Transakcije

Više SQL naredbi može da se grupiše u jednu transakciju. Ta transakcija se posmatra kao jedna celina.

Primeri u ovoj skripti će biti realizovani nad tabelom ***racuni*** koja čuva podatke o iznosima na bankovnim računima različitih klijenata.

```
create table racuni
(
id int primary key not null,
iznos int not null
)

begin tran
insert into racuni values (1,1000)
insert into racuni values (2,100)
update racuni set iznos=iznos+100 where id=1
commit tran
--rollback tran

select * from racuni
```

BEGIN TRAN – početak transakcije
COMMIT TRAN – potvrđivanje svih promena od početka transakcije
ROLLBACK TRAN – prekidanje transakcije, poništavanje svih promena od njenog početka

Transakcije najbolje opisuju njihove četiri osobine – Atomicity, Consistency, Isolation, Durability, skraćeno ACID. Na srpskom – atomičnost, konzistentnost, izolacija, trajnost.

Atomičnost – Ili će se transakcija u celosti izvršiti do kraja, ili se nijedna naredba iz transakcije neće izvršiti.

Primer.

```
SET XACT_ABORT ON;
begin tran
insert into racuni values(3,500)
insert into racuni values(4,200)

waitfor delay '00:00:10'

insert into racuni values(5,1500)
insert into racuni values(6,2000)
commit tran
```

Ako posle prve dve naredbe prekinemo izvršavanje, dobijamo:

	id	iznos
1	1	1100
2	2	100

SET XACT_ABORT ON govori SQL serveru da u slučaju greške u izvršavanju treba automatski da poništi celu transakciju. Ako ova naredba nije navedena, programer mora sam da vodi računa o atomičnosti pravilnim korišćenjem naredbe *ROLLBACK*.

Ako se prethodna transakcija izvrši do kraja i commit-uje, dobijamo:

	id	iznos
1	1	1100
2	2	100
3	3	500
4	4	200
5	5	1500
6	6	2000

Konzistentnost – Baza podataka mora da ostane u konzistentnom stanju nakon transakcije, bilo da je transakcija izvršena uspešno ili ne. To znači da sve promene nastale u bazi moraju da budu validne, da ispoštuju sva pravila i ograničenja baze.

Primer. Tabela čuva podatke o količini nekog proizvoda u magacinu. Ako dva kupca istovremeno pokušaju da kupe poslednji proizvod, ne sme da se desi da se proizvod proda i jednom i drugom, i da količina u tabeli postane negativna. Ako to dopustimo, baza će postati nekonzistentna.

Izolacija – Transakcija koja se trenutno izvršava i još uvek nije commit-ovana, mora ostati izolovana od drugih transakcija. Rezultat konkurentno izvršenih transakcija treba da bude isti kao da su transakcije izvršavane serijski. SQL server dozvoljava ublažavanje ovog uslova zbog performansi. Transakcijama se mogu zadavati različiti nivoi izolacije.

Primer. Koristimo tabelu ***racuni***. Trenutno se na računu *r1* nalazi 1100 dinara, a na računu *r2* ima 100 dinara. Neka se dve transakcije, *t1* i *t2* izvršavaju konkurentno nad tabelom. Transakcija *t1* treba da obavi prenos 500 dinara sa računa 1 na račun 2. Transakcija *t2* ima zadatak da podigne 200 dinara sa računa klijenta 2 ako je taj iznos raspoloživ. Postoji mogućnost da redosled operacija bude kao na sledećoj tabeli:

Transakcija *t1*	**Transakcija *t2***
račun 2 +500 din (600 din)	
	račun 2 -200 din (400 din)
GREŠKA! ROLLBACK!	

Ovde postoji opasnost od dovođenja baze u nekonzistentno stanje. Transakcija *t1* je dodala na račun *r2* iznos od 500 dinara. Transakcija *t2* sada može da podigne 200 dinara sa računa jer ima dovoljno novca na računu. Međutim, šta ako se zatim transakcija *t1* rollback-uje pre nego što se sa računa *r1* skine 500 dinara i kompletira prenos? Prenos novca bi bio u potpunosti poništen, iznos na *r2* bi bio smanjen za 500 dinara i završio bi u minusu!

Osobina izolacije štiti transakcije od ovakvih pojava nekonzistentnosti.

```
--transakcija t1
begin tran t1

update racuni
set iznos=iznos+500
where id=2

waitfor delay '00:00:10'

update racuni
set iznos=iznos-500
where id=1
```

```
--transakcija t2
begin tran t2

update racuni
set iznos=iznos-200
where id=2 and iznos>=200

commit tran t2
```

Pokrenimo ove dve transakcije u dve različite instance Management Studija, prvo *t1*, pa odmah zatim *t2*. Primetimo da, dok se u *t1* odbrojava delay, *t2 je* blokirana, ne može da izvrši update dok se transakcija *t1* ne završi, bilo commit-om ili rollback-om. Na ovaj način transakcije ostaju izolovane i konzistentnost baze je sačuvana. Ako se *t1* rollback-uje, *t2* ne može da skine 200 dinara jer na računu *r2* nema dovoljno novca.

Trajnost – Jednom kada se transakcija potvrdi, te promene se čuvaju, makar posle toga pao sistem. Ovo se postiže čuvanjem svih promena u dnevniku transakcija (*transaction log*). Ako sistem padne pre nego što se na disk upišu promene koje je napravila transakcija, pri restartovanju računara dnevnik će vratiti ove promene jer ih je sačuvao. Zahvaljujući dnevniku je moguće i izvršavanje naredbe *rollback*, dnevnik u sebi ima zapis o stanju baze pre početka transakcije.

Rollback

Ako koristimo naredbu rollback, nije nam potrebno da SQL server sam automatski vrši poništavanje transakcije pri *runtime* grešci.

```
set xact_abort off
```

Uostalom, *off* je podrazumevana vrednost za *XACT_ABORT*.
Runtime greške je najbolje obraditi okruživanjem transakcije *try* i *catch* blokovima, gde će commit uvek da se nalazi u try bloku, a rollback u catch bloku.

```
begin try
	begin tran t3
	insert into racuni values(7,500)
	insert into racuni values(8,200)
	insert into racuni values(8,1500)
	insert into racuni values(9,2000)
	commit tran t3
end try
begin catch
	print 'Greska'
	rollback tran t3
end catch
```

Ako pokušamo da unesemo u tabelu vrednost primarnog ključa koja već postoji, dolazi do *runtime* greške, prelazi se u *catch* blok, poziva se *rollback* i poništava se cela transakcija.

```
(1 row(s) affected)

(1 row(s) affected)

(0 row(s) affected)
Greska
```

Savepoint

U nekim slučajevima, kada dođe do rollback-a u transakciji, ne želimo da se poništi sav posao koji je izvršen od početka transakcije, već samo da se vratimo do određene tačke (*savepoint*). U takvim situacijama koristimo naredbu *SAVE TRAN*.

```
begin tran t4
insert into racuni values(7,500)
save tran s1
insert into racuni values(8,200)
save tran s2
insert into racuni values(9,2000)
rollback tran s2
commit tran t4
```

Posle trećeg insert-a se ne poništava cela transakcija, već samo naredbe izvršene nakon pravljenja savepoint-a *s2*. Ovakav rollback ne označava kraj transakcije, i dalje je potrebno potvrditi naredbe izvršene pre savepoint-a *s2,* otuda na kraju skripte naredba commit. Dobijamo:

	id	iznos
1	1	1100
2	2	100
3	3	500
4	4	200
5	5	1500
6	6	2000
7	7	500
8	8	200