**Vježba 1: Transakcija**

U ovoj ćemo vježbi ispitati mehanizam transakcije. Transakcija je skup instrukcija čiji se učinak na podatke u bazi prihvaća po načelu sve ili ništa. Pretpostavimo sljedeću transakciju:

**TRANSAKCIJA**

 Instrukcija 1;

 Instrukcija 2;

 **AKO** su uspjele Instrukcije 1 i 2 **ONDA**

 **COMMIT**;

 **INAČE**

 **ROLLBACK**;

Operacijom COMMIT se promjene koje su proizvele Instrukcija 1 i 2 prihvaćaju i postaju trajne, dok operacija ROLLBACK vraća bazu u stanje prije izvršavanja instrukcija transakcije, kao da se i nisu izvršavale. Uzroci neuspjeha instrukcija mogu biti različiti, primjerice:

* Kvar hardvera
* Nestanak napajanja
* Neuspjeh promjene podatka.

Neuspjeh promjene podatka ovisi o aplikaciji, što ćemo prikazati u primjeru prebacivanja novčanog iznosa s jednog računa na drugi. Prije toga, pojasnimo što znači da transakcija ima svojstva ACID.

**Atomarnost (engl. Atomicity)**

Transakcije se često sastoje od više instrukcija. Atomarnost jamči da se svaka transakcija tretira kao jedna nedjeljiva "jedinica", koja ili uspijeva u potpunosti, ili u potpunosti ne uspije: ako se bilo koja od instrukcija koja sačinjava transakciju ne dovrši, cijela transakcija ne uspije i baza podataka ostaje nepromijenjena. Sustav mora jamčiti atomarnost u svakoj situaciji, uključujući nestanak struje, greške i krahiranja sustava.

**Konzistentnost (engl. Consistency)**

Na temelju podataka u bazi vrednujemo, s istina ili laž, određene tvrdnje. Primjerice:

***Na bankovnom računu čiji je broj 100 ima iznos od 1000 kuna. (1)***

Jasno, ova je tvrdnja istinita ako na računu broj 100 ima 1000 kuna, inače je neistinita. Pretpostavimo da tablica Račun sadrži sljedeće podatke:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Id#*** | ***Broj\_racuna*** | ***Iznos*** | ***Zadnja\_promjena*** |
| 1 | 100 | 1000 | 3.3.2019 10:00 |
| 2 | 200 | 5000 | 3.3.2019 10:00 |
| 3 | 100 | 2000 | 4.3.2019 09:30 |

Primarni ključ tablice je Id, dok atribut Broj\_racuna nije ključ kandidat, odnosno, vrijednosti ovog atributa nisu jedinstvene. U tablici postoje dva retka čiji je broj računa 100, jedan s iznosom 1000 (Id=1), a drugi s 2000 kuna (Id=3). Kako se sada vrednuje tvrdnja (1)? U isto vrijeme, na temelju retka Id=1 ona je istinita, a na temelju retka Id=3 ona je neistinita (lažna), što je kontradikcija. Kažemo da gornja tablica nije konzistentna (dosljedna) stoga što istu tvrdnju u isto vrijeme možemo vrednovati s istina i laž. Jasno je da na temelju ovakvih podataka ne možemo valjano odlučivati.

Dakle, na temelju podataka u bazi možemo valjano odlučivati ako je baza u valjanom stanju, kažemo konzistentna.

Invarijanta transakcije je tvrdnja čiju istinitost transakcija ne mijenja. Dakle, ista tvrdnja vrijedi prije i poslije transakcije. Mnoge su tvrdnje istinite prije i poslije transakcije, ali ovdje nas zanimaju one tvrdnje koje uključuju atribute kojima se mijenjaju podaci. Primjerice, ako Instrukcija 1 oduzima s izvornog računa (npr. Račun broj 100) iznos od 1000 kuna, a Instrukcija 2 pribraja isti iznos odredišnom računu (npr. Račun broj 200), onda pogodna invarijanta transakcije može biti: zbroj iznosa na izvornom i odredišnom računu je konstantan, tj:

***Prije transakcije(Izvorni.Iznos + Odredišni.Iznos) = Poslije transakcije(Izvorni.Iznos + Odredišni.Iznos) (2)***

Dakle, konzistentnost transakcije osigurava da transakcija može dovesti bazu podataka iz jednog valjanog stanja u drugo, održavajući invarijante baze podataka, a to je da svi podaci zapisani u bazu podataka moraju biti valjani prema svim definiranim pravilima. Ovo svojstvo transakcije jamči da će baza podataka biti konzistentna u svakom momentu u kojem su podaci dostupni (nisu zaključani). Primijetimo da konzistentnost transakcije nije isto što i korektnost transakcije s obzirom na specifikaciju. Njena korektnost se to mora verificirati posebno, primjerice testiranjem.

**Izolacija (engl. Isolation)**

Često se više transakcija izvršava istovremeno (npr. čitanje i pisanje više tablica u isto vrijeme). Izolacija osigurava da istodobno izvršavanje transakcija ostavlja bazu podataka u istom stanju koje bi bilo dobiveno da su transakcije izvršene sekvencijalno. Izolacija je glavni cilj kontrole konkurentnosti; ovisno o korištenoj metodi, učinci nedovršene transakcije neće biti vidljivi drugim transakcijama. Svojstvo izolacije transakcija se ostvaruje primjenom lokota, tako da se reci baze podataka čije će se vrijednosti mijenjati zaključaju. Razlikujemo sljedeće vrste lokota:

* Dijeljeni lokot (engl. Shared lock)
* Isključivi lokot (engl. Exclusive lock)
* Namjeravani lokot (engl. Intention lock)

SQL instrukcija SELECT postavlja dijeljeni lokot tako da on ne brani pristup zaključanim recima drugim SELECT instrukcijama. Instrukcije, INSERT, UPDATE, DELETE trebaju isključivi lokot te ne mogu pristupiti recima zaključanim dijeljenim lokotom. Kada je redak zaključan isključivim lokotom, njemu može pristupiti samo ona instrukcija koja ga je zaključala.

Instrukcije jedne transakcije se mogu izvršiti paralelno, ali i slijedno. U primjeru prijenosa novca s jednog na drugi bankovni račun to znači da se u kratkom vremenskom intervalu, nakon završetka instrukcije 1 a prije završetka instrukcije 2, baza nalazi u stanju u kojem nije zadovoljena invarijanta (2). Budući su reci uključeni u transakciju zaključani, stanje u kojem invarijanta nije zadovoljena ne može vidjeti niti jedna instrukcija, pa kažemo da izoliranost omogućava zadovoljenje invarijanti baze podataka.

Ipak, spomenimo kako je moguće i tzv. prljavo čitanje redaka tablice (engl. Dirty read) koje se obavlja bez obzira na stanje lokota. Posljedica takvog čitanja mogu biti podaci koji ne zadovoljavaju invarijante baze. To je poseban slučaj koji treba pažljivo koristiti.

**Trajnost (engl. Durability)**

Kada se pomoću instrukcije COMMIT prihvate promjene podataka postignute instrukcijama transakcije, one se nalaze u radnoj memoriji i još nisu spremljene na medije za trajnu pohranu, kao što je magnetski ili čvrsti disk utemeljen na EEPROM tehnologiji. Svojstvo trajnosti jamči da se stanje baze neće izgubiti čak i u slučaju nenadanih kvarova i ispada sustava, primjerice radi nestanka napajanja. To osigurava mehanizam loga (žurnala) koji radi na ovaj način:

* Log je sekvencijalni zapis u koji se reci dodaju uvijek na kraj, dok se uz pomoć funkcije *fn\_dblog* on prikazuje kao tablica.
* U log se zapisuju podaci o transakciji, uključujući promijenjene podatke (vrijednost prije i poslije transakcije).
* Transakcija se može prihvatiti samo ako zapisivanje u log uspije, inače će biti odbačena.
* Izvršavanjem instrukcije COMMIT transakcija se prihvaća i novi podaci su zapisani u redak tablice sadržan u RAM-u. Stranica (od 8192 byte) koja sadrži retke tablice se označava kao *zaprljana* (engl. dirty).
* Kada je sustav za upravljanje bazom podataka besposlen (engl. Idle), zaprljane stranice se iz RAM-a spremaju na disk. Isto će se desiti i ako ponestane slobodnog prostora u dijelu RAM-a dodijeljenog sustavu.
* Transakcija (ili više njih) koja je zaprljala stranicu označi se u logu kako bi se znalo da su njeni podaci spremljeni na disk. Ova se oznaka zove *checkpoint*.
* Ukoliko se desi kvar ili pad sustava, primjerice zbog nestanka napajanja, moguće je da sve zaprljane stranice nisu spremljene na disk, tako da neki reci imaju zastario podatak u tablici na disku. Srećom, njegova promjena je registrirana u logu i nalazi se poslije *checkpointa*.
* Kada se sustav za upravljanje bazom ponovo pokrene, nekonzistentna baza prelazi u stanje oporavka (engl. Recovery, pripadni simbol baze obojan u sivo) i još se ne može koristiti. Oporavak se sastoji u ponovnom izvršavanju primitivnih operacija prema logu i to od *checkpointa* do kraja. Nakon toga je baza ponovo konzistentna i može se koristiti.

Sadržaj loga se može vidjeti sljedećom, nedokumentiranom funkcijom koja vraća tablicu i ima dva argumenta:

select \*

from fn\_dblog(null, null);

Više detalja o sadržaju loga na <https://www.sqlshack.com/reading-sql-server-transaction-log/>.

Navode o transakciji ćemo provjeriti kroz zadatke koki slijede.

*Zadatak 1*: Napraviti direktorij C:\DATA (ako već ne postoji). Koristeći sustav za upravljanje relacijskom bazom podataka MS SQL Server (verzija 2017) napraviti relacijsku bazu BANKA tako da su njene datoteke smještene u direktoriju C:\DATA. U bazi BANKA napraviti relaciju (tablicu) ***Racun*** sa sljedećim atributima (kolonama):

|  |
| --- |
| **Tablica Racun** |
| ***Naziv atributa*** | ***Podatkovni tip*** | ***Nullable*** | ***Uloga*** |
| Broj\_racun | Int | Not null | Primarni ključ, autoinkrement |
| Iznos | Decimal(10, 2) | Not null | Stanje računa u kunama |
| Zadnja\_promjena | Smalldatetime | Not null | Datum i vrijeme kada je stanje računa zadnji put promijenjeno |

*Zadatak 2*: Insertirati u tablicu podatke, tako da stanje tablice bude:

|  |  |  |
| --- | --- | --- |
| ***Broj\_racuna*** | ***Iznos*** | ***Zadnja\_promjena*** |
| 1 | 1000.00 | 1.3.2019 12:29 |
| 2 |  2000.00 | 1.3.2019 10:10 |
| 3 | 20000.00 | 1.3.2019 8:10 |

*Zadatak 3*. Otvorimo novi dokument upita (New Query) i provjerimo identifikator procesa sesije (SPID, session process identifier):

select @@SPID

instrukcija će prikazati neki broj, npr 55. Otvorimo novi dokument i ponovimo isti upit. Sada će instrukcija prikazati neki drugi broj, npr. 58, što znači da su dokumentima pridruženi različiti procesi.

Provjerimo još koji su procesi pokrenuti pomoću spremljene procedure (engl. Stored procedure):

 exec sp\_who;

te detaljnije pomoću:

exec sp\_who2

*Zadatak 4*. U dokument uređivača naredbi prve sesije upisat i izvršiti transakciju:

BEGIN TRANSACTION

 UPDATE Racun

 SET Iznos = Iznos + 1000

 WHERE Broj\_racuna=1;

Primijetimo da transakcija nije zaključena s COMMIT niti s ROLLBACK;

U dokument uređivača naredbi druge sesije upišimo:

SELECT \*

FROM Racun;

**Pitanja:**

1. Objasniti što će prikazati SELECT iz druge sesije?
2. Što će prikazati ista SELECT instrukcija ako je upišemo u prvu sesiju.
3. Što je potrebno učiniti kako bi SELECT druge sesije prikazao tablicu Racun?
4. Provjeriti učinke instrukcija COMMIT i ROLLBACK.
5. Ponovo izvršiti transakciju Zadatka 4 bez COMMIT I ROLLBCK. Potom u drugoj sesiji izvršiti: SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Što će SELECT prikazati nakon promjene izolacijske razine iako nije izvršen COMMIT/ROLLBACK u prvoj sesiji?

*Zadatak 5*. Potrebno je napraviti transakciju koja će prebaciti 1000 kuna s računa broj 1 na račun broj 2. Prebacivanje se može izvršiti samo ako je na izvornom računu iznos veći ili jednak onom koji se uplaćuje na odredišni račun. Uzeti u obzir da se instrukcije izvršavaju paralelno, tako da je moguće da netko upravo malo prije skidanja iznosa s izvornog računa taj račun izbriše iz tablice ili podigne raspoloživi iznos. Isto tako, možda ne postoji odredišni račun pa se transakcija treba odbaciti u cijelosti. Diskutirati rješenje.

*Rješenje*:

declare @x decimal(10,2);

begin transaction

 update Racun set Iznos=Iznos + 1000

 where Broj\_racuna=2;

 if @@ROWCOUNT = 0

 begin

 rollback;

 print 'Odredišni racun br. ' + convert(varchar, @naRacun) + ' ne postoji!';

 return;

 end

 update Racun set Iznos=Iznos - 1000

 where Broj\_racuna=1;

 if @@ROWCOUNT = 0

 begin

 rollback;

 print 'Izvorišni racun br. ' + convert(varchar, @saRacuna) + ' ne postoji!';

 return;

 end

 select @x=Iznos

 from Racun

 where Broj\_racuna=1;

 if @x < 0

 begin

 rollback;

 print 'Na računu je manje od 1000 kuna pa se transakcija ne može izvršiti!';

 end

 else

 begin

 commit;

 print 'Ok';

 end