Nivoi izolacije

U kompleksnim poslovnim sistemima, brojne transakcije se izvršavaju konkurentno, često i nad istom tabelom. Primeri: elektronsko bankarstvo, onlajn rezervacija letova i sl. Ove transakcije bi trebalo da budu izolovane jedne od drugih, jer postoji velika opasnost od narušavanja konzistentnosti. Međutim, ako zaključamo celu tabelu dok jedna transakcija radi nad njom, može da se desi da veliki broj transakcija čeka u redu da dobije pristup tabeli i da se sistem prilično uspori. Da se to ne bi dešavalo, SQL Server pruža mogućnost zadavanja različitih nivoa izolacije. Što je nivo izolacije viši, konzistentnost je sigurnija, ali je konkurentnost manja i sistem je sporiji. Što je nivo izolacije niži, performanse sistema se povećavaju, ali se povećava i opasnost od gubljenja konzistentnosti. Nivo izolacije mora biti veoma pažljivo odabran u zavisnosti od situacije. Cilj je da odaberemo najniži mogući nivo za koji smo sigurni da će sačuvati konzistentnost.

Nivo izolacije se zadaje pre poćetka izvršavanja transakcije naredbom:

SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED| REPEATABLE READ | SERIALIZABLE | SNAPSHOT }

Read Uncommitted - najniži nivo izolacije. Transakcija sa ovim nivoom može da pročita podatke koje je neka druga transakcija modifikovala, ali nije još *commit*-ovala.

Primer 1. Tabela ***racuni*** sadrži podatke o iznosima na bankovnim računima. Neka je trenutno stanje tabele:

	id	iznos
1	1	1100
2	2	100
3	3	500

Dve transakcije (*t1* i *t2*) se izvršavaju konkurentno. Transakcija *t1* vrši prenos 400 dinara sa računa 1 (označimo ga sa *r1*) na račun 2 (označimo ga sa *r2*). Pre nego što *t1* potvrdi prenos naredbom *commit*, transakcija *t2* sa nivoom izolacije *read uncommitted* započinje svoje izvršavanje i čita nepotvrđeni iznos sa računa *r2*.

```
begin tran t1

update racuni
set iznos=iznos-400
where id=1

update racuni
set iznos=iznos+400
where id=2

waitfor delay '00:00:20'

commit tran t1
```

```
set transaction isolation level
read uncommitted

begin tran t2

select iznos from racuni
where id=2

commit tran t2
```

Rezultat transakcije *t2* nam govori da je iznos na računu *r2* uvećan za 400 dinara:

	iznos
1	500

Ako se transakcija *t1* sada iznenada *rollback*-uje, prenos novca će biti poništen, a baza će biti vraćena u prethodno stanje, gde je na računu *r2* stari iznos od 100 dinara.

	id	iznos
1	1	1100
2	2	100
3	3	500

To znači da transakcija *t2* prikazuje netačne rezultate

Ova pojava se zove **prljavo čitanje** (*dirty read*) i može da dovede do nekonzistentnog stanja baze.

Primer 2. Transakcija *t1* vrši prenos 400 dinara sa *r1* na *r2* kao u prethodnom primeru. Pre nego što *t1* potvrdi promene, *read uncommitted* transakcija *t2* pročita iznos sa računa *r2*, i ako ima dovoljno novca, podiže 200 dinara.

```
begin tran t1

update racuni
set iznos=iznos-400
where id=1

update racuni
set iznos=iznos+400
where id=2

waitfor delay '00:00:20'

commit tran t1
```

```
set transaction isolation level
read uncommitted

begin tran t2

declare @iznos int

select @iznos=iznos from racuni
where id=2

if @iznos>=200
update racuni
set iznos=iznos-200
where id=2

commit tran t2
```

Transakcija *t1* poveća iznos na računu *r2* za 400. Transakcija *t2* pročita da ima 500 dinara na raspolaganju i skine 200.

Ako se sada prekine izvršavanje transakcije *t1*, prenos novca će se poništiti i 400 dinara će biti vraćeno sa *r2* na *r1*. Na računu *r2* je sada negativan iznos, baza ostaje u nekonzistentnom stanju!

	id	iznos
1	1	1100
2	2	-100
3	3	500

Primetimo da ovde UPDATE iz *t1* nije sprečio SELECT transakcije *t2*, ali je sprečio UPDATE iz *t2*. Tek kad se transakcija *t1* završila (bilo sa COMMIT ili ROLLBACK), transakcija *t2* je izvršila svoj UPDATE. Dakle, *read uncommitted*, dozvoljava prljavo čitanje, ali ne i tzv. "**slepo pisanje"**.

Read committed - Podrazumevani nivo izolacije. Transakcija sa ovim nivoom izolacije ne može da čita modifikovane podatke koji nisu *commit*-ovani.

Primer 3. Ponovimo primer 1, samo ovoga puta stavljamo nivo izolacije *read committed* za transakciju *t2*.

```
begin tran t1

update racuni
set iznos=iznos-400
where id=1

update racuni
set iznos=iznos+400
where id=2

waitfor delay '00:00:20'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

select iznos from racuni
where id=2

commit tran t2
```

Transakcija *t2* uopšte neće moći da izvrši *select* dok transakcija *t1* ne uradi *commit*. Ovime je prljavo čitanje sprečeno. Ali to nije jedina anomalija koja može da se pojavi.

Primer 4. Vratimo bazu u konzistentno stanje:

	id	iznos
1	1	1100
2	2	100
3	3	500

Transakcije *t1* (sa nivoom izolacije *read committed*) i *t2* se izvršavaju konkurentno. Prva transakcija u dva navrata čita iznos na računu *r2*. Između ova dva čitanja, transakcija *t2* menja vrednost na računu *r2* i potvrđuje promenu.

```
set transaction isolation level
read committed
begin tran t1

select iznos from racuni
where id=2

waitfor delay '00:00:10'

select iznos from racuni
where id=2

commit tran t1
```

```
begin tran t2

update racuni
set iznos=iznos+200
where id=2

commit tran t2
```

Prva transakcija prilikom izvršavanja druge SELECT naredbe čita novu potvrđenu vrednost za *r2.* Dakle, u okviru iste transakcije, dva ista čitanja su dala dva različita rezultata.

	iznos
1	100

	iznos
1	300

Ova anomalija se naziva neponovljeno čitanje (*non-repeatable read*).

Repeatable read - Ovaj nivo izolacije zadržava sve osobine nivoa *read committed* i povrh toga sprečava neponovljena čitanja.

Primer 5. Trenutno stanje tabele ***racuni*** je:

	id	iznos
1	1	1100
2	2	300
3	3	500

Ponovimo primer 4, sa jedinom razlikom u nivou izolacije za transakciju *t1*, ovoga puta stavljamo *repeatable read*.

```
set transaction isolation level
repeatable read

begin tran t1

select iznos from racuni
where id=2

waitfor delay '00:00:10'

select iznos from racuni
where id=2

commit tran t1
```

```
begin tran t2

update racuni
set iznos=iznos+200
where id=2

commit tran t2
```

Sada će se transakcija *t2* blokirati i čekati da se transakcija *t1 commit*-uje. Tek tada će moći da izvrši UPDATE naredbu. Ovim su neponovljena čitanja sprečena. Transakcija *t1* kao rezultate prikazuje:

	iznos
1	300

	iznos
1	300

Kada se i transakcija *t2* završi i doda 200 dinara na *r2*, tabela ***racuni*** će izgledati ovako:

	id	iznos
1	1	1100
2	2	500
3	3	500

Kada jedna transakcija sa nivoom izolacije *repeatable read* izvrši SELECT nad nekim podacima, drugim transakcijama je zabranjen UPDATE nad tim istim podacima sve dok se prva transakcija ne završi. Isto važi i za DELETE. Međutim, INSERT nije zabranjen što može da dovede do anomalije zvane **fantomska vrsta**.

Primer 6.

```
set transaction isolation level
repeatable read

begin tran t1

select * from racuni

waitfor delay '00:00:10'

select * from racuni

commit tran t1
```

```
begin tran t2

insert into racuni
values (4,1000)

commit tran t2
```

Dobija se:

	id	iznos
1	1	1100
2	2	500
3	3	500

	id	iznos
1	1	1100
2	2	500
3	3	500
4	4	1000

Iako smo sprečili neponovljena čitanja, ponovo se desilo da dve identične SELECT naredbe u okviru iste transakcije daju različite rezultate. Ovoga puta je razlika u broju vrsta, pojavila se jedna nova „fantomska vrsta“.

Serializable - najviši nivo izolacije. Ako se transakcije izvršavaju serijalizabilno, to znači da će njihov rezultat biti kao da su se izvršile serijski. Ovaj nivo izolacije sprečava pojavu fantomskih vrsta i ne dopušta nijednu anomaliju. Ako transakcija sa ovim nivoom pročita neke podatke, ti podaci su zaključani u potpunosti za sve ostale transakcije dok se *serializable* transakcija ne *commit*-uje.

Primer 7. Ponovimo prethodni primer uz zamenu nivoa izolacije za *t1* sa *serializable*.

```
set transaction isolation level
serializable

begin tran t1

select * from racuni

waitfor delay '00:00:10'

select * from racuni

commit tran t1
```

```
begin tran t2

insert into racuni
values (5,2000)

commit tran t2
```

Ovoga puta, SELECT naredba u *t1* blokira INSERT u *t2*. Rezultat transakcije *t1*:

	id	iznos
1	1	1100
2	2	500
3	3	500
4	4	1000

	id	iznos
1	1	1100
2	2	500
3	3	500
4	4	1000

Tek kada se *t1* završi, *t2* će moći da ubaci novu vrstu u tabelu:

	id	iznos
1	1	1100
2	2	500
3	3	500
4	4	1000
5	5	2000