4. domača naloga pri predmetu Tehnologija upravljanja podatkov

Neža Belej (63120340)

January 4, 2017

1 Prvi del naloge se navezuje na vašo dosedanjo serijo domačih nalog. Za vašo opišite vsaj eno smiselno aktivnost z več opravili (npr. zapis rezultatov preiskav) in jo realizirajte v obliki ene ali več transakcij. Vašo rešitev ustrezno dokumentirajte.

Primer transakcije: Ob postavitvi diagnoze v transakciji vstavimo zapis v tabelo Diagnoza, nato preberemo največjo številko diagnoze v tej obravnavi ter posodobimo prej vstavljen zapis z novo številko diagnoze.

Problem lahko nastane, če sočasno vstavimo dve diagnozi za isto obravnavo. Pri branju največje številke diagnoze se lahko v obeh primerih prebere ista vrednost, kar bi povzročilo, da imata obe diagnozi isto številko diagnoze. To rešimo z ekskluzivnim zaklepanjem SELECT .. FOR UPDATE, ki pisalno zaklene vrstice.

2 Drugi del sloni na nalogah, ki smo jih opravljali na vajah v okviru sočasnega dostopa do PB. Z uporabo večnitnega programiranja v Pythonu simulirajte množico sočasnih dostopov do podatkov. Z uporabo podprogramov iz vaj (po potrebi s popravki) nad podatki iz prejšnjih domačih nalog reproducirajte štiri težave sočasnega dostopa

2.1 Branje nepotrjenega podatka (dirty read)

Pri uporabi izolacijske stopnje Read Uncommited se lahko pri sočasnem dostopu zgodi branje nepotrjenega podatka, kot je to demonstrirano v programu dirty-read.py. Simboličen potek programa je podan v spodnji tabeli:

Time	T1	T2	vrednostPreiskave
t0	begin transaction	begin transaction	
t1	read(VP_{57398})		0
t2		write(VP_{57398}, 10)	10
t3	VP = read(VP_{57398})		10
t4		rollback	0
t5	write(VP_{57398}, VP + 5)		15
t6	commit		15

Težavo lahko rešimo, da v prvi transakciji uporabimo višjo stopnjo izolacije transakcije (vsaj Read Commited).

Time	T1	T2	vrednostPreiskave
t0	begin transaction	begin transaction	
t1	read(VP_{57398})		0
t2		write(VP_{57398}, 10)	10
t3	VP = read(VP_{57398})		0
t4		rollback	0
t5	write(VP_{57398}, VP + 5)		5
t6	commit		5

2.2 Nekonsistentna analiza (non-repeatable read)

Pri uporabi izolacijske stopnje Read Committed ali Read Uncommitted se lahko pri sočasnem dostopu zgodi branje nepotrjenega podatka, kot je to demonstrirano v programu non-repeatable-read.py.

ISOLATION LEVEL READ COMMITED/UNCOMMITED:

Time	T1	T2	vrednostPreiskave
t0	begin transaction	begin transaction	
t1	read(VP_{57398})		0
t2		write(VP_{57398}, 10)	10
t6		commit	10
t4	read(VP_{57398})		10
t6	commit		

ISOLATION LEVEL REPEATABLE READ/SERIALIZABLE:

Time	T1	T2	vrednostPreiskave
t0	begin transaction	begin transaction	
t1	read(VP_{57398})		0
t2		write(VP_{57398}, 10)	10
t6		commit	10
t4	read(VP_{57398})		0
t6	commit		

2.3 Fantomsko branje (phantom read)

Podobno kot pri nekonsistentni analizi, le da se tu uporabi insert namesto update in posledica je nova, fantomska vrstica. Težavo rešimo z najvišjo stopnjo izolacije.

ISOLATION LEVEL READ COMMITED, UNCOMMITED, REPEATABLE READ:

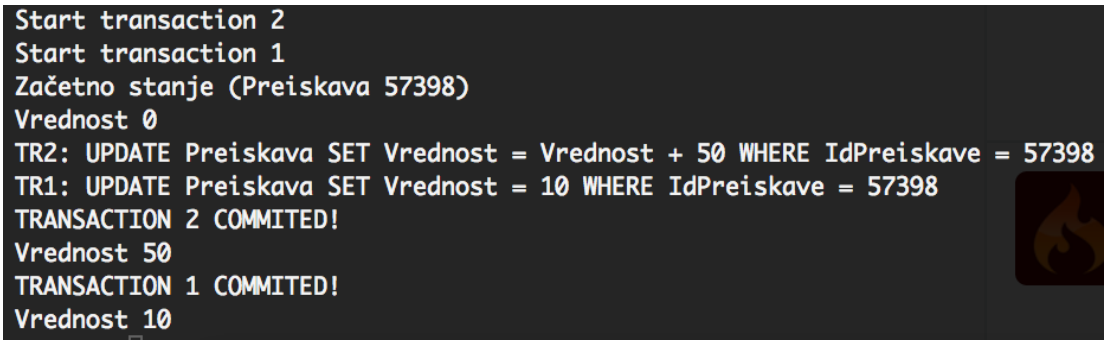
Time	T1	T2	len(preiskave)
t0	begin transaction	begin transaction	
t1	preiskave = read(Preiskava WHERE KZZ = 500587)		3
t2		insert(Preiskava (KZZ = KZZ = 500587))	
t6		commit	
t4	preiskave = read(Preiskava WHERE KZZ = 500587)		4
t6	commit		

ISOLATION LEVEL SERIALIZABLE:

Time	T1	T2	len(preiskave)
t0	begin transaction	begin transaction	
t1	preiskave = read(Preiskava WHERE KZZ = 500587)		3
t2		insert(Preiskava (KZZ = KZZ = 500587))	
t6		commit	
t4	preiskave = read(Preiskava WHERE KZZ = 500587)		3
t6	commit		

2.4 Izgubljeno ažuriranje (lost update)

ISOLATION LEVEL READ COMMITTED.



V tem specifičnem primeru, se pri uporabi SERIALIZABLE nivoja tran-

skacije zgodi deadlock, zato moramo uporabiti FOR UPDATE zaklep pri branju v drugi transakciji, ki prepreči prvi transakciji branje in pisanje na prebrani vrstici.

ISOLATION LEVEL SERIALIZABLE + FOR UPDATE

```
Start transaction 2
Start transaction 1
Začetno stanje (Preiskava 57398)
TR2: UPDATE Preiskava SET Vrednost = Vrednost + 50 WHERE IdPreiskave = 57398
Vrednost 50
TRANSACTION 2 COMMITED!
Vrednost 50
TR1: UPDATE Preiskava SET Vrednost = Vrednost + 10 WHERE IdPreiskave = 57398
TRANSACTION 1 COMMITED!
Vrednost 60
➜  faks
```

2.5 (dodatno) preizkusite odkrivanje in reševanje mrtve zanke

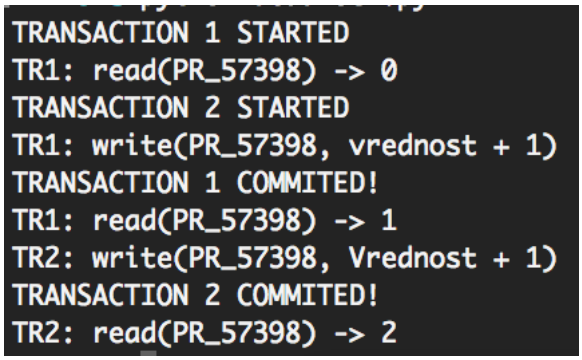
Primer deadlocka je predstavljen v programu deadlock.py. V obeh primerih je uporabljen Serializable nivo izolacije. V prvem primeru pride do deadlocka v drugi transakciji, na kar nas tudi opozori sam program. Druga (mlajša) transakcija se zaradi deadlocka razveljavi in tako sprosti konfliktno vrstico, nato pa se prva transakcija potrdi.

```
TRANSACTION 1 STARTED
TRANSACTION 2 STARTED
TR1: read(PR_57398)
TR2: write(PR_57398, Vrednost + 1)
TR1: write(PR_57398, vrednost + 1)

Exception in thread Thread-2:
Traceback (most recent call last):
conn2.query("UPDATE Preiskava SET Vrednost = Vrednost + 1 WHERE IdPreiskave = 57398")
OperationalError: (1213, 'Deadlock found when trying to get lock; try restarting transaction')

TRANSACTION 1 COMMITED!
```

Deadlock lahko preprečimo z ročnim zaklepom tabele za pisanje (LOCK TABLE Preiskava WRITE) v prvi transakciji in tako bo druga transakcija počakala s pisanjem. Ta rešitev je sicer performančno zelo slaba. Deadlock lahko rešujemo tudi v samem programu, ko v try catch bloku zaznamo, da je prišlo do deadlocka in ponovno poskusimo s transakcijo, saj je verjetno prva transakcija že sprostila vrstico. Ta dobro deluje ob predpostavki, da je deadlockov relativno malo. Je pa performančno boljša od zaklepanja tabele.


TRANSACTION 1 STARTED
TR1: read(PR_57398) -> 0
TRANSACTION 2 STARTED
TR1: write(PR_57398, vrednost + 1)
TRANSACTION 1 COMMITED!
TR1: read(PR_57398) -> 1
TR2: write(PR_57398, Vrednost + 1)
TRANSACTION 2 COMMITED!
TR2: read(PR_57398) -> 2