Lock-ovi

Izolacija transakcija se realizuje postavljanjem različitih *lock*-ova na resurse.

Ekskluzivni lock-ovi (exclusive locks) – Kada transakcija modifikuje podatke tj. kada izvršava neku od *write* naredbi (INSERT, UPDATE, DELETE), SQL Server na te podatke postavlja ekskluzivni *lock*.

Deljeni lock-ovi (shared locks) – Kada transakcija čita podatke, ali ih ne modifikuje, tj. kada izvršava *read* naredbu (SELECT), SQL Server na te podatke postavlja deljeni *lock.*

U daljem tekstu, ekskluzivni *lock*-ovi će skraćeno biti obeležavani kao X *lock*-ovi, a deljeni kao S *lock*-ovi.

Kada jedna transakcija postavi X lock na neki resurs, druge transakcije ne mogu ni da modifikuju ni da pročitaju taj resurs.

X lock: ~~read~~, ~~write~~

X lock zapravo ne dopušta postavljanje nijednog drugog lock-a na resurs, ni ekskluzivnog ni deljenog. Transakcija koja postavi X lock na resurs ima ekskluzivni pristup.

Sa druge strane, kada jedna transakcija postavi S lock na neki resurs, druge transakcije ne mogu da modifikuju, ali zato mogu da pročitaju taj resurs.

S lock: read, ~~write~~

S lock dopušta postavljanje drugog S lock-a nad istim resursom, zato se i zove deljeni lock. Postavljanje X lock-a nad resursom koji već ima S lock nije moguće, jer X lock zahteva ekskluzivan pristup i ne dopušta postojanje bilo kakvih drugih lock-ova nad istim resursom.

	S	**X**
S	može	ne može
X	ne može	ne može

Lock-ovi se uglavnom postavljaju na nivou redova (*row locks*), a mogu još biti i na nivou opsega (*range locks*), nivou tabele, baze i sl. SQL Server koristi više od 20 vrsta lock-ova. Pored deljenih i ekskluzivnih, najbitniji su *update* i *intent lock*-ovi.

Primer 1. write-write konflikt

```
set transaction isolation level
read committed

begin tran t1

update racuni
set iznos=iznos+100
where id=1

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

update racuni
set iznos=iznos+100
where id=1

commit tran t2
```

Pokrenimo izvršavanje transakcije *t1*. Njena UPDATE naredba postavlja X lock nad vrstom u kojoj se nalazi račun sa id-jem 1. Zatim pokrenimo izvršavanje transakcije *t2*. Da bi *t2* izvršila svoju UPDATE operaciju, ona pokušava da postavi svoj X lock. Međutim, vrsta sa računom 1 već ima na sebi X lock prve transakcije, pa se *t2* blokira i čeka da se resurs oslobodi. Tek kad se *t1* završi, *t2* može da nastavi sa izvršavanjem, da postavi X lock i da izvrši naredbu modifikacije.

Ovaj primer nam govori da X lock uvek traje do kraja transakcije.

Naredbe INSERT i DELETE takođe stavljaju X lock nad vrstama koje ubacuju, odnosno brišu.

Primer 2. write-read konflikt

```
set transaction isolation level
read committed

begin tran t1

update racuni
set iznos=iznos+100
where id=2

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

select * from racuni
where id=2

commit tran t2
```

Pokrenimo transakciju *t1*. Kao i u prethodnom primeru, i ovde *t1* postavlja X lock. Pokrenimo sada transakciju *t2*. Ona pokušava da postavi S lock na račun 2. Međutim, vrsta sa podacima o računu sa id-jem 2 već ima na sebi X lock koji ne dozvoljava postavljanje nekog drugog lock-a, čak ni deljenog, pa se *t2* blokira i nastavi sa izvršavanjem tek kada se *t1* završi i oslobodi resurs.

Dakle, X lock blokira i naredbe čitanja.

Primer 3. read-write konflikt

```
set transaction isolation level
read committed

begin tran t1

select * from racuni
where id=2

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

update racuni
set iznos=iznos+100
where id=2

commit tran t2
```

Pokrenimo izvršavanje transakcije *t1*. Ona čita podatke o iznosu na računu 2 i postavlja S lock. Zatim započnimo transakciju *t2*. UPDATE naredba se izvršava bez ikakvog blokiranja, pre nego što se *t1* commit-uje!

Šta se ovde zapravo desilo? I dalje važi pravilo da postavljanje S lock-a sprečava X lock. Međutim, ovde S lock ne traje do kraja transakcije, već samo tokom izvršavanja SELECT naredbe. Čim je SELECT završen, transakcija *t2* je mogla da postavi X lock i modifikuje resurs.

Postavljanjem nivoa izolacije *repeatable read*, trajanje S lock-a se produžava do kraja transakcije:

```
set transaction isolation level
repeatable read

begin tran t1

select * from racuni
where id=2

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

update racuni
set iznos=iznos+100
where id=2

commit tran t2
```

Ovoga puta, UPDATE naredba se blokira i čeka se kraj transakcije *t1*. S lock koji je postavila *t1* traje i posle izvršavanja SELECT naredbe, sve do kraja transakcije.

Primer 4. read-read

```
set transaction isolation level
repeatable read

begin tran t1

select * from racuni
where id=2

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

select * from racuni
where id=2

commit tran t2
```

Postavljanje S lock-a na resurs, ne sprečava drugu transakciju da i ona učini to isto. Ako *t1* prva postavi S lock i pročita podatke, *t2* će postaviti svoj S lock i bez blokiranja pročitati te iste podatke.

Primer 5. Row locks

Lock-ovi se ne postavljaju nužno nad celom tabelom, dovoljno je da se postave nad vrstama nad kojima se izvršava operacija. Ostale vrste iz iste tabele nisu zaključane i druge transakcije im mogu slobodno pristupiti.

```
set transaction isolation level
read committed

begin tran t1

update racuni
set iznos=iznos+100
where id=2

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

update racuni
set iznos=iznos+100
where id=1

commit tran t2
```

Transakcija *t1* zaključava vrstu sa id-jem 2. To ne sprečava transakciju *t2* da modifikuje vrstu sa id-jem 1 bez blokiranja.

Primer 6. Lock-ovi i fantomske vrste

```
set transaction isolation level
repeatable read

begin tran t1

select * from racuni

waitfor delay '00:00:10'

select * from racuni

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

insert into racuni
values (6,1000)

commit tran t2
```

Prvo transakcija *t1* pročita celu tabelu i postavi S lock-ove nad svim vrstama. S lock-ovi traju do kraja transakcije i sprečavaju konkurentno izvršavanje *write* naredbi nad bilo kojom postojećom vrstom u tabeli. Međutim, ubacivanje nove vrste nije zabranjeno, jer se time nikako ne utiče na zaključane vrste. Zato *t2* može da izvrši INSERT naredbu bez blokiranja, što dovodi do pojave fantomskih vrsta.

Rezultat izvršavanja transakcije *t1* je:

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

	id	iznos
1	1	1100
2	2	500
3	3	500
4	4	1000
5	5	2000
6	6	1000

Fantomske vrste se sprečavaju postavljanjem nivoa izolacije na *serializable*. Sa ovakvom izolacijom, lock-ovi se ne postavljaju na nivou redova, već na nivou opsega (*range locks*). Ako *t1* pročita celu tabelu, neće biti dozvoljeno ubacivanje u nju dok se ne sklone lock-ovi:

```
set transaction isolation level
serializable

begin tran t1

select * from racuni

waitfor delay '00:00:10'

select * from racuni

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

insert into racuni
values (7, 1000)

commit tran t2
```

Na serijalizabilnom nivou izolacije, S lock se postavlja nad opsegom vrednosti koji obuhvata SELECT naredba. Nad tim opsegom su zabranjene operacije, ali ne i van njega.

Npr. ako *t1* SELECT-uje sve vrste sa id-jem manjim od 3 i *t2* pokuša da ubaci vrstu čiji je id takođe manji od 3:

```
set transaction isolation level
serializable

begin tran t1

select * from racuni
where id<3

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

insert into racuni
values (0, 1000)

commit tran t2
```

iNSERT će biti blokiran dok se *t1* ne završi, tj. dok se ne oslobodi opseg vrednosti za id-jeve.

Sa druge strane, ako nakon zaključavanja vrsta sa id-jem manjim od 3 od strane transakcije *t1*, pokušamo u drugoj transakciji da izvršimo bilo koju *write* naredbu van zaključanog opsega:

```
set transaction isolation level
serializable

begin tran t1

select * from racuni
where id<3

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read committed

begin tran t2

update racuni
set iznos=iznos+100
where id=6

commit tran t2
```

Write naredba će biti izvršena bez blokiranja jer se vrši nad vrstom koja ne spada u opseg S lock-a.

Primer 7. Lock-ovi i *read uncommitted*

Transakcije sa nivoom izolacije *read uncommitted* postavljaju X lock-ove, ali **ne postavljaju** S lock-ove. Posledica nepostojanja S lock-a je nepoštovanje X lock-a, odnosno „prljavo čitanje“.

```
set transaction isolation level
read committed

begin tran t1

update racuni
set iznos=iznos+100
where id=1

waitfor delay '00:00:10'

commit tran t1
```

```
set transaction isolation level
read uncommitted

begin tran t2

select * from racuni
where id=1

commit tran t2
```

Pokrenimo transakciju *t1*. Ona ažurira iznos na računu sa id-jem 1 i stavlja X lock na njegovu vrstu. Kao što znamo, sada je zabranjeno postavljanje bilo kakvog lock-a na tu vrstu dok se *t1* ne završi. Trebalo bi da su zabranjene i *read* i *write* naredbe nad zaključanim resursom. Međutim, *read uncommitted* transakcije uopšte ne postavljaju S lock pri čitanju resursa. Dakle, *t2* može pročitati ažurirani iznos pre nego što je on commit-ovan („prljavo čitanje”).

Rezime

Read uncommitted - Ne postavlja S lock-ove i ne poštuje X lock-ove.

Read committed - poštuje tuđe lock-ove i postavlja lock-ove na svoje resurse. S lock-ove ne drži tokom cele transakcije, već samo dok traje čitanje. X lock-ove drži sve vreme trajanja transakcije.

Repeatable read - drži S lock-ove sve vreme transakcije, ne samo za vreme čitanja.

Serializable - dodatno postavlja i range lock-ove da bi se sprečile fantomske vrste.