

Obrada transakcija kod Oracle sistema za upravljanje bazama podataka

Seminarski rad

Uroš Vukić

Broj indeksa: 1086

Sistemi za upravljanje bazama
podataka

Elektronski fakultet u Niš

Sadržaj

1	Uvod	4
2	Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje	5
2.1	Transakcije, operacije čitanja i upisa u DBMS-u, baferi DBMS-a	5
2.2	Problemi koji se javljaju pri konkurentnom pristupu bazi podataka	6
2.2.1	Problem izgubljenog ažuriranja (The lost update problem)	6
2.2.2	Problem prljavog čitanja (The dirty read problem)	6
2.2.3	Problem netačnog sumiranja (The incorrect summary problem)	7
2.2.4	Problem neponovljivog čitanja (The unrepeatable read problem)	8
2.3	Proces obrade transakcije	8
2.4	Sistemska Log	9
2.5	Osobine transakcija	10
2.6	Planovi izvršenja i vrste planova izvršenja	11
2.6.1	Oporavljivost planova izvršenja	11
2.6.2	Serijabilni (Serializable) planovi izvršenja	11
2.7	Dvofazno zaključavanje kao tehnika kontrole pristupa	13
2.7.1	Binarni lock	13
2.7.2	Deljeni/Ekskluzivni (Shared/Exclusive) ili Read/Write lock	14
2.7.3	Garantovanje serijabilnosti pomoću dvofaznog protokola zaključavanja	14
2.8	Obrada deadlock-ova	15
2.9	Granularnost stavki podataka	17
2.10	Kontrola konkurentnosti u B+ stablima	19
3	Obrada transakcija kod Oracle baze podataka	20
3.1	System change number (SCN)	20
3.2	Commit transakcije	20
3.3	Rollback transakcije	21
3.4	Autonomne transakcije	21
3.5	Konzistentnost čitanja primenom više verzija	22
3.5.1	Konzistentnost čitanja na nivou naredbe	22
3.5.2	Konzistentnost čitanja na nivou transakcije	22
3.5.3	Konzistentnost čitanja i undo segmenti	22
3.6	Izolacioni nivoi transakcija	23
3.7	Nivoi izolacije kod Oracle-a:	23

3.7.1	Read committed izolacioni nivo	23
3.7.2	Serializable izolacioni nivo	24
3.7.3	Read-Only izolacioni nivo	24
3.8	Vrsite lock-ova.....	24
3.9	Konverzija lock-ova i njihova eskalacija	24
3.9.1	Vrste automatskih lock-ova	25
3.9.2	DML lock-ovi.....	25
3.9.3	DDL lock-ovi	25
3.9.4	Sistemske lock-ovi	25
3.10	Primer izvršenja dve transakcije sa READ COMMITED nivoom izolacije:	26
3.11	Primer izvršenja transakcije sa SERIALIZABLE izolacionim nivoom:.....	31
4	Zaključak.....	36
5	Literatura.....	37

1 Uvod

Transakcije predstavljaju mehanizam za opis logičke celine za obradu nad podacima u bazi podataka. Sistemi sa podrškom za obradu pomoću transakcija su sistemi velikih baza podataka sa stotinama konkurentnih korisnika koji uz pomoć transakcija pristupaju bazi podataka. Primeri takvih sistema uključuju sisteme za rezervisanje online karata, sisteme za razmenu novca u bakama, sisteme za online kupovinu, sisteme za prodaju deonica na berzama i mnoge druge. Ovakvi sistemi zahtevaju veliku dostupnost (high availability) i kratko vreme odgovora (response time) za sve korisnike koji konkurentno pristupaju sistemu.

Zbog svega ovoga ćemo se u ovom radu baviti pojmom transakcija, njihovom obradom, planovima izvršenja, međusobnom izolacijom i zaključavanjem objekata baze podataka.

2 Obrada transakcija, planovi izvršavanja transakcija, izolacija i zaključavanje

2.1 Transakcije, operacije čitanja i upisa u DBMS-u, baferi DBMS-a

Transakcija je program u izvršenju koji formira logičku celinu pri obradi podataka nad bazom. Transakcija uključuje jednu ili više naredbi nad bazom podataka (tu spadaju naredbe za pribavljanje podataka, naredbe za dodavanje, ažuriranje/modifikovanje i brisanje podataka). Ove naredbe se mogu ili ugraditi (embedded) u sam izvršni program aplikacije ili izraziti uz pomoć nekog jezika visokog nivoa poput SQL-a.

Jedan način za određivanje granica transakcije je eksplicitnim navođenjem naredba **begin transaction** (započni transakciju) i **end transaction** (završi transakciju) u samom programu. U tom slučaju, sve operacije između ovih dveju naredbi se smatraju delom jedne transakcije. Jedan program može da kreira veći broj transakcija u toku svog izvršenja. Ukoliko se u transakciji ne nalaze naredbe koje menjaju sadržaj baze podataka (nema dodavanja, ažuriranja i brisanja podataka), takva transakcija se naziva **read-only** transakcija, u suprotnom je **read-write** transakcija.

Da bi smo objasnili koncepte transakcija, koristićemo pojednostavljeni model baze podataka. Bazu podataka možemo posmatrati kao imenovanu kolekciju stavki podataka¹ (named data items). Veličina ovih stavki podataka se naziva granularnost. Stavke podataka mogu biti različitih veličina - zapis (record), ceo blok na hard disku ili pojedinačno polje unutar zapisa. Koncepti transakcija su nezavisni u odnosu na granularnost podataka. Svaka stavka podataka ima svoje jedinstveno ime (unique name) na osnovu koga se taj podatak identifikuje (npr. ako je stavka podataka blok na disku onda ime podatka može biti njegova adresa). U ovom pojednostavljenom modelu osnovne operacije koje transakcija može koristiti su:

- `read_item(X)` – Čita stavku podataka sa imenom X iz baze u programsku promenljivu X.
- `write_item(X)` – Upisuje vrednost programske promenljive X u stavku podataka sa imenom X.

`Read_item(X)` operacija se izvršava kroz sledećih nekoliko koraka:

1. Naći adresu bloka koji sadrži podatak X
2. Kopirati blok u bafer (buffer) u glavnoj memoriji (ukoliko se traženi blok već ne nalazi u njoj)
3. Kopirati stavku podatka X iz bafera u programsku promenljivu X

`Write_item(X)` operacija se izvršava kroz sledećih nekoliko koraka:

1. Naći adresu bloka koji sadrži podatak X
2. Kopirati blok u bafer (buffer) u glavnoj memoriji (ukoliko se traženi blok već ne nalazi u njoj)
3. Kopirati vrednost programske promenljive X u promenljivu u baferu podataka
4. Sačuvati vrednost bloka iz bafera na disk (odmah ili naknadno u nekom trenutku)

Odluka o tome kada treba sačuvati izmenjeni blok podataka koji se nalazi u baferu nazad na disk je posao kojim se bavi menadžer oporavka (recovery manager) DBMS-a u kooperaciji sa operativnim sistemom. DBMS održava određen broj bafera podataka u svojoj keš memoriji. Jedan bafer sadrži čuva vrednost jednog bloka sa diska. Kada su svi dostupni baferi popunjeni, a novi blok treba da se prebaci u

¹ Stavka podataka (data item) je apstraktna definicija podatka, odnosno predstavlja podatak različitih veličina.

glavnu memoriju, primenjuje se neka od tehnika za zamenu blokova. Ako je blok izmenjen, on se snima na disk.

Transakcija koristi `read_item` i `write_item` operacije kako bi pristupila i ažurirala podatke u bazi podataka. Skup čitanja (`read-set`) transakcije je skup svih podataka koje transakcija čita u toku izvršenja, a skup upisa (`write-set`) je skup svih podataka koje transakcija upisuje u toku izvršenja.

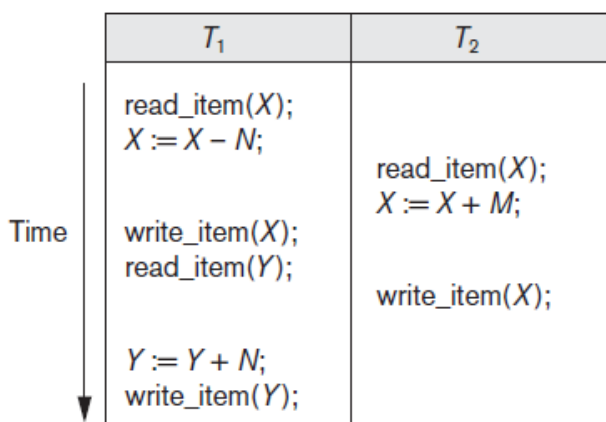
Sistem za upravljanje konkurentnim pristupom i mehanizmi oporavka se uglavnom bave transakcijama. Transakcije se konkurentno izvršavaju od strane većeg broja korisnika i pri tom mogu ažurirati iste podatke. Nekontrolisano konkurentno izvršavanje transakcija može dovesti do problema poput nekonzistentnosti baze podataka.

2.2 Problemi koji se javljaju pri konkurentnom pristupu bazi podataka

Nekoliko problema se mogu javiti pri konkurentnom izvršenju transakcija bez kontrole pristupa.

2.2.1 Problem izgubljenog ažuriranja (The lost update problem)

Ovaj problem se javlja kada dve transakcije isprepletano pristupaju istim podacima u bazi podataka i na taj način vrednost nekog podatka postane ne validna. Pretpostavimo da su transakcije T_1 i T_2 izdate u približno istom trenutku i da su njihove operacije isprepletane kao na Ilustracija 1. Onda će finalna vrednost podataka X biti ne validna zato što T_2 čita vrednost X pre nego što je transakcija T_1 promeni u bazi i stoga je vrednost koja se dobija iz transakcije T_1 izgubljena.

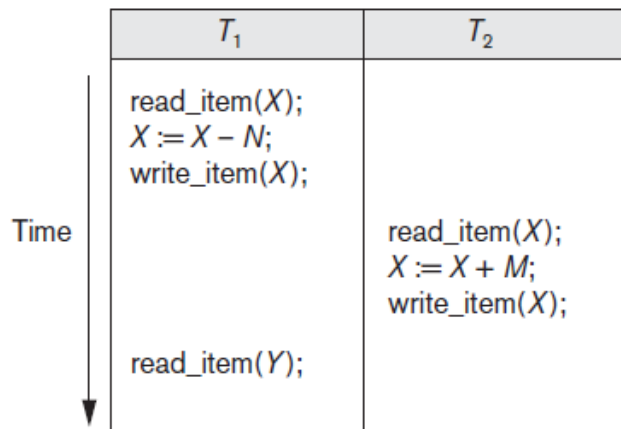


Ilustracija 1 - Primer izvršenja dve transakcije koji dovodi do pojave problema izgubljenog upisa

2.2.2 Problem prljavog čitanja (The dirty read problem)

Ovaj problem se javlja kada jedna transakcija ažurira vrednost podatka u bazi i ne uspe da se izvrši do kraja (podbaci u nekom trenutku - fail). U isto vreme ažurirani podatak čita druga transakcija pre nego što se njegova vrednost vrati na originalnu vrednost. Na Ilustracija 2 je dat primer gde transakcija T_1 ažurira vrednost X i onda ne uspe da se završi i sistem mora da vrati X na početnu vrednost. Pre nego što sistem to odradi, transakcija T_2 čita privremenu vrednost promenljive X , koja nije snimljena u bazu zbog

podbacivanja transakcije T1. Podataka koji je T2 pročitala se naziva prljav podatak, jer je kreiran od strane neuspešno završene transakcije, pa se ovaj problem zove problem prljavog čitanja.

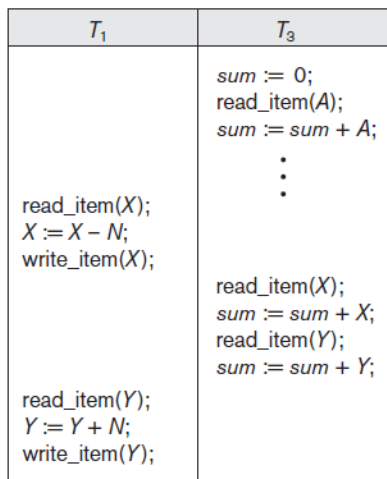


Ilustracija 2 - Primer izvršenja dve transakcije koji dovodi do pojave problema prljavog čitanja

2.2.3 Problem netačnog sumiranja (The incorrect summary problem)

Ukoliko jedna od transakcija računa vrednost neke agregacione funkcije nad nekom grupom podataka u bazi dok druga transakcija ažurira neke od tih podataka, agregaciona funkcija može uzeti vrednosti podataka pre njihovog ažuriranja i neke druge nakon njihovog ažuriranja.

Neka transakcija T3 računa broj rezervacija na svim letovima, a u međuvremenu se izvršava transakcija T1. Ako izvršavanje ovih transakcija prati redosled prikazan na Ilustracija 3, onda će rezultat koji transakcija T3 računa biti manji za N, jer T3 čita vrednost X nakon što se oduzmu N sedišta i čita vrednost promenljive Y pre nego što se dodaju N sedišta toj promenljivoj.



Ilustracija 3 - Primer izvršenja dve transakcije koji dovodi do pojave problema netačnog sumiranja

2.2.4 Problem neponovljivog čitanja (The unrepeatable read problem)

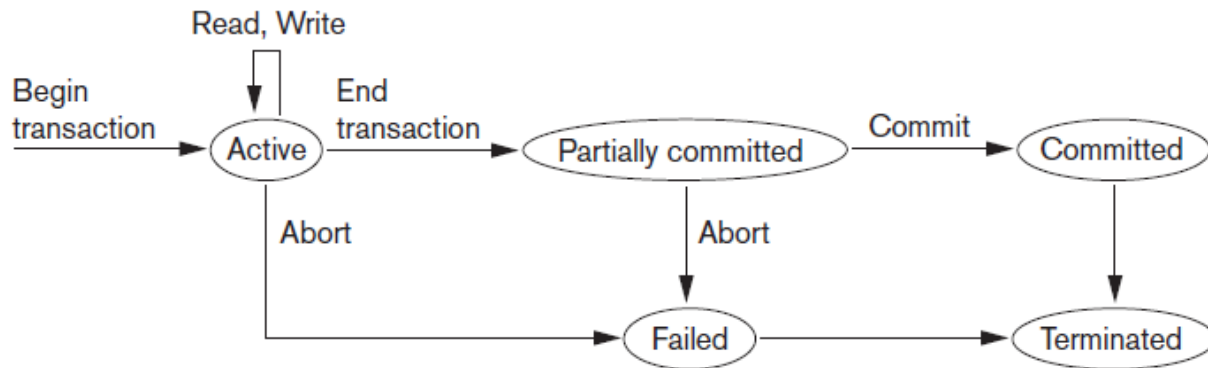
Ovaj problem se javlja kada transakcija T dva puta pročita isti podatak i pročitani podatak je promenjen od strane transakcije T' u periodu između dva čitanja. Stoga, transakcija T dobija dve različite vrednosti za isti podatak. Ovaj problem se može javiti prilikom transakcije za rezervaciju karata za prevoz. Korisnik pošalje zahtev za broj slobodnih mesta na određenoj liniji. Pre nego što obavi konačnu rezervaciju mesta, korisnik još jedanput šalje zahtev za brojem slobodnih mesta i tada može dobiti drugu vrednost ukoliko je neko u međuvremenu veće rezervisao određen broj mesta.

2.3 Proces obrade transakcije

Transakcija je atomična celina koja se ili izvršava u potpunosti ili se ne izvršava uopšte. Da bi se obezbedila mogućnost oporavka, sistem mora pratiti kada svaka transakcija počinje, završava se, kada potvrđuje (commit) izvršenje i kada otkazuje (abort) izvršenja. Iz tog razloga, menadžer oporavka DBMS (komponenta DBMS-a) mora pratiti sledeće operacije:

- BEGIN_TRANSACTION – Ova operacija označava početak izvršenja transakcije
- READ ili WRITE – Ove dve operacije specificiraju operacije čitanja i pisanja nad bazom podataka koje se izvršavaju kao delovi transakcije
- END_TRANSACTION – Ova operacija specificira da su READ i WRITE operacije završene i označava kraj izvršenja transakcije. U ovom trenutku može biti potrebno izvršiti proveru da li promene nastale u toku izvršenja transakcije mogu biti trajno sačuvane (commit-ovane) ili moraju biti otkazane jer izvršena transakcija ne poštuje neko pravilo poput serijabilnosti izvršenja transakcija (videćemo u kasnije delu značenje ovog pojma).
- COMMIT_TRANSACTION – Ova operacija označava uspešan kraj izvršenja transakcije i da sve promene napravljene u toku njenog izvršenja treba bezbedno sačuvati u bazi podataka – „commit-ovati“ i da ove promene neće biti obrisane
- ROLLBACK (ili ABORT) – Ova operacija signalizira da je transakcija neuspešno završila sa izvršenjem i da sve promene koje je transakcija napravila u toku svog izvršenja moraju biti vraćene (undone) na pređašnje stanje

Na Ilustracija 4 možemo videti dijagram koji ilustruje kako transakcija prolazi kroz svako stanje izvršenja. Transakcija prelazi u aktivno stanje odmah nakon što započne sa izvršenjem. U ovom stanju ona može da izvršava READ i WRITE operacije. Kada transakcija završi sa izvršenjem, ona prelazi u delimično commit-ovano stanje (partially committed state). U ovom stanju se izvršavaju razni protokoli oporavka koji obezbeđuju da otkaz sistema neće dovesti do nemogućnosti da se promene transakcije trajno snime (najčešće se promene transakcije upisuju u sistemski log kako bi se upamtile promene koje je transakcije obavila). Takođe, ukoliko se koristi optimistična kontrola konkurentnog pristupa podacima (optimistic concurrency control), u ovoj fazi se vrši provera da li je došlo do međusobnog uticaja između ove i neke druge transakcije koja se izvršava. Kada su sve provere uspešne, transakcija dolazi u committed stanje. U tom stanju transakcija je uspešno završila sa izvršenjem i sve promene moraju trajno biti snimljene u bazu podataka (čak i u slučaju otkaza sistema).



Ilustracija 4 - Prikaz procesa obrade transakcije

Sa druge strane, transakcija će završi u neuspešnom (failed) stanju ukoliko ne prođe neku od provera ili ukoliko je otkazana u toku njenog izvršenja. U tom slučaju, poništavaju se sve WRITE operacije koje je transakcija napravila nad bazom. Na samom kraju transakcija prelazi u završeno stanje (terminated state) koje označava da transakcija napušta sistem tj. sve informacije koje su se čuvala u toku njenog izvršenja se uklanjaju iz sistemskih tabela. Neuspešne (failed) ili otkazane (aborted) transakcije se mogu kasnije ponovo pokrenuti – automatski ili ponovo poslati na izvršenje od strane korisnika kao potpuno nove transakcije.

2.4 Sistemski Log

Da bi se omogućio oporavak od pada sistema, sistem čuva informacije o svim operacijama transakcije koje su menjale vrednosti podataka u bazi. Ove informacije se čuvaju u fajlu koji se naziva sistemski log. Sistemski log je sekvencijalni file u koji se može samo upisivati i koji se čuva na disku i na taj način smanjuje verovatnoća otkaza ovog mehanizma. Uglavnom jedan ili više bafera unutar memorije, koji se još nazivaju i log baferi, služe za upis podataka u ovaj log tako što se sve operacije transakcija sekvencijalno dodaju unutar ovih bafera. Kada se ovi baferi napune ili pod određenim uslovima (npr. commit-ovanje neke transakcije), oni se snimaju na disk i na taj način obezbeđuje dugotrajnost upisanih operacija. U bafere se upisuju specijalne stavke – koje se nazivaju log rekordi (log records) – koji predstavljaju operacije izvršene unutar transakcije. Svaki rekord sadrži identifikator transakcije T koji se generiše automatski od strane sistema na početku izvršenja transakcije. Moguće su sledeće vrste rekorda:

1. [start_transaction, T] – Označava da je transakcija T počela sa izvršenjem
2. [write_item, T, X, old_value, new_value] – Označava da je transakcija T promenila vrednost podatka X sa old_value (stara vrednost) na new_value (nova vrednost)
3. [read_item, T, X] – Označava da je transakcija T pročitala vrednost podataka X
4. [commit, T] – Označava da je transakcija T uspešno završila sa izvršenjem i potvrđuje da su njene izmene prihvaćene i trajno snimljene u bazu podataka
5. [abort, T] – Označava da transakcija T neuspešno završena

Protokoli za oporavak koji izbegavaju kaskadne rollback-ove (koji uključuju gotovo sve danas primenljive protokole) ne zahtevaju upisivanje READ operacija u sistemski log. Sa druge strane, ove operacije se ipak mogu čuvati ukoliko je cilj praćenje svih operacija koje se izvršavaju nad bazom podataka. Pored toga, neki protokoli za oporavak zahtevaju jednostavnije WRITE zapise nego one gore predstavljene koji čuvaju samo jednu vrednost umesto dve vrednosti (old_value ili new_value). Ukoliko dođe do pada sistema, log sadrži dovoljno informacija o svakoj WRITE operaciji tako da je moguće poništiti operacije transakcije T koja nije uspešno završena do kraja ili dovršiti izvršenje transakcije tako što se vrednosti podataka u bazi prepisuju odgovarajućim vrednostima iz log-a. Commit tačka je trenutak nakon koje se promene transakcije mogu oporaviti - sve promene transakcije su uspešno izvršene i sve promene su snimljene u sistemskom log-u. Kada transakcije dostigne commit tačku, u log se upisuje [commit, T] vrednost, što znači da je transakcija trajno snimljena. Prilikom oporavka, sistem završava sve nedovršene transakcije koje su upisale commit tačku i rollback-uje sve one koje to nisu uradile.

2.5 Osobine transakcija

Transakcije treba da poseduju svojstva poznatija kao ACID svojstva. Ova svojstva moraju biti podržana od strane sistema za upravljanje konkurentnim pristupom podacima i od strane sistema za oporavak sistema. ACID svojstva su sledeća:

- Atomičnost (Atomicity) – Transakcija je atomična operacija koja se izvršava ili ne u celosti
- Konzistentnost ili očuvanje konzistentnosti (Consistency preservation) – Transakcija treba da očuva konzistentnost sistema nakon njenog potpunog izvršenja tj. svojim izvršenjem ona prevodi sistem iz jednog u drugo konzistentno stanje
- Izolacija (Isolation) – Transakcija se izvršava u izolaciji od drugih transakcija. Prilikom izvršenja, ona vidi samo promene transakcija koje su izvršene pre nje, a ne vidi promene koje druge transakcije izvršavaju konkurentno sa njom.
- Trajnost (Durability) – Promene koje commit-ovana transakcija napravi u toku svojeg izvršenja su trajne i moraju biti sačuvane u bazi podataka. Ove promene ne smeju biti izgubljene ukoliko dođe do pada sistema.

Atomičnost zahteva da transakcije izvršavamo u celosti. Ovim se bavi sistem za oporavak sistema, tako što u slučaju pada sistema ili završava izmene commit-ovane transakcije ili poništava izmene ne commit-ovane transakcije.

Konzistentnost se obezbeđuje aplikacionom logikom i na odgovornosti je aplikacionih programera. Transakcije moraju biti napisane tako poštuju ograničenja koja su nametnuta od strane aplikacije (npr. da suma novca na svim računima u banci bude konstantna).

Izolacija se obezbeđuje sistemom za upravljanje konkurentnim pristupom DBMS-u. Postoje više nivoa izolacije koji dopuštaju/ograničavaju pojavu problema koji se javljaju pri konkurentnom pristupu DBMS-u.

Trajnost se obezbeđuje sistemom za oporavak od pada sistema.

2.6 Planovi izvršenja i vrste planova izvršenja

Plan izvršenja ili istorija (schedule ili history) S skupa n transakcija T_1, T_2, \dots, T_n je uređenje operacija transakcija. Operacije različitih transakcija mogu biti međusobno isprepletane u planu izvršenja S , ali se sve operacije transakcije T_i moraju naći u planu izvršenja S u istom redosledu u kojem se operacije nalaze u transakciji T_i . Plan uređenja može biti totalno/potpuno uređen – za svake dve operacije u planu može se odrediti koja se od njih izvršila pre druge. Ukoliko se ne može odrediti uređenje svake dve operacije u pitanju je delimično uređen plan. U ovom radu ćemo se baviti isključivo potpuno uređenim planovima izvršenja.

Kao skraćeni zapis operacija **begin_transaction**, **read_item**, **write_item**, **end_transaction**, **commit** i **abort** koristićemo simbole **b**, **r**, **w**, **e**, **c**, i **a** respektivno. Subscript brojem označavamo identifikator transakcije koja obavlja operaciju. Primer: sa $r_1(X)$ označavamo da transakcija sa id=1 čita podataka X . Primer jednog potpuno plana izvršenja: $S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y)$.

Za dve operacije se kaže da su u konfliktu koliko zadovoljavaju sledeća tri uslova:

1. Operacije pripadaju različitim transakcijama
2. Operacije pristupaju istom podatku X
3. Barem jedna od operacija je **write_item(X)** operacija

Intuitivno dve operacije su u konfliktu ukoliko se zamenom njihovog izvršenja mogu dobiti različiti rezultati. Na osnovu definicije konflikta mogu se izdvojiti read-write konflikti – uređenje $r_1(X); w_2(X)$ ne daje iste rezultate kao $w_2(X); r_1(X)$, i write-write konflikti - $w_2(X); w_1(X)$; se razlikuje od $w_1(X); w_2(X)$.

2.6.1 Oporavljivost planova izvršenja

Za planove izvršenja važno je da budu lako oporavljivi, odnosno da se na osnovu plana izvršenja može izvršiti oporavak sistema uz primenom nekog algoritma. Jednom commit-ovana transakcija ne bi trebalo da bude rollback-ovana. Ovo je zahtevano svojstvom trajnosti transakcija. Za plan izvršenja koji poštuje ovaj uslov kaže se da je oporavljiv (može se oporaviti). Iz tog razloga se ne oporavljivi planovi izvršenja ne smeju dopustiti od strane DBMS-a. Plan izvršenja je oporavljiv ukoliko nijedna transakcija T u planu izvršenja S ne commit-uje svoje promene sve dok sve transakcije T' čije promene je transakcija T pročitala ne izvrše operaciju commit. Transakcija T čita promene transakcije T' ukoliko u planu izvršenja operacija upisa podataka X koju obavlja transakcija T' prednjači u odnosu na operaciju čitanja transakcije T . Plan S_a' je oporavljiv iako pati od problema izgubljenog ažuriranja. Plan izvršenja S_c nije oporavljiv jer T_2 čita vrednost X koju je upisala T_1 i T_2 commit-uje pre T_1 .

Primer oporavljivog plana izvršenja: $S_a' : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$

Primer ne oporavljivog plana izvršenja: $S_c: r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$

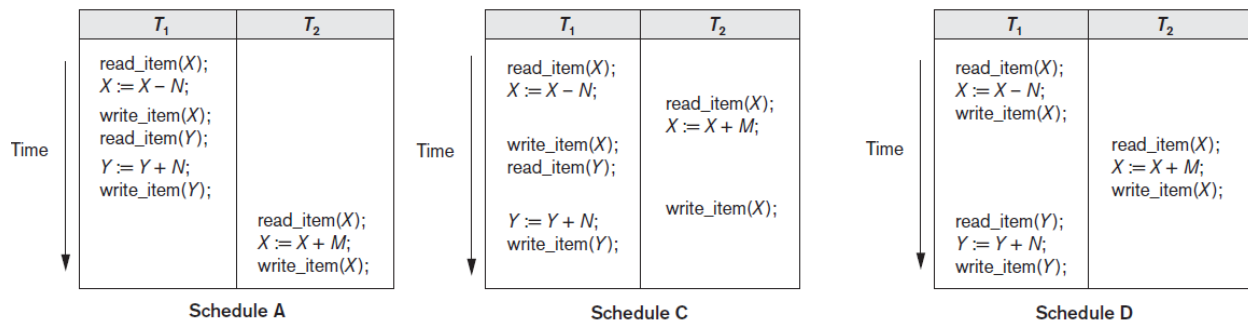
2.6.2 Serijabilni (Serializable) planovi izvršenja

Redni/serijski planovi izvršenja se uvek smatraju korektnim po pitanju konkurentnih transakcija. Kod serijskih planova nema preplitanja operacija različitih transakcija u toku izvršenja. Za plan izvršenja kaže se da je serijski ukoliko se kod svake transakcije T u planu izvršenja sve operacije transakcije T izvršavaju sekvencijalno jedna za drugom. U suprotnom je plan ne serijski.

Problem sa serijskim planovima je to što ograničava konkurentno izvršenje transakcija, pa su stoga serijski planovi ne prihvatljivi u praksi. Umesto njih se koriste planovi koji su ekvivalentni nekom

serijskom planu izvršenja (serijabilni). Plan izvršenja S od n transakcija je serijabilan ukoliko je ekvivalentan nekom serijskom planu izvršenja. Postoji više načina da se definiše ekvivalentnost planova izvršenja, a najjednostavniji uključuje poređenje rezultate izvršenja oba plana nad bazom podataka. Dva plana izvršenja su ekvivalentna po rezultatu (result equivalent) ukoliko dovode do istih rezultata nad istom početnom bazom podataka. U oba plana je redosled individualnih operacija isti nad istim podatkom. Najčešće se umesto ekvivalentnosti po rezultatu koristi ekvivalentnost po konfliktu i ekvivalentnost po pogledu.

Dva plana su ekvivalentna po konfliktu ako je uređenje bilo koje dve operacije koje se nalaze u konfliktu isto u oba plana izvršenja. Na osnovu definicije o ekvivalentnosti po konfliktu, za plan S kaže se da je konflikt serijabilan ukoliko je plan izvršenja S konflikt ekvivalentan nekom serijalnom planu izvršenja S'. U tom slučaju možemo preurediti operacije koje se ne nalaze u konfliktu u planu izvršenja S dok ne dobijemo ekvivalentan serijalni plan izvršenja S'. (Bilo bi lepo da se ubaci slika kao primer).



Ilustracija 5 - Različiti planovi izvršenja istih dveju transakcija T_1 i T_2 . Plan A je striktan. Plan D je serijabilan po konfliktu, a plan C nije serijabilan po konfliktu.

Manje restriktivna definicija ekvivalentnosti je ekvivalentnost po pogledu (view equivalence). Dva plana izvršenja S i S' su ekvivalentna po pogledu ukoliko ispunjavaju sledeće uslove:

1. Planovi izvršenja S i S' imaju isti skup transakcija kao i isti skup operacija tih transakcija
2. Za bilo koju operaciju $r_i(X)$ iz transakcije T_i plana izvršenja S, ako je pročitana vrednost X upisana od strane $w_j(X)$ transakcije T_j (ili je originalna vrednost X, pre početka plana izvršenja S), isti uslov mora važiti i za operaciju $r_i(X)$ iz transakcije T_i plana izvršenja S'
3. Ako je operacija $w_k(Y)$ transakcije T_k poslednja operacija koja upisuje vrednost u Y u planu izvršenja S, onda $w_k(Y)$ transakcije T_k mora takođe biti poslednja operacija koja upisuje vrednost u Y u planu izvršenja S'

Ideja ekvivalentnosti po pogledu je da sve operacije čitanja u transakcijama u oba plana izvršenja vide iste upisane rezultate, odnosno operacije upisa proizvode iste rezultate. Plan izvršenja je serijabilan po pogledu ukoliko je ekvivalentan po pogledu nekom serijalnom planu izvršenja.

Primer: S_g : $r_1(X)$; $w_2(X)$; $w_1(X)$; $w_3(X)$; c_1 ; c_2 ; c_3 ;

Kod plana izvršenja S_g operacije $w_2(X)$ i $w_3(X)$ su operacije praznog upisa pošto transakcije T_2 i T_3 ne čitaju vrednost X. Plan izvršenja S_g je serijabilan po pogledu jer je ekvivalentan serijalnom planu izvršenja T_1 , T_2 ,

T₃. Sa druge strane, plan izvršenja S_g nije serijabilan po konfliktu, jer nije ekvivalentan po konfliktu nijednom serijalnom planu izvršenja.

2.7 Dvofazno zaključavanje kao tehnika kontrole pristupa

Dvofazno zaključavanje je jedna od glavnih tehnika za upravljanje konkurentnog izvršenja transakcija i zasniva se na konceptu zaključavanja stavki podataka. Lock je posebna promenljiva povezana sa stavkom podataka koja govori da li je promenljiva slobodna za korišćenje u određenim operacijama. Postoji po jedan lock za svaku podatka u bazi podataka i ove promenljive se koriste za sinhronizaciju pristupa od strane konkurentnih transakcija.

2.7.1 Binarni lock

Binarni lock može imati samo dve vrednosti: locked (zaključano) i unlocked (otključano) odnosno vrednosti 1 i 0. Po jedan lock je rezervisan za svaki podatak X iz baze podataka. Ako je vrednost lock-a za podatak X jednaka 1, onda se podatku X ne može pristupiti pomoću operacije koja zahteva taj podatak. Ukoliko je vrednost lock-a jednaka 0, onda se tom podatku može pristupiti uz pomoć zahteva pri čemu se vrednost lock promenljive menja na 1. Trenutnu vrednost lock promenljive pridružene promenljivoj X označavamo sa lock(X).

Dve operacije koje su moguće nad binarnim lock-om su lock_item i unlock_item. Kada transakcija želi da pristupi podatku X, ona prvo izvršava operaciju lock_item(X). Ako je vrednost lock(X)=1, transakcija je primorana da čeka. Ako je lock(X)=0, onda se vrednost lock-a postavlja na 1 i transakcija može da pristupi podatku X. Kada transakcija završi sa upotrebom podataka X, ona izvršava operaciju unlock_item(X) koja vraća vrednost lock(X) promenljive nazad na 0 i na taj način omogućava pristup drugim transakcijama. Na ovaj način binarni lock obezbeđuje međusobno isključivanje nad istim podatkom. Operacije lock_item i unlock_item treba implementirati kao atomične operacije tj. ne sme se dopustiti preplitanje prilikom izvršenja ovih operacija kako dve transakcije ne bi zaključale isti podatak u isto vreme.

U svojoj najprostijoj formi binarni lock se implementira uz pomoć 3 vrednosti: ime podataka koji se zaključava, vrednost lock promenljive i red u kome transakcije čekaju na pristup lock-u. Sistem čuva samo vrednosti lock-ova podataka koji su trenutno zaključani u posebnoj lock tabeli koja se može organizovati kao heš file po imenu podataka.

Ukoliko se konkurentnost implementira binarnim lock-om, svaka transakcija mora da ispunjava sledeće uslove:

1. Transakcija T mora da izvrši operaciju lock_item(X) pre nego što može da izvrši operacije read_item(X) ili write_item(X)
2. Transakcija T mora izvršiti operaciju unlock_item(X) nakon što su sve operacije read_item(X) i write_item(X) završene
3. Transakcija ne sme da šalje zahtev za zaključavanje podataka X ukoliko ima već pribavljeni lock (ova operacija može i da se implementira tako da uvek vraća da je lock pribavljen ukoliko transakcija T sadrži lock nad X)
4. Transakcija T ne sme da izvrši operaciju unlock_item(X) osim ako nema već pribavljen lock za promenljivu X

2.7.2 Deljeni/Ekskluzivni (Shared/Exclusive) ili Read/Write lock

Problem sa binarnim lock-ovima je što su previše restriktivni po pitanju konkurentnog pristupa jer samo jedna transakcija u datom trenutku može da pristupa podatku. Možemo dopustiti svim transakcijama da zajedno pristupaju podatku ukoliko sve one obavljaju operaciju čitanja jer dve ili više operacije čitanja ne prouzrokuju konflikt. Sa druge strane ako transakcija želi da vrši upis nad podatkom X, moramo da obezbedimo da samo ona u datom trenutku ima pristup kako bi se izbegli mogući konflikti. Iz tog razloga, definiše se drugačija vrsta lock mehanizma koji podržava više vrsta zaključavanja. Ovakav mehanizam se naziva deljeni/ekskluzivni (shared/exclusive) lock i kod njega postoje tri moguće operacije: `read_lock(X)` – zaključava podatak X radi operacije čitanja, `write_lock(X)` zaključava podatak X radi operacije upisa i `unlock(X)` – uklanja bilo kakav lock koji je transakcija pribavila nad podatkom X. `Read_lock` se takođe naziva i deljeni lock zato što više transakcija u istom trenutku mogu da čitaju isti podatak. Drugi naziv za `write_lock` je ekskluzivni lock jer samo jedna transakcija može držati ovakav lock u datom trenutku.

Razlika u implementaciji u odnosu na binarni lock je to što je u ovom slučaju potrebno pamtitu listu transakcija jer u slučaju kada više transakcija drži lock za čitanje moramo pamtitu svaku od njih.

Uslovi koje transakcije moraju ispoštovati kod primene deljenog/ekskluzivnog lock mehanizma su:

1. Transakcija T mora izvršiti `read_lock(X)` ili `write_lock(X)` pre nego što izvrši bilo koju `read_item(X)` operaciju. Ove dve operacije pribavljaju deljeni i ekskluzivni lock respektivno.
2. Transakcija T mora izvršiti `write_lock(X)` pre nego što izvrši bilo koju `write_item(X)` operaciju
3. Transakcija T mora da izvrši `unlock(X)` operaciju nakon što završi sa svim `read_item(X)` i `write_item(X)` operacijama
4. Transakcija T ne može da izvrši operaciju `read_lock(X)` ili `write_lock(X)` ukoliko već poseduje deljeni ili ekskluzivni lock respektivno (ovo pravilo može biti definisano malo ležernije što ćemo videti kasnije)
5. Transakcija T ne sme da izvrši operaciju `write_lock(X)` ukoliko već poseduje deljeni ili ekskluzivni lock nad podatkom X (ovo pravilo može biti definisano malo ležernije što ćemo videti kasnije)
6. Transakcija T ne sme da obavi operaciju `unlock(X)` ukoliko ne poseduje deljeni ili ekskluzivni lock nad podatkom X

Ponekad je potrebno popustiti uslove 4 i 5 iz prethodne liste kako bi dopustili konverziju lock-ova tj. ukoliko transakcija već poseduje deljeni (shared) lock nad podatkom X ona može pod određenim uslovima da konvertuje deljeni lock u ekskluzivni lock. Da bi se ova „nadogradnja“ (upgrade) dopustila transakcija T koja traži ovu nadogradnju mora biti jedina transakcija koja poseduje lock nad tim podatkom. U suprotnom transakcija mora da čeka da se taj uslov ispuni. Takođe je moguće izvršiti konverziju tako da se postojeći ekskluzivni lock „unazadi“ (downgrade), tj. da se konvertuje u deljeni lock.

2.7.3 Garantovanje serijabilnosti pomoću dvofaznog protokola zaključavanja

Za transakciju se kaže da poštuje dvofazni protokol zaključavanja ukoliko sve operacije zaključavanja (`read_lock` i `write_lock`) prethode prvoj operaciji otključavanja (`unlock`) u transakciji. Kod takvih transakcija možemo uočiti dve faze: fazu ekspanzije ili rasta tokom koje se mogu pribaviti novi lock-ovi nad podacima i fazu skupljanja tokom koje se postojeći lock-ovi mogu osloboditi, ali se ne mogu zatražiti

novi. Ukoliko nadogradnja lock-ova moguća, onda se ona mora obaviti za vreme faze ekspanzije, a ako je downgrade lock-ova moguć onda se on mora obaviti u fazi skupljanja.

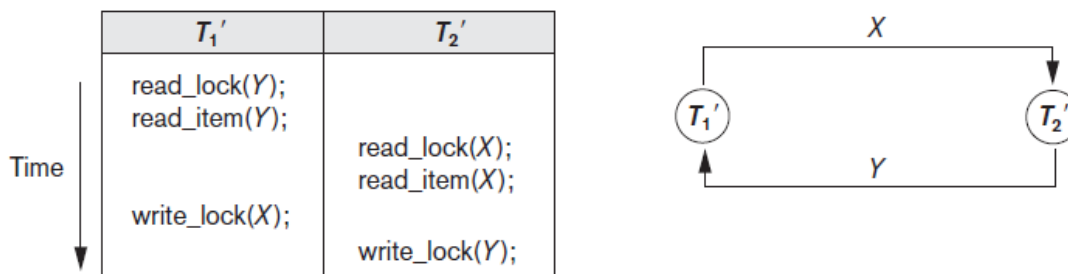
Može se pokazati da ukoliko svaka transakcija u planu izvršenja poštuje protokol dvofaznog zaključavanja, da će dobijeni plan izvršenja biti garantovano serijabilan (i na taj način ukloni potrebu za proveru serijabilnosti). Iako ovaj protokol garantuje serijabilnost, on ne dopušta sve moguće serijabilne planove izvršenja. Ovaj protokol je poznatiji kao osnovni dvofazni lock protokol (basic 2PL – osnovni protokol dvofaznog zaključavanja, dve faze su faza rasta i faza skupljanja).

Varijacija ovog protokola poznatija kao konzervativni 2PL (conservative 2PL ili static 2PL) zahteva da transakcija zatraži sve svoje lock-ove pre početka svojeg izvršenja, tako što će naznačiti skup podataka koje će pročitati i skup podataka koje će upisivati. Ukoliko transakcije ne može da pribavi lock nad nekim podatkom, ona je primorana da čeka. Ova varijanta dvofaznog zaključavanja je otporna na deadlock-ove jer se svi zahtevi transakcije ispunjuju na samom početku. Sa druge strane ovaj protokol je teško primenjivati u praksi jer nije moguće u svim situacijama naznačiti skup podataka koji će se koristiti.

U praksi mnogo popularnije rešenje je primenom striktnog 2PL koji garantuje striktne planove izvršenja. U ovoj varijaciji transakcija ne sme da oslobodi nijedan ekskluzivni lock sve dok ne commit-uje ili otkaže svoje promene. Striktni 2PL nije otporan na deadlock-ove. Restriktivnija implementacija ovog protokola je rigorozni 2PL koji takođe garantuje striktne planove izvršenja. Kod rigoroznog 2PL transakcija T ne sme da oslobodi nijedan od svojih lock-ova sve do trenutka kada commit-uje ili otkaže svoje promene (pa je iz tog razloga i lakši za implementaciju). Važan detalj koji možemo primetiti kod ove dve varijacije 2PL je to da prva (konzervativna implementacija) počinje u fazi skupljanja transakcije, a kod druge (striktna implementacija) faza ekspanzije traje tokom čitavog trajanja transakcije.

2.8 Obrada deadlock-ova

Deadlock nastaje kada svaka transakcija T u skupu od dve ili više transakcija čeka na neki podatak koji je zaključen od strane druge transakcije T' iz ovog skupa. Svaka transakcija sedi u redu čekanja na podatak i čeka da se podatak oslobodi, ali pošto i druga transakcija takođe čeka, do oslobađanja tog podatka nikada neće doći. Na Ilustracija 6 može se videti primer jednog plana izvršenja koji dovodi do pojave deadlock-a. Postoje dva načina ophođenja prema deadlock-ovima: prevencija deadlock-ova – primenom određenih protokola sprečavaju se planovi izvršenja u kojima su deadlock-ovi mogući, i detektovanje deadlock-ova i ukoliko su nastali otkazati jednu od transakcija koja je uključena u deadlock.



Ilustracija 6 - Primer izvršenja transakcija T_1' i T_2' koji dovodi do pojave deadlock-a (levo), graf deadlock-a (desno)

2.8.1.1 Protokoli za prevenciju deadlock-ova

Kao što smo već spomenuli konzervativni dvofazni protokol zaključavanja garantuje da neće doći do deadlock-a jer transakcija zahteva sve lock-ove na samom početku, pa ukoliko ne može da pribavi određeni lock ni ne počinje sa izvršenjem.

Deo protokola za prevenciju koristi koncept vremenskih markica (timestamp) $TS(T)$ koji predstavlja jedinstveni identifikator transakcije dodeljen na samom početku njenog izvršenja. Za vremenske markice važi sledeće: ukoliko je transakcija T_1 počela pre transakcije T_2 , onda je $TS(T_1) < TS(T_2)$ (starija transakcija ima manju vremensku markicu). Postoje dva protokola zasnovana na vremenskim markicama: čekaj-umri (wait-die) i Rani-čekaj (wound-wait). Pretpostavimo da transakcija T_i pokušava da zaključa podatak X, ali ne može to da uradi jer transakcija T_j već poseduje traženi lock. Dva protokola se razlikuju u sledećem:

- Čekaj-umri (wait-die) – Ako je $TS(T_i) < TS(T_j)$, onda (T_i je starija od T_j) T_i može da čeka na, u suprotnom (T_i je mlađa od T_j) otkazati T_i (T_i umire) i restartovati je kasnije sa istom vremenskom markicom.
- Rani-čekaj (wound-wait) – Ako je $TS(T_i) < TS(T_j)$, onda (T_i je starija od T_j) otkazati T_j (T_j je ranjena) i restartovati je kasnije sa istom vremenskom markicom, u suprotnom (T_i je mlađa od T_j) T_i može da čeka.

Kod čekaj-umri protokola, starija transakcija može da čeka na mlađu da oslobodi lock, dok se mlađa transakcija otkazuje i restartuje ukoliko zahteva podatak zaključan od strane neke starije transakcije. Rani-čekaj pristup radi suprotno, mlađa transakcija može da čeka na stariju da oslobodi lock, dok ukoliko starija zatraži podatak zaključan od strane mlađe, mlađa transakcija se otkazuje i restartuje. U oba slučaja se otkazuje mlađa transakcija pod pretpostavkom da je obavila manje posla.

Još jednu grupu protokola za prevenciju deadlock-ova čine protokol bez čekanja (no waiting) i protokol opreznog čekanja (cautious waiting). Kod protokola bez čekanja, ako transakcija ne uspe da odmah pribavi lock, onda se ona otkazuje i restartuje nakon nekog vremena. U ovom slučaju ni jedna transakcija nikad ne čeka pa se ne javljaju deadlock-ovi. Kod opreznog čekanja transakcija sme da čeka jedino ako transakcija koja poseduje lock nad traženim podatkom nije blokirana. U suprotnom se otkazuje i restartuje posle nekog vremena.

2.8.1.2 Detekcija deadlock-ova

Malo praktičniji pristup obrade deadlock-ova je njihova detekcija, odnosno dopuštamo da se deadlock-ovi jave i tada ih detektujemo i rešavamo. Ovaj način je primamljiviji jer u najvećem broju slučajeva postoji mala smetnja između konkurentnih transakcija pa se i deadlock-ovi retko javljaju.

Jednostavan način za detektovanje deadlock-ova je kreiranjem i održavanjem grafova čekanja. Za svaku transakciju koja se trenutno izvršava kreira se po jedan čvor u grafu. Ukoliko transakcija T_i čeka na podatak X koji je zaključan od strane transakcije T_j , onda u grafu postoji grana $T_i \rightarrow T_j$. Kada se lock nad podatkom X ukloni, odgovarajuća grana u grafu se briše. Deadlock postoji ukoliko u grafu čekanja postoji ciklus. Problem sa ovim pristupom je kada vršiti proveru ciklusa. Jedna mogućnost je nakon dodavanja svake grane u graf, ali bi to znatno uticalo na cenu izvršenja. Drugi prihvatljiviji uslov može biti promena broja transakcija koje se trenutno izvršavaju ili broja transakcija koje čekaju na neki lock.

Ukoliko se deadlock detektuje, neke od transakcija se moraju otkazati. Za određivanje koju transakciju treba otkazati koristi se algoritam za odabir žrtve (victim selection algorithm). Ovaj algoritam bi generalno trebao da izbegava transakcije koje se izvršavaju već duže vreme ili one koje su ažurirale veliki broj podataka, jer je cena otkazivanja takvih transakcija velika.

Jedan jednostavnijih algoritama za detekciju deadlock-ova je primenom timeout-a. Prednosti ove metode su jednostavnost i mala složenost. Kod ove metode, ako transakcija čeka duži vremenski period od neke predefinisane vrednosti, onda sistem pretpostavlja da transakcija učestvuje u deadlock-u i otkazuje je bez obzira da li deadlock zaista postoji.

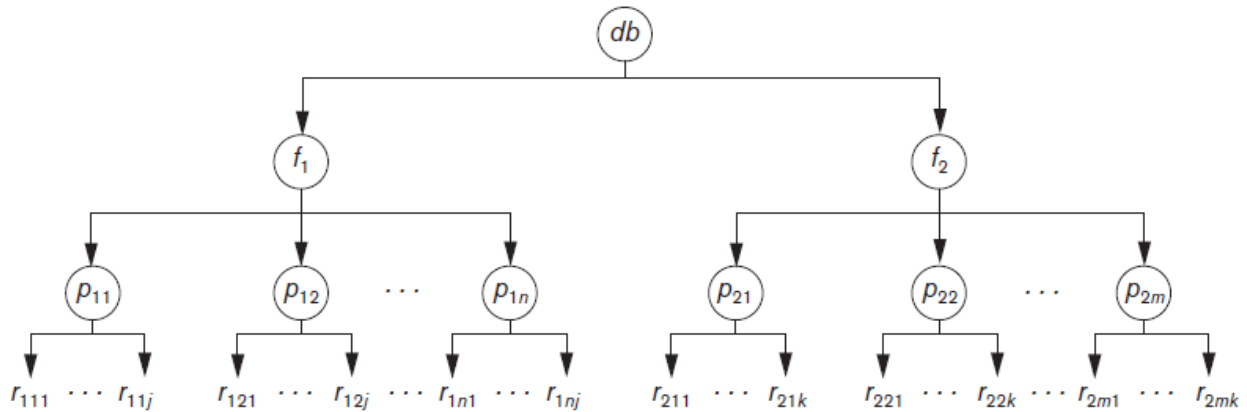
2.9 Granularnost stavki podataka

Sistemi za upravljanje konkurentnosti zasnivaju se na pretpostavci da se baza podataka može posmatrati kao imenovani skup stavki podataka. Stavka podataka može biti:

- Zapis u bazi podataka (database record)
- Vrednost atributa unutar zapisa
- Blok na disku
- Ceo fajl
- Cela baza podataka

Veličina stavke podataka se još naziva i granularnost podataka. Pojam fine granularnosti (fine granularity) odnosi se na podatke manje veličine dok se pojam grube granularnosti (coarse granularity) odnosi na podatke većih veličina. Postoje prednosti i mane primene svake vrste granularnosti prilikom zaključavanja podatka. Što je veća stavka podataka, smanjuje se nivo konkurentnosti sistema jer transakcije zaključava veću stavku podataka pa verovatno i one delove koje neće koristiti u toku svog izvršenja. Sa druge strane, ukoliko koristimo manje stavke podataka imamo više podataka. Pošto svaki podatak ima svoj lock, moramo imati više lock-ova kako bi pokrili isti broj podataka, a sa više lock-ova moramo izvršavati više operacija zaključavanja i otključavanja. Ne postoji granularnost koja odgovara u svim slučajevima. Zato se tip granularnosti određuje od vrste primene. Ukoliko transakcija koristi samo jedan ili par zapisa u toku svojeg izvršenja, onda možemo koristiti finu granularnost kako bi oslobodili ostale podatke drugim transakcijama. Ako transakcija koristi veliki broj podataka iz istog fajla, možda je lakše zaključati nekoliko blokova ili ceo fajl prilikom izvršenja transakcije.

Pošto najpogodniji tip granularnosti zavisi od obrade koju transakcija obavlja, najbolje je da DBMS podržava više nivoa granularnosti i da se odgovarajući tip odabere za konkretnu transakciju. Protokol 2PL koji podržava ovo naziva se 2PL protokol sa više nivoa granularnosti. Kod ovog protokola su različiti stavke podataka uređene u hijerarhiju po veličini podataka. Na samom vrhu nalazi se čvor koji predstavlja celu bazu podataka, a listovi stabla su pojedinačni zapisi ili njihovi atributi.



Ilustracija 7 - Ilustracija hijerarhije granularnosti stavki podataka

Za implementaciju ovog protokola biće nam potrebne dodatne vrste lock-ova koji se nazivaju lock-ovi namere (intention locks). Postoje tri vrste lock-ova namere:

1. Namera za deljeni lock (intention-shared – IS) – ovaj lock označava da će transakcija zatražiti jedan ili više deljenih lock-ova za potomke trenutnog čvora
2. Namera za ekskluzivni lock (intention-exclusive – IX) – ovaj lock označava da će transakcija zatražiti jedan ili više ekskluzivnih lock-ova za potomke trenutnog čvora
3. Deljeni sa namerom za ekskluzivni lock (shared-intention-exclusive – SIX) – ovaj lock označava da je trenutni čvor zaključan deljenim lock-om, ali da će transakcija zatražiti jedan ili više ekskluzivnih lock-ova za potomke trenutnog čvora

	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

Ilustracija 8 - Kompatibilnost između različitih lock-ova kod 2PL protokolasa više nivoa granularnosti

Ovaj protokol se sastoji od sledećih pravila:

1. Kompatibilnost međusobnih lock-ova mora poštovati uslove navedene u tabeli na Ilustracija 8
2. Uvek se prvo zaključava koren stabla (primenom bilo kojeg tipa lock-a)
3. Transakcija T može zaključati čvor N pomoću S (deljenog) lock-a jedino ako je roditeljski čvor čvora N već zaključan od strane transakcije T korišćenjem IS ili IX lock-a

4. Transakcija T može zaključati čvor N pomoću X, IX ili SIX lock-a jedino ako je roditeljski čvor čvora N već zaključan od strane transakcije T korišćenjem IX ili SIX lock-a
5. Transakcija T može da zaključa bilo koji čvor jedino ako nije otključala nijedan drugi čvor (kako bi se obezbedio 2PL protokol)
6. Transakcija T može da otključa čvor N jedino ako nijedan potomak čvora N nije trenutno zaključan od strane transakcije T

2.10 Kontrola konkurentnosti u B+ stablima

Najtrivijalniji pristup upravljanja konkurentnosti u B+ stablima bio bi ignorisanje celokupne strukture stabla i tretirati svaku stranicu kao zaseban podatak koji se može zaključati. Ovaj jednostavan protokol doveo bi do velikog broja smetnji u višim delovima stabla. Malo efikasniji protokoli iskorišćavaju hijerarhijsku strukturu stabla indeksa i obezbeđuju serijabilnost i oporavljivost. Postoje dve važne činjenice koje se koriste pri implementaciji ovih protokola:

1. Viši nivoi stabla uglavnom služe za usmeravanje traženja jer se svi podaci nalaze u listovima stabla
2. Prilikom upisa, unutrašnji čvor mora biti zaključan primenom ekskluzivnog lock-a jedino ako se podela čvorova potomaka može propagirati od njega

Pretrage u indeksu zahtevaju da transakcija pribavi deljene lock-ove počevši od korena stabla pa sve do odgovarajućeg lista koji sadrži traženi podatak. Na osnovu prve činjenice možemo uočiti da se u ovom slučaju lock nad čvorom može osloboditi odmah nakon što se pribavi lock nad njegovim detetom (jer se nikada ne vraćamo pri traženju).

Prilikom dodavanja moramo pribaviti ekskluzivne lock-ove od korena sve do lista stabla gde se zapis dodaje jer se podelom potomaka pri dodavanju podaci mogu propagirati naviše u stablu. Međutim, ukoliko čvor potomak (kroz koji se proteže put dodavanja zapisa) ima dovoljno slobodnog prostora za dodavanje tako da ne dolazi do njegove podele, možemo da oslobodimo lock nad trenutnim čvorom pošto se dodavanje neće propagirati do njega. Na taj način lock-ovi koji se pribavljaju prilikom dodavanja obezbeđuju da transakcije, koje prate put dodavanja podatka, čekaju na najranijoj tački koja može biti pod uticajem dodavanja.

3 Obrada transakcija kod Oracle baze podataka

Kod Oracle baze transakcija započinje prvom izvršnom SQL naredbom. Izvršne SQL naredbe su naredbe, uključujući DML (data manipulation language) i DDL (data definition language) naredbe i izraz SET TRANSACTION naredbu, koje generišu poziv ka bazi. Kada transakcija započne sa izvršenjem, Oracle dodeljuje transakciji slobodan tablespace (tabelarni prostor – logička jedinica za čuvanje podataka, sastoji se od više fajlova podataka) gde će čuvati entitete za slučaj rollback-a transakcija. Transakcija se završava kada se ispuni jedan od sledećih uslova:

- Korisnik pošalje COMMIT ili ROLLBACK naredbu bez SAVEPOINT klauzule
- Korisnik pokrene neku od DDL naredbi poput CREATE, DROP, RENAME ili ALTER naredbu. Ako trenutna transakcija sadrži neku od DML naredbi, Oracle prvo završava i commit-uje transakciju, a zatim počinje sa izvršenjem DDL naredbe
- Korisnik izgubi konekciju sa Oracle bazom. U tom slučaju se trenutna transakcija automatski commit-uje
- Korisnički proces se abnormalno završi. U ovom slučaju se transakcija rollback-uje

Nakon što se transakcija završi, prva sledeća SQL naredba automatski pokreće novu transakciju.

3.1 System change number (SCN)

Broj promene u sistemu (SCN - System change number) predstavlja logičku vremensku markicu (timestamp) koja se koristi isključivo unutar Oracle baze podataka. SCN-ovi služe za uređenje redosleda događaja unutar baze podataka kako bi se obezbedila ACID svojstva. Generišu se monotono rastućem poretku i iz tog razloga se mogu koristiti poput sata jer SCN pokazuje logičku tačku u vremenu. Svako transakciji se dodeljuje jedan SCN. Kada transakcija ažurira neki red, baza zapisuje SCN kada se promena desila. Izmene iste transakcije imaju isti SCN. Kada se transakcija commit-uje, baza upisuje SCN za taj commit.

SCN-ovi se generišu inkrementiranjem u sistemskom globalnom prostoru (SGA - system global area). Kada transakcija izmeni neke podatke, baza upisuje novi SCN u undo segment podataka (undo data segment) koji je dodeljen toj transakciji. Nakon toga proces za upis log-ova upisuje zapis za commit-naredbu u online redo log. Ovaj redo zapis sadrži jedinstveni SCN transakcije.

3.2 Commit transakcije

Commit naredbom izmene napravljene transakcijom postaju trajne. Kada se transakcija commit-uje dešava se sledeće:

- Baza generiše SCN za commit naredbu. U internu tabelu transakcije koja služi za pamćenje undo zapisa, dodaje se zapis da je transakcija commit-ovana. Odgovarajući jedinstveni SCN transakcije se dodeljuje i upisuje u tabelu transakcija.
- Proces za upis log-ova (log writer process – LGWR) upisuje preostale redo log zapise u bafere online redo log-a i upisuje SCN u online redo log. Ova atomična operacija označava da je transakcija commit-ovana.

- Oracle baza oslobađa lock-ove pribavljene nad redovima i tabelama (korisnici koji su čekali na neki od ovih podataka sada nastavljaju sa izvršenjem)
- Oracle baza briše savepoint-e
- Oracle baza obavlja commit cleanout („čišćenje nakon commit-a)
- Oracle označava transakciju završenom

LGWR upisuje redo operacije u online redo log sinhrono i transakcija čeka da svi podaci iz bafera budu upisana na disk pre nego što vrati commit korisniku. Pored sinhronog, ovaj upis se može obaviti i asinhrono tako da transakcija ne mora da čeka da ceo redo log bude upisan na disku, već odmah vraća commit korisniku.

3.3 Rollback transakcije

Prilikom rollback-a transakcije koja nije commit-ovana, vraćaju se sve izmene koje je transakcija napravila prilikom svog izvršavanja. Oracle obavlja sledeće korake prilikom rollback-a:

- Poništavaju se sve izmene napravljene SQL naredbama u transakciji korišćenjem odgovarajućih undo segmenta. Unutar tabele transakcija čuva se pokazivač na undo podatke transakcije (u obrnutom redosledu od rada u aplikaciji). Baza čita undo zapise iz undo segmenta, poništava promene napravljene transakcijom i označava ove undo zapise kao obavljene.
- Oslobađaju se svi lock-ove koje je transakcija pribavila
- Brišu se svi savepoint-ovi kreirani unutar transakcije
- Završava se izvršenje transakcije

3.4 Autonomne transakcije

Autonomna transakcija je nezavisna transakcija koja se poziva iz neke druge glavne transakcije. Izvršenje glavne transakcije se može suspendovati, izvrše se SQL naredbe autonomne transakcije i ona se commit-uje ili rollback-uje i zatim se nastavlja sa izvršenjem glavne transakcije.

Autonomne transakcije su korisne za akcije koje se izvršavaju nezavisne od pozivajuće transakcije. Na primer prilikom kupovine deonice, nezavisno od toga da li je kupovina uspeła ili ne želimo da sačuvamo podatke o prodavcu. Autonomne transakcije imaju sledeća svojstva:

- Autonomna transakcija ne vidi ne commit-ovane promene glavne transakcije i ne deli lock-ove ili resurse sa glavnom transakcijom
- Promene napravljene unutar autonomne transakcije su vidljive drugim transakcijama nakon commit-a autonomne transakcije, nezavisno od toga da li je glavna transakcija završena ili ne.
- Autonomne transakcije mogu kreirati nove autonomne transakcije. Ne postoje ograničenja po pitanju broja autonomnih transakcija osim ograničenja resursa sistema.

3.5 Konzistentnost čitanja primenom više verzija

U Oracle bazi podataka, multiverzionisanje (multiversioning) je sposobnost istovremenog materijalizovanja više verzija istog podataka. Oracle baza podataka održava konzistentnost čitanja primenom više verzija podataka. Upiti ka Oracle bazi podataka imaju sledeće karakteristike:

- Konzistentnost čitanja upita – Podaci vraćeni upitom su commit-ovani i konzistentni u jednom trenutku u vremenu. Oracle nikada ne dozvoljava prljavo čitanje (dirty read) koja se dešavaju kada transakcija pročita ne commit-ovane podatke druge transakcije.
- Ne blokirajući upiti – Čitaoci i pisci se međusobno ne blokiraju

3.5.1 Konzistentnost čitanja na nivou naredbe

Oracle baza podataka uvek obezbeđuje konzistentnost čitanja na nivou naredbe odnosno garantuje da su podaci vraćeni jednim upitom commit-ovani i konzistentni u jednom trenutku u vremenu. Trenutak u vremenu sa kojim je izvršenje jedne SQL naredbe konzistentno zavisi od izolacionog nivoa transakcije i od prirode samog upita:

- Prilikom read committed izolacionog nivoa, naredba je konzistentna sa trenutkom kada je naredba počela sa izvršenjem. Na primer, ako je SELECT naredba počela sa izvršenjem u trenutku SCN=1000 onda je ona konzistentan sa SCN=1000.
- Prilikom serializable ili read-only izolacionog nivoa transakcija, naredba je konzistentna sa trenutkom početka transakcije. Na primer ako je transakcija počela sa SCN=1000 i za vreme transakcije se izvrše višestruke SELECT naredbe, onda je svaka naredba konzistentna sa SCN=1000.
- Prilikom Flashback upita, SELECT naredba specificira trenutak u vremenu sa kojim će biti konzistentna. Na primer, može se poslati upit bazi u trenutku kakva je ona bila prethodnog četvrtka u 2 sata popodne.

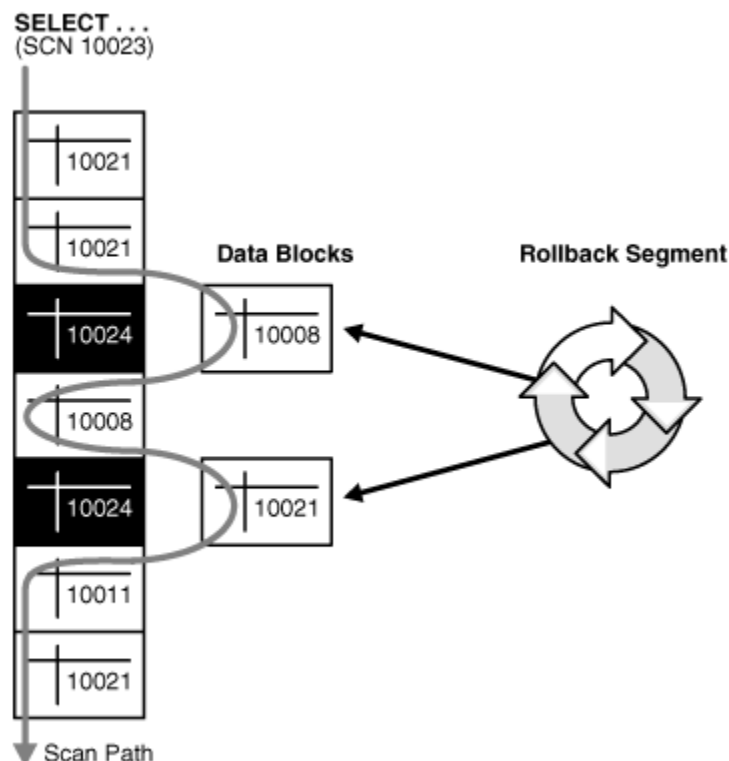
3.5.2 Konzistentnost čitanja na nivou transakcije

Oracle baza podataka takođe obezbeđuje konzistentnost čitanja za sve upite jedne transakcije. U ovom slučaju sve naredbe unutar transakcije vide podatke u istom trenutku u vremenu kada je transakcija počela sa izvršenjem.

3.5.3 Konzistentnost čitanja i undo segmenti

Da bi obezbedila model sa više verzija podataka, baza mora kreirati konzistentan skup podataka kada se simultano šalju upiti za čitanje tabele i njeno ažuriranje. Oracle baza podataka ovo postiže primenom undo podataka.

Kada god dođe do promene podatka, Oracle baza podataka kreira undo zapise koje upisuje u undo segmente. Undo segmenti sadrže stare vrednosti izmenjenih podataka još ne commit-ovanih transakcija. Na taj način imamo više verzija istih podataka u različitim trenucima u vremenu. Oracle može da koristi snapshot podataka u različitim trenucima u vremenu i na taj način obezbedi prikaz konzistentnih pogleda i ne blokirajućih upita.



Ilustracija 9 - Primer izvršavanja SELECT upita sa SCN=10023 i način primene više verzija za obezbeđivanje konzistentnosti izvršenja upita

3.6 Izolacioni nivoi transakcija

ANSI/ISO standard definiše četiri nivoa izolacije transakcija: Read uncommitted, Read committed, Repeatable read i Serializable. U tabeli ispod dat je svaki od ovih nivoa izolacije kao i problemi koji se mogu javiti prilikom primene određenog nivoa izolacije.

NIVO IZOLACIJE	DIRTY READ	NONREPEATABLE READ	PHANTOM READ
READ UNCOMMITTED	Da	Da	Da
READ COMMITTED	Ne	Da	Da
REPEATABLE READ	Ne	Ne	Ne
SERIALIZABLE	Ne	Ne	Ne

3.7 Nivoi izolacije kod Oracle-a:

Oracle baza podataka podržava sledeće nivoe izolacije transakcija:

- Read committed
- Serializable
- Read-Only

3.7.1 Read committed izolacioni nivo

Kod read committed izolacije transakcija, svaki upit u transakciji vidi samo podatke koji su commit-ovani pre početka izvršenja tog upita (ne pre početka transakcije). Ovo je default-ni nivo izolacije transakcija.

Pri upotrebi ovog izolacionog nivoa može doći do konflikta pri upisu kada transakcija pokuša da ažurira podatak koji je već ažuriran od strane neke ne commit-ovane transakcije. Ova ne commit-ovana transakcija koja sprečava modifikaciju, naziva se blokirajuća transakcija. Transakcija sa read committed izolacijom čeka na završetak blokirajuće transakcije i da oslobodi sve lock-ove. Ukoliko je blokirajuća transakcija rollback-ovana, read committed transakcija se izvršava kao da prethodne transakcije nije ni bilo. Ukoliko se blokirajuća transakcija uspešno commit-uje, read committed transakcija će upis obaviti nad novim podacima.

3.7.2 Serializable izolacioni nivo

Kod serializable izolacionog nivoa, transakcija vidi samo promene commit-ovane pre početka izvršenja transakcije (ne pre početka samog upita) kao i promene napravljene unutar transakcije. Serializable transakcije se izvršava u okruženju kao da nijedan drugi korisnik ne modifikuje podatke. Oracle dozvoljava da serializable transakcija modifikuje zapis (red podataka) jedino ako su sve izmene nad tim zapisom commit-ovane pre početka njenog izvršenja. Baza generiše grešku (ORA-08177: Cannot serialize access for this transaction) ukoliko serializable transakcija pokuša da ažurira ili obriše podatak koji je izmenjen nakon početka njenog izvršavanja. Ukoliko aplikacija naiđe na ORA-08177 grešku, mogući su sledeći koraci:

- Commit izmena koje su obavljene do tog trenutka
- Izvršenje dodatnih naredbi, na primer nakon rollback-ovanja na neki savepoint
- Rollback cele transakcije

3.7.3 Read-Only izolacioni nivo

Read-Only izolacioni nivo je sličan serializable izolacionom nivou, osim što transakcija ne dozvoljava izvršenje DML naredbi osim ako nije u pitanju SYS korisnik. Read-Only transakcije nisu podložne ORA-08177 greški. Konzistentnost se obezbeđuje rekonstrukcijom podataka na osnovu undo segmenta.

3.8 Vrsite lock-ova

Oracle baza podataka uvek automatski koristi što je moguće manju restriktivnost kako bi obezbedila što bolju konkurentnost. Oracle podržava dva tipa zaključavanja:

- Ekskluzivni (exclusive) lock – Ovaj tip zaključavanja sprečava da asocirani resurs bude deljen sa drugim transakcijama. Ekskluzivni lock se koristi kada transakcija menja podatke. Prva transakcija koja pribavi ekskluzivni lock je jedina koja može menjati asocirani resurs.
- Deljeni (shared) lock – Deljeni lock omogućava da asocirani resurs bude deljen od strane više transakcija. Više korisnika mogu da čitaju isti deljeni podatak. Ovaj lock sprečava pribavljanje ekskluzivnog lock-a nad istim resursom.

3.9 Konverzija lock-ova i njihova eskalacija

Oracle obavlja konverziju lock-ova po potrebi. Konverziju obavlja baza automatski pretvarajući lock manje restriktivnosti u lock veće restriktivnosti. Na primer ukoliko transakcija izda SELECT ... FOR UPDATE naredbu nekog radnika i kasnije ažurira zaključani red, baza će zameniti deljeni lock na nivou tabele ekskluzivnim lock-om na nivou tabele. Transakcija pribavlja ekskluzivni lock na nivou reda za sve redove koje dodaje, ažurira ili briše unutar transakcije. Pošto su lock-ovi na nivou redova uvek najveće moguće restriktivnosti, nema potrebe za njihovom konverzijom.

Eskalacija lock-ova se dešava kada postoji veći broj lock-ova na određenom nivou granularnosti (na primer red) i baza poveća nivo granularnosti na viši nivo (na primer tabela). Ukoliko neka sesija zaključa veliki broj redova u tabeli, neke baze automatski eskaliraju lock-ove na nivou reda na lock na nivou tabele.

Oracle baza podataka nikad ne eskalira lock-ove jer se eskalacijom drastično povećavaju šanse za pojavu deadlock-a.

3.10 Vrste automatskih lock-ova

Oracle baza podataka automatski zaključava resurse zahtevane od transakcije kako bi sprečila druge transakcije od ekskluzivnog pristupa istom resursu. Oracle podržava sledeće vrste lock-ova:

- DML lock-ovi – štite podatke
- DDL lock-ovi – štite strukturu šeme podataka
- System lock-ovi – štite internu strukturu baze podataka

3.10.1 DML lock-ovi

DML lock garantuje integritet podataka kome pristupaju više konkurentnih korisnika. Mogu biti:

- Lock reda (row lock)
- Lock tabele (table lock)

Lock reda je lock nad jednim redom u tabeli. Transakcije pribavljaju lock reda kada izvršavaju neku modifikaciju poput INSERT, UPDATE, DELETE, MERGE ili SELECT..FOR UPDATE naredbu. Lock reda postoji sve dok transakcija ne commit-uje ili rollback-uje promene. Prilikom pribavljanja lock-a reda, transakcija takođe pribavlja i lock nad tabelom u kojoj se taj red nalazi. Lock tabele sprečava izvršenje DDL operacija nad tabelom dok se trenutna transakcija ne završi. Lock tabele se pribavlja prilikom izvršenja naredbi za izmenu podataka poput INSERT, UPDATE, DELETE, MERGE ili SELECT..FOR UPDATE ili izvršenjem LOCK TABLE naredbe.

3.10.2 DDL lock-ovi

Data dictionary lock (DDL) štiti definiciju šema objekata za vreme izvršenja DDL operacije ili ukoliko neko referencira objekat. Samo individualni objekti koji se menjaju ili koji su referencirani se zaključavaju.

Baza nikad ne zaključava ceo data dictionary. DDL lock-ovi se dele na:

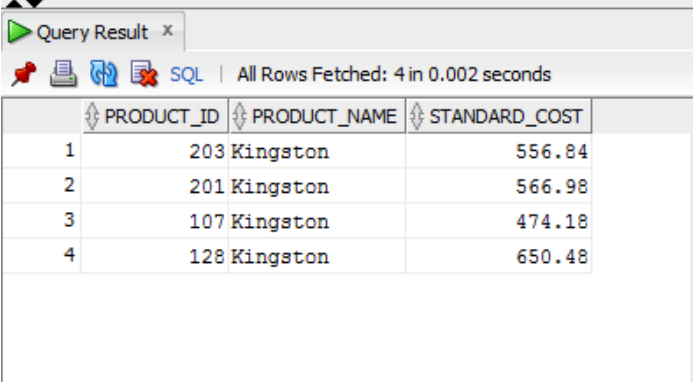
- Ekskluzivne DDL lock-ove – sprečavaju druge sesije da pribave DDL ili DML lock nad istim resursom
- Deljene DDL lock-ove – sprečavaju druge sesije da pribave ekskluzivni lock nad istim resursom i omogućavaju deljenje resursa ukoliko su DDL operacije koje se obavljaju slične
- Lock-ovi parsiranja (parse lock) – ovi lock-ovi se pribavljaju od strane SQL naredbi ili PL/SQL delova programa, nad svakim schema objektom koji je referenciran u naredbi. Koriste se kako bi se keš o prevedenoj SQL naredbi obrisao u slučaju izmene objekata scheme koje ta naredba referencira.

3.10.3 Sistemski lock-ovi

Oracle baza podataka koristi različite vrste sistemskih lock-ova kako bi zaštitila pristup unutar strukturama unutar same baze podataka. Ovi mehanizmi su nepristupačni za krajnjeg korisnika i ne mogu se kontrolisati. U njih spadaju lečevi (latches), muteksi (mutexes) i interni lock-ovi.

3.11 Primer izvršenja dve transakcije sa READ COMMITED nivoom izolacije:

```
select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```



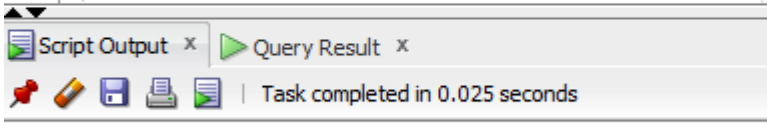
Query Result x

SQL | All Rows Fetched: 4 in 0.002 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	556.84
2	201	Kingston	566.98
3	107	Kingston	474.18
4	128	Kingston	650.48

Ilustracija 10 - Sesija 1 šalje upit za listom proizvoda koji se zovu Kingston

```
update products
set standard_cost = 500
where product_id = 203;
```



Script Output x | Query Result x

Task completed in 0.025 seconds

1 row updated.

Ilustracija 11 - Sesija 1 implicitno započinje transakciju 1 ažuriranjem cene proizvoda sa id = 203. Default-ni nivo izolacije je READ COMMITED

Conn

```

from products
where product_name like 'Kingston';

update products
set standard_cost = 500
where product_id = 203;

```

Script Output x Query Result x

Task completed in 0.025 seconds

1 row updated.

OT2

```

set transaction isolation level read committed;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

```

Script Output x Query Result x

All Rows Fetched: 4 in 0.012 seconds

PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203 Kingston	556.84
2	201 Kingston	566.98
3	107 Kingston	474.18
4	128 Kingston	650.48

Ilustracija 12 - Sesija 2 (sa desne strane) započinje eksplicitno transakciju sa READ COMMITED-ed izolacionim nivoom. Sesija 2 ne vidi promene sesije 1 (pisci ne blokiraju čitaoc)

Conn

```

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 500
where product_id = 203;

```

Script Output x Query Result x

Task completed in 0.025 seconds

1 row updated.

OT2

```

set transaction isolation level read committed;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 400
where product_id = 107;

```

Script Output x Query Result x

Task completed in 0.02 seconds

Transaction ISOLATION succeeded.

>>Query Run In:Query Result

1 row updated.

Ilustracija 13 - Transakcija 2 uspešno ažurira podatak sa id = 107 jer je transakcija 1 zaključala samo red sa id = 203.

TRANSACTION 1Query Builder

```
select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 500
where product_id = 203;

insert into products
(product_id,product_name,description,
standard_cost, list_price, category_id)
values(289, 'Kingston','description',
200,300,5);
```

Script Output xQuery Result x

Task completed in 0.136 seconds

1 row updated.

Error starting at line : 9 in command -
insert into products
(product_id,product_name,description,
standard_cost, list_price, category_id)
values(289, 'Kingston','description'
200,300,5)
Error at Command Line : 13 Column : 1
Error report -
SQL Error: ORA-00917: missing comma
00917. 00000 - "missing comma"
*Cause:
*Action:

1 row inserted.

TRANSACTION 2Query Builder

```
set transaction isolation level read committed;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 400
where product_id = 107;

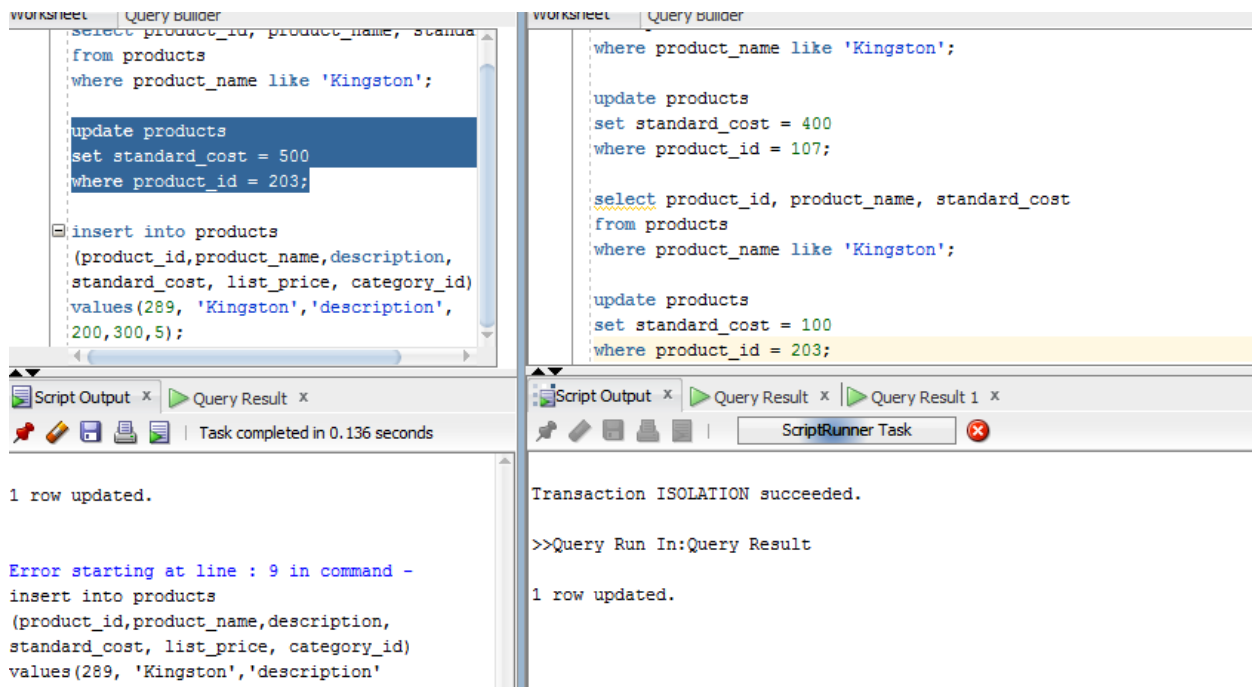
select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```

Script Output xQuery Result xQuery Result 1 x

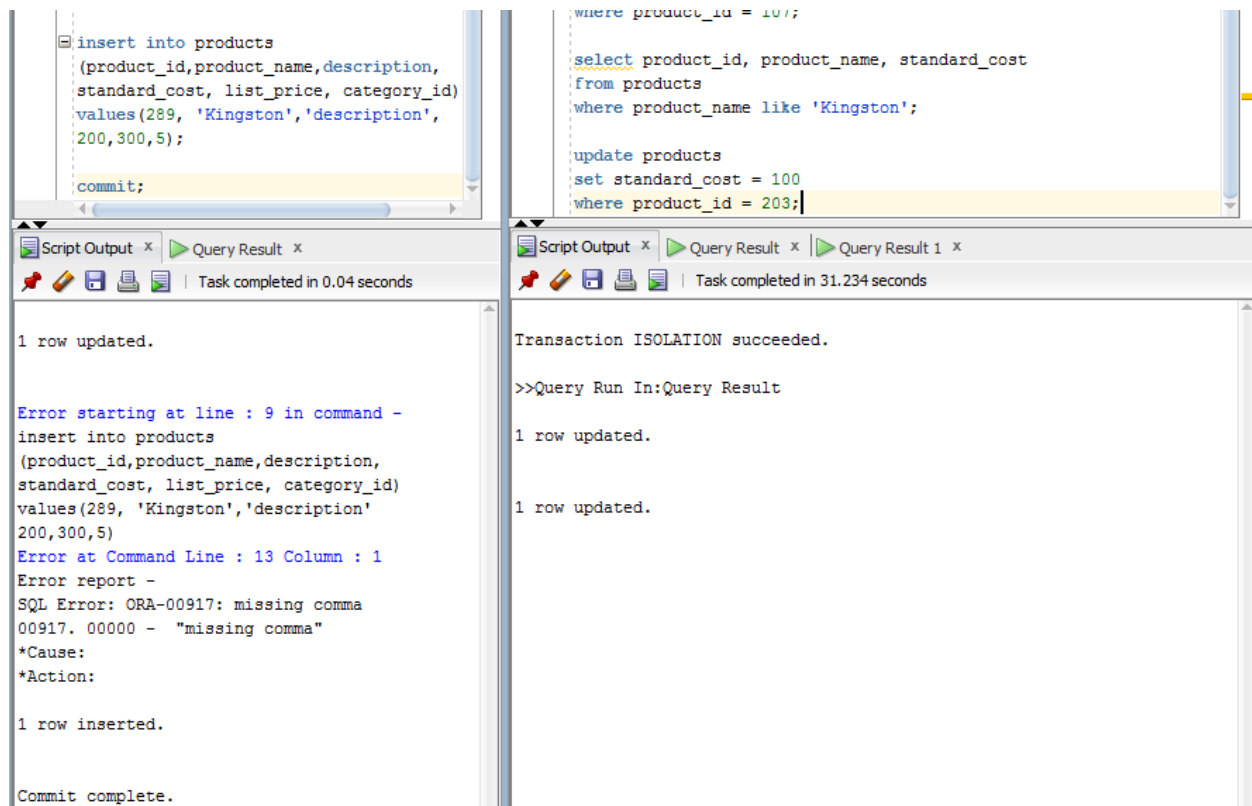
SQL | All Rows Fetched: 4 in 0.04 seconds

PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203 Kingston	556.84
2	201 Kingston	566.98
3	107 Kingston	400
4	128 Kingston	650.48

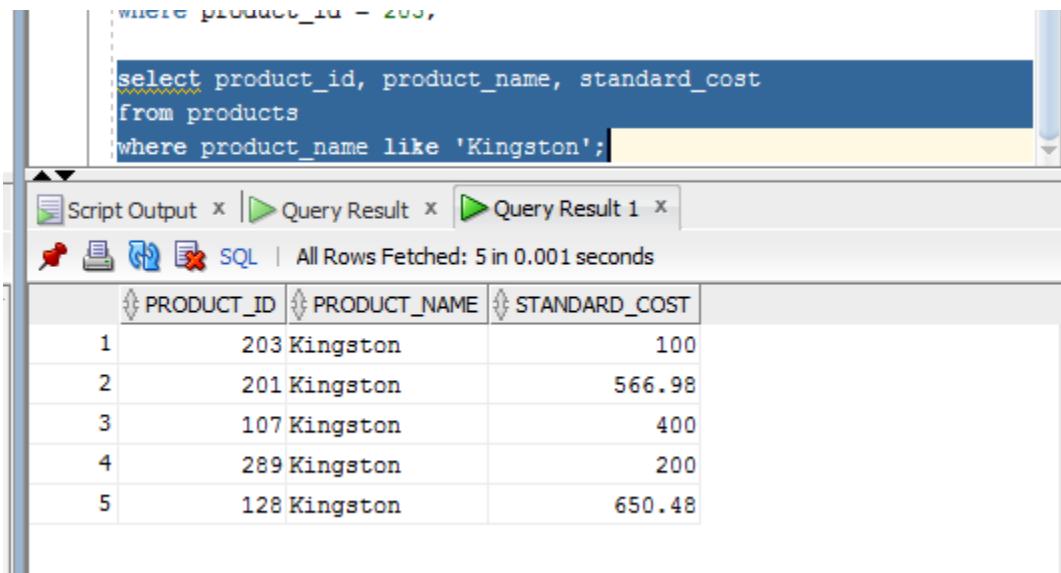
Ilustracija 14 - Transakcija 1 dodaje novi proizvod sa imenom Kingston ali još uvek ne commit-uje. Transakcija 2 šalje upit za listu proizvoda i ne vidi promene ni dodavanje transakcije 1.



Ilustracija 15 - Transakcija 2 pokušava da ažurira proizvod sa id = 203 koji je trenutno zaključak ekskluzivnim lock-om od strane transakcije 1, pa time kreira konflikt. Transakcija 2 čeka na transakciju 1 da commit-uje ili rollback-uje promene kako bi nastavila sa izvršenjem



Ilustracija 16 - Transakcija 1 commit-uje svoje promene. Odmah nakon toga transakcija 2 završava ažuriranje podatka sa id = 203 (primer izgubljenog ažuriranja).



The screenshot shows a SQL query window with the following SQL code:

```

where product_id = 203,

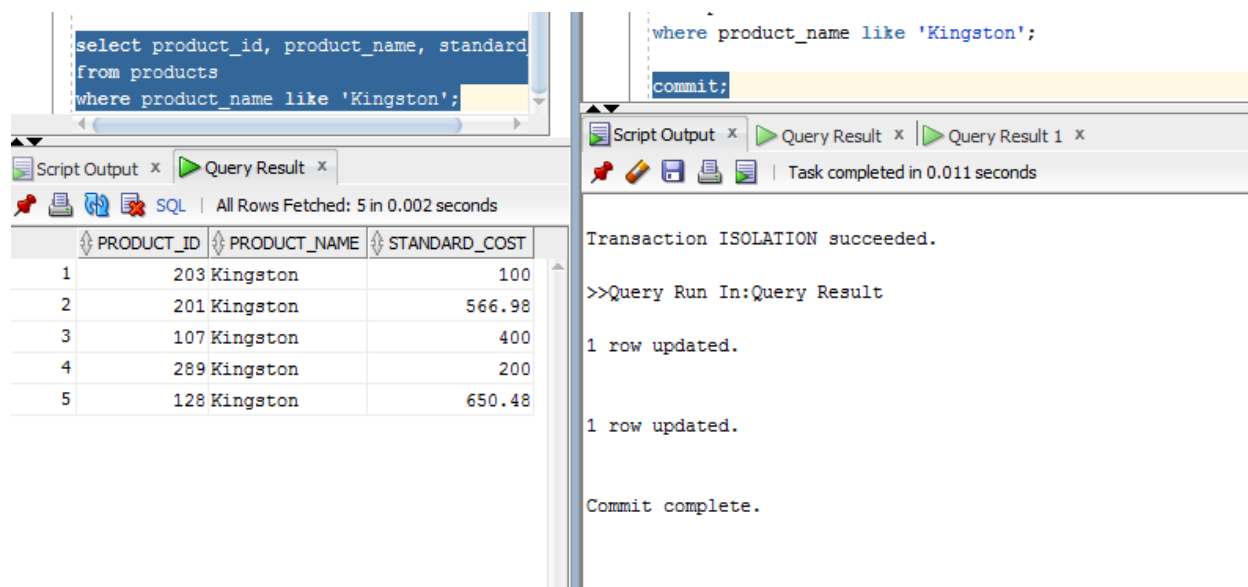
select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

```

The results pane shows the following data:

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	100
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	200
5	128	Kingston	650.48

Ilustracija 17 - Transakcija 2 šalje upit za listu proizvoda i vidi promene transakcije 1 kao i svoje promene.



The screenshot shows a SQL query window with the following SQL code:

```

select product_id, product_name, standard
from products
where product_name like 'Kingston';

commit;

```

The results pane shows the following data:

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	100
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	200
5	128	Kingston	650.48

The script output pane shows the following messages:

```

Transaction ISOLATION succeeded.

>>Query Run In:Query Result

1 row updated.

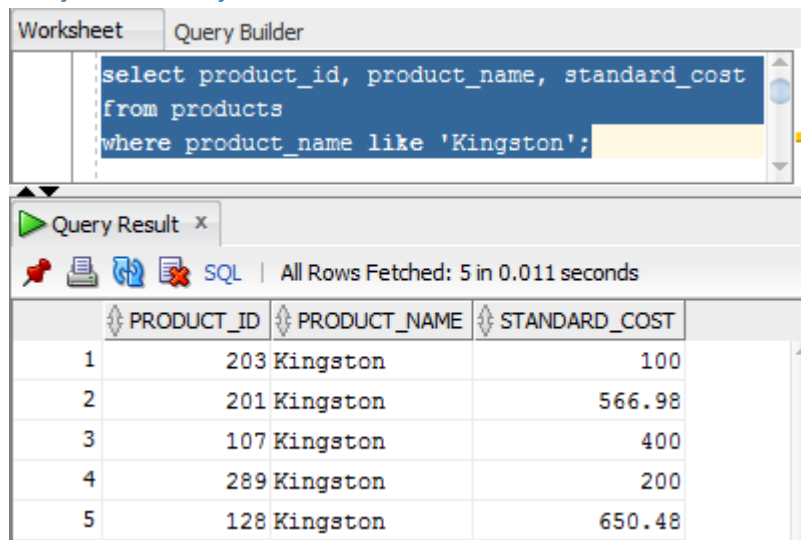
1 row updated.

Commit complete.

```

Ilustracija 18 - Transakcija 2 commit-uje svoje promene. Sesija 1 šalje novi upit za listom proizvoda sa imenom Kingston i vidi promene transakcija 1 i 2.

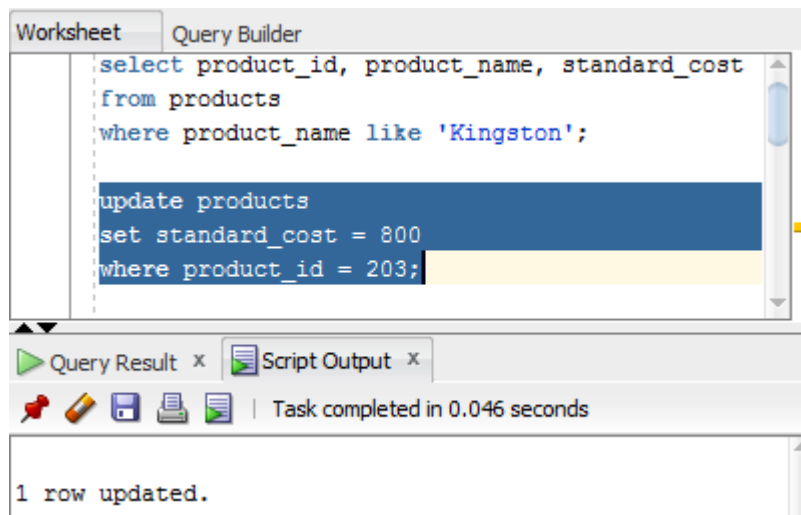
3.12 Primer izvršenja transakcije sa SERIALIZABLE izolacionim nivoom:



The screenshot shows the SQL Developer interface. The 'Query Builder' tab is active, displaying a SQL query: `select product_id, product_name, standard_cost from products where product_name like 'Kingston';`. Below the query, the 'Query Result' tab shows the results of the query. The status bar indicates 'All Rows Fetched: 5 in 0.011 seconds'. The results are displayed in a table with three columns: PRODUCT_ID, PRODUCT_NAME, and STANDARD_COST.

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	100
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	200
5	128	Kingston	650.48

Ilustracija 19 - Sesija 1 šalje upit za prikaz liste proizvoda sa njihovim cenama koji imaju ime Kingston.



The screenshot shows the SQL Developer interface. The 'Query Builder' tab is active, displaying a SQL query: `select product_id, product_name, standard_cost from products where product_name like 'Kingston';`. Below the query, the 'Script Output' tab is active, showing the results of the query. The status bar indicates 'Task completed in 0.046 seconds'. The results are displayed in a table with one row: '1 row updated.'

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	100
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	200
5	128	Kingston	650.48

Ilustracija 20 - Sesija 1 ažurira proizvod sa id = 203 i implicitno započinje transakciju 1.

Worksheet

Query Builder

```

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 800
where product_id = 203;

```

Query Result x

Script Output x

Task completed in 0.046 seconds

1 row updated.

Worksheet

Query Builder

```

set transaction isolation level serializable;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

```

Script Output x

Query Result x

SQL | All Rows Fetched: 5 in 0.002 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	100
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	200
5	128	Kingston	650.48

Ilustracija 21 - Sesija 2 započinje transakciju 2 sa SERIALIZABLE izolacionim nivoom i šalje upit za prikaz liste proizvoda sa imenom Kingston. Transakcija 2 ne vidi promene transakcije 1.

Worksheet

Query Builder

```

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 800
where product_id = 203;

```

Query Result x

Script Output x

Task completed in 0.046 seconds

1 row updated.

Worksheet

Query Builder

```

set transaction isolation level serializable;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 300
where product_id = 289;

```

Script Output x

Query Result x

Task completed in 0.03 seconds

Transaction ISOLATION succeeded.

1 row updated.

Ilustracija 22 -Transakcija 2 ažurira proizvod sa id = 289 bez konflikta zato što je transakcija 1 zaključala samo proizvod sa id = 203.

Worksheet

Query Builder

```
where product_id = 203;

insert into products
(product_id,product_name,description,
standard_cost, list_price, category_id)
values(290, 'Kingston','description',
50,50,5);

commit;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```

Script Output x

Query Result x

SQL | All Rows Fetched: 6 in 0.002 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203 Kingston		800
2	201 Kingston		566.98
3	107 Kingston		400
4	289 Kingston		200
5	290 Kingston		50
6	128 Kingston		650.48

Worksheet

Query Builder

```
set transaction isolation level serializable;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

update products
set standard_cost = 300
where product_id = 289;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```

Script Output x

Query Result x

SQL | All Rows Fetched: 5 in 0.003 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203 Kingston		100
2	201 Kingston		566.98
3	107 Kingston		400
4	289 Kingston		300
5	128 Kingston		650.48

Ilustracija 23 - Transakcija 1 dodaje novi proizvod sa imenom Kingston i commit-uje svoje promene. Sesija 1 i transakcija 2 šalju upit za prikaz liste proizvoda sa imenom Kingston. Sesija 1 vidi commit-ovane promene transakcije 1. Transakcija 2 ne vidi commit-ovane promene transakcije 1.

Worksheet

Query Builder

```
where product_id = 203;





insert into products
(product_id,product_name,description,
standard_cost, list_price, category_id)
values(290, 'Kingston','description',
50,50,5);

commit;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```

Script Output x

Query Result x

    SQL

All Rows Fetched: 6 in 0.002 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	800
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	300
5	290	Kingston	50
6	128	Kingston	650.48

Worksheet

Query Builder

```
update products
set standard_cost = 300
where product_id = 289;





select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

commit;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';
```

Script Output x

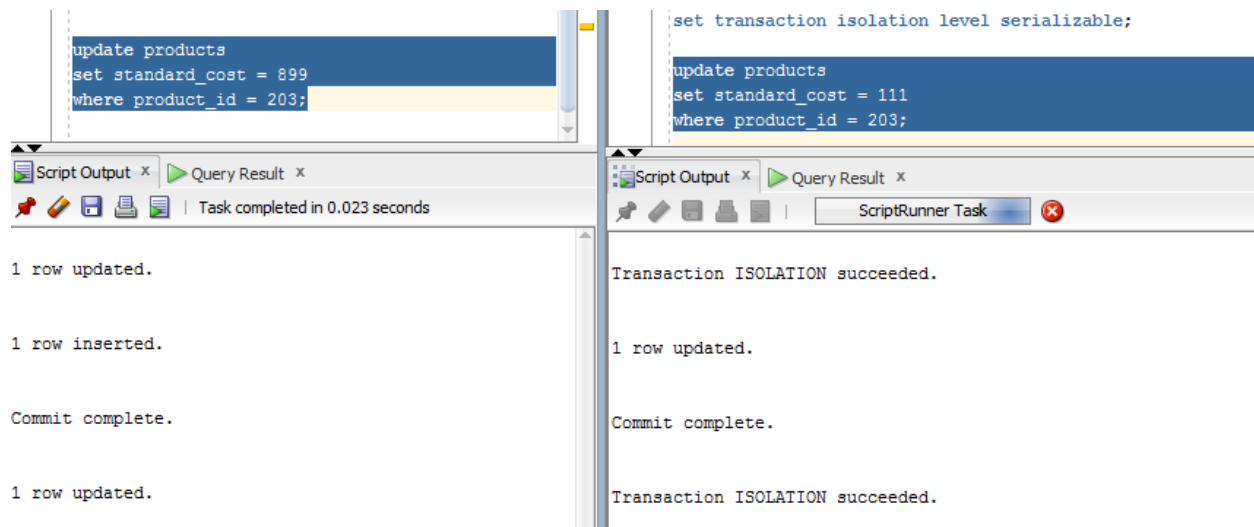
Query Result x

    SQL

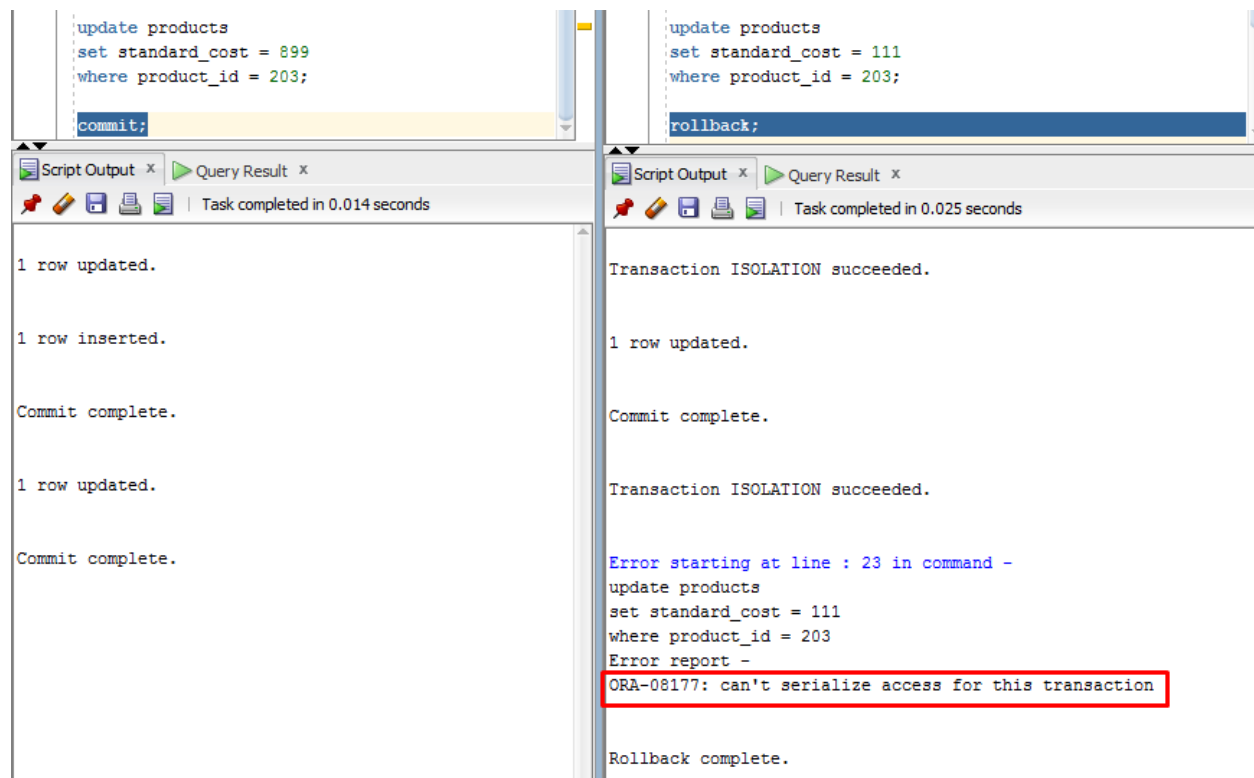
All Rows Fetched: 6 in 0.002 seconds

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	800
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	300
5	290	Kingston	50
6	128	Kingston	650.48

Ilustracija 24 - Transakcija 2 commit-uje promene. Sesije 1 i 2 šalju upit za prikaz proizvoda sa imenom Kingston. Obe sesije vide promene transakcija 1 i 2.



Ilustracija 25 - Sesija 1 ažurira cenu podataka sa id = 203 i implicitno započinje novu transakciju (broj 3). Sesija 2 explicitno započinje novu transakciju (broj 4) sa SERIALIZABLE nivoom izolacije i pokušava da ažurira proizvod sa id = 203. Pošto je transakcija 3 već zaključala taj podatak, transakcija 4 čeka na završetak transakcije 3.



Ilustracija 26 - Transakcija 3 commit-uje svoje promene. Ažuriranje proizvoda sa id = 203 u transakciji 4 nije uspeo zato što je taj podatak ažuriran od strane transakcije 3 u toku izvršenja transakcije 4. Javlja se ORA-08177 greška. Transakcija 4 rollback-uje svoje promene.

```

set transaction isolation level serializable;

select product_id, product_name, standard_cost
from products
where product_name like 'Kingston';

```

	PRODUCT_ID	PRODUCT_NAME	STANDARD_COST
1	203	Kingston	899
2	201	Kingston	566.98
3	107	Kingston	400
4	289	Kingston	300
5	290	Kingston	50
6	128	Kingston	650.48

Ilustracija 27 - Sesija 2 započinje novu transakciju sa SERIALIZABLE izolacionim nivoom. Transakcija 5 šalje upit za prikaz liste proizvoda sa imenom Kingston i vidi promene napravljene od strane transakcije 3.

```

update products
set standard_cost = 111
where product_id = 203;

commit;

```

1 row updated.

Commit complete.

Ilustracija 28 - Transakcija 5 ažurira proizvod sa id = 203 i uspešno commit-uje svoje promene. Da je opet neka transakcija ažurirala proizvod sa id = 203 pre ažuriranja transakcije 5, opet bi se javila ORA-08177 greška.

4 Zaključak

Transakcije igraju važnu ulogu u realizaciji savremenih enterprise sistema kojima pristupa veliki broj korisnika. Bez upotrebe transakcija realizacija ovakvih sistema ne bi bila moguća.

Videli smo osnovne probleme koji se javljaju pri konkurentnom pristupu bazi podataka - problem izgubljenog ažuriranja, problem prljavog čitanja, problem netačnog sumiranja i problem neponovljivog čitanja. Videli smo proces obrade jedne transakcije kao i stanja kroz koja transakcija prolazi prilikom obrade - aktivno, delimično commit-ovano, commit-ovano/otkazano, završeno. Videli smo šta su to planovi izvršenja transakcija i koji su to serijabilni planovi izvršenja. Obradili smo glavnu tehniku za implementaciju konkurentnog pristupa pomoću deljenih i ekskluzivnih lock-ova. Videli smo šta su to deadlock-ovi, kako nastaju i načine za njihovu obradu. Dali smo pregled različitih granularnosti podataka prilikom zaključavanja i objasnili konkurentnost pristupa u indeksima.

Oracle je relaciona baza podataka sa implementacijom konkurentnog pristupa pomoću više verzija podataka (multiversion concurrency control). Ovakvom implementacijom čitaoci i pisci se međusobno ne blokiraju. Oracle implementira 3 nivoa izolacije transakcija - read committed, serializable i read only. Oracle koristi dve vrste lock-ova - deljene i ekskluzivne lock-ove. Ovi lock-ovi se koriste za implementaciju različitih DML, DDL i sistemskih lock-ova.

5 Literatura

- [1] S. B. N. Ramez Elmasri, Fundamentals of Database Systems 6th edition.
- [2] R. a. G. J. Ramakrishnan, Databse Management Systems, 3th Edition.
- [3] Oracle, "Data Concurrency and Consistency," [Online]. Available:
<https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/data-concurrency-and-consistency.html#GUID-E8CBA9C5-58E3-460F-A82A-850E0152E95C>.
- [4] Oracle, "Transactions," [Online]. Available: <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/transactions.html#GUID-B97790CB-DF82-442D-B9D5-50CCE6BF9FBD>.