Predavanje IX.

Transakcije

BAZE PODATAKA II

doc. dr. sc. Goran Oreški
Fakultet informatike,
Sveučilište Jurja Dobrile, Pula

Sadržaj

- ponavljanje prethodnih predavanja
  - optimizacija upita
  - pravila ekvivalencije
  - procjena troška
  - odabir evaluacijskog plana
- transakcije
- ACID svojstva
- stanje transakcije
- istodobno izvršavanje
- rasporedi
- serijalizacija
- kontrola istodobnosti
- transakcije u SQL-u

Ponavljanje

- optimizacija upita je proces odabira najučinkovitijeg plana za evaluaciju upita između mnogo strategija koje su moguće za određeni upit
- generiranje planova evaluacije upita uključuje tri koraka:
  1. generiranje izraza koji su logički ekvivalentni zadanom izrazu
  2. označavanje generiranih izraza na različite načine da bi se dobili različiti planovi evaluacije
  3. odabir onog plana koji ima najmanji procijenjeni trošak
- dva aspekta optimizacije:
  1. na razini relacijske algebre
  2. odabir detaljne strategije evaluacije

Ponavljanje

- pravila ekvivalencije
  - definiraju ekvivalentnost dva izraza
  - ekvivalentna ali ne i jednako skupa (cilj pravila)
- primjeri osnovnih pravila
- korištenje pravila
  - više pravila se mogu koristiti jedno iza drugoga
- pronalaženje ekvivalentnih izraza
  - optimizacija transformacije izraza
    - prostorna
    - vremenska

Ponavljanje

- procjena troška
  - stvarni trošak != procijenjeni trošak
- statistike relacija i indeksa
  - informacije kataloga
- odabir evaluacijskog plana
  - odabir algoritama
  - uzeti u obzir interakciju evaluacijskih tehnika
- dva načina odabira
  - troškovna optimizacija
  - heuristika

Transakcije

- transakcije se definiraju kao **jedinice** posla koje pristupaju i (ponekad) mijenjaju podatke u bazi podataka
- transakcije se sastoje od skupa operacija koje čine logičku cjelinu posla
- primjer transakcije:
  - „prebacivanje novca” ili T_1
  - prebacivanje 50kn s računa A na račun B
  - koriste se dvije operacije *read* i *write* (*nije SQL jezik*)

T_1: **read**(A)
A := A – 50
write(A)
read(B)
B := B + 50
write(B)

Transakcije

- prilikom izvođenja transakcija pojavljuju se dva izazova:
  1. različiti razlozi prekida transakcije
  2. istodobno izvršavanje više transakcija
- rezultat može biti **nekonzistentno stanje** (*engl. inconsistent state*), tj. stanje koje ne odgovara stvarnom stanju svijeta kojeg baza predstavlja
- u SQL jeziku transakcija se može definirati pomoću bloka:

```
begin transaction
   operacije…
end transaction
```

Transakcije

- vratimo se na transakciju prebacivanja novca s računa A na račun B

T_1:
1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)

- *zahtjev za nedjeljivosti transakcije*, atomičnost (*engl. atomicity*)
  - ukoliko transakcija pukne nakon 3. koraka a prije 6. koraka rezultat će biti "izgubljen novac", tj. *nekonzistentno stanje*
    - razlog pucanja može biti zbog hardware-a ili software-a
  - sustav treba osigurati da update parcijalno izvršenih transakcije ne bude evidentiran u bazi podataka

Transakcije

- zahtjev za trajnosti transakcije (*engl. durability*)
  - jednom kada je korisnik obaviješten o izvršenju transakcije (tj. da su novci prebačeni) novo stanje baze podataka mora biti trajno zapisano čak i ako se dogodi neki kvar
- zahtjev za konzistentnosti (*engl. consistency*)
  - u promatranom primjeru, suma računa A i računa B mora biti ostati nepromijenjena nakon izvršavanja transakcije
  - u pravilu zahtjev za konzistentnosti uključuje:
    - eksplicitno definirana ograničenja integriteta (primarni ključ, strani ključ…)
    - implicitna ograničenja integriteta (suma računa…)
- transakcija kada započne izvršavanje mora vidjeti konzistentnu bazu
- tijekom izvršavanja stanje može biti privremeno nekonzistentno
- nakon izvršavanja baza mora biti u konzistentnom stanju

Transakcije

- zahtjev za izolacijom (*engl. isolation*)
  - ukoliko između koraka 3. i 6. druga transakcija T_2 ima dostup do parcijalno izmijenjene baze podataka, vidjeti će nekonzistentno stanje
    - suma A+B će biti manja nego što treba biti

T_1:		T_2:
1.	read(A)	
2.	A := A – 50	
3.	write(A)	
		read(A), read(B), print(A+B)
4.	read(B)	
5.	B := B + 50	
6.	write(B)	

- izolacija trivijalno može biti osigurana izvođenjem transakcija serijski
- međutim istodobno izvođenje transakcija ima značajne prednosti

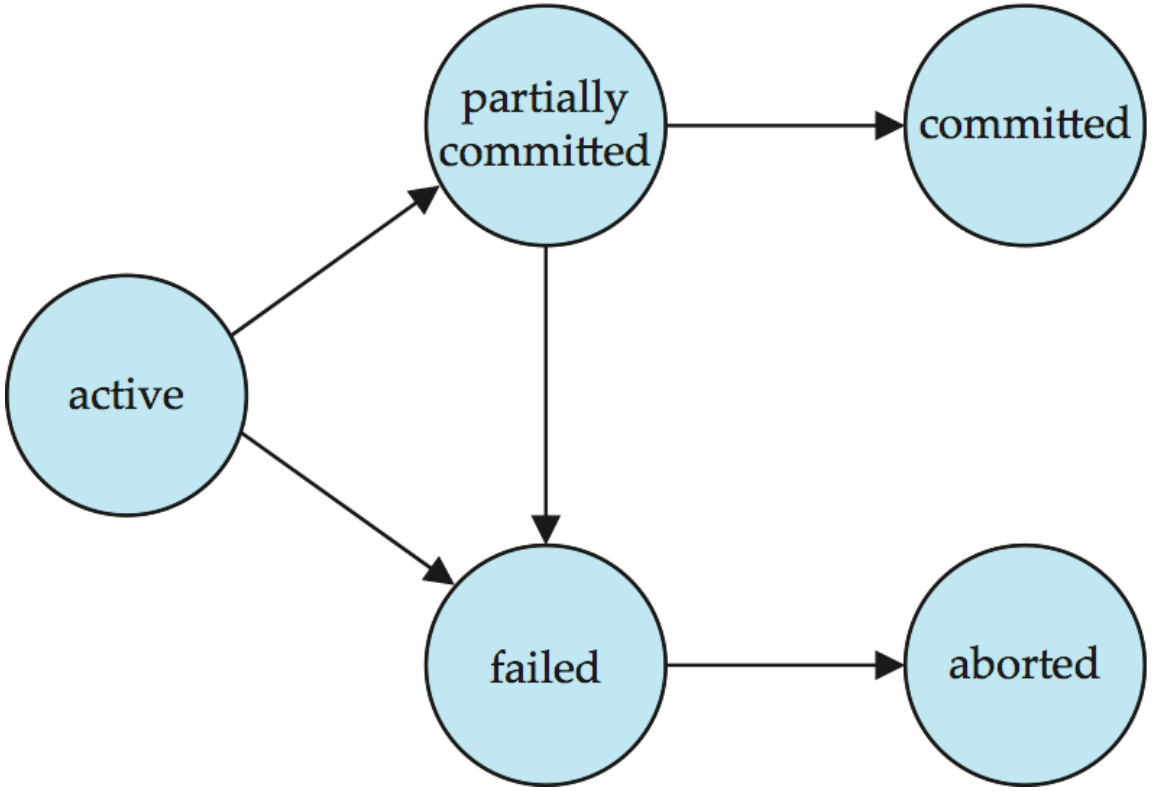
ACID svojstva

- da bi se zaštitio integritet podataka sustav baze podataka mora osigurati:
  - nedjeljivost transakcije: ili su sve operacije transakcije uspješno izvršene u bazi podataka ili nije niti jedna
  - konzistentnost: izvršavanje transakcije u izolaciji održava konzistentnost baze podataka
  - izolacija: iako se mnoge transakcije mogu izvršiti istodobno, transakcija ne smije biti svjesna drugih transakcija koje se izvršavaju u istom vremenu
    - međurezultati moraju biti skriveni od drugih istodobnih transakcija
    - tj. za svaki par transakcija T_i i T_j, transakciji T_i se čini ili da je transakcija T_j završila prije nego što je T_i počela ili je T_j počela izvršavanje nakon što je T_i završila
  - trajnost: nakon što transakcija uspješno završi, promjene koje su u bazi učinjene su trajne čak i ako se sustav sruši

Stanje transakcije

- **aktivna** – inicijalno stanje; transakcija ostaje u tom stanju dok se izvršava
- **djelomično potvrđena** (engl. partially committed) – nakon što je zadnja naredba izvršena
- **neuspješna** (engl. failed) – nakon otkrivanja da se normalno izvršavanje ne može nastaviti
- **prekinuta** – nakon što je transakcija poništena (engl. rolled back) i baza podataka je vratila stanja prije pokretanja transakcije
  - mogućnosti: ponovno pokrenuti ili odustati (engl. kill the transaction)
- **potvrđena** (engl. commited) – uspješno izvođenje

Stanje transakcije



Slika 17.1 [DSC]

Istodobno izvršavanje

- više transakcija se mogu istodobno izvršavati na sustavu; prednosti su:
  - povećanje iskoristivosti diska i procesora, što dovodi do bolje propusnosti (*engl. throughput*)
  - smanjenje prosječnog vremena potrebnog za izvršavanje transakcije, kratke transakcije ne moraju čekati iza dugih
- sheme za kontrolu istodobnosti (*engl. concurrency control schemes*) – mehanizmi za postizanje izolacije
  - kontrola interakcije između istodobnih transakcija s ciljem očuvanja konzistentnosti baze podataka
  - na slijedećem predavanju

Rasporedi

- **raspored** (izvođenja) – sekvence instrukcija koje definiraju kronološki redoslijed izvođenja instrukcija unutar istodobnih transakcija
  - raspored za skup transakcija mora sadržavati sve instrukcije istih transakcija
  - redoslijed u kojem se pojavljuju instrukcije unutar transakcija mora biti očuvan
- transakcija koja uspješno završi izvođenje sadrži commit kao zadnju naredbu
- transakcija koja neuspješno završi izvođenje sadrži abort kao zadnju naredbu

Raspored 1

- uzmimo da T_1 transakcija prebacuje 50 kuna s računa A na račun B, a da transakcija T_2 prebacuje 10% iznosa s računa A na B
- prikazan je **serijski** raspored u kojem se prvo izvršava T_1 transakcija te potom T_2 transakcija

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Slika 17.2 [DSC]

Raspored 1

- uzmimo da T_1 transakcija prebacuje 50 kuna s računa A na račun B, a da transakcija T_2 prebacuje 10% iznosa s računa A na B
- prikazan je **serijski** raspored u kojem se prvo izvršava T_1 transakcija te potom T_2 transakcija

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

A_{krajT1} = ?
B_{krajT1} = ?

A_{krajT2} = ?
B_{krajT2} = ?

Slika 17.2 [DSC]

Raspored 1

- uzmimo da T_1 transakcija prebacuje 50 kuna s računa A na račun B, a da transakcija T_2 prebacuje 10% iznosa s računa A na B
- prikazan je **serijski** raspored u kojem se prvo izvršava T_1 transakcija te potom T_2 transakcija

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

A_{krajT1} = 950 kn
B_{krajT1} = 150 kn

A_{krajT2} = ?
B_{krajT2} = ?

Slika 17.2 [DSC]

Raspored 1

- uzmimo da T_1 transakcija prebacuje 50 kuna s računa A na račun B, a da transakcija T_2 prebacuje 10% iznosa s računa A na B
- prikazan je **serijski** raspored u kojem se prvo izvršava T_1 transakcija te potom T_2 transakcija

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

A_{krajT1} = 950 kn
B_{krajT1} = 150 kn

A_{krajT2} = 855 kn
B_{krajT2} = 245 kn

Slika 17.2 [DSC]

Raspored 2

- prikazan je **serijski** raspored u kojem se prvo izvršava T_2 transakcija te potom T_1 transakcija

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Slika 17.3 [DSC]

Raspored 2

- prikazan je **serijski** raspored u kojem se prvo izvršava T_2 transakcija te potom T_1 transakcija

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

A_{krajT2} = ?
B_{krajT2} = ?

A_{krajT1} = ?
B_{krajT1} = ?

Slika 17.3 [DSC]

Raspored 3

- prikazani raspored nije **serijski** raspored ali je ekvivalentan rasporedu 1
- raspored 1, raspored 2 i raspored 3 čuvaju sumu A+B

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Slika 17.4 [DSC]

Raspored 3

- prikazani raspored nije **serijski** raspored ali je ekvivalentan rasporedu 1
- raspored 1, raspored 2 i raspored 3 čuvaju sumu A+B

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Slika 17.4 [DSC]

A_{krajT2} = ?
B_{krajT2} = ?

Raspored 4

- raspored 4 ne čuva sumu A+B

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Slika 17.5 [DSC]

Raspored 4

- raspored 4 ne čuva sumu A+B

A_{start} = 1000 kn
B_{start} = 100 kn

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

A_{krajT2} = ?
B_{krajT2} = ?

Slika 17.5 [DSC]

Serijalizacija

- osnovna pretpostavka – svaka transakcija čuva konzistentnost baze podataka
- stoga, serijsko izvođenje transakcija čuva konzistentnost baze podataka
- raspored je serijabilan ukoliko je ekvivalentan serijskom rasporedu
- različiti forme ekvivalencije rasporeda :
  - conflict serializability
    - forma koju ćemo proučavati u nastavku
  - view serializability

Pojednostavljeni pogled na transakcije

- u suštini transakcije su programi
  - stoga može biti teško, točno odrediti koje operacije transakcija sve izvodi i način interakcije različitih operacija transakcije
- iz tog razloga nećemo razmatrati sve tipove operacija koja transakcija može izvoditi na podatkovnom elementu, nego se usredotočiti na dvije instrukcije: *read*, *write*
- pretpostaviti ćemo da transakcije mogu izvoditi različite operacije na podacima u lokalnim međuspremnicima između *read* i *write* instrukcija

Konfliktne instrukcije

- neka su I_i i I_j dvije slijedne instrukcije transakcija T_i i T_j respektivno, nekog rasporeda S
- instrukcije I_i i I_j su u konfliktu ako i samo ako postoji neki element Q kojem pristupaju I_i i I_j, i barem jedna od tih instrukcija zapisuje (*write*) Q
  1. I_i = **read**(Q), I_j=**read**(Q). I_i i I_j nisu u konfliktu
  2. I_i = **read**(Q), I_j=**write**(Q). I_i i I_j su u konfliktu
  3. I_i = **write**(Q), I_j=**read**(Q). I_i i I_j su u konfliktu
  4. I_i = **write**(Q), I_j=**write**(Q). I_i i I_j su u konfliktu
- ukoliko su instrukcije u konfliktu bitan je raspored između njih
  - obratno nije

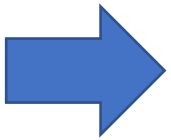
Konflikt serijalizacija

- ukoliko raspored S može biti transformiran u raspored S' nizom zamjena ne-konfliktnih instrukcija, možemo reći da su S i S' ***konflikt ekvivalentni*** (*engl. conflict equivalent*)
- kažemo da je raspored S **konflikt serijabilan** (*engl. conflict serializable*) ukoliko je konflikt ekvivalentan serijskom rasporedu
- da bismo utvrdili da li je neki raspored konflikt ekvivalentan serijskom rasporedu potrebno je pratiti definiciju konflikt instrukcije
  - tj. kada su neke instrukcije konfliktne?
  - i pravilo zamjene

Konflikt serijalizacija

- Raspored 3 može biti transformiran u serijski raspored gdje T_2 slijedi T_1
- transformacija se izvodi nizom zamjena ne-konfliktnih instrukcija
- stoga se može reći da je Raspored 3 konflikt serijabilan

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)



T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Konflikt serijalizacija

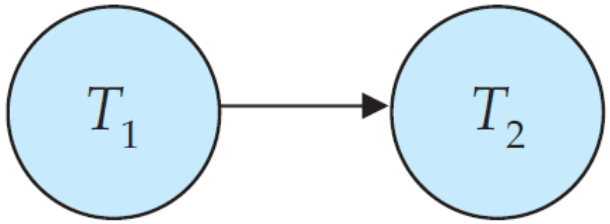
- primjer rasporeda koji nije konflikt serijabilan

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- ne možemo zamijeniti instrukcije u gornjem rasporedu da bi ostvarili serijski raspored <T_3, T_4>, ili serijski raspored <T_4, T_3>

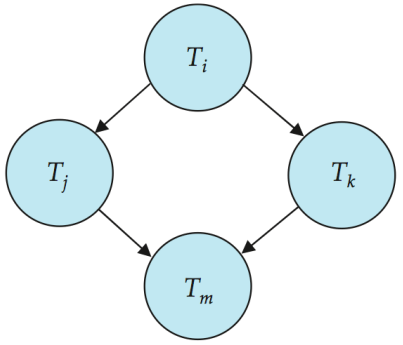
Graf prioriteta

- razmotrimo raspored koji sadrži skup transakcija T_1, T_2,…,T_n
- graf prioriteta (engl. precedence graph) – usmjereni graf u kojem su čvorovi transakcije
- vezu između T_1 i T_2 crtamo ukoliko su dvije transakcije konfliktne i T_1 pristupa podatkovnom elementu na kojem je konflikt nastao
- vezi se može dodati naziv elementa kojem se pristupa

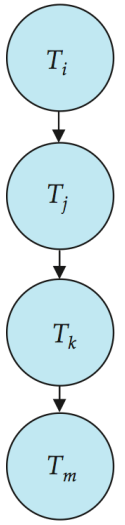


Konflikt serijalizacija

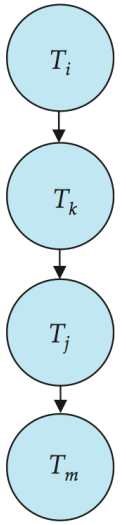
- raspored je konflikt serijabilan ako i samo ako je graf prioriteta acikličan
- da bi testirali konflikt serijalizaciju potrebno je:
  - stvoriti graf prioriteta
  - koristiti algoritam za detekciju ciklusa
- ukoliko je graf prioriteta aciklički, redoslijed serijalizacije se može utvrditi pomoći topološkog sortiranja grafa



(a)



(b)



(c)

Prekid transakcija

- raspored koji omogućava povrat na početno stanje (*engl. recoverable schedule*)
  - ukoliko transakcija T_j čita podatkovni element prethodno zapisan od transakcije T_i, tada se commit operacija transakcije T_i mora pojaviti prije commit operacije T_j
- slijedeći raspored ne omogućava povrat ukoliko T_9 napravi commit nakon *read(A)* operacije

- ukoliko bi se transakcija T_8 prekinula, T_9 transakcija bi koristila (i možda proslijedila korisniku) nekonzistentno stanje, stoga baza podataka mora osigurati da rasporedi omogućavaju povrat

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

Kaskadni rollback

- kaskadni rollback (engl. *cascading rollback*) – neuspjeh jedne transakcije dovodi do serije rollback operacija na različitim transakcijama
- razmotrimo slijedeći raspored u kojem niti jedna transakcija nije napravila commit (recoverable schedule)

- ukoliko se transakcija T_{10} prekine, T_{11} i T_{12} transakcije moraju napraviti rollback
- čime se može dovesti do poništenja značajnog dijela posla

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

Kaskadni rollback

- cascadeless schedules – za svaki par transakcija T_i i T_j, takav da T_j čita podatkovni element prethodno zapisan od T_i , commit operacija transakcije T_i se pojavljuje prije operacije čitanja transakcije T_j
- svaki cascadeless raspored je ujedno i recoverable

- poželjno je ograničiti rasporede na one koji su cascadeless
- primjer rasporeda koji nije cascadeless

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

Kontrola istodobnosti

- sustav baze podataka mora pružiti mehanizam koji osigurava da su mogući rasporedi:
  - conflict serializable
  - recoverable i cascadeless
- strategija, u kojoj se izvodi pojedinačno svaka transakcija stvara serijski raspored, ali ne omogućava viši stupanj istodobnosti
- sheme za kontrolu istodobnosti biraju između količine istodobnosti koje dopuštaju i količine opterećenja kao rezultata te količine
- testiranje serijabilnosti nakon izvršavanja je kasno
- cilj – razviti protokole za kontrolu istodobnosti koji će osigurati serijabilnost

Niska razina konzistentnosti

- neke aplikacije su spremne koristiti niske razine konzistentnosti, dopuštajući rasporede koji nisu serijabilni, na primjer:
  - read-only transakcije koje žele dohvatiti približni ukupni saldo svih računa
  - statistika baze podataka koja se računa za optimizaciju upita može biti približna
  - takve transakcije ne moraju biti serijabile u odnosu na druge transakcije
- odricanje točnosti za performanse

Razine konzistentnosti prema SQL-92

- serijalizacija – u pravilu osiguravaju serijabilo izvršavanje transakcija (najviša razina izolacije)
- ponovno čitanje (engl. repeatable read) – dopušta čitanje samo potvrđenih podataka, i dodatno između dva čitanja u transakciji ne dozvoljava drugoj transakciji izmjenu podatka
- potvrđene n-torke (engl. read committed) – dopušta čitanje samo potvrđenih podataka, ali commit može napraviti duga transakcija
- prljavo čitanje (engl. read uncommitted) – dopušta čitanje ne potvrđenih podataka
- *sve razine izolacije zabranjuju "prljavo pisanje" (engl. dirty writes)*

Definicija transakcija u SQL-u

- DML uključuje naredbu koja definira skup akcija od kojih se transakcija sastoji
- u SQL, transakcije najčešće počinju implicitno (opcija, može se isključiti)
  - ako je isključeno, transakcija se definira pomoću BEGIN i END bloka
- transakcija završava kao:
  - commit - potvrđuje trenutnu transakciju i započinje novu
  - rollback – rezultira prekidanjem trenutne transakcije
- u većini baza podataka, SQL naredbe (većina njih), radi implicitni commit ukoliko su uspješno završene

Literatura

- Pročitati
  - [DSC] poglavlje 14.
  - prezentacija Poglavlje 14. – Database System Concepts; Silberschatz, Korth i Sudarshan
  - Transaction management Oracle; https://docs.oracle.com/cd/B19306_01/server.102/b14220/transact.htm
- Slijedeće predavanje
  - [DSC] poglavlje 15.
  - prezentacija Poglavlje 15. – Database System Concepts; Silberschatz, Korth i Sudarshan