pri izvršenju operacije COMMIT, slog prvo fizički upisuje u log datoteku, pa se zatim podaci upisuju iz bafera podataka u bazu.

Kao što je rečeno, u checkpoint tački, fizički se upisuju podaci i informacije iz log bafera i bafera podataka (koji su u unutrašnjoj memoriji) u log datoteku i u bazu podataka, redom, i upisuje se slog checkpoint-a u log datoteku. Ovaj slog sadrži informaciju o svim aktivnim transakcijama u momentu tačke pamćenja, adrese poslednjih slogova tih transakcija u log datoteci, a može sadržati i niz drugih informacija o stanju baze podataka u tom trenutku. Adresa checkpoint sloga upisuje se u datoteku ponovnog startovanja (eng. **restart file**).

Fizički upis u checkpoint tački znači da će se sadržaj bafera prepisati na spoljni medijum bez obzira da li su baferi puni ili ne. To dalje znači da se fizički upis svih ažuriranih podataka

uspešno kompletirane transakcije može garantovati tek u tački pamćenja koja sledi za uspešnim kompletiranjem transakcije.

Tako dolazimo do zaključka da su kompletiranje transakcije (upis COMMIT sloga u log datoteku) i definitivni upis svih ažuriranja te transakcije u bazu – dve odvojene radnje, za koje se ne sme dogoditi da se jedna izvrši a druga ne.

Mehanizam koji to obezbeđuje je protokol upisivanja u log koje se vrši unapred (eng. WAL – Write Ahead Log). Prema ovom protokolu, pri izvršenju operacije COMMIT prvo se odgovarajući slog fizički upisuje u log datoteku, pa se zatim podaci upisuju iz bafera podataka u bazu. Ako dođe do pada sistema posle upisa COMMIT sloga u log datoteku a pre nego što je sadržaj bafera podataka prepisan u bazu, taj se sadržaj može restaurirati iz sistemskog loga REDO logikom.

Pri ponovnom startovanju sistema, posle pada sistema, moguće je prema sadržaju log datoteke identifikovati neuspele transakcije – kandidate za poništavanje, i uspele transakcije – kandidate za ponovno izvršavanje.

▼ Poglavlje 3

Oporavak BP od pada transakcija

PLANIRANO I NEPLANIRANO IZVRŠENJE TRANSAKCIJE

Do planiranog završetka dolazi izvršenjem COMMIT operacije ili eksplicitne ROLLBACK operacije; Neplanirani završetak nastaje kada dođe do greške za koju ne postoji provera.

Jedan aplikativni program može se sastojati od većeg broja transakcija. Izvršenje jedne transakcije može da se završi planirano ili neplanirano.

1. Do planiranog završetka dolazi izvršenjem COMMIT operacije kojom se uspešno kompletira transakcija, ili eksplicitne ROLLBACK operacije, koja se izvršava kada dođe do greške za koju postoji programska provera (tada se radnje transakcije poništavaju a program nastavlja sa radom, izvršenjem sledeće transakcije).
2. Neplanirani završetak izvršenja transakcije događa se kada dođe do greške za koju ne postoji programska provera; tada se izvršava implicitna (sistemska) ROLLBACK operacija, radnje transakcije se poništavaju a program prekida sa radom.

Izvršavanjem operacije COMMIT:

1. efekti svih ažuriranja transakcije postaju trajni, tj. više se ne mogu poništiti procedurom oporavka.
2. u log datoteku se upisuje odgovarajući slog o kompletiranju transakcije (COMMIT slog), a svi katanci koje je transakcija držala nad objektima – oslobađaju se.

Izvršenje COMMIT operacije ne podrazumeva fizički upis svih ažuriranih podataka u bazu (neki od njih mogu biti u baferu podataka, pri čemu se, efikasnosti radi, sa fizičkim upisom u bazu čeka dok se baferi ne napune). Činjenica da efekti ažuriranja postaju trajni znači da se može garantovati da će se upis u bazu dogoditi u nekom narednom trenutku; u slučaju pada sistema, na primer, posle COMMIT operacije a pre fizičkog upisa podataka iz bafera u bazu, upis se garantuje REDO-logikom.

KAKO SE OMOGUĆAVA OPORAVAK OD PADA TRANSAKCIJE?

Vraćanje baze u konzistentno stanje omogućava se upisom svih informacija o ovim radnjama i operacijama u sistemski log.

Pad transakcije nastaje kada transakcija ne završi svoje izvršenje planirano. Tada sistem izvršava implicitnu – prinudnu ROLLBACK operaciju, tj. sprovodi aktivnost oporavka od pada transakcije.

Pojedinačne radnje ažuriranja u okviru jedne transakcije, kao i operacija početka transakcije (eksplicitna ili implicitna BEGIN TRANSACTION operacija), izvršavaju se na način koji omogućava oporavak baze u slučaju pada transakcije.

Oporavak podataka u bazi i vraćanje baze u konzistentno stanje omogućava se upisom svih informacija o ovim radnjama i operacijama u sistemski log.

Pri izvršavanju operacija ažuriranja (u užem smislu), osim što se ažurira baza, u log se beleže vrednosti svakog ažuriranog objekta pre i posle ažuriranja (uz ostale informacije o tom ažuriranju).

- Izvršavanjem operacije *BEGIN TRANSACTION* (bilo da je eksplicitna ili implicitna) u log datoteku se upisuje slog početka transakcije.
- Operacija *ROLLBACK*, bilo eksplicitna ili implicitna, sastoji se od poništavanja učinjenih promena nad bazom. Izvršava se čitanjem unazad svih slogova iz log datoteke koji pripadaju toj transakciji, do *BEGIN TRANSACTION* sloga. Za svaki slog, promena se poništava primenom odgovarajuće *UNDO*-operacije.

Aktivnost oporavka od pada transakcije ne uključuje REDO logiku.

▼ Poglavlje 4

Kontrola konkurentnosti

ZAŠTO SE PREDUZIMAJU MERE KONTROLE KONKURETNOSTI ?

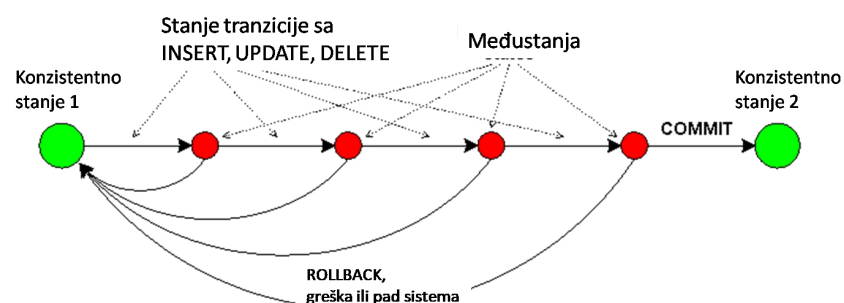
Da bi se sprečilo da rad jednog korisnika nad bazom podataka loše utiče na rad drugog.

Mere kontrole konkurentnosti se preduzimaju da bi se sprečilo da rad jednog korisnika loše utiče na rad drugog. U mnogim slučajevima, mere konkurentnosti obezbeđuju da korisnik u obradi u kojoj učestvuje više korisnika dobije iste rezultate koje bi dobio kada bi radio sam. Kontrola konkurentnosti se vezuje za pojam transakcije koja se definiše kao skup akcija koje predstavljaju logičku jedinicu posla nad bazom podataka.

Primer: U sistemu baza podataka jedne banke, prenošenje novca sa jednog računa na drugi bi predstavljalo jednu transakciju. Ona se sastoji od bar dve akcije: isplate sa jednog i uplate na drugi račun, ali su te radnje logički nedeljive i samo se zajedno smatraju jednim poslom. Na efekte transakcije ne sme da utiče eventualno istovremeno izvršenje drugih transakcija, ili nestanak struje.

Transakcija se može definisati i kao logička jedinica posla pri radu sa podacima. Ona predstavlja niz, tj. sekvencijalnu kompoziciju radnji koja ne narušava uslove integriteta. Kontrola konkurentnosti znači da po izvršenju kompletne transakcije stanje baze treba da bude konzistentno, tj. da nisu narušeni uslovi integriteta baza podataka.

Dakle, posmatrana kao jedinica posla, transakcija prevodi jedno konzistentno stanje baze u drugo takvo stanje baze, dok u međukoracima transakcije, konzistentnost podataka može biti i narušena. Transakcija tako predstavlja i bezbedno sredstvo za interakciju korisnika sa bazom.



Slika 4.1 Konzistentnost baze podataka [Izvor: Autor]

ACID SVOJSTVA TRANSAKCIJE

Atomičnost, konzistentnost, izolacija, trajnost.

Transakcija se karakteriše sledećim važnim svojstvima (poznatim kao ACID svojstva):

1. **atomičnost** (engl. **Atomicity**): transakciju treba izvršiti u celosti ili je uopšte ne treba izvršiti (nijednu njenu radnju);
2. **konzistentnost** (engl. **Consistency**): transakcija prevodi jedno konzistentno stanje baze u drugo konzistentno stanje baze;
3. **izolacija** (engl. **Isolation**): efekti izvršenja jedne transakcije su nepoznati drugim transakcijama sve dok se ona uspešno ne kompletira;
4. **trajnost** (engl. **Durability**): svi efekti uspešno kompletirane transakcije su trajni, tj. mogu se poništiti samo drugom transakcijom.

Ove osobine znače da kada se transakcija završi, njeni podaci su konzistentni i stabilni na disku. Ovo je dobro za one koji rade na razvoju aplikacija, ali zahteva sofisticirano **zaključavanje** koje može prouzrokovati nedostupnost podataka i obično predstavlja težak uzorak (eng. **pattern**) za većinu slučajeva korišćenja.

Kao osnovne komponente transakcije, posmatraćemo objekte (npr. slogove) i dve radnje:

1. čitanje *i*
2. upis (ili ažuriranje).

Dve radnje u ovom modelu su konfliktne ako se obavljaju nad istim objektom a jedna od njih je radnja upisa.

To znači da parovi konfliktnih radnji mogu biti samo parovi

1. (čitanje, upis) i
2. (upis, upis).

IZVRŠENJE TRANSAKCIJA U KONKURETNOM OKRUŽENJU: PRIMER

Problem konzistentnosti se može proučiti na primeru.

Problem konzistentnosti se može proučiti na primeru prikazanom na slici 4. 1 .



Slika 4.2 Primer načina izvršenja transakcija u konkurentnom okruženju [Izvor: NM IT350 - 2020/2021.]

Korisnik transakcije A čita stavku 100, menja je i ponovo zapisuje u bazi podataka. Korisnik transakcije B izvršava iste akcije, ali nad stavkom 200. CPU obrađuje transakciju korisnika A sve dok se ne pojavi neki prekid zbog I/O operacije ili nekog drugog kašnjenja kada operativni sistem prenosi kontrolu na korisnika B. CPU tada obrađuje transakcije korisnika B do pojave drugog prekida kada operativni sistem ponovo prenosi kontrolu na korisnika A. Korisnicima izgleda da je obrada transakcija istovremena ali ona je konkurentna, što je prikazano na slici 4.1.

Prilikom konkurentnog izvršenja transakcija nad bazom podataka može doći do sledećih problema:

1. Nekonzistentne analize
2. Izgubljenog ažuriranja
3. Poništenog ažuriranja

✓ Poglavlje 5

Nekozistentna analiza

DEFINISANJE TRANSAKCIJA T1 I T2

Transakcije se izvršavaju nad tabelama KI i I.

K_SIF	I_SIF	IZDANJE	GODINA	TIRAZ	VRIZDANJA
k1	i1	2	1965	10000	
k2	i1	2	1974	7000	
k3	i1	1	1975	10000	
k4	i1	2	1979	10000	
k5	i2	4	1986	5000	
k6	i3	1	1966	3000	
k6	i4	3	1988	5000	

Slika 5.1 Relacija KI [Izvor: Autor]

Neka je **relacija KI** proširena atributom VRIZDANJA koji se odnosi na vrednost jednog izdanja (u dinarima), jedne knjige, jednog izdavača, i neka je relacija I:

I_SIF	NAZIV	STATUS	DRŽAVA	VREDNOST
i1	Prosveta	30	Srbija	
i2	Addison Wesley Publ. Comp	20	SAD	
i3	Dečije novine	10	Srbija	
i4	Wiley Computer Publishing	30	SAD	

Slika 5.2 Relacija I [Izvor: Autor]

Nad ove dve tabele se mogu izvršavati dve transakcije:

Transakcija T1- koja zahteva promenu vrednosti atributa VRIZDANJA drugog Prosvetinog izdanja knjige sa šifrom k1 u relaciji KI, kao i odgovarajuću promenu vrednosti atributa VREDNOST u relaciji I. U nekim SQL dijalektima ova transakcija može se izraziti na sledeći način:

```
BEGIN TRANSACTION
(A) UPDATE KI
    SET VRIZDANJA = VRIZDANJA + 4000
    WHERE K_SIF = 'k1' AND I_SIF = 'i1' AND IZDANJE = 2;
(B) UPDATE I
    SET VREDNOST = VREDNOST + 4000
    WHERE I_SIF = 'i1';
END TRANSACTION
```

Transakcija T2- kojom se povećava vrednost svih izdanja izdavača i1 za 10%. Transakcija T2 se u SQL-u može zapisati na sledeći način:

```
BEGIN TRANSACTION
(A')      UPDATE KI
              SET VRIZDANJA = VRIZDANJA * 1.1
              WHERE KI.I_SIF = 'i1';
(B')      UPDATE I
              SET VREDNOST = VREDNOST * 1.1
              WHERE I.I_SIF = 'i1';
END TRANSACTION
```

KADA T1 I T2 MOGU BAZU DOVESTI U NEKONZISTENTNO STANJE?

Ako se jedna transakcija kompletno izvrši između prve i druge radnje druge transakcije.

Ako jedna transakcija počne po završetku druge (redosled radnji: A, B, A', B' ili A', B', A, B), ili ako im se radnje izvršavaju naizmenično (redosled radnji: A, A', B, B' ili A', A, B', B), njihovo izvršenje je serijsko ili ekvivalentno serijskom izvršenju T1;T2 (ili T2;T1). Posle tog izvršenja baza će biti u konzistentnom stanju.

Ako se jedna transakcija kompletno izvrši između prve i druge radnje druge transakcije (redosled radnji: A, A', B', B ili A', A, B, B'), baza će se posle izvršenja tog skupa transakcija naći u nekonzistentnom stanju.

Da bismo se uverili u ovo tvrđenje, razmotrimo efekte koje na bazu proizvodi redosled radnji A, A', B', B.

Primer: Pretpostavimo, pri tom, da je vrednost 2. izdanja knjige k1 izdavača i1 jednaka a, a da je vrednost svih izdanja izdavača i1 jednaka b (i u relaciji I – vrednost atributa VREDNOST, i u relaciji KI – suma vrednosti atributa VRIZDANJA za izdavača i1). Da bi izvršenje skupa transakcija {T1,T2} ostavilo bazu u konzistentnom stanju, iznos b u obe relacije mora da se promeni za istu vrednost.

Promene nad relacijama I i KI možemo da posmatramo na sledeći način:

```
Stanje baze pre izvršenja transakcija T1, T2:
KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VRIZDANJA]      = a;
SUM (KI [VRIZDANJA]) za I_SIF = i1                = b;
I [I_SIF=i1] [VREDNOST]                          = b;
Stanje baze posle izvršenja radnje A (transakcija T1):
KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VRIZDANJA]      = a+4000;
SUM (KI [VRIZDANJA]) za I_SIF = i1                = b+4000;
Stanje baze posle izvršenja radnje A' (transakcija T2):
KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VRIZDANJA]      = (a + 4000) * 1.1
                                                    = a * 1.1 + 4400
SUM (KI [VRIZDANJA]) za I_SIF = i1                = (b+4000)*1.1
```


$$= b * 1.1 + 4400$$

(svim izdanjima izdavača i1 vrednost je uvećana za 10%);

Vrednost atributa VREDNOST relacije I, za izdavača sa šifrom i1, manja za 400 ,dinara nego što bi trebalo da bude, što se vidi iz sledećeg:

Stanje baze posle izvršenja radnje B' (transakcija T2):
I [I_SIF=i1] [VREDNOST] = $b * 1.1$;
Stanje baze posle izvršenja radnje B (transakcija T1):
I [I_SIF=i1] [VREDNOST] = $b * 1.1 + 4000$.

DOKAZ O NEKONZISTENTNOM STANJU BAZE

*Vrednost atributa VREDNOST relacije I za izdavača sa šifrom i1 je $b*1.1+4000$, a zbir vrednosti svih izdanja (VRIZDANJA) tog izdavača u relaciji KI je $b * 1.1 + 4400$.*

Dakle,

1. vrednost atributa VREDNOST relacije I za izdavača sa šifrom i1 je $b*1.1+4000$, a
2. zbir vrednosti svih izdanja (VRIZDANJA) tog izdavača u relaciji KI je $b * 1.1 + 4400$.

Prva vrednost (VREDNOST) uvećana je za 10% (zbog uvećanja vrednosti svih izdanja tog izdavača za 10%) i za 4000 dinara (jer je 2. izdanju knjige k1 izdavača i1 uvećana vrednost i za 4000) ali nije uvećana za 10% od tih 4000 dinara (mada je vrednost 2. izdanja knjige k1 izdavača i1 porasla i za 10% na uvećanje od 4000 dinara). Zato je vrednost atributa VREDNOST relacije I, za izdavača sa šifrom i1, manja za 400 dinara nego što bi trebalo da bude.

▼ Poglavlje 6

Izgubljeno i poništeno ažuriranje

KAKO NASTAJE IZGUBLJENO AŽURIRANJE?

Nastaje ako se svaka od transakcija T1, T2 sastoji od čitanja i upisa istog sloga datoteke, pri čemu se prvo izvrše radnje čitanja transakcija T1 i T2, a zatim radnje upisa u istom redosledu.

Neka je relacija KI prikazana na slici 6.1.

K_SIF	I_SIF	IZDANJE	GODINA	TIRAZ	VRIZDANJA
k1	i1	2	1965	10000	
k2	i1	2	1974	7000	
k3	i1	1	1975	10000	
k4	i1	2	1979	10000	
k5	i2	4	1986	5000	
k6	i3	1	1966	3000	
k6	i4	3	1988	5000	

Slika 6.1 Relacija KI [Izvor: Autor]

proširena atributom VRIZDANJA koji se odnosi na vrednost jednog izdanja (u dinarima), jedne knjige, jednog izdavača, i neka je relacija I:

I_SIF	NAZIV	STATUS	DRŽAVA	VREDNOST
i1	Prosveta	30	Srbija	
i2	Addison Wesley Publ. Comp	20	SAD	
i3	Dečije novine	10	Srbija	
i4	Wiley Computer Publishing	30	SAD	

Slika 6.2 Relacija I [Izvor: Autor]

proširena atributom VREDNOST koji se odnosi na vrednost (u dinarima) svih izdanja jednog izdavača (prikazana na slici 6.2). U tom slučaju, uslov integriteta baze može biti zadat kao: **vrednost atributa VREDNOST relacije I, za svakog pojedinačnog izdavača je jednaka zbiru vrednosti svih izdanja svih knjiga tog izdavača u relaciji KI.**

Nad ove dve tabele se mogu izvršavati dve transakcije:

Neka se svaka od transakcija T₁, T₂ sastoji od čitanja i upisa istog sloga datoteke, pri čemu se prvo izvrše radnje čitanja transakcija T₁ i T₂, a zatim radnje upisa u istom redosledu.

Na primer: transakcija T₁ može da pročita slog koji se odnosi na izdavača sa šifrom I_SIF = i2, zatim transakcija T₂ može da pročita isti slog (obe transakcije čitaju identične podatke; posebno, za atribut STATUS, obe čitaju vrednost 20).

Posle toga transakcija T_1 može da promeni vrednost atributu *STATUS* tog sloga (sa 20 na 30) i da završi sa radom, npr. da zatvori datoteku, a zatim transakcija T_2 može da promeni vrednost atributu *STATUS* (sa pročitane vrednosti 20 na 40) i da završi sa radom (npr. da zatvori svoju verziju iste datoteke).

U tom slučaju ostaju zapamćene samo promene transakcije T_2 koja je poslednja izvršila upis i zatvorila datoteku (vrednost polja *STATUS* sloga o izdavaču i2 ostaje 40), a transakciji T_1 se bez obaveštenja poništava efekat upisa, tj. gubi se vrednost 30 polja *STATUS* posmatranog sloga.

Izgubljeno ažuriranje nastaje ako se svaka od transakcija T_1 , T_2 sastoji od čitanja i upisa istog sloga datoteke, pri čemu se prvo izvrše radnje čitanja transakcija T_1 i T_2 , a zatim radnje upisa u istom redosledu.

PRIMER IZGUBLJENOG AŽURIRANJA

Problem se može analizirati i na sledećem primeru.

Ovaj problem se može analizirati i na sledećem primeru. Korisnik A želi da naruči pet komada stavke 100 a korisnik B tri komada iste stavke, što je prikazano na slici 6.3.

1. Čita stavku 100 (broj stavki je 10)	1. Čita stavku 100 (broj stavki je 10)
2. Smanjuje količinu stavke za 5	2. Smanjuje količinu stavke za 3
3. Zapisuje stavku 100	3. Zapisuje stavku 100

Redosled obrade na serveru baze podataka

1. Čita stavku 100 za korisnika A
2. Čita stavku 100 za korisnika B
3. Stavlja količinu na 5 (za korisnika A)
4. Piše stavku 100 (za korisnika A)
5. Stavlja količinu na 7 (za korisnika B)
6. Piše stavku 100 (za korisnika B)

Slika 6.3 Primer izgubljenog ažuriranja [Izvor: NM-IT350-2020/2021.]

Na kraju, u bazi podataka je zapisano da je da je količina stavke 100 iznosi 7, što nije tačno.

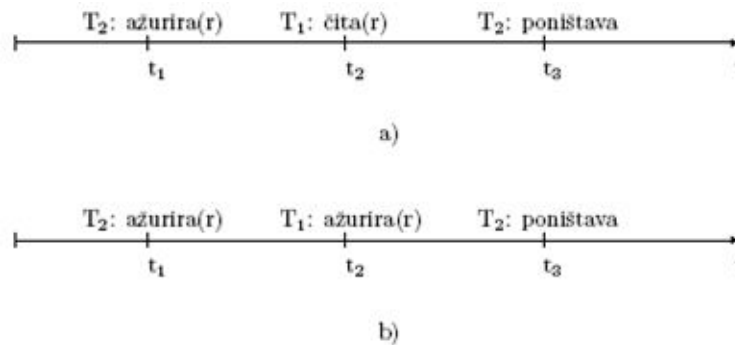
KADA DOLAZI DO PONIŠTAVANJA AŽURIRANJA?

Npr. kada 1. transakcija ažurira slog, a zatim 2. pročita slog i završi svoje izvršavanje. Ako 1. transakcija poništi ažuriranje sloga, 2. transakcija je pročitala pogrešnu vrednost.

Pretpostavimo da transakcija T_1 uključuje čitanje nekog sloga r , a da transakcija T_2 uključuje upis (tj. ažuriranje) istog sloga. Neka pri konkurentnom izvršavanju ovih transakcija prvo transakcija T_2 ažurira slog r , a zatim transakcija T_1 pročita slog r i završi svoje izvršavanje.

U nekom kasnijem trenutku, iz bilo kog razloga, transakcija T_2 poništi sve efekte koje je do tog trenutka proizvela na bazu, tj. poništi ažuriranje sloga r . U tom slučaju transakcija T_1 je pročitala ažuriranu vrednost sloga r koja nije ni trebalo da bude upisana u bazu.

Slično, neka transakcija T_1 uključuje ažuriranje nekog sloga r , a transakcija T_2 se sastoji od istih radnji kao i u prethodnom slučaju. Pri istom redosledu izvršavanja radnji i poništenju transakcije T_2 izgubilo bi se, bez obaveštenja, ažuriranje transakcije T_1 , jer bi poništavanjem efekata transakcije T_2 vrednost sloga r bila vraćena na vrednost pre ažuriranja (od strane transakcije T_2).



Slika 6.4 Zavisnost a) čitanja b) ažuriranja od poništenog ažuriranja [Izvor: NM IT350 - 2020/2021.]

Poništavanje ažuriranja nastaje npr. kada prva transakcija ažurira slog, a zatim druga pročita slog i završi svoje izvršavanje. Ako prva transakcija poništi ažuriranje sloga, druga transakcija je pročitala pogrešnu vrednost.

▼ Poglavlje 7

Rešenje problema konkurentnosti: zaključavanje

ZAKLJUČAVANJE

Obezbeđuje serijsko izvršenje transakcija, koje ne poznaje ni jedan od pomenutih problema kontrole konkurentnosti.

Problemi konkurentnosti opisani u prethodnom odeljku mogu se rešiti mehanizmom dinamičkog zaključavanja i otključavanja objekata, koji obezbeđuje linearizovanost izvršenja. **Linearizovana izvršenja, s obzirom da su ekvivalentna serijskim, ne poznaju ni jedan od pomenutih problema.**

Moguće je realizovati različitu granularnost, tj. veličinu objekata zaključavanja; na primer, objekti zaključavanja mogu biti cele tabele, pojedinačne n-torke relacije, delovi n-torki, grupe atributa, skupovi n-torki koje zadovoljavaju zadati uslov, itd.

Zaključavanje objekta (postavljanje katanca na objekat) je postupak koji obezbeđuje transakciji pristup objektu, i kojim transakcija istovremeno sprečava druge transakcije da pristupe tom objektu.

Svaka transakcija na kraju svog izvršavanja otključava sve objekte koje je sama zaključala (a koje nije već otključala). Pretpostavimo da u jednostavnom modelu postoji samo jedna vrsta katanca, i da pri izvršenju skupa transakcija nijedna transakcija ne može zaključati već zaključani objekat. U realnim modelima transakcija postoji više vrsta katanaca, od kojih su neki *deljivi* a neki *ekskluzivni*; pri tom više transakcija može da postavi deljivi katanac na jedan objekat, ali najviše jedna može da postavi ekskluzivni katanac.

Ekskluzivno zaključavanje zaključava objekat za bilo kakav drugi pristup, nijedna druga transakcija ne može da čita ili da menja taj objekat.

Deljivo zaključavanje zaključava objekat za menjanje ali ne i za čitanje, što znači da druge transakcije mogu da ga čitaju ali ne i da ga menjaju.

PRIMER

Dat je sledeći primer zaključavanja.

Slika 7.1 prikazuje redosled obrade na prethodnom primeru ažuriranja iste stavke od strane dva korisnika, korišćenjem komande lock (zaključavanje)

KORISNIK A	KORISNIK B
1. Zaključava stavku 100	1. Zaključava stavku 100
2. Čita stavku 100	2. Čita stavku 100
2. Smanjuje količinu stavke za 5	3. Smanjuje količinu stavke za 3
3. Zapisuje stavku 100	4. Zapisuje stavku 100

Redosled obrade na serveru baze podataka

1. Zaključava stavku 100 za korisnika A
2. Čita stavku 100 za korisnika A
3. Zaključava stavku 100 za korisnika B; ne može pa stavlja B u stanje čekanja
4. Stavlja količinu na 5 za korisnika A
5. Piše stavku 100 za korisnika A
6. Otključava stavku 100 za korisnika A
7. Zaključava stavku 100 za korisnika B
8. Čita stavku 100 za korisnika B
9. Stavlja količinu na 2 za korisnika B
10. Piše stavku 100 za korisnika B
11. Otključava stavku 100 za korisnika B

Slika 7.1.1 Primer zaključavanja [Izvor: NM IT350-2020/2021.]

Kod zaključavanja, postoje dve faze:

1. faze zaključavanja objekata i
2. faze otključavanja objekata.

U fazi zaključavanja nema nijedne radnje otključavanja objekta, odnosno u fazi otključavanja više nema radnji zaključavanja (naravno, obe ove faze uključuju i druge operacije nad objektima kao što su čitanje, obrada, upis objekata).

Ova strategija omogućava da se zaključavanje izvrši kada je to potrebno, ali kada se otključa prvi slog, zaključavanja više nema. Faza otključavanja objekata počinje prvom radnjom otključavanja.

Specijalan slučaj dvofaznog zaključavanja je korišćenje komandi COMMIT i ROLLBACK što je prisutno u mnogim RDBMS sistemima. U tom slučaju se zaključavanje vrši samom transakcijom, ali se zaključavanje ne ukida sve dok se ne izvrši komanda COMMIT ili ROLLBACK.

TEOREMA

Dovoljan uslov za linearizovanost izvršenja i skupa transakcija $\{T_1, T_2\}$ je da su one dvofazne.

Teorema>: Dovoljan uslov za linearizovanost izvršenja skupa transakcija $\{T_1, T_2\}$ je da su one dvofazne.

Dokaz: Pretpostavimo da za neki objekat o trojka (T_1, o, T_2) jeste element relacije zavisnosti izvršenja I , tj. $(T_1, o, T_2) \in Z(I)$. Pokazaćemo da je onda izvršenje I ekvivalentno serijskom T_1 ,

T_2 , tj. da je linearizovano (sasvim analogno se pokazuje da, ako je trojka $(T_2, o, T_1) Z(I)$, onda je izvršenje I ekvivalentno serijskom T_2, T_1 , tj. opet je linearizovano).

U suprotnom, neka postoji objekat o' takav da je $(T_2, o', T_1) Z(I)$. To znači da:

1. zbog $(T_1, o, T_2) Z(I)$ postoje koraci k, m ($k < m$) izvršenja I u kojima transakcija T_1 otključava objekat o (korak k), odnosno transakcija T_2 zaključava objekat o (korak m)
2. zbog $(T_2, o', T_1) Z(I)$, postoje koraci p, q ($p < q$) izvršenja I u kojima transakcija T_2 otključava objekat o' (korak p), odnosno transakcija T_1 zaključava objekat o' (korak q)
3. iz dvofaznosti transakcija T_1 i T_2 sledi da je $q < k$ i $m < p$
4. iz 1) i 3) sledi $q < k < m < p$ što je u kontradikciji sa 2). To znači da ne postoji objekat o' takav da je $(T_2, o', T_1) Z(I)$, tj. za svaki objekat o' nad kojim operišu obe transakcije T_1, T_2 važi: $(T_1, o', T_2) Z(I)$. Dakle, izvršenje I je linearizovano.

Indukcijom se može dokazati važenje prethodnog tvrđenja i za svaki konačan skup transakcija $\{T_1, T_2, \dots, T_n\}$.

Dvofaznost skupa transakcija nije potreban uslov za linearizovanost proizvoljnog izvršenja, što se može jednostavno dokazati primerom.

▼ 7.1 Deadlock

ŠTA JE DEADLOCK?

Uzajamno blokiranje transakcija.

Osim što obezbeđuje linearizovanost, dvofaznost transakcija može proizvesti i neželjeni efekat poznat kao **uzajamno blokiranje transakcija** (eng. **deadlock**).

Primer 1: Za dve dvofazne transakcije

T_1 :	zaključati KI	T_2 :	zaključati I
	upisati KI		upisati I
	zaključati I		zaključati KI
	upisati I		upisati KI
	otključati KI		otključati I
	otključati I		otključati KI,

Slika 7.2.1 Primer transakcija T_1 i T_2 [Izvor: Autor]

Jedna mogućnost njihovog delimičnog izvršenje je prikazana na slici 8.2.

Nijedna transakcija ne može da nastavi sa radom, pa se jedna transakcija nasilno prekida uz poništenje svih njenih radnji i oslobađanje katanaca; zatim se ta transakcija ponovno aktivira.

T₁: zaključati KI
T₁: upisati KI
T₂: zaključati I
T₁: upisati I.

Slika 7.2.2 Dead lock za transakcije T1 i T2 [Izvor: Autor]

Primer 2: Razmotrimo još jedan primer blokiranja transakcija u slučaju da dva korisnika žele da naruče dve stavke sa zaliha. Pretpostavimo da korisnik A želi da naruči papir i ako ga dobije, želi da naruči olovke. Korisnik B želi da naruči olovke i ako ih dobije želi da naruči papir. Redosled obrade je prikazan na slici 8.3.

KORISNIK A	KORISNIK B
1. Zaključava papir	1. Zaključava olovke
2. Uzima papir	2. Uzima olovke
3. Zaključava olovke	3. Zaključava papir

Redosled obrade na serveru baze podataka
1. Zaključava papir za korisnika A
2. Zaključava olovke za korisnika B
3. Obradjuje zahtev korisnika A; upisuje slog za papir
4. Obradjuje zahtev korisnika B; upisuje slog za olovke
5. Stavlja korisnika A u stanje čekanja za olovke
6. Stavlja korisnika B u stanje čekanja za papir
** ZAKLJUČANO **

Slika 7.2.3 Primer zaključavanja [Izvor: Autor]

Skoro svi DBMS sistemi imaju algoritme za rešavanja situacija u kojima su transakcije blokirane. Prvo, DBMS mora da detektuje da je došlo do blokiranja a zatim da prekine jednu od transakcija i iz baze podataka ukloni sve promene koje je ona uzrokovala.

OPTIMISTIČNO I PESIMISTIČNO ZAKLJUČAVANJE

Optimistično: Pretpostavka je da se konflikt neće dogoditi.

Pesimistično: pretpostavka je da će se konflikt pojaviti.

Generalno, postoje dva osnovna stila zaključavanja:

1. **optimistično i**
2. **pesimistično.**

Kod optimističnog zaključavanja, pretpostavka je da se konflikt neće dogoditi. Podaci se čitaju, transakcije se obrađuju, vrše se ažuriranja i tada se proverava da li je došlo do konflikta. Ako nije, transakcija se završava. Ako se konflikt pojavio, transakcija se ponavlja sve dok se ona ne obradi bez konflikta.

Sledi primer optimističnog zaključavanja

```
SELECT PRODUCT.Name, PRODUCT.Quantity
  FROM PRODUCT
WHERE PRODUCT.Name = 'Pencil'
Set NewQuantity = PRODUCT.Quantity - 5
{Obradi transakciju, pretpostavka da je sve OK}
LOCK PRODUCT
UPDATE PRODUCT
SET PRODUCT.Quantity = NewQuantity
  WHERE PRODUCT.Name = 'Pencil'
AND PRODUCT.Quantity = OldQuantity;
UNLOCK PRODUCT
{Proveri da li je azuiranje uspesno, ako nije, ponovi transakciju}
```

Kod pesimističnog zaključavanja, pretpostavka je da će se konflikt pojaviti. Najpre se vrši zaključavanje, zatim obrađuje transakcija, a zatim se vrši otključavanje.

Sledi primer pesimističnog zaključavanja za istu transakciju.

```
LOCK PRODUCT
SELECT PRODUCT.Name, PRODUCT.Quantity
  FROM PRODUCT
WHERE PRODUCT.Name = 'Pencil'
Set NewQuantity = PRODUCT.Quantity - 5
{Obradi transakciju, pretpostavka da je sve OK}

UPDATE PRODUCT
SET PRODUCT.Quantity = NewQuantity
  WHERE PRODUCT.Name = 'Pencil'
AND PRODUCT.Quantity = OldQuantity
UNLOCK PRODUCT
{Nema potrebe proveriti da li je azuiranje uspesno}
```

Prednost optimističnog zaključavanja je da zaključavanje traje mnogo kraće nego u slučaju pesimističnog zaključavanja. Ova prednost je naročito značajna ako je granularnost zaključavanja velika - na nivou cele tabele (u ovom slučaju tabele PRODUCT).

Primenom mehanizma zaključavanja uz dvofaznost transakcija, mogu se rešiti sva tri problema konkurentnosti.

OPTIMALNA STRATEGIJA ZAKLJUČAVANJA: COMMIT I ROLLBACK

Naredbe Commit i Rollback mogu obezbediti da DBMS uključuje i isključuje zaključavanja i menja nivo i tip zaključavanja.

Ponekad optimalna strategija zaključavanja zavisi od toga koje su transakcije aktivne i šta one rade. Zbog toga se **za aplikacije koje rade nad bazom podataka ne može definisati generalna strategija zaključavanja**. Umesto toga se mogu naznačiti granice transakcija i za njih deklarirati tip zaključavanja. Na taj način RDBMS može da uključuje i isključuje

zaključavanja i menja nivo i tip zaključavanja. U te svrhe se koriste komande COMMIT i ROLLBACK, od kojih svaka označava kraj transakcije.

Kao što je poznato:

1. *Naredba COMMIT označava uspešan kraj transakcije i trajan upis efekata svih radnji transakcije u bazu*
2. *Naredba ROLLBACK označava neuspešan kraj transakcije i poništenje svih efekata koje su proizvele radnje te transakcije.*

Tako je transakcija deo aplikacije od početka programa do prve radnje COMMIT ili ROLLBACK, odnosno između dve susedne takve radnje u programu, odnosno od poslednje takve radnje do kraja programa.

Ako se nijedna od radnji COMMIT, ROLLBACK ne uključi u program, onda čitav program predstavlja jednu transakciju; na kraju programa se sistemski generiše jedna komanda COMMIT ako je kraj programa uspešan, odnosno jedna komanda ROLLBACK ako je kraj programa neuspešan. Ugnježdene transakcije u većini sistema nisu moguće.

Jedan primer korišćenja ovih naredbi dat je na sledeći način:

```
BEGIN TRANSACTION
SELECT PRODUCT.Name, PRODUCT.Quantity
      FROM PRODUCT
WHERE PRODUCT.Name = 'Pencil'
Set NewQuantity = PRODUCT.Quantity - 5
{Obradi transakciju}

UPDATE PRODUCT
SET PRODUCT.Quantity = NewQuantity
  WHERE PRODUCT.Name = 'Pencil'
  AND PRODUCT.Quantity = OldQuantity ....
IF transakcija je normalno završena
  THEN .....
      COMMIT transakcije
ELSE
      ROLLBACK transakcije
END IF
```

▼ Poglavlje 8

Pokazna vežba

NAČIN ORGANIZACIJE POKAZNIH VEŽBI

Vežba je organizovana kroz uvod deo i deo za samostalni rad studenata

Vežba je organizovana kroz uvod deo i deo za samostalni rad studenata.

1. U uvodnom delu pokaznih vežbi se daje pokazni primer koji studentima treba da pomogne u samostalnom rešavanju zadataka.
2. Zadatke koji su zadati za samostalni rad student samostalno rešava uz pomoć asistenta.

▼ 8.1 Pokazni primer - 1 deo

OPORAVAK BAZA PODATAKA MYSQL (5 MIN.)

Ako radite sa MySQL bazama podataka i imate podatke koje ne želite izgubiti, veoma je važno da redovno pravite sigurnosne kopije vaših podataka (backup).

Oporavak baze podataka je vraćanje baze podataka u korektno stanje posle nekog otkaza. Mogući otkazi su npr. prestanak napajanja, oštećenja diskova na kojima se čuva baza, softverske greške.

Da bi se oporavak baze podataka mogao da izvrši neophodno je da SUBP obezbedi neke redundantne podatke. Jedan skup redundantnih podataka se čuva u logu ili žurnalu. Log datoteka se koristi za otkaze koji fizički ne oštećuju memorijske jedinice (diskove) na kojima se čuva baza, npr. kod pada sistema ili pada transakcije. Drugi skup redundantnih podataka se čuva u tzv. arhivskoj memoriji, u koju se povremeno prenosi sadržaj čitave baze podataka (Database Backup). Arhivska memorija služi za oporavak posle otkaza memorijske jedinice.

Potrebno je zapamtiti da se povratak baze (backup baze) pravi često. To je najsigurniji način da povratimo podatke. Pogotovo je potrebno ukoliko se često vrši izmena nad bazom i samim ti dolazi do mogućnosti oštećenja baze.

Ako radite sa MySQL bazama podataka i imate podatke koje ne želite izgubiti, veoma je važno da redovno pravite sigurnosne kopije vaših podataka (backup) kako bi se zaštitili od

gubitka podataka. Ovaj tutorial će vam pokazati dva jednostavna načina za backup i vraćanje podataka u MySQL bazu podataka. Takođe možete koristiti ovaj postupak kako bi premestili svoje podatke na novi web server.

1. Backup iz komandne linije (koristeći mysqldump)
2. Backup MySQL baza podataka sa kompresijom
3. Vraćanje MySQL baze podataka
4. Backup i vraćanje podataka (restore) pomoću phpMyAdmin

BACKUP MYSQL BAZE PREKO COMMAND PROMPT-A (7 MIN.)

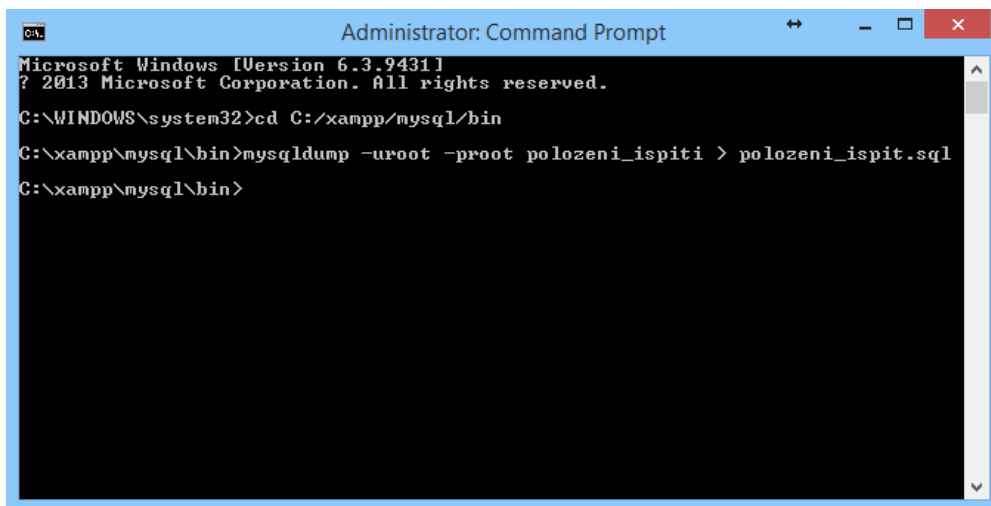
Koristi se klijentski program Mysqldump.

MySql dump klijent je program za backup-ovanje baze originalno napisan od strane Igora Romenaenkovog. MySQL dump može generisati i CSV i XML i naravno SQL fajl.

Spisak komandi može se videti: <http://dev.mysql.com/doc/refman/5.1/en/mysqldump.html>.

Da bi smo backup-ovali bazu podataka preko mysqldump-a (slika 9.1) korišćemo command prompt pri čemu:

1. *u* označava user name (root)
2. *p* je sifra (root u ovom slučaju isto)
3. *Polozeni_ispiti* je ime baze
4. *> polozeni_spit.sql* je file u koji se snima baza.



```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9431]
? 2013 Microsoft Corporation. All rights reserved.
C:\WINDOWS\system32>cd C:\xampp\mysql\bin
C:\xampp\mysql\bin>mysqldump -uroot -proot polozeni_ispiti > polozeni_ispit.sql
C:\xampp\mysql\bin>
```

Slika 8.1.1 Backup baze podataka u MySql-u [Izvor: NM IT350 - 2020/2021.]

DA LI TREBA ČUVATI BAZE ONLINE ILI OFFLINE? (5 MIN.)

Online backup se pravi kada je MySQL server pokrenut; Offline backup se pravi kada je server nedostupan.

Online backup se pravi kada je MySQL server pokrenut, tako što se informacije iz baze dobiju sa servera. Što se tiče offline backup-a, on se pravi i kada je server stopiran. Postoji i naziv kao što je "hot" iliti topao backup i "cold" iliti hladan backup.

Kod online backup-a potrebno je određeno zaključavanje zbog modifikacija kako ne bi kompromitovale integritet backup-a.

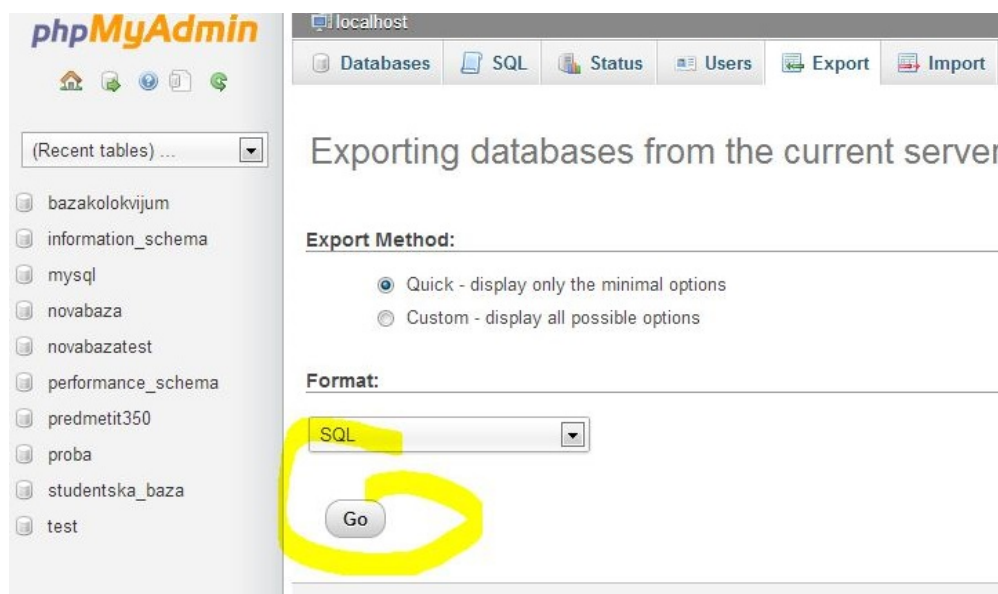
Kod offline backup-a klijenti ne mogu negativno da utiču jer je server nedostupan tokom backup-a i procedura je prostija jer ne dolazi do mešanja aktivnosti klijenta. Kada se vrši backup offline, baza se samo skladišti i ne izvršavaju se u isto vreme neke izmene.

Backup se može izvršiti na mnoštvo načina. Ukoliko koristimo phpMyAdmin može uz pomoć eksportovanja baze. Takođe, možemo koristiti i skripte koje će svakog dana sa servera automatski da backup-uju našu bazu itd.

PHP MY ADMIN BACKUP-SLUČAJ I (7 MIN.)

Slučaj kada postoji samo jedna baza.

Ukoliko postoji samo jedna baza koju želimo da backup-ujemo iz PHP MyAdmin-a, u tom slučaju se baza ne bira, već se samo klikne na export a zatim izabere format u kojem želimo da eksportujemo bazu (SQL, CSV, LATEX itd...). U ovom slučaju biramo SQL. Nakon toga kliknemo na dugme GO (slika 9.2).



Slika 8.1.2 PHP My Admin backup za samo jednu bazu [Izvor: NM IT350 - 2020/2021.]

PROBLEMI PRI KONKURENTNOM IZVRŠENJU SKUPA TRANSAKCIJA (5 MIN.)

Nekonzistentna analiza, izgubljeno ažuriranje i zavisnost od poništenog ažuriranja.

Kako bazu podataka koristi istovremeno veći broj korisnika, to znači da se nad bazom istovremeno može izvršavati više transakcija. Mada svaka od njih može ispunjavati uslove integriteta baze podataka, ponekad njihovo istovremeno izvršenje može da naruši te uslove.

Pri konkurentnom (istovremenom) izvršenju skupa transakcija mogu nastati sledeći problemi:

- *Nekonzistentna analiza*
- *Izgubljeno ažuriranje*
- *Zavisnost od poništenog ažuriranja*

Prva dva problema konkurentnosti vezana su za redosled konkurentnog izvršavanja radnji dve ili više transakcija.

Treći problem je, osim za konkurentnost, vezan i za oporavak baze podataka pri poništavanju transakcije u konkurentnom izvršenju. Do poništavanja transakcije može doći, na primer, zbog greške u transakciji ili zbog pada sistema.

FIZIČKI I LOGIČKI BACKUP MYSQL BAZE (8 MIN.)

Backup sadrži kopije direktorijuma i fajlova baze podataka.

Metode fizičkog backup-a imaju sledeće karakteristike:

1. *Backup sadrži kopije direktorijuma i fajlova baze podataka. Obično je ova kopija deo MySQL direktorijuma podataka. Podatak iz MEMORY tabele ne može biti backup-ovana na ovom načinu jer njihov sadržaj se ne čuva na disku.*
2. *Metode fizičkog backup-a su brže od logičkog zato što samo uključuju samo kopiranje bez konvertovanja.*
3. *Izlaz je više kompaktan nego kod logičkog backup-a.*
4. *Backup-ovanje i povratak podrazumeva povratak granularnosti direktorijuma od direktorijuma do fajlova.*
5. *Može sadržati i fajlove kao što su log i konfiguracioni fajlovi za razliku od logičkog.*

Logički povratak baze - ili kako je već kod nas prihvaćeno logički backup, snima informacije prezentovane u logičku strukturu baze podataka (CREATE DATABASE, CREATE TABLE upita) i sadržaja (INSERT upita).

Što se tiče logičkog povratka baze podataka ima nekoliko karakteristika:

1. *Backup se izvršava tako što se izvršavaju upiti nad MySQL serverom kako bi se dobila struktura baze podataka i sadržaj baze podataka*
2. *Backup je sporiji od fizičkih metoda zato što server mora da pristupi informacijama iz baze podataka i konvertovati ga u logički format. Ukoliko je izlaz pisan na klijentskom delu, server mora takođe da pošalje do programa za izvršavanje backup.*
3. *Izlaz mora biti veći od fizičkog backup-a, posebno kada se čuva u tekstualnom formatu.*
4. *Backup ne sadrži logove ili konfiguracione fajlove, ili druge fajlove koji nisu delovi baze podataka nad kojim se vrši backup.*
5. *Kako bi se vratio logički backup, dump fajlovi u SQL formatu moraju biti procesirani korišćenjem MySQL klijenta.*

To znači da ukoliko koristimo MySQL bazu a format backup-a je SQL obavezno uz pomoć MySQL klijenta je potrebno importovati bazu. Moramo da imamo mesta gde sačuvamo bazu. Vrlo često ukoliko je baza kompresovana (zip, rar) nakon otvaranja kompresije fajl baze podataka zauzimati dosta više nego kompresovani fajl.

PHP MY ADMIN BACKUP-SLUČAJ II (9 MIN.)

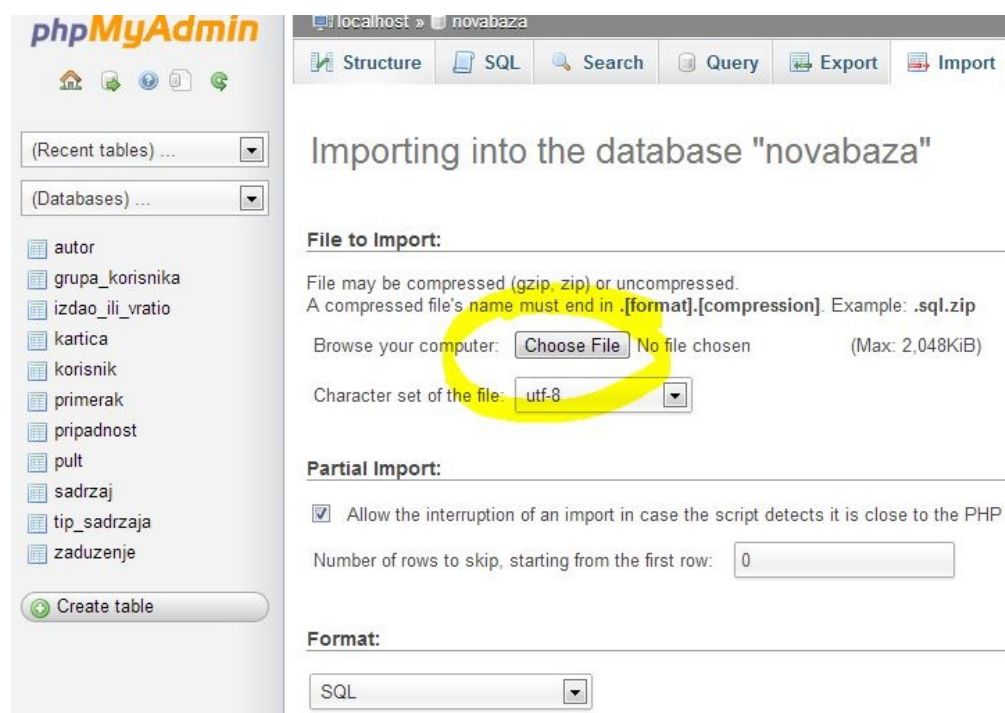
Slučaj kada postoji više baza.

Ukoliko postoji više baza a želimo samo jednu bazu da backup-ujemo, prvo izaberemo bazu a nakon toga kliknemo na export (slika 9.3). Nakon toga treba primeniti prethodno opisanu proceduru. U ovom slučaju smo backup-ovali samo jednu bazu.

Ukoliko želimo da backup-ujemo samo jednu tabelu, biramo tabelu tabelu koju želimo.

Ukoliko je baza velika, preporučuje se zipovanje baze.

Kada želimo da uvezemo (importujemo) bazu, potrebno je prvo predhodnu bazu obrisati a nakon toga importovati nov sadržaj baze.



Slika 8.1.3 Backup-ovanje baze eksportovanjem njenog sadržaja [Izvor: NM IT350 - 2020/2021.]

▼ 8.2 Pokazni primer - II deo

KORIŠĆENJE NAREDBE COMMIT (15 MIN.)

Služi za trajno čuvanje podataka u bazi podataka nakon zadnje izvršenje transakcije (insert, delete, update).

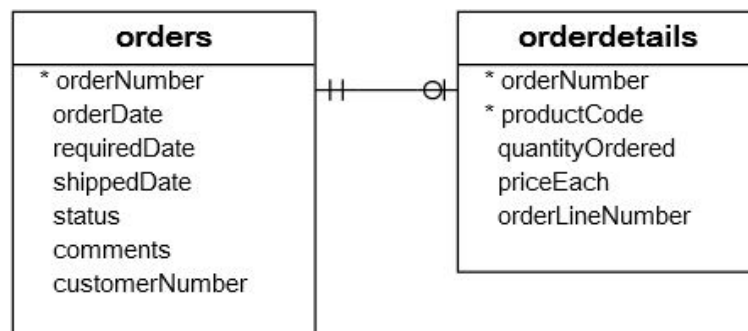
Prema podrazumevanim podešavanjima, MySQL automatski izvršava svaku naredbu pojedinačno i trajno čuva podatke u bazi. Da bi smo ovo zaobišli, potrebno je pre početka transakcije izvršiti naredbu.

```
SET autocommit = 0;
ili
SET autocommit = OFF;
```

Na kraju transakcije, potrebno je omogućiti autocommit opciju naredbom

```
SET autocommit = 1;
ili
SET autocommit = ON;
```

Primer COMMIT-a u MySQL: Nad datom bazom izvršiti transakciju unosa podataka u obe tabele prikazane na slici 1.



Slika 8.2.1 Tabele kad kojima se izvršava komanda COMMIT [Izvor: NM IT350 - 2020/2021.]

```
SET autocommit = 0;
START TRANSACTION;

SELECT
    @orderNumber:=MAX(orderNumber)+1
FROM
    orders;

INSERT INTO orders(orderNumber, orderDate, requiredDate,
                  shippedDate, status, customerNumber)
VALUES(@orderNumber, '2019-05-31', '2019-06-10', '2019-06-11',
      'In Process', 145);

INSERT INTO orderdetails(orderNumber, productCode,
                        quantityOrdered, priceEach,
                        orderLineNumber)
VALUES(@orderNumber, 'S18_1749', 30, '136', 1),
      (@orderNumber, 'S18_2248', 50, '55.09', 2);

COMMIT;
SET autocommit = 0;
```


KORIŠĆENJE NAREDBE ROLLBACK (15 MIN.)

Vraća sadržaj baze na stanje koje je postajalo pre izvršenja transakcije (u trenutku zadnjeg izvršenog commit-a).

Primer 1: Prost primer korišćenja naredbi ROLLBACK u MySQL-u je dat u nastavku:

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT]
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHANI] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHANI] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

Primer 2: Korišćenje naredbe ROLLBACK sa SAVEPOINT (CHECKPOINT) mogućnošću prikazan je nad tabelom tabela123. Kreiranje tabele table123 nad kojoj se vrši ROLLBACK prikazan je na slici 4. dok je na slici 5. prikazano adekvatno korišćenje naredne ROLLBACK za primer 2.

```
CREATE TABLE test (id INT NOT NULL PRIMARY KEY) ENGINE=InnoDB;
```

Opis procedure za korišćenje naredne ROLLBACK sa SAVEPOINT u primeru 2:

Počinjemo sesiju, pravimo savepoint i pozivamo ga. Kod je:

```
SET autocommit = 0;
START TRANSACTION;
INSERT INTO tabela123VALUES (1);
SELECT *FROM tabela123;
SAVEPOINT tran2;
INSERT INTO tabela123VALUES (2);
SELECT *FROM tabela123;
ROLLBACK TO tran2;
SET autocommit = 1;
```

Koji sadržaj baza podataka ima nakon izvršenja ovih naredbi?

PRIMER TRANSKCIJE U PHP-U (15 MIN.)

Puna moć transakcija moguća je tek kada se primeni sa nekim programskim jezikom

Zadatak: Kreirati program koji će vršiti transfer novca sa jednog naloga na drugi.

Rešenje: Da bi rešili zadatak, potrebna nam je tabela koja će čuvati naloge i količinu novca korisnika.

```
CREATE TABLE accounts (
    id      INT AUTO_INCREMENT PRIMARY KEY,
    name    VARCHAR (50)      NOT NULL,
```

```
amount DECIMAL (19, 4) NOT NULL
);
```

Za test primer, dodajemo i dva zapisa u tabelu *accounts*

```
INSERT INTO accounts(name,amount)
VALUES('Pera',25000),
      ('Arsa',95000)
```

Zadatak ćemo rešiti upotrebom OO PHP-a. Kreiramo klasu *TransactionDemo* koja će sadržati metodu ***transfer(idFrom, idTo, amount)*** za prenos sredstava sa jednog naloga na drugi. Prilikom kreiranja objekta ove klase, otvaramo konekciju nad bazom koja sadrži tabelu ***accounts***, dok prilikom uništavanja zatvaramo konekciju.

```
<?php

/**
 * PHP MySQL Transaction Demo
 */
class TransactionDemo {

    const DB_HOST = 'localhost';
    const DB_NAME = 'test';
    const DB_USER = 'root';
    const DB_PASSWORD = 'root';

    /** Prilikom konstruisanja objekta, otvaramo konekciju nad bazom podataka
     * Za konekciju, koristimo PDO
     */
    public function __construct() {
        // Otvaranje konekcije
        $conStr = sprintf("mysql:host=%s;dbname=%s", self::DB_HOST, self::DB_NAME);
        try {
            $this->pdo = new PDO($conStr, self::DB_USER, self::DB_PASSWORD);
        } catch (PDOException $e) {
            die($e->getMessage());
        }
    }

    private $pdo = null;

    /**
     * Transfer novca sa jednog naloga na drugi
     * @param int $from
     * @param int $to
     * @param float $amount
     * @return true ako je uspešan transfer ili false ukoliko nije.
     */
    public function transfer($from, $to, $amount) {
        try {
            $this->pdo->beginTransaction();
```

```

        // Provera da li na nalogu sa koga isplaćujemo
        // ima dovoljno novca za prenos
        $sql = 'SELECT amount FROM accounts WHERE id=:from';
        $stmt = $this->pdo->prepare($sql);
        $stmt->execute(array(":from" => $from));
        $availableAmount = (int) $stmt->fetchColumn();
        $stmt->closeCursor();

        if ($availableAmount < $amount) {
            echo 'Nedovoljno sredstava za prenos!';
            return false;
        }
        // smanjujemo količinu novca sa naloga sa koje isplaćujemo novac
        $sql_update_from = 'UPDATE accounts SET amount = amount - :amount WHERE
id = :from';
        $stmt = $this->pdo->prepare($sql_update_from);
        $stmt->execute(array(":from" => $from, ":amount" => $amount));
        $stmt->closeCursor();

        // Dodajemo novac na nalog na koji prima uplatu
        $sql_update_to = 'UPDATE accounts SET amount = amount + :amount WHERE
id = :to';
        $stmt = $this->pdo->prepare($sql_update_to);
        $stmt->execute(array(":to" => $to, ":amount" => $amount));

        // commit transakcije
        $this->pdo->commit();

        echo 'Uspešan prenos sredstava';

        return true;
    } catch (PDOException $e) {
        // Ukoliko dođe do greške, uradićemo rollback na početno stanje oba
računa
        $this->pdo->rollBack();
        die($e->getMessage());
    }
}

// Zatvaramo konekciju nad bazom prilikom uništavanja objekta
public function __destruct() {
    $this->pdo = null;
}
}

// Kreiramo test objekat
$obj = new TransactionDemo();

// Sa naloga sa id-jem 1 vršimo transfer novca na nalog sa id-jem 2
$obj->transfer(1, 2, 30000);

```

```
// Sa naloga sa id-jem 1 vršimo transfer novca na nalog sa id-jem 2  
$obj->transfer(1, 2, 5000);
```

▼ 8.3 Zadatak za samostalni rad

TEKST ZADATKA (45 MIN.)

Samostalno uradite sledeći zadatak:

Nad studentskom bazom podataka koja je korišćena u prethodnim domaćim zadacima (sastoji se od tabela DOSIJE, ISPIT, PREDMET i ISPITNI_ROK), napisati transakciju kojom će se izvršiti sledeće naredbe:

1. Napisati naredbu kojom se u tabelu DOSIJE dodaju 5 novih redova. **(7 minuta)**
2. Napisati naredbu kojom se u tabelu ISPIT dodaju 6 novih redova. **(8 minuta)**
3. Napisati naredbu kojom se u tabelu ISPITNI_ROK dodaju 3 nova reda. **(6 minuta)**
4. Napisati naredbu kojom se u tabelu PREDMET dodaju 5 novih redova. **(6 minuta)**
5. Potvrditi transakcije po izvršavanju svih naredbi, ako se naredbe uspešno izvrše. **(6 minuta)**
6. Poništite naredbe po izvršavanju, ako niste zadovoljni rezultatom. **(6 minuta)**
7. Uradite backup baze podataka po izvršenju naredbi. **(6 minuta)**

▼ Poglavlje 9

Domaći zadatak

DOMAĆI ZADATAK 11 - VREME IZRADE 150 MIN.

Uradite sledeći zadatak

Nad bazom podataka koju ste kreirali u DZ08, poštujući referencijalne integritete, napisati transakcije kojima će se izvršite sledeće naredbe:

1. Naredbu kojom se u jednu od tabela dodaje nova kolona koja ne može biti NULL. Definirati podrazumevanu vrednost ove kolone.
2. Naredbu kojom se u jednu od tabela briše jedna od kolona .
3. Naredbu kojom ćete update-ovati vrednost neke kolone u nekoj od tabela.
4. Potvrdite transakcije po izvršenju naredbi, ako ste zadovoljni rezultatom.
5. Poništite transakciju po izvršavanju naredbi, ako niste zadovoljni rezultatom.
6. Uradite backup baze podataka po izvršenju naredbi

Rešenje zadatka snimate u dokument IT250-DZ11-BrIndeksa-Ime_Prezime.sql, gde su Ime, Prezime i brIndexa vaši podaci.

Prilikom slanja domaćih zadatka, neophodno je da ispunite sledeće:

- Subject mail-a mora biti IT250-DZbr (u slučaju kada šaljete domaći za ovu nedelju to je IT250-DZ11)
- U prilogu mail-a treba da se nalazi arhiviran projekat koji se ocenjuje imenovan na sledeći način: IT250-DZbr-BrojIndeksa-Ime Prezime. Na primer, IT250-DZ11-1234-VeljkoGrkovic
- Telo mail-a treba da ima pozdravnu poruku
- Arhivu sa zadatkom poslati na adresu predmetnog asistenta:
milica.vlajkovic@metropolitan.ac.rs (studenti u Beogradu i online studenti) ili
tamara.vukadinovic@metropolitan.ac.rs (studenti u Nišu).

Svi poslati mail-ovi koji ne ispunjavaju navedene uslove NEĆE biti pregledani. Za sva pitanja ili nedoumice u vezi zadatka, možete se obratiti asistentu

▼ Zaključak

ZAKLJUČAK

Šta smo naučili u ovoj lekciji?

U ovom predavanju se govorilo o načinu na koji se može izvršiti oporavak baza podataka i to u slučaju pada transakcije ili pada sistema.

Izvršenje jedne transakcije može da se završi planirano ili neplanirano. Do planiranog završetka dolazi izvršenjem COMMIT operacije kojom se uspešno kompletira transakcija, ili eksplicitne ROLLBACK operacije, koja se izvršava kada dođe do greške za koju postoji programska provera.

Neplanirani završetak izvršenja transakcije dovodi do izvršenja implicitne (sistemske) ROLLBACK operacije, radnje transakcije se poništavaju a program prekida sa radom. U slučaju pada sistema, sadržaj unutrašnje memorije je izgubljen. Zato se, po ponovnom startovanju sistema, za oporavak koriste podaci iz log datoteke da bi se poništili efekti transakcija koje su se obrađivale u trenutku pada sistema.

Kontrola konkurentnosti je takođe jedno od važnih pitanja u analizi baza podataka. Problemi koji mogu nastati zbog nedovoljne kontrole su nekonzistentna analiza, izgubljeno ažuriranje, zavisnost od poništenog ažuriranja o čemu se u predavanju govori kroz navođenje odgovarajućih primera. Problem konkurentnosti se rešava zaključavanjem koje može biti optimistično i pesimistično.

LITERATURA

Za pisanje ove lekcije je korišćena sledeća literatura:

1. David M. Kroenke, Database Processing – fundamentals, design and implementation, Prentice Hall, 2004.
2. C. J. Date, An introduction to Database Systems, Addison-Wesley Publishing Company, 1990