Univerzitet u Nišu, Elektronski fakultet
Katedra za računarstvo

Obrada transakcija, planovi izvršenja transakcija, izolacija i zaključavanje kod Oracle baze podataka

Sistemi za upravljanje bazama podataka

Seminarski rad

Mentor:

Doc. dr Aleksandar Stanimirović

Student:

Vladana Stojiljković 1135

Niš, 2021.

Sadržaj

1. Uvod

Transakcije su jedna od glavnih karakteristika koja razlikuje baze podataka od sistema datoteka (eng. *file system*). Ako u datotečkom sistemu dođe do pada operativnog sistema usred upisa u neku datoteku, ta datoteka će verovatno biti oštećena, ali postoje načini za vraćanje datoteke u neko prethodno stanje. Međutim, ukoliko je potrebno sinhronizovati upis u dve datoteke i dođe do zakazivanja sistema pre ažuriranja druge datoteke, datoteke se neće sinhronizovati, tj. sistem neće biti konzistentan. To je glavna svrha transakcija - one prevode bazu podataka iz jednog konzistentnog stanja u drugo. U situacijama kada jedan ili više korisnika pokušava da pristupi i promeni podatke istovremeno u istoj tabeli, dolazi do preplitanja akcija i mogućnosti nekonzistentne promene istih podataka ili netačnosti podataka. U cilju rešavanja ovog problema koriste se transakcije da bi se obezbedilo konkurentno izvođenje akcija, ali i u cilju oporavka baze podataka. Transakcija je skup SQL naredbi takav da se ili izvršavaju sve naredbe iz skupa ili nijedna. Glavna svrha transakcija je da omoguće da baza podataka ostane konzistentna čak i u slučaju sistemskih grešaka, kada se izvršavanje prekine (u potpunosti ili delimično) i mnoge operacije nad bazom ostanu nedovršene ili sa nejasnim statusom, kao i da omoguće izolaciju među programima koji istovremeno pristupaju bazi podataka [1]. Bez izolacije, rezultati programa mogu da budu pogrešni. Postoji više nivoa izolacije transakcija i oni određuju ishod transakcije. U cilju održanja konzistentnosti koristi se i mehanizam zaključavanja koji služi da spreči destruktivne interakcije. Interakcije su destruktivne kada netačno ažuriraju podatke ili pogrešno promene osnovne strukture podataka, između transakcija koje pristupaju deljenim podacima [2]. Brave (eng. *locks*) igraju presudnu ulogu u održavanju istovremenosti i konzistentnosti baze podataka.

Oracle baza podataka je relacioni sistem za upravljanje bazama podataka (eng. *Database Managemnt System, DBMS*) koji proizvodi i prodaje kompanija Oracle korporacija. Za demonstraciju primera korišćena je verzija 19c. Transkacije u Oracle-u imaju ACID svojstva i njihove osobine, način u poglavlju 2, a izolacija u poglavlju 3. Mehanizam zaključavanja i tipovi brava opisani su poglavlju 4, zaključak je dat u poglavlju 5, a spisak korišćene literature u poglavlju 6.

2. Transakcije u Oracle bazi podataka

Transakcija je logična i atomična jedinica izvršenja koja sadrži jednu ili više SQL naredbi. Transakcija grupiše SQL naredbe tako da su ili sve izvršene i potvrđene (eng. *committed*), što znači da su primenjene na bazu podataka ili poništene (eng. *rollback*), odnosno opozvane iz baze podataka. Svaka transakcija u Oracle bazi podataka označena je jedinstvenim identifikatorom (*transaction ID*) [2]. Kao što je već rečeno, Oracle transakcije imaju ACID svojstva:

- atomičnost (eng. *Atomicity*) – Sve naredbe u transakciji se izvršavaju ili se ne izvršava nijedna. Nema delimičnih transakcija. Na primer, ako transakcija treba da ažurira 200 vrsti, ali sistem otkaže nakon 150 ažuriranja, tada baza podataka poništava promene u ovih 150 vrsti (vraća prethodno stanje).
- konzistentnost (eng. *Consistency*) - Transakcija prevodi bazu podataka iz jednog konzistentnog stanja u drugo. Na primer, u bankarskoj transakciji kojom se tereti štedni račun i kreditira tekući račun, neuspeh ne sme prouzrokovati da baza podataka kreditira samo jedan račun, jer bi to dovelo do nekonzistentnih podataka.
- izolacija (eng. *Isolation*) - Učinak transakcije nije vidljiv drugim transakcijama dok se transakcija ne izvrši. Na primer, jedan korisnik koji ažurira tabelu *zaposleni* ne vidi nepotvrđene (eng. *uncommitted*) promene zaposlenih koje je istovremeno napravio drugi korisnik. Stoga se korisnicima čini kao da se transakcije izvršavaju serijski.
- trajnost (eng. *Durability*) - Promene izvršene transakcijom su trajne. Po završetku transakcije, baza podataka kroz svoje mehanizme oporavka osigurava da se promene iz transakcije neće izgubiti.

Transakcija može da bude skup jedne ili više naredbi za manipulaciju podacima (eng. *Data Manipulation Language*, *DML*) koje zajedno predstavljaju atomičnu promenu nad bazom podataka, ili jedna naredba za definisanje struktura podataka (eng. *Data Definition Language*, *DDL*). Sve transakcije imaju svoj početak i kraj. U Oracle-u nema potrebe za eksplicitnim definisanjem početka transakcije korišćenjem naredbe *SET TRANSACTION*, već transakcija implicitno započinje kad se naiđe na prvu izvršnu naredbu (eng. *executable*). Izvršna naredba je SQL naredba koja poziva bazu podataka, uključujući DML, DDL i *SET TRANSACTION* naredbe [2]. Informacije o transakcijama vidljive su iz pogleda *V$TRANSACTION* koja sadrži listu svih aktivnih transakcija i njihovih atributa [3]. Pogodno je dati ime transakciji kako bi ona bila uočljivija prilikom izlistavanja sadržaja *V$TRANSACTION*. Zadavanjem imena transakciji olakšava se praćenje njenog izvršenja, što je značajno kod dugih transakcija, a moguće je pretraživanje logova kako bi se našla određena transakcija. Na slici 1 dat je primer na kom se vidi da se nakon izvršenja *UPDATE* naredbe u pogledu *V$TRANSACTION* nalazi jedna aktivna transakcija. Aktivna transakcija je transakcija koja je započela sa izvršenjem, ali još nije potvrđena ili poništena [2].

```
SQL> update test set id=7 where id=3;

1 row updated.

SQL> select xid, xidusn undo_seg, xidslot slot, xidsqn seq, status st
  2  from v$transaction;

XID              UNDO_SEG       SLOT        SEQ ST
---------------- ---------- ---------- ---------- ----------------
0A001B0090070000         10         27       1936 ACTIVE
```

Slika 1: V$TRANSACTION

Transkacija može da se završi pod različitim okolnostima:

- korisnik pozove *COMMIT* ili *ROLLBACK* naredbu bez *SAVEPOINT* naredbe;
- korisnik pozove neku DDL naredbu (*CREATE*, *RENAME*, *DROP*, *ALTER*);
- korisnik normalno izađe iz Oracle uslužnih programa i alata, što dovodi do toga da je trenutna transakcija implicitno potvrđema (*committ*-ovana). Ponašanje izvršenja *commit* naredbe kada korisnik prekine vezu zavisi od aplikacije i može da se konfiguriše;
- klijentski proces se abnormalno završava, što dovodi do toga da se transakcija implicitno poništava koristeći metapodatke iz tabele transakcija i segmenata poništavanja (eng. *undo segment*).

2.1. Kontrola transakcija

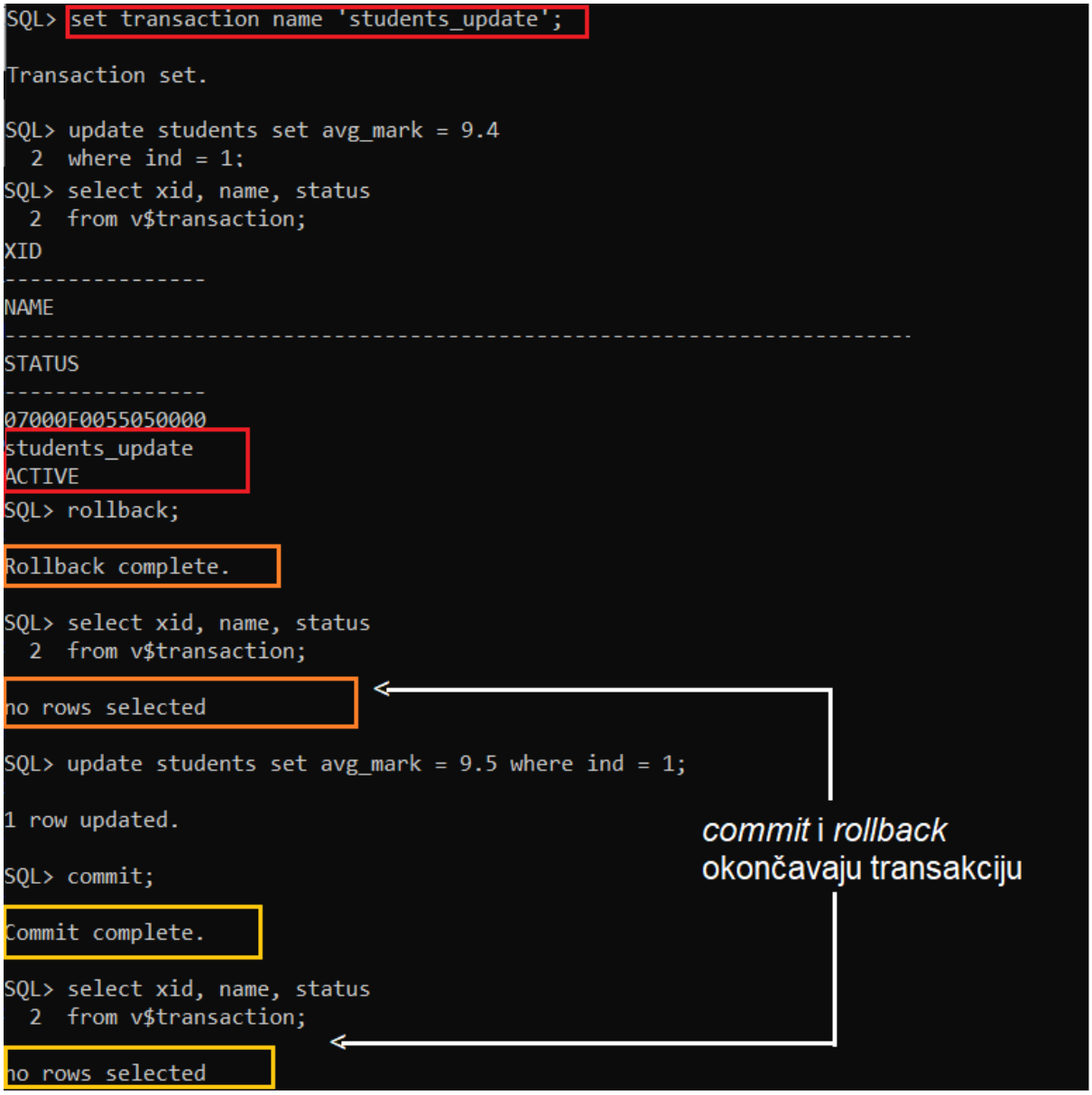
Transakcije su u Oracle-u atomične, što znači da je svaka naredba koja čini transakciju potvrđena (učinjena trajnom) ili su sve naredbe vraćene unazad. To se odnosi i na pojedinačne naredbe [4]. Ili je naredba u potpunosti izvršena ili u potpunosti vraćena unazad. U naredbe za upravljanje transakcijama (eng. *Transaction Control Language*, *TCL*) spadaju:

- *COMMIT* naredba – poziv ove naredbe ekvivalentan je pozivu *COMMIT WORK*. Njome se završava transakcija i sve promene u bazi podataka postaju trajne. Po izvršenju *COMMIT*-a promene koje je izvršio jedan korisnik vidljive su i svim ostalim korisnicima. Oracle implicitno poziva ovu naredbu pre i posle svake DDL naredbe. Ako trenutna transakcija sadrži DML naredbu, Oracle najpre izvrši *COMMIT*, a onda svaku DDL naredbu izvršava kao novu, zasebnu transakciju [2].
- *ROLLBACK* naredba – poziv ove naredbe ekvivalentan je pozivu *ROLLBACK WORK*. Efekat ove naredbe je završetak transakcije i poništavanje svi nekomitovanih promena. To se vrši čitanjem podataka iz segmenata poništavanja koji čuvaju blokove baze podataka u stanju u kom su bili pre nego što je započela transkacija.
- *SAVEPOINT* naredba – ovom naredbom se označava tačka čuvanja u transakciji na koju je moguće vratiti se kasnije *ROLLBACK* naredbom [2].
- *ROLLBACK TO <SAVEPOINT>* - naredba ima isti efekat kao *ROLLBACK* s tim što se ne vrši vraćanje na stanje pre početka transakcije, već na tačku u transakciji koja je

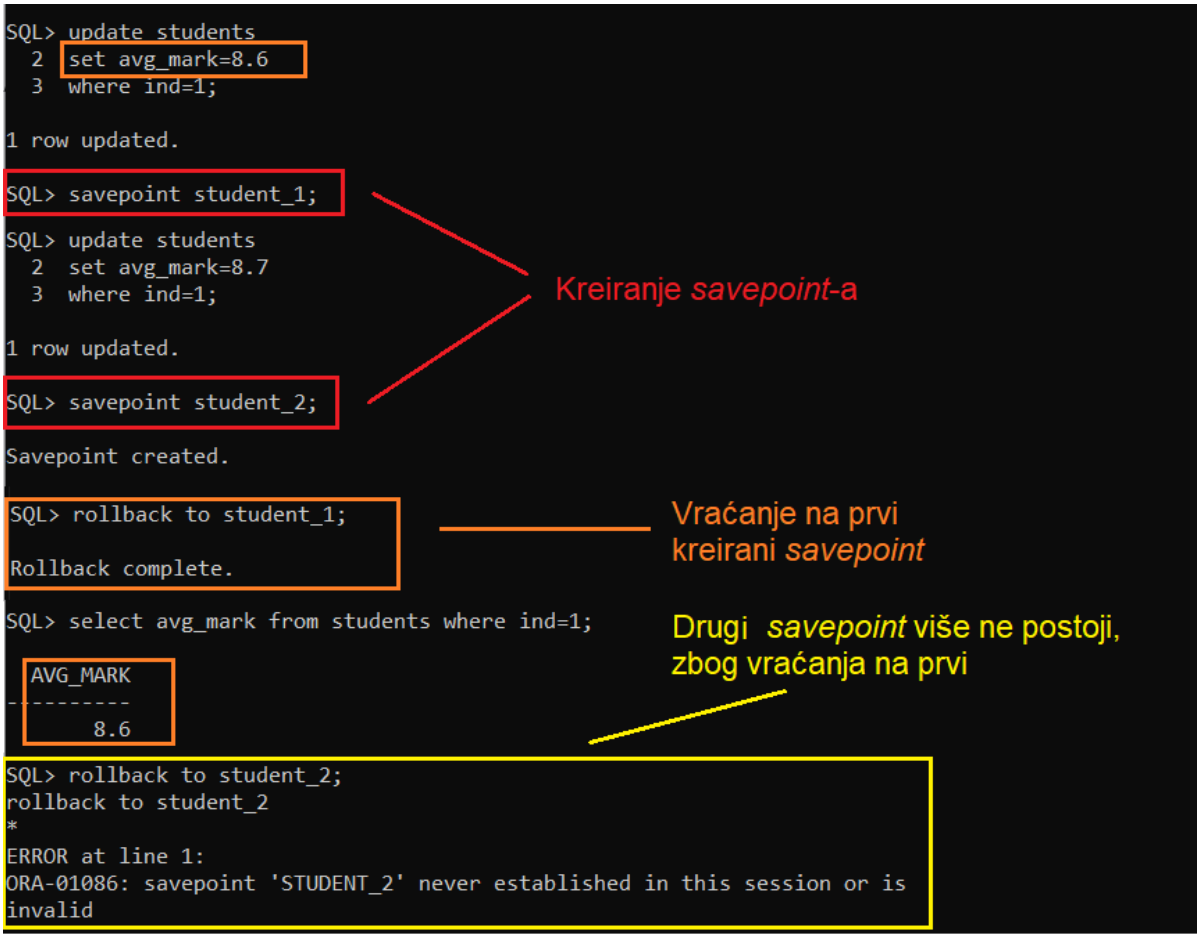
prethodno kreirana *SAVEPOINT* naredbom. Poništavaju se sve akcije nakon označene tačke, čuva se označena tačka i odbacuju sve kreirane nakon nje, oslobađaju se sve tabele i brave vrsta nabavljene nakon označene tačke, a zadržavaju brave nad podacima nabavljene pre te tačke u transakciji. Transakcija ostaje aktivna i može da nastavi sa izvršenjem.

- *SET TRANSACTION* – ovom naredbom se omogućava postavljanje različitih atributa transakcija, kao što su ime transakcije, nivo izolacije transakcije i da li je samo za čitanje ili za čitanje i pisanje.

Na slici 2 prikazani su primeri poziva naredbi *SET TRANSACTION*, *COMMIT* i *ROLLBACK*, a na slici 3 primeri kreiranja tačaka čuvanja i vraćanja na njih.



Slika 2: TCL naredbe



Slika 3: TCL naredbe

2.1.1. Poništavanje transakcija i UNDO segmenti

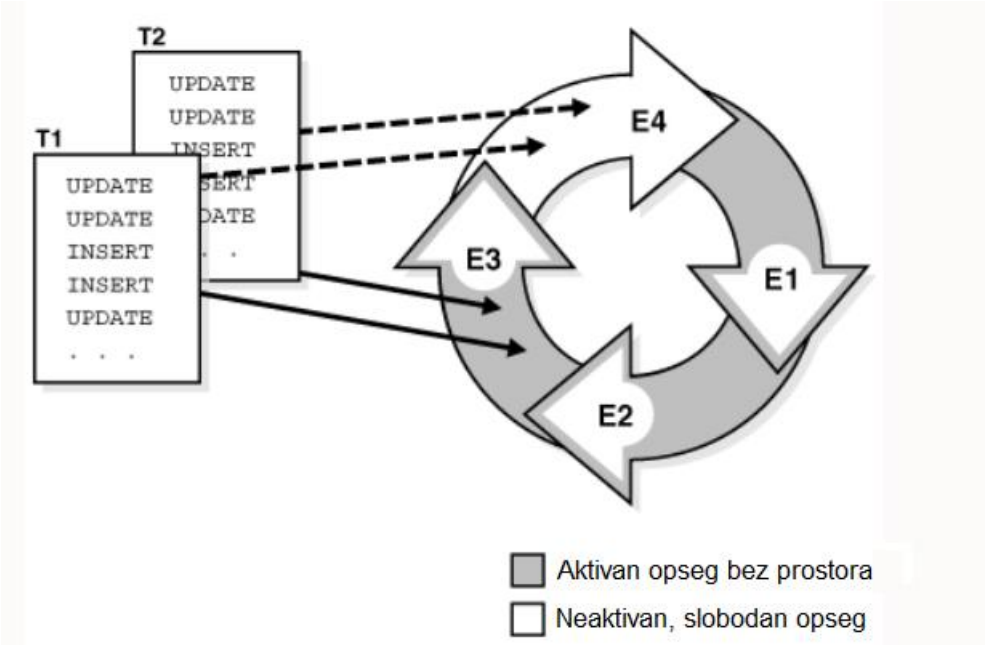
Kao što je pomenuto, poništavanje nepotvrđene (nekomitovane) transakcije opoziva sve promene podataka koje su izvršile SQL naredbe unutar transakcije. Nakon poništavanja transakcije, efekti posla obavljenog u transakciji više ne postoje. Vraćajući celu transakciju unazad, bez referenciranja bilo kojih tačaka čuvanja (*savepointa*-a), Oracle vrši sledeće:

- Poništava sve promene koje su izvršili svi SQL izrazi u transakciji upotrebom odgovarajućih segmenata poništavanja (*undo* segemenata). Tabela transakcija za svaku aktivnu transakciju ima slog koji sadrži pokazivač na sve podatke o poništavanju (u obrnutom redosledu) za transakciju. Baza podataka čita podatke iz segmenta poništavanja, poništava operaciju, a zatim označava slog poništavanja kao primenjen.
- Oslobađa sve brave podataka koje poseduje transakcija.
- Briše sve tačke čuvanja kreirane u transakciji.
- Završava transakciju čime se baza vraća u stanje nakon poslednje izvršene *COMMIT* naredbe.

Trajanje poništavanja transakcije zavisi od količine izmenjenih podataka.

Ključna struktura podataka za poništavanje transkacija jesu segmenti poništavanja. Oracle čuva zapise o akcijama transakcija koji se zovu podaci poništavanja (eng. *undo data*) [5]. Podaci poništavanja se koriste kod poništavanja transakcija, oporavka izvršenih transakcija i za omogućavanje konizistencije prilikom čitanja [5]. Ovi podaci čuvaju se u bazi u vidu blokova koji se ažuriraju kao i blokovi podataka, pri čemu promene kod ovih blokova generišu *redo* zapise (eng. *redo data*). Podaci poništavanja trajnih objekata čuvaju se u tabelarnom prostoru za poništavanje (eng. *undo tablespace*). Oracle ima potpuno automatizovan mehanizam, poznat kao režim automatskog poništavanja, za upravljanje segmentima poništavanja i tabelarnog prostora. Podaci poništavanja dele se u dva toka: jedan za privremene i jedan za permanentne objekte [5]. Baza podataka upravlja privremenim i trajnim poništavanjem nezavisno.

Kada transakcija započne, baza podataka vezuje transakciju za segment poništavanja, a samim tim i za tabelu transakcija, u trenutnom prostoru tabela poništavanja. U retkim slučajevima, ako instanca baze podataka nema naznačeni prostor za poništavanje tabela, tada se transakcija vezuje za sistemski segment poništavanja. Više aktivnih transakcija može istovremeno biti vezano za isti segment za poništavanja i samim tim i upisuju u njega, ili za različite segmente. Konceptualno, opsezi (eng. *extent*) u segmentu za poništavanje čine prsten. Transakcije ciklično upisuju u jedan opseg, a zatim u sledeći u prstenu. Na slici 4 dat je primer dve transakcije koje su započele upis u jednom opsegu, a nastavile u drugom u okviru istog segmenta.



Slika 4: Opsezi u undo segmentu [5]

Kada se izda *ROLLBACK* naredba, iščitavaju se podaci poništavanja, odbacuju se promene nastale nekomitovanom transakcijom i baza se vraća u prethodno konzistentno stanje.

2.1.2. *Commit* transakcije

Potvrđivanje ili *commit*-ovanje završava trenutnu transakciju i čini trajnim sve promene izvršene u transakciji. Prilikom izdavanja *COMMIT* naredbe, baza generiše sistemski broj promene (eng. *System Change Number, SCN*) za *COMMIT* [2]. U internoj tabeli transakcija se beleži jedinstveni SCN transakcije, kao i to da je transakcija izvršena.

Sistemski broj promene je logičan, interni vremenski žig koji koristi Oracle[2]. SCN-ovi uređuju događaje koji se javljaju u bazi podataka, što je neophodno za zadovoljenje ACID svojstava transakcije. SCN-ovi se javljaju u monotono rastućem nizu. Oracle baza može da koristi SCN poput sata, jer posmatrani SCN ukazuje na logičku tačku u vremenu, a ponovljena posmatranja vraćaju jednake ili veće vrednosti. Ako jedan događaj ima manji SCN od drugog događaja, to znači da se on desio ranije u bazi podataka. Nekoliko događaja može deliti isti SCN, što znači da su se dogodili istovremeno. Svaka transakcija ima SCN. Na primer, ako transakcija ažurira vrstu, tada baza podataka beleži SCN na kojem je došlo do ovog ažuriranja. Ostale modifikacije u ovoj transakciji imaju isti SCN. Po završetku transakcije, baza podataka beleži SCN za *COMMIT*. Kada transakcija modifikuje podatke, baza zapisuje novi SCN u segment poništavanja koji je dodeljen transakciji.

Nakon dodeljivanja SCN-a *COMMIT*-u, proces za upis logova (eng. *log writer process*, *LGWR*) upisuje zaostale *redo* zapise iz bafera u log, uz odgovarajući SCN transakcije. Ovaj atomičan događaj predstavlja potvrđivanje transakcije [2]. Potom se oslabađaju brave nad vrstama i tabelama, brišu se tačke čuvanja i vrši se čišćenje (eng. *commit cleanout*). Ako su modifikovani blokovi koji sadrže podatke iz transakcije i dalje u globalnom sistemskom prostoru (eng. *System Global Area, SGA*) i ako ih nijedna druga sesija ne modifikuje, tada se uklanjaju transakcione informacije o bravama iz blokova (ITL zapisi[1]). U idealnom slučaju, *COMMIT* čisti blokove tako da naredni *SELECT* ne mora da izvrši ovaj zadatak. Ako za određenu vrstu ne postoji unos ITL, ona nije zaključana. Ako ITL unos postoji za određenu vrstu, onda postoji mogućnost da je ona zaključana, pa sesija mora da proveri zaglavlje segmenta poništavanja da bi utvrdila da li je transakcija izvršena. Ako jeste, tada sesija čisti blok, ali ako je *COMMIT* prethodno očistio ITL, onda su provera i čišćenje suvšni.

Na samom kraju, transakcija se označava kao izvršena. Nakon potvrđivanja transakcije, svi korisnici mogu da vide promene. Ova operacija je obično brza, bez obzira na veličinu transakcije i količinu podataka koju ona modifikuje.

[1] ITL – *Interested Transacion List* je lista svih vrsti unutar bloka koje je zaključala transakcija.

2.2. Autonomne transakcije

Autonomna transakcija je nezavisna transakcija koja se može pozvati iz druge, glavne transakcije [2]. Moguće je suspendovati pozivajuću transakciju, izvoditi SQL naredbe i potvrđivati ih ili poništavati u autonomnoj transakciji, a zatim nastaviti sa glavnom transakcijom. Autonomne transakcije su korisne za radnje koje treba izvršiti nezavisno, bez obzira na to da li će glavna transakcija biti potvrđena ili poništena. Za autonomne transakcije važi sledeće:

1. autonomna transakcija ne vidi nepotvrđene promene koje je izvršila glavna transakcija i ne deli brave ili resurse sa glavnom transakcijom,
2. promene koje izvrši autonomna transakcija vidljive su čim se nad njom izvrši *COMMIT*, tj. ne čeka se potvrđivanje glavne transakcije,
3. iz jedne autonomne transakcije moguće je pozvati drugu.

U PL/SQL-u, autonomna transakcija se izvršava u autonomnom opsegu, što je rutina označena pragmom *AUTONOMOUS_TRANSACTION*. U ovom kontekstu, rutine uključuju anonimne PL/SQL blokove najvišeg nivoa, samostalne pakete ili ugnježdene potprogame, metode apstraktnih tipova podataka i okidače. Pragma je direktiva koja nalaže kompajleru da izvrši opciju kompajliranja, a pragma *AUTONOMOUS_TRANSACTION* nalaže bazi podataka da se deo koda izvrši autonomno, tj. nezavisno od glavne transakcije [2]. Kad autonomna transakcija krene sa izvršenjem, glavna se suspenduje sve dok se autonomna ne izvrši. Pošto autonomna transakcija ne deli resurse sa glavnom, ukoliko pokuša da pristupi resursima koje drži glavna transakcija, može doći do uzajamnog blokiranja (eng. *deadlock*).

Sledeći primeri ilustruju rad autonomnih transakcija. Kreirane su dve procedure koje vrše jednostavno dodavanje vrste u tabelu, s tim što je jedna procedura obeležena kao autonomna. U okviru anonimnog PL/SQL bloka prvo se poziva *INSERT* naredba, onda obična (neautonomna) procedura i na kraju *ROLLBACK*. Po izvršenju ovog bloka, u tabeli su prisutne obe dodate vrste. To je posledica toga što *COMMIT* iz neautonomne procedure potvrđuje promene koje je izvršila glavna transakcija, pa ne postoje nesačuvane promene koji bi mogle da se poništе *ROLLBACK*-om (slika 5).

U drugom slučaju (slika 6), u tabeli je prisutna samo jedna vrsta, tj. podaci dodati autonomnom transakcijom. Pošto se ona izvršava potpuno nezavisno od glavne transakcije, *COMMIT* unutar nje ne potvrđuje promene izvršene od strane glavne transakcije, tako da su one poništene *ROLLBACK* naredbom iz bloka.

```
SQL> create or replace procedure non_auto_insert
  2  as
  3  begin
  4  insert into moja (id) values (5);
  5  commit;
  6  end;
  7  /

Procedure created.
SQL> begin
  2  insert into moja (id) values (10);
  3  non_auto_insert;
  4  rollback;
  5  end;
  6  /

PL/SQL procedure successfully completed.

SQL> select * from moja;

        ID
----------
        10
         5
```

Slika 5: Neautonomna procedura

```
SQL> create or replace procedure auto_insert
  2  as
  3  pragma autonomous_transaction;
  4  begin
  5  insert into moja (id) values (3);
  6  commit;
  7  end;
  8  /

Procedure created.

SQL> delete from moja;

2 rows deleted.

SQL> commit;

Commit complete.

SQL> begin
  2  insert into moja (id) values (13);
  3  auto_insert;
  4  rollback;
  5  end;
  6  /

PL/SQL procedure successfully completed.

SQL> select * from moja;

        ID
----------
         3
```

Slika 6: Autonomna transakcija

2.3. Distribuirane transakcije

Distribuirana transakcija je transakcija koja uključuje skup naredbi koje ažuriraju podatke na dva ili više različitih čvorova distribuirane baze podataka, koristeći objekat šeme koji se naziva veza baze podataka (eng. *database link*) [2]. Distribuirana baza podataka je skup baza podataka u distribuiranom sistemu koji sa strane aplikacije izgleda kao jedan izvor podataka. Link baze podataka opisuje kako se jedna instanca baze podataka povezuje na drugu instancu. Za razliku od transakcije u lokalnoj bazi podataka, distribuirana transakcija menja podatke u više baza podataka. Shodno tome, distribuirana obrada transakcija je složenija jer baza mora da obavi potvrđivanje ili poništavanje promena u transakciji atomično. Cela transakcija mora da se izvrši ili poništi i podaci moraju da ostanu konzistentni i u slučaju pada mreže ili samog sistema.

Distribuirane transakcije imaju dvofazno potvrđivanje (eng. *two-phase commit, 2PC*) koje obezbeđuje da se promene ili zapamte ili poništе na svim čvorovima u distribuiranoj bazi. U dvofaznom *commit*-u, jedna baza podataka koordinira distribuiranu transakciju. Inicirajući čvor naziva se globalni koordinator [2]. Koordinator pita ostale baze podataka da li su spremne da izvrše *COMMIT*. Ako bar jedna od baza nije, tada se cela transakcija poništava. Dvofazni mehanizam potvrđivanja je transparentan za korisnike koji izdaju distribuirane transakcije. U stvari, korisnici ne moraju da budu svesni da se transakcija distribuira. Naredba *COMMIT* koja označava kraj transakcije automatski pokreće opisani dvofazni mehanizam. Pisanje distribuiranih transakcija ne razlikuje se od pisanja lokalnih:

update local_table set x = 5;

update remote_table@another_database set y = 10; commit;

Ili će promene biti sačuvane i u lokalnoj bazi i u *another_database*, ili neće ni u jednoj.

Ukoliko dođe do prekida na mreži ili u sistemu tokom 2PC-a, takva transakcija naziva se sumnjiva (eng. *in-doubt transaction*) [2]. Na primer, dve baze podataka obaveštavaju glavnog koordinatora da su bile spremne za *commit*, ali instanca koordinacione baze podataka otkaže odmah nakon prijema poruka. Pozadinski proces oporavka (RECO) automatski rešava ishod nesumnjivo distribuiranih transakcija. Nakon što se kvar popravi i komunikacija ponovo uspostavi, proces RECO svake lokalne Oracle baze automatski potvrđuje ili poništava sve sumnjive distribuirane transakcije konzistentno na svim uključenim čvorovima. U slučaju dugotrajnog kvara, Oracle omogućava svakom lokalnom administratoru da ručno potvrdi ili poništi bilo koju distribuiranu transakciju koja je sumnjiva zbog kvara. Ova opcija omogućava lokalnom administratoru baze podataka da oslobodi zaključane resurse koji se zadržavaju na neodređeno vreme zbog dugotrajnog kvara.

3. Izolacija kod Oracle-a

Oracle podržava tri nivoa izolacije transakcije, sa različitim mogućim ishodima za isti scenario transakcije. Odnosno, isto delo izvedeno na isti način sa istim ulaznim podacima može dati različite rezultate, u zavisnosti od nivoa izolacije. Ovi nivoi izolacije definisani su na osnovu tri „pojave“ koje su ili dozvoljene ili nisu na datom nivou izolacije:

1. prljava čitanja (eng. *dirty reads*) – jedna tranksacija čita podatke koje je upisala druga, neizvršena transakcija,

2. neponavljajuća čitanja (eng. *nonrepeatable, fuzzy reads*) - transakcija ponovo čita podatke koje je prethodno pročitala i utvrđuje da je druga izvršena transakcija izmenila ili izbrisala podatke,

3. fantomska čitanja (eng. *phantom reads*) - transakcija ponovo pokreće upit koji vraća vrste koje zadovoljavaju uslov pretraživanja i utvrđuje da je druga izvršena transakcija ubacila dodatne vrste koji zadovoljavaju uslov.

Zavisno od toga koje od ovih čitanja je moguće, Oracle definiše sledeće nivoa izolacije:

- *READ COMMITTED*,
- *READ ONLY*,
- *SERIALIZABLE*.

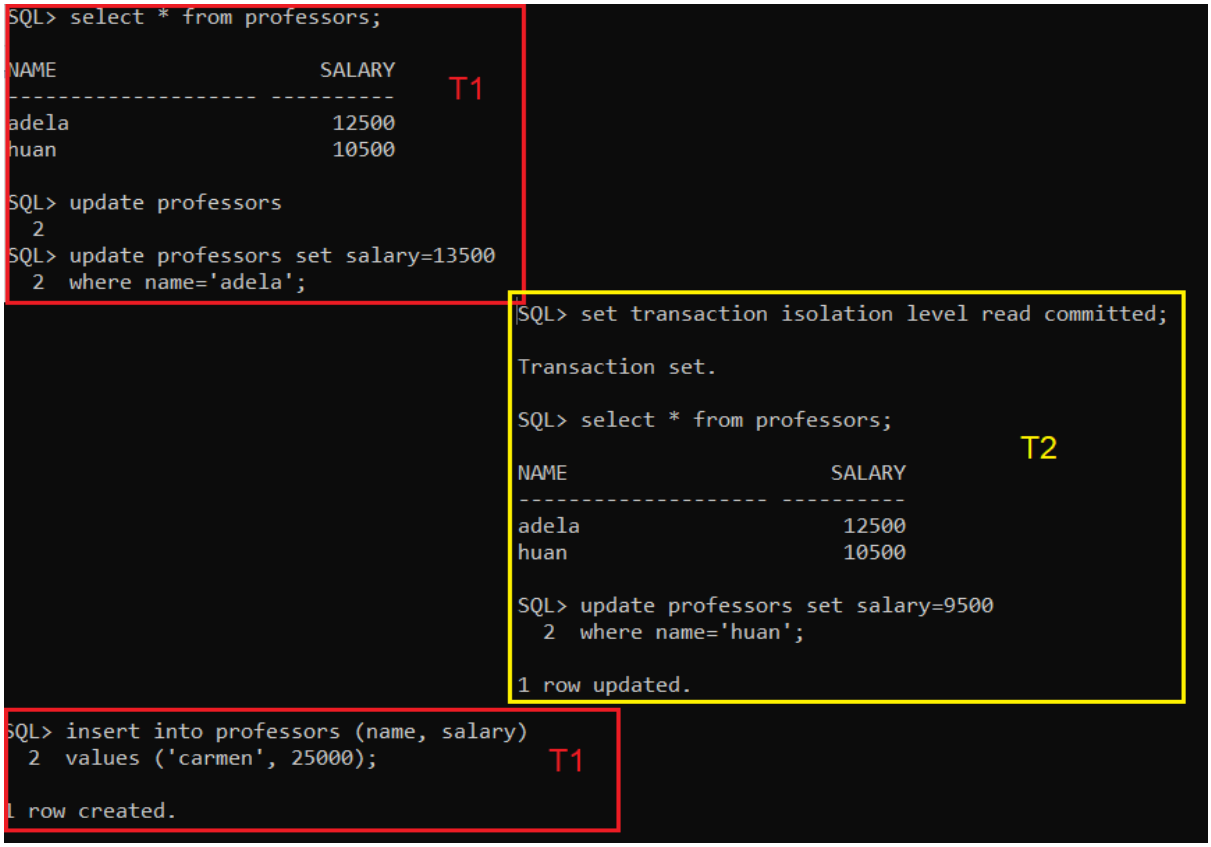
3.1. READ COMMITTED nivo izolacije

Kod *READ COMMITTED* nivoa izolacije, svaki upit koji se izvrši u transakciji vidi samo promene izvršene pre početka upita, ne cele transakcije [6]. Ovaj nivo izolacije je podrazumevani i pogodan je za okruženja baza podataka u kojima je mala verovatnoća da će doći do konflikta između transakcija. Upit u pročitanoj transakciji izbegava čitanje podataka koji se *commit*-uju dok se upit izvršava. Na primer, ako je upit koji čita tabelu sa milion vrsti stigao do pola i ako druga transakcija izvrši ažuriranje u 950000. vrsti, upit neće videti ovu promenu kada dođe do te vrste. Međutim, s obzirom da baza podataka ne sprečava druge transakcije da modifikuju podatke koji se čitaju upitom, druge transakcije mogu da promene podatke između izvršavanja upita. Dakle, transakcija koja izvrši isti upit dva puta može imati nejasna ili fantomska čitanja. Baza podataka pruža konzistentan skup rezultata za svaki upit, garantujući doslednost podataka, bez ikakvih intervencija korisnika. Implicitni upit, poput upita koji se podrazumeva klauzulom *WHERE* u *UPDATE* izrazu, garantuje konzistentan skup rezultata. Međutim, izraz u implicitnom upitu ne vidi promene koje je načinila sama DML naredba, već vidi podatke onakve kakvi su bili pre nego što su izvršene promene.

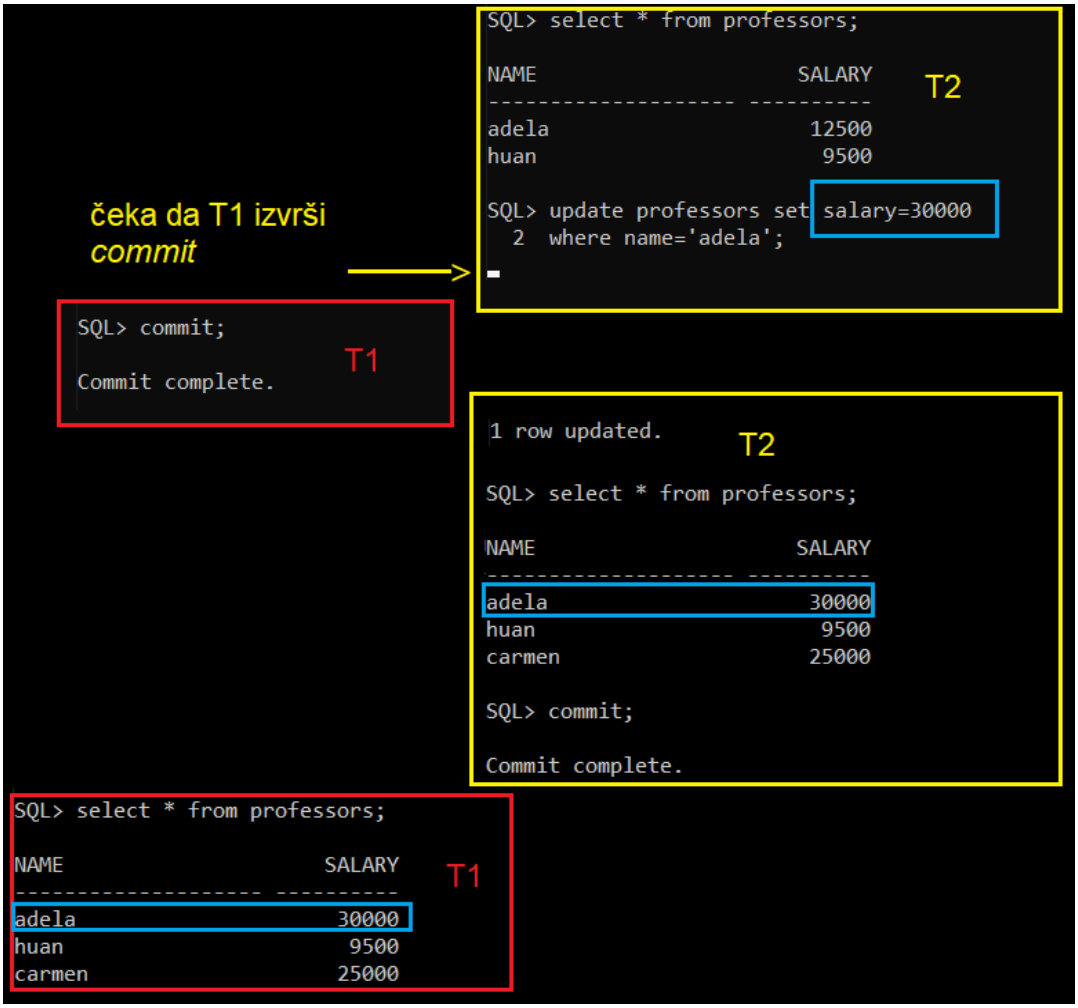
Kod transakcija s ovim nivoom izolacije može da dođe do konflikta pri upisu ukoliko transakcija modifikuje vrstu u koju upisuje još neka, neizvršena transakcija. U tom slučaju,

jedna transakcija blokira drugu da nastavi sa upisom i ta transakcija zove se blokirajuća. *Read committed* transakcija čeka da se blokirajuća izvrši i oslobodi bravu nad vrstom. Ako se blokirajuća transakcija poništi, tada transakcija na čekanju menja prethodno zaključani red kao da druga transakcija nikada nije postojala, a ako se blokirajuća transakcija potvrdi i oslobodi svoje brave, tada transakcija na čekanju nastavlja sa predviđenim ažuriranjem, ali nad promenjenom vrstom.

U nekim slučajevima, ukoliko postoji više transakcija od kojih jedna ima *READ COMMITTED* nivo izolacije, može da dođe do izgubljenih upisa (eng. *lost update*). To ilustruje scenario na slikama 7 i 8. Pokrenute su dve sesije i u obe po jedna transakcija. Prva transakcija (T1) može da ima *READ ONLY* ili *READ COMMITTED* nivo, dok druga transakcija (T2) ima *READ COMMITTED* nivo izolacije. Prva sesija vrši pretraživanje po nekom upitu i ažurira podatke, čime se pokreće transakcija sa podrazumevanim nivoom izolacije, a onda se u drugoj sesiji kreira transakcija kojoj se ekplicitno zadaje *READ COMMITTED* nivo izolacije. Druga transakcija izvršava isti upit kao i prva, ali ne vidi ažuriranje od strane prve transakcije, jer nije potvrđeno. Nakon toga i T2 vrši ažuriranje podataka, ali nad drugom vrstom iz tabele. T1 dodaje novu vrstu u tabelu, opet bez potvrđivanja, a T2 ponovo izvršava isti upit. Rezultati su takvi da T2 vidi svoje ažuriranje, ali ne vidi ništa što je uradila T1. Ukoliko pokuša da modifikuje vrstu koju je ažurirala T1, doći će do konflikta i T2 se blokira sve dok T1 ne izvrši *COMMIT* i oslobodi bravu nad vrstom. Nakon toga i T2 menja istu vrstu i potvrđuje promene, čime se u stvari gubi ažuriranje koje je izvršila T1 (izgubljen upis).



Slika 7: READ COMMITTED nivo izolacije



Slika 8: READ COMMITTED nivo izolacije

3.2. *SERIALIZABLE* nivo izolacije

Na *SERIALIZABLE* nivou izolacije, transakcija vidi samo promene počinjene u trenutku kada je transakcija započela (ne upit), kao i promene koje je izvršila sama transakcija. Transakcija se izvršava tako da izgleda kao da nijedan drugi korisnik ne menja podatke u bazi podataka, tj. kao da se transakcije izvršavaju serijski, jedna za drugom [6]. Serijalizovana izolacija je pogodna u sledećim slučajevima:

- kod velikih baza podataka i kratkih transakcija koje ažuriraju mali broj vrsti,
- kad je šansa da će dve istovremene transakcije izmeniti iste vrste relativno mala,
- u sistemi gde su relativno dugotrajne transakcije prvenstveno samo za čitanje.

U serializabilnoj izolaciji, konzistentnost čitanja koja se obično dobija na nivou naredbe proteže se na celu transakciju. Za bilo koju vrstu koju pročita transakcija važi da će biti ista i pri sledećem čitanju. Bilo koji upit garantuje vraćanje istih rezultata tokom trajanja transakcije, tako da promene izvršene drugim transakcijama nisu vidljive upitu bez obzira na to koliko dugo

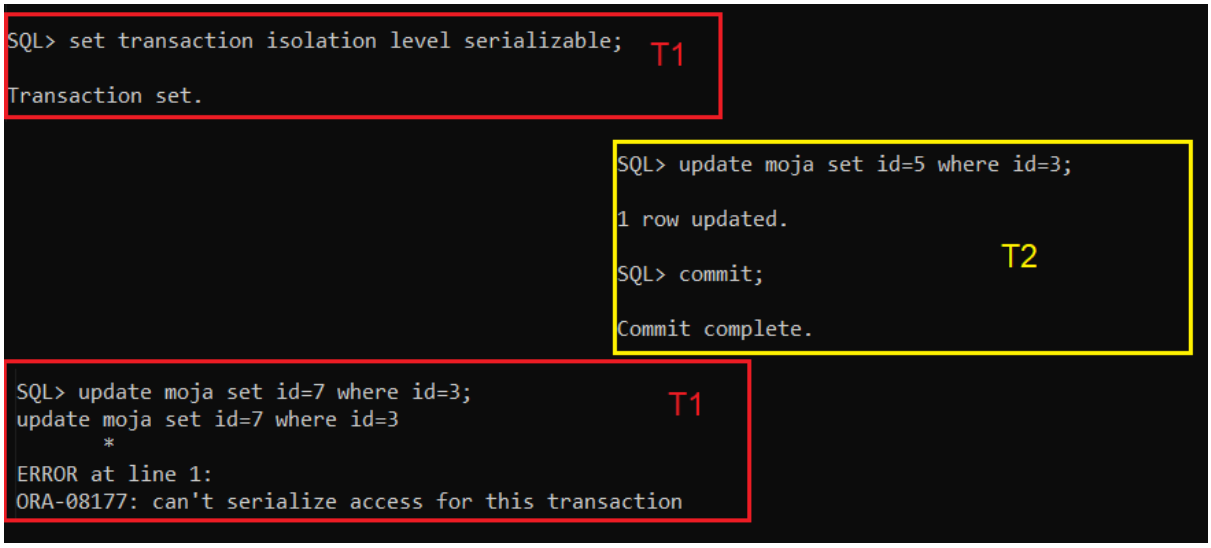
se izvršava. Kod ovih transakcija nisu dozvoljena prljava čitanja, nejasna čitanja ili fantomska čitanja [6].

Oracle dozvoljava serializabilnoj transakciji da ažurira vrstu samo ako su promene u vrsti napravljene drugim transakcijama već izvršene kada je serijalizabilna transakcija započela. Postavljanje ovog nivoa izolacije vrši se kao na slici 9:

```
SQL> set transaction isolation level serializable;

Transaction set.
```

Slika 9: Postavljanje SERIALIZABLE nivoa izolacije

Baza podataka generiše grešku kada serijalizabilna transakcija pokušava da ažurira ili izbriše podatke promenjene drugom transakcijom koja je izvršila *COMMIT* tek nakon što je serijalizabilna transakcija počela, što se vidi sa slike 10:

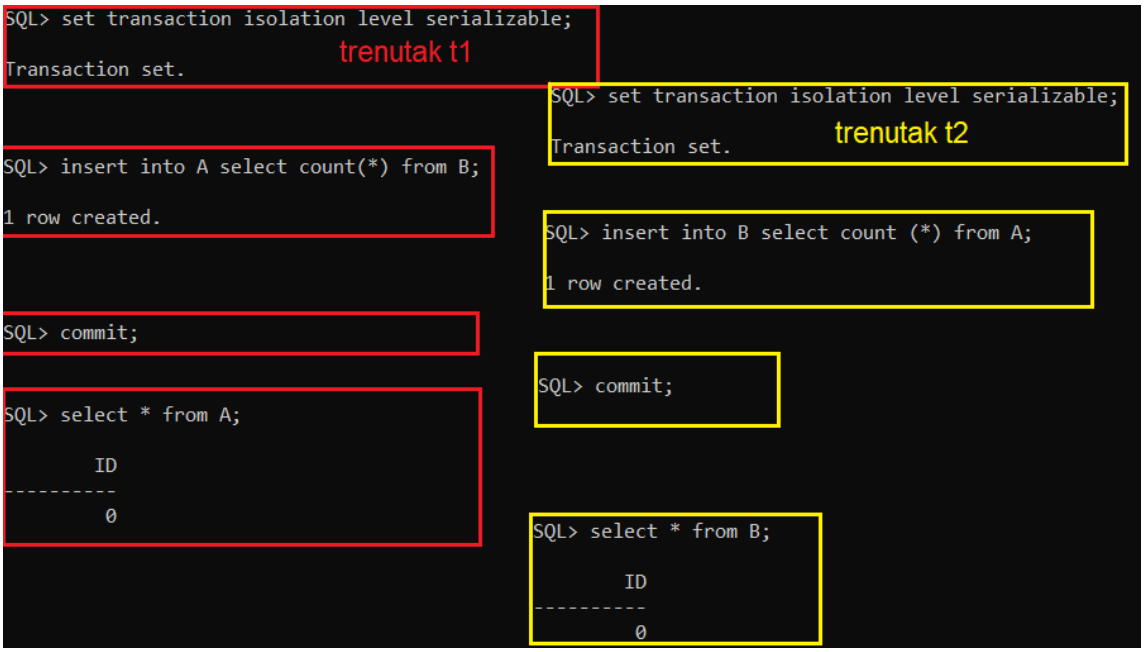


Slika 10: ORA- 08177 kod SERIALIZABLE nivoa izolacije

Ako se desi ova greška, aplikacija će da potvrdi promene izvršene do tog trenutka, da izvrši dodatne naredbe (npr. posle vraćanja na neku tačku čuvanja u transakciji) ili da poništi čitavu transakciju [6].

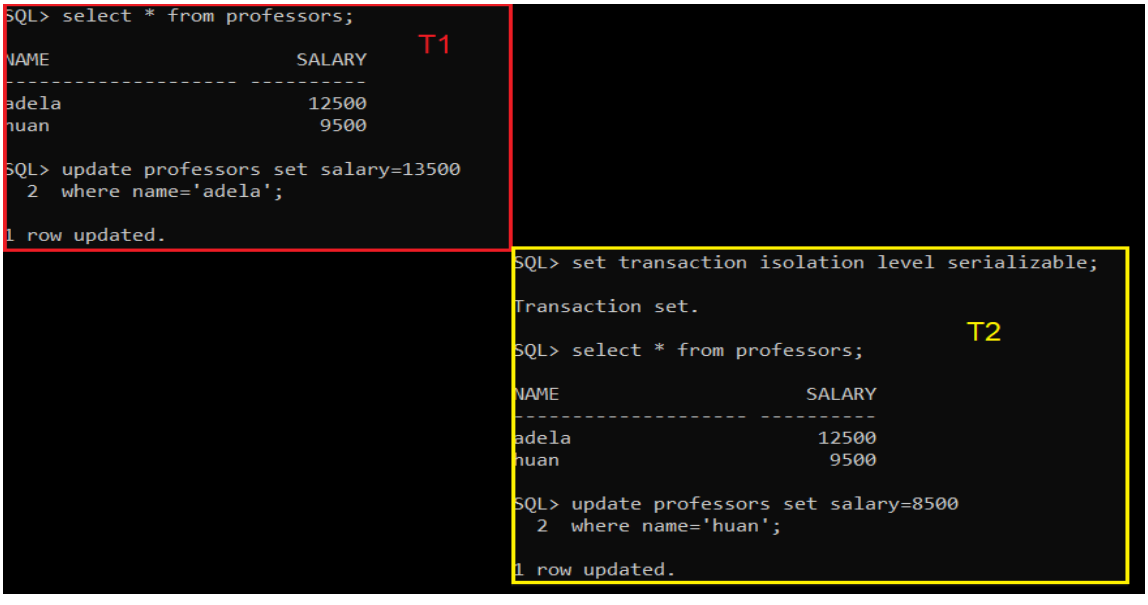
Ovaj nivo izolacije ne znači striktno da se sve transakcije ponašaju kao da su se izvršile jedna za drugom u serijskom maniru. To ilustruje sledeći primer. Kreirane su dva tabele, a onda su izvršene instrukcije sa slike broj 11. Sada i tabela *A* i *B* imaju po jednu dodatu vrstu sa vrednošću 0. Kad bi u potpunosti važio serijski poredak transakcija, ne bismo mogli da imamo obe tabele koje sadrže vrednost 0. Ako je prva transakcija izvršena u celosti pre druge, tada bi tabela *B* imala vrstu sa vrednošću 1 (ili obrnuto). Međutim, obe tabele će imati vrste sa vrednošću 0. Razlog je to što se serijalizabilna transakcija izvršava kao da je tada jedina transakcija u bazi podataka u datom trenutku. Bez obzira koliko puta sesija 1 izvršava upit nad tabelom *B* i bez obzira na potvrđeno stanje sesije 2, *count* će biti broj podataka koji je postojao

u tabeli *B* u trenutku *t1*. Isto tako, bez obzira koliko puta druga sesija izvrši upit nad tabelom *A*, *count* će biti isti kao i u trenutku *t2* (trenutak kad je transakcija započeta).

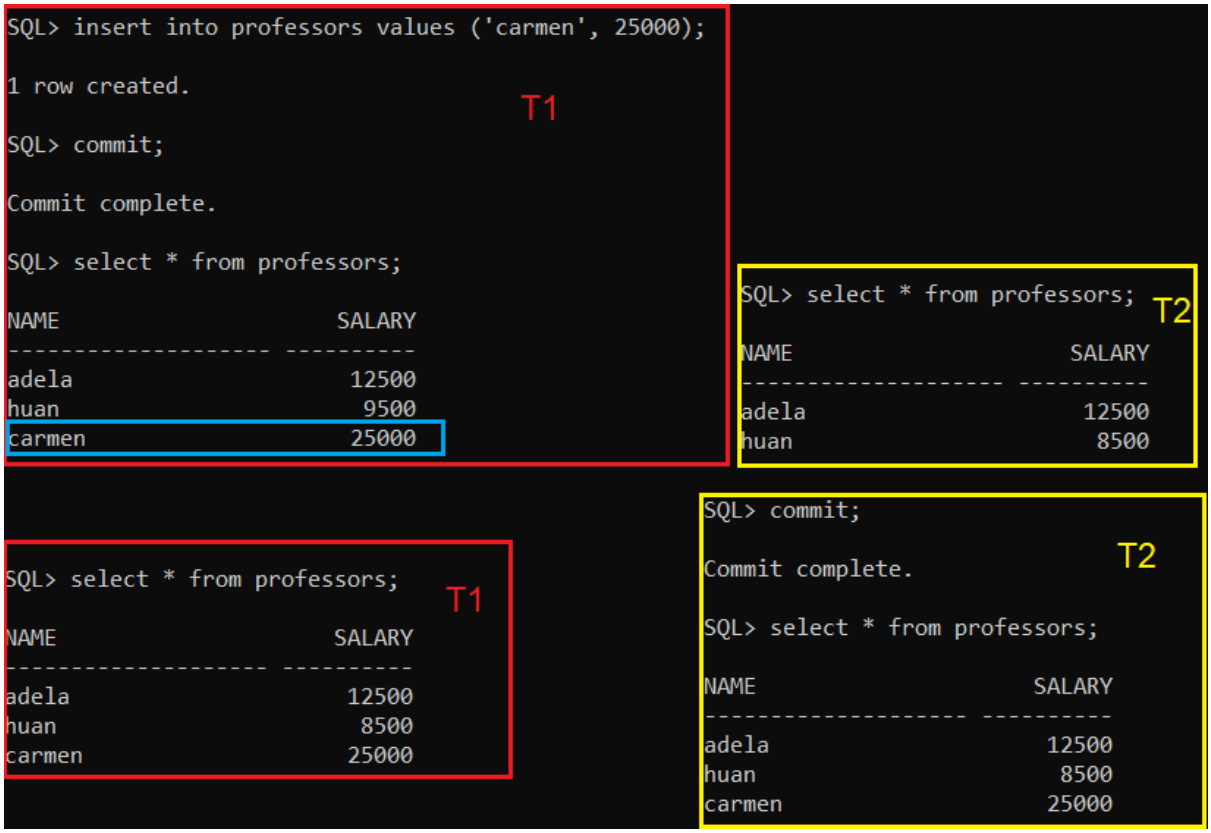


Slika 11: Serijsko izvršenje transakcija

Sledeći primer ilustruje interakciju serijalizabilne sa drugim transakcijama. Izvršava se isti scenario kao u primeru za *READ COMMITTED* transakcije, s tim što se nivo izolacije za T2 postavlja na *SERIALIZABLE*. Kad T2 izvrši upit koji je prethodno izvršila T1, dobija podatke bez ažuriranja koje je izvršila T1. T2 ažurira neku vrstu iz tabele, a onda T1 dodaje novu vrstu i izvršava *COMMIT*. Kad nakon toga i T1 i T2 izvrše upit, dobijaju različite podatke. Obe transakcije vide svoje promene, ali zbog serijalizabilnosti T2 ne vidi komitovane promene koje je načinila T1. Kad T2 izvrši *COMMIT*, a onda obe ponovo izvrše upit, dobiće identične, konzistentne podatke.



Slika 12: Serializable nivo izolacije



Slika 13:Serializable nivo izolacije

3.3. *READ ONLY* nivo izolacije

Nivo izolacije samo za čitanje (*READ ONLY*) sličan je serijalizabilnom nivou, ali transakcije samo za čitanje ne dozvoljavaju izmenu podataka u transakciji, osim ako je korisnik *SYS*. Kod ovih transakcija ne može da dođe do greške *ORA-08177*. Transakcije samo za čitanje korisne su za generisanje izveštaja u kojima sadržaj mora biti u skladu s vremenskim trenutkom kada je transakcija započela. . U transakciji sa *READ ONLY* nivoom izolacije logički snimak baze podataka kreira se na početku transakcije i otpušta na kraju transakcije. To garantuje da će sva čitanja podataka u okviru transakcije da daju konzistentne podatke iz baze.

Oracle baza održava konzistentnost čitanja rekonstruisanjem podataka po potrebi iz segmenata za poništavanje. Kao što je rečeno ranije, segmenti poništavanja se koriste u kružnom redosledu, pa baza podataka može da upiše podatke preko već postojećih. Kad su u pitanju dugotrajne transakcije, postoji mogućnost da drugi korisnici menjaju podatke koji se čitaju u *READ ONLY* transakciji. Tada se vrši i upis u segmente poništavanja, pa može da dođe do gubitka informacija potrebnih za rekonstrukciju podataka. U tim slučajevima dolazi do *snapshot too old* greške (*ORA-01555*). Nju je moguće izbeći postavljanjem odgovarajuće veličine tabelarnog prostora poništavanja.

4. Mehanizam zaključavanja u Oracle-u

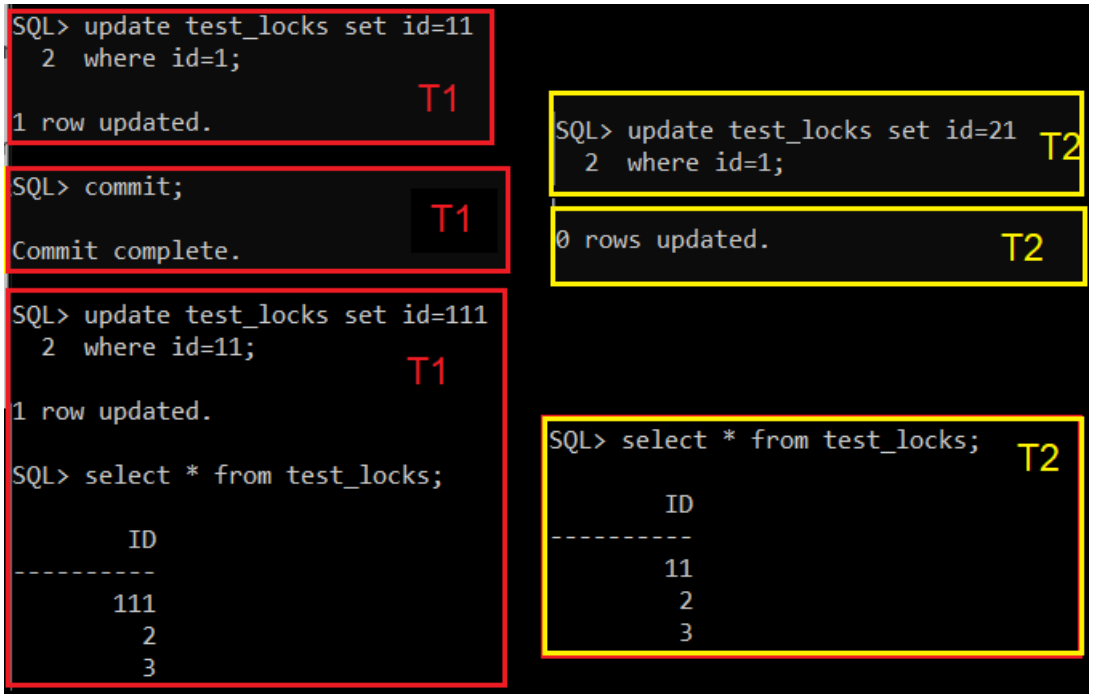
Jedan od ključnih izazova u razvoju višekorisničkih aplikacija zasnovanih na bazama podataka je omogućavanje istovremenog pristupa, ali tako da svaki korisnik može čitati i menjati podatke na konzistentan način. U jednokorisničkoj bazi podataka zaključavanje nije potrebno jer samo jedan korisnik menja informacije. Međutim, kada više korisnika pristupa istim podacima i menja ih, baza mora da obezbedi način za sprečavanje istovremene izmene istih podataka. Mehanizmi zaključavanja koji omogućavaju da se ovo dogodi ključne su odlike svake baze podataka. Brave (eng. *locks*) su mehanizmi koji se koriste za regulisanje istovremenog pristupa zajedničkom resursu i imaju ključnu ulogu u održavanju konzistentnosti baze podataka [7]. Oracle vrši zaključavanje podataka tabele na nivou vrste, ali i na mnogim drugim nivoima da bi se istovremeno obezbedio pristup različitim resursima. Zaključavanje se događa automatski i ne zahteva radnju korisnika. Na primer, dok se izvršava uskladištena procedura, sama procedura je zaključana u režimu koji omogućava drugima da je izvršavaju, ali neće dopustiti drugom korisniku da izmeni instancu te uskladištene procedure na bilo koji način. Brave se koriste u bazi podataka da bi im se omogućio istovremeni pristup zajedničkim resursima, istovremeno pružajući integritet i konzistentnost podataka.

Brave utiču na interakciju čitača i pisaca. Čitač (eng. *reader*) je upit koji se izvršava nad resursom, dok je pisac naredba koja modifikuje resurs. Sledeća pravila rezimiraju ponašanje zaključavanja Oracle baze podataka za čitače i pisce [7]:

- Vrsta se zaključava samo kada je pisac modifikuje. Kada se naredbom ažurira samo jedna vrsta, transakcija pribavlja bravu samo za tu vrstu. Zaključavanjem podataka tabele na nivou vrste, baza podataka smanjuje konflikte za iste podatke. U normalnim okolnostima baza podataka ne proširuje zaključavanje vrste do nivoa bloka ili tabele.
- Pisac vrste blokira istovremenog pisca iste vrste. Ako jedna transakcija ažurira vrstu, tada zaključavanje vrste sprečava drugu transakciju da istovremeno modifikuje istu vrstu.
- Čitač nikada ne blokira pisca. Budući da ga čitač vrste ne zaključava, pisac može da ažurira vrstu. Jedini izuzetak je naredba *SELECT ... FOR UPDATE*, koja je posebna vrsta *SELECT* naredbe koja zaključava vrstu koji čita.
- Pisac nikada ne blokira čitača. Kada pisac ažurira vrstu, baza podataka koristi podatke poništavanja kako bi čitačima pružila konzistentan prikaz vrste.

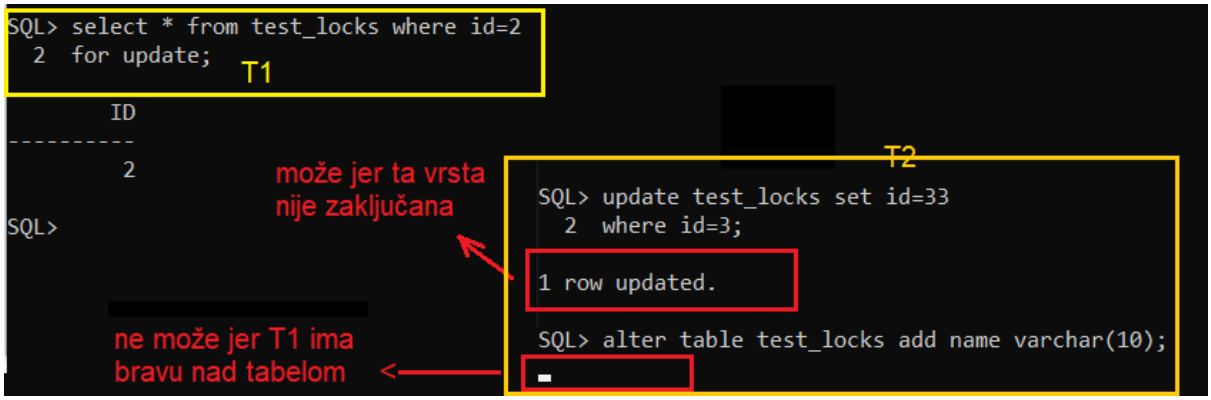
Oracle automatski pribavlja brave prilikom izvršenja SQL naredbi. Budući da su mehanizmi zaključavanja usko povezani sa kontrolom transakcija, dizajneri aplikacija treba samo pravilno da definišu transakcije, a Oracle će automatski upravljati zaključavanjem. Korisnici nikada ne treba da eksplicitno zaključavaju nijedan resurs, iako je i to moguće. Scenario na slici 14 ilustruje kako brave funkcionišu. Pokreću se dve sesije i prva vrši ažuriranje vrste čime se pokreće transakcija i pribavlja brava nad vrstom. Nakon toga druga sesija pokušava da ažurira istu vrstu, ali pošto prva sesija nije izvršila *commit*, brava nije oslobođena, pa dolazi do blokiranja. Nakon što prva izvrši *commit*, druga sesija se odblokira, ali nijedna vrsta nije ažurirana. Razlog je to što originalni podatak više ne postoji, već je ažuriran povrđivanjem

transakcije. Kad prva sesija ponovo započne ažuriranje iste vrste (kao kod prethodne *UPDATE* naredbe), opet se pribavlja brava nad njom. Međutim, kad druga sesija pročita tu vrstu, ne vidi nepotvrđene promene, već poslednje konzistentno stanje (konzistentnost čitanja).



Slika 14: Zaključavanje vrste

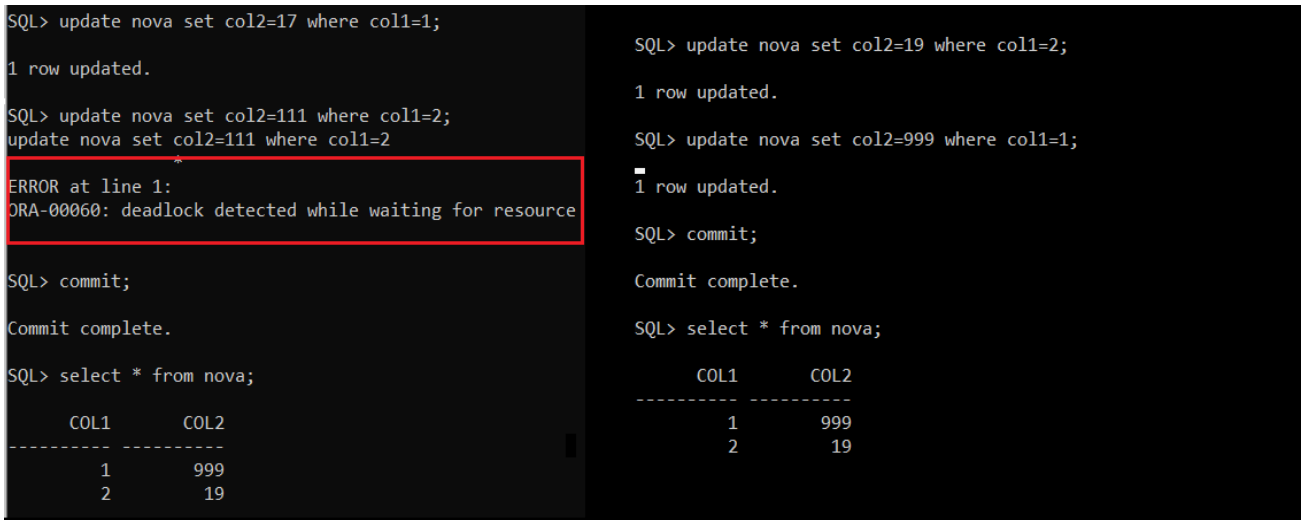
Oracle baza podataka ima nekoliko različitih vrsta brava, u zavisnosti od operacije koja je nabavila bravu. Generalno, koriste se dve vrste brava: ekskluzivne brave i zajedničke brave. Nad resursom poput vrste ili tabele moguće je pribaviti samo ekskluzivnu bravu, ali nad jednim resursom moguće je pribaviti više zajedničkih brava. Po potrebi, Oracle može automatski da izvrši konverziju brave niže restriktivnosti u bravu sa višom [7]. Ekskluzivan režim sprečava deljenje resursa. Transakcija dobija ekskluzivnu bravu prilikom modifikacije podataka. Prva transakcija koja je ekskluzivno zaključala resurs je jedina transakcija koja može izmeniti resurs dok se ne oslobodi ekskluzivna brava. S druge strane, deljive brave omogućavaju deljenje povezanog resursa, u zavisnosti od operacija koje se izvršavaju. Više korisnika može istovremeno da deli podatke radi čitanja, pri čemu svaki ima deljivu bravu kako bi se sprečio istovremeni pristup pisca kome je potrebna ekskluzivna brava. Ako transakcija koristi naredbu *SELECT ... FOR UPDATE* za odabir jedne vrste tabele, ona stiče ekskluzivnu bravu vrste i deljivu bravu tabele. Zaključavanje vrste omogućava ostalim sesijama da modifikuju bilo koje nezaključane vrste, dok zaključavanje tabele sprečava sesije da promene strukturu tabele. Time baza podataka teži da omogući izvršavanje što više naredbi (slika 15).



Slika 15: Zaključavanje vrsta i tabela

Baza automatski oslobađa zaključane resurse kada se desi neki događaj takav da transakcija više ne zahteva resurs. Uobičajeno, brave pribavljene naredbama u okviru transakcije održavaju se dok traje transakcija (dok se ne potvrdi ili poništi). Ove brave sprečavaju destruktivne smetnje kao što su prljava čitanja, izgubljena ažuriranja i destruktivni DDL od istovremenih transakcija. Oracle takođe oslobađa brave stečene nakon tačke čuvanja prilikom vraćanja na nju. Međutim, samo transakcije koje ne čekaju prethodno zaključane resurse mogu da steknu brave na sada dostupnim resursima. Transakcije koje čekaju i dalje čekaju dok se originalna transakcija ne potvrdi ili se u potpunosti poništi.

Ukoliko dva ili više korisnika čeka na podatke koje je zaključao onaj drugi, dolazi do uzajamnog blokiranja (eng. *deadlock*), odnosno, transakcije ne mogu da nastave sa izvršenjem. Oracle automatski otkriva uzajamna blokiranja i rešava ih poništavanjem jedne naredbe koja je uključena u blokiranje, oslobađajući jedan skup brava nad vrstama koje su u konfliktu. Poništena naredba pripada transakciji koja je detektovala *deadlock*. Odgovarajući primer dat je na sledećoj slici.



Slika 16: Primer uzajamnog blokiranja

4.1. Automatske brave

Oracle automatski zaključava resurs u ime transakcije da bi sprečio druge transakcije da rade nešto što zahteva ekskluzivni pristup istom resursu. Postoji nekoliko tipova brava u zavisnosti od toga prilikom izvršenja kojih naredbi se pribavljaju:

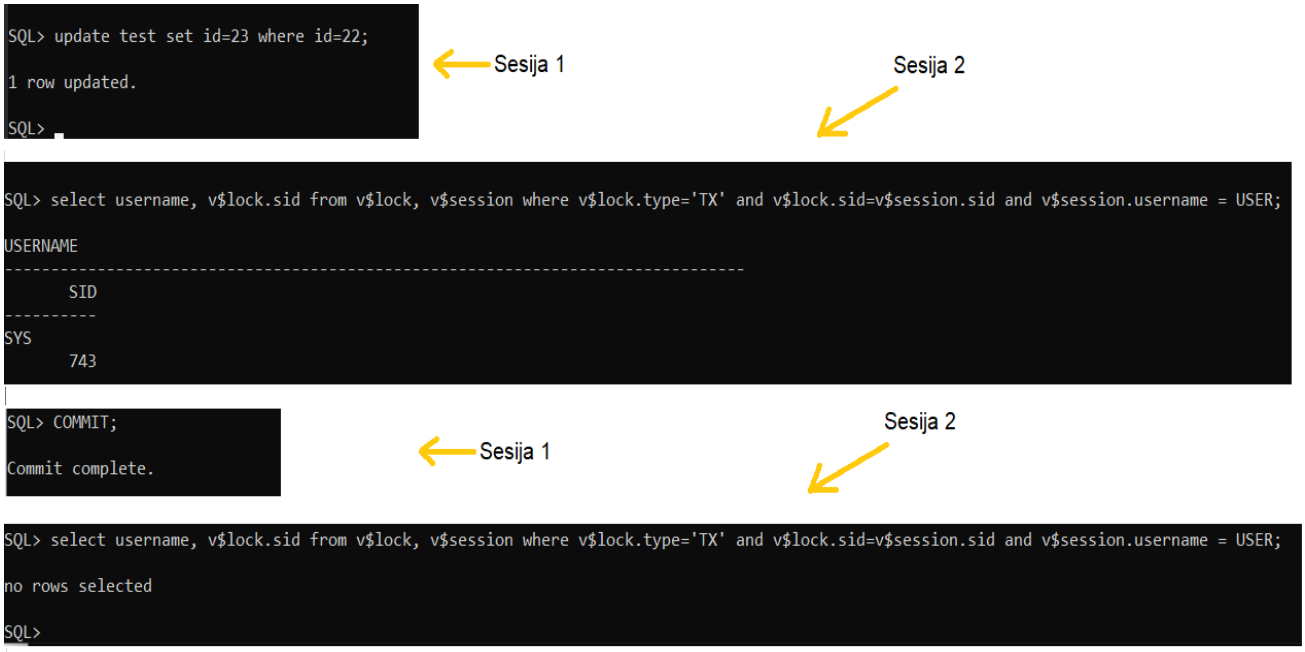
1. DML brave – služe za zaštitu podataka (npr. brava nad vrstom ili nad tabelom),
2. DDL brave – štite strukturu podataka (npr. definicije tabela i pogleda),
3. sistemske brave – štite internu strukturu baze podataka (poput fajlova podataka), lečevi, muteksi i interne brave su potpuno automatske.

4.1.1. DML brave

DML brava, poznata i kao brava nad podacima, služi za očuvanje integriteta podataka prilikom višekorisničkog pristupa. DML naredbe automatski pribavljaju brave nad vrstom (*TX* brave) ili brave nad tabelom (*TM* brave) [7].

Brava nad vrstom, koja se naziva i TX brava, je brava nad jednom vrstom tabele. Transakcija dobija bravu za svaku vrstu modifikovanu nekom od naredbi *INSERT*, *UPDATE*, *DELETE*, *MERGE* ili *SELECT ... FOR UPDATE*. Brava postoji sve dok se transakcija ne potvrdi ili poništi. TX brave prvenstveno implementiraju mehanizam čekanja kako bi se sprečilo da dve transakcije menjaju istu vrstu. Oracle uvek zaključava modifikovanu vrstu u ekskluzivnom režimu, tako da druge transakcije ne mogu modifikovati red dok se transakcija ne završi [7]. Kad transakcije dobije bravu nad vrstom, ona pribavlja i bravu nad tabelom kako bi se zabranile DDL naredbe koje bi poništile izvršene promene nad podacima.

Za razliku od drugih baza koje koriste menadžera zaključavanja (eng. *lock manager*) i podatke o bravama čuvaju u memoriji, Oracle čuva infromacije u blokovima podataka koji sadrže zaključanu vrstu. Koristi se mehanizam čekanja za pribavljanje brava nad vrstama. Ako transakcija zahteva zaključavanje za otključanu vrstu, tada transakcija pribavlja bravu, tj. postavlja se njen ID u okviru bloka podataka. Zato svaka vrsta modifikovana ovom transakcijom ukazuje na kopiju ID-a transakcije koja je sačuvana u zaglavlju bloka, umesto na same podatke [7]. Kada se transakcija završi, ID transakcije ostaje u zaglavlju bloka. Ako druga transakcija želi da modifikuje red, ona koristi ID da utvrdi da li je transakcija koja drži bravu i dalje aktivna. Ako jeste, sesija traži da bude obaveštena kada se brava otpusti, inače pribavlja i zaključava vrstu. Pregled pribavljenih brava na koje neko čeka i sesija koje čekaju na bravu dostupan je iz pogleda *V$LOCK*. Ona sadrži podatke poput identifikatora sesije koje ima bravu, tipa brave, moda zaključavanja i slično [8]. Na slici 17 dat je primer u kom jedna sesija ažurira vrstu iz tabele, a druga sesija na osnovu pogleda *V$LOCK* i *V$SESSION* vidi da postoji pribavljena TX brava. Kad prva sesija izvrši *COMMIT*, vrsta više nije zaključana.

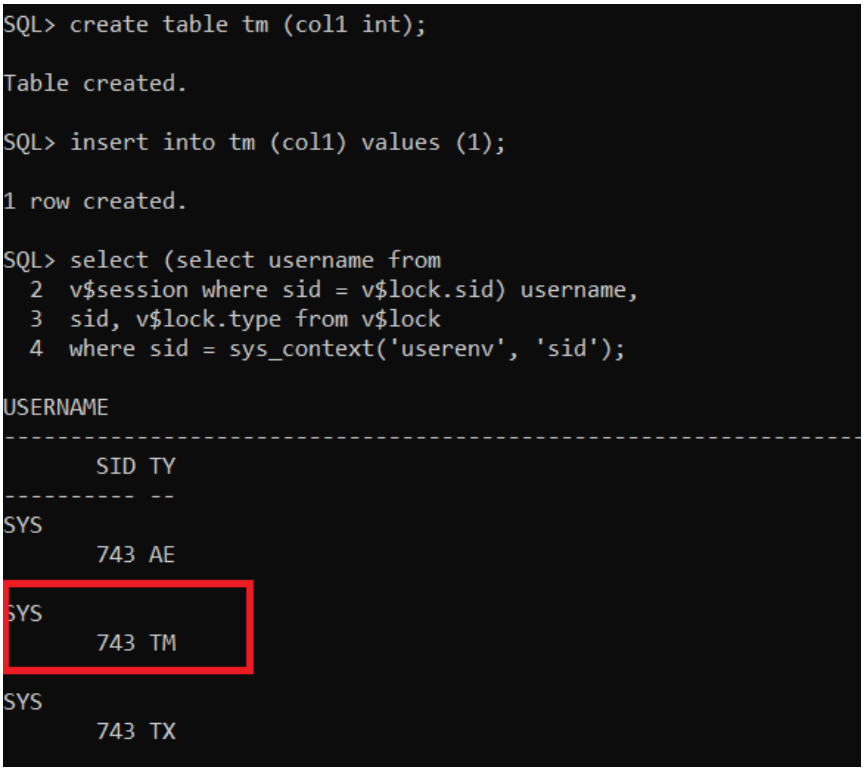


Slika 17: V$LOCK

Pored brava nad vrstom, u DML brave spada i brava nad tabelom, tj. TM brava. TM brave služe da osiguraju da neće doći do promene strukture tabele dok se menja njen sadržaj (npr. *ALTER* ili *DROP* naredbom). Transakcija pribavlja TM bravu kad izvršava *INSERT, UPDATE, MERGE, SELECT, DELETE, SELECT FOR UPDATE* ili *LOCK TABLE* naredbu. TM brava može da bude u nekom od sledećih modova rada [7]:

- *Row Share (RS)* mod – RS brava ukazuje da je transakcija koja drži bravu nad tabelom zaključala vrste u tabeli i namerava da ih ažurira. Deljiva brava vrste je najmanje restriktivni način zaključavanja tabele, i nudi najviši stepen konkurentnosti za tabelu.
- *Row Exclusive Table Lock (RX)* - ova brava, označava da je transakcija koja je pribavila bravu ažurirala vrste tabele ili izdala *SELECT ... FOR UPDATE*. RX brava omogućava drugim transakcijama da istovremeno pretražuju, dodaju, ažuriraju, brišu ili zaključavaju vrste u istoj tabeli. Prema tome, RX brave omogućavaju da više transakcija dobije istovremeno RX ili RS brave za istu tabelu.
- *Share Table Locks (S)* – Deljiva brava nad tabelom koju pribavlja transakcija omogućava drugim transakcijama da pretražuju tabelu (bez korišćenja *SELECT ... FOR UPDATE*), ali su ažuriranja dozvoljena ako samo jedna transakcija ima S bravu.
- *Share Row Exclusive Table Lock (SRX)* – ova brava je restriktivnija od S brave. Samo jedna transakcija istovremeno može da dobije SRX bravu nad nekom tabelom. Ako neka transakcija ima SRX bravu, ostale transakcije mogu jedino da pretražuju tabelu (osim *SELECT ... FOR UPDATE*).
- *Exclusive Table Lock (X)* – najrestriktivniji tip zaključavanja, zabranjuje drugim transakcijama da izvrše bilo koju DML naredbu ili zaključaju tabelu.

Primer na slici 18 ilustruje kako se nakon kreiranja tabele i dodavanja vrste u nju pribavlja TM brava nad tabelom.



Slika 18: TM brava

Po transakciji je moguće pribaviti samo jednu TX bravu, ali kad su u pitanju TM brave, pribavlja se onoliko koliko se objekata modifikuje. Ukupan broj TM brava može da se zada unapred, i može da bude postavljen i na 0 [4]. U tom slučaju, nije moguće izvršenje DDL naredbi nad bazom. Kreiranje TM brava može da se zabrani i naredbom *ALTER TABLE imeTabele DISABLE TABLE LOCK*. Da bi korisnik mogao npr. da obriše tabelu, prvo mora da omogući TM brave.

4.1.2. DDL brave

Brava rečnika podataka (eng. *Data Dictionary Lock, DDL*) štiti definiciju objekta šeme dok tekuća DDL operacija deluje ili se odnosi na objekat. Tokom DDL operacija zaključavaju se samo pojedinačni objekti šeme koji su izmenjeni ili referencirani. Baza podataka nikada ne zaključava ceo rečnik podataka. Oracle automatski pribavlja DDL brave u ime bilo koje DDL transakcije koja to zahteva, a korisnici ne mogu ekskluzivno da ih zahtevaju. DDL brave se zadržavaju tokom trajanja DDL operacija i oslobađaju se po njihovom završetku. To je obezbeđeno interno umotavanjem DDL izraza u implicitna potvrđivanja (*commit*) ili par potvrđivanje/poništavanje (*commit*/rollback). Iz tog razloga, DDL operacije se uvek potvrđuju u Oracle-u, čak i kad je izvršenje neuspešno. Takođe, DDL i započinje izvršenje *COMMIT* naredbom, da se u slučaju greške ne bi izgubile promene. Postoji tri tipa DDL brava: ekskluzivne, deljive i raščlanjive brave.

Ekskluzivne DDL brave sprečavaju druge sesije u dobijanju DDL ili DML brave. Većina DDL operacija zahteva ekskluzivne DDL brave za resurs. Na primer, *DROP TABLE* ne sme da obriše tabelu dok joj *ALTER TABLE* dodaje kolonu i obrnuto. Ekskluzivne DDL

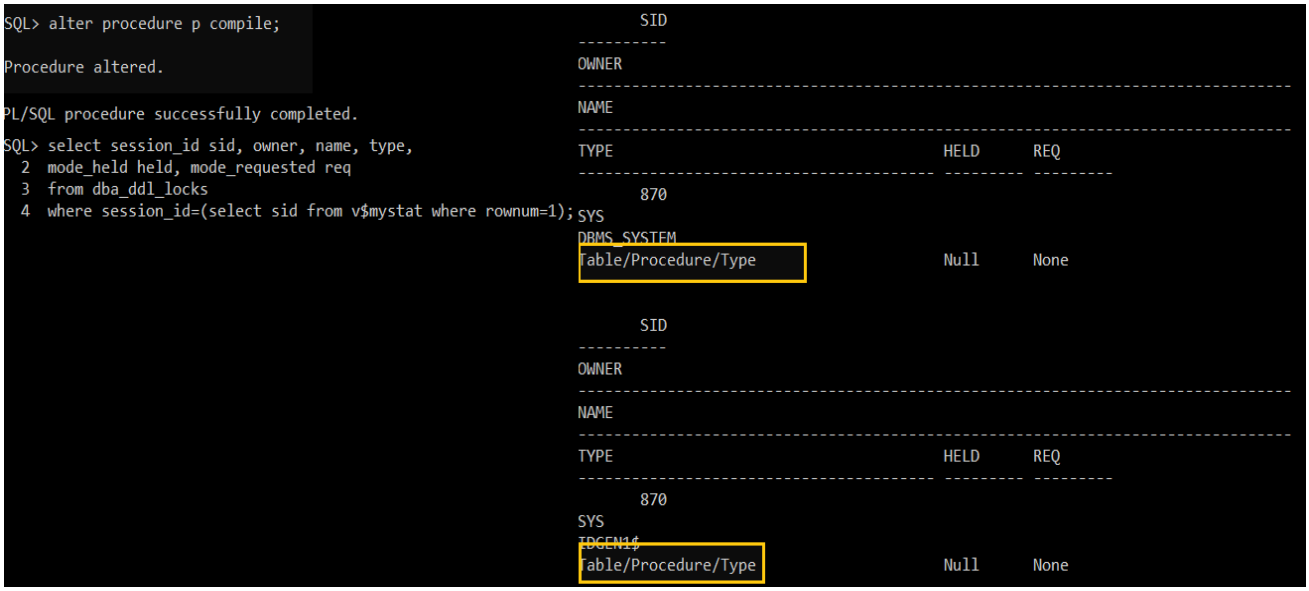
brave traju tokom izvršenja DDL operacije i automatskog *commit*-a. Tokom pribavljanja ekskluzivne DDL brave, ako je druga DDL operacija zaključala objekat šeme, pribavljanje čeka da se starija DDL brava oslobodi, a onda nastavlja sa izvršenjem. Najviše DDL naredbi pribavlja ekskluzivnu bravu.

Deljive DDL brave za resurs sprečavaju destruktivne smetnje u sukobljenim DDL operacijama, dok s druge strane omogućavaju istovremeno konkurentnost za slične DDL operacije. Na primer, kada se izvrši naredba *CREATE PROCEDURE*, transakcija koja je sadrži stiče DDL brave za sve referencirane tabele. Druge transakcije mogu istovremeno da kreiraju procedure koje upućuju na iste tabele i pribavljaju DDL brave nad istim tabelama, ali nijedna transakcija ne može da dobije ekskluzivnu DDL bravu nad bilo kojom referenciranom tabelom. Deljiva DDL brava traje isto kao i ekskluzivne. Prema tome, transakcija koja ima deljenu DDL bravu garantuje da će definicija referenciranog objekta šeme da ostane konstantna tokom transakcije.

Raščlanjive DDL brave pribavljaju SQL naredbe ili PL/SQL programi za svaki objekat šeme koji se referencira. Raščlanjive brave zovu se i lomljive, jer ne zabranjuju nijednu DDL operaciju i mogu se prekinuti kako bi se omogućile sukobljene DDL operacije. Ove brave se pribavljaju u zajedničko spremište (eng. *shared pool*) tokom faze raščlanjivanja u izvršenju SQL naredbe i traju sve dok deljivo SQL područje (eng. *SQL area*) za tu naredbu ostaje u deljenom spremištu. Kad se referencirani objekat izbriše ili promeni, ovo spremište se invalidira. Sledeći primer (slika 19) prikazuje ovaj tip brava. Kreirana je i izvršena procedura. Pretraživanjem *dba_ddl_lock* tabele (sadrži listu DDL brava) vidi se da postoje 3 DDL brave tipa *Table/Procedure/Type*. Procedura se kompajlira, a onda se ponovo prikazuju DDL brave. Sada ih ima 2, što znači da je jedna „slomljena“.

```
SQL> create or replace procedure p
  2  as
  3  begin
  4  null;
  5  end;
  6  /

Procedure created.

SQL> exec p;

PL/SQL procedure successfully completed.

SQL> select session_id sid, owner, name, type,
  2  mode_held held, mode_requested req
  3  from dba_ddl_locks
  4  where session_id=(select sid from v$mystat where rownum=1);

------------------------------------------ --------- ---------
       870
SYS
P
Table/Procedure/Type                      Null      None


       SID
----------
OWNER
--------------------------------------------------------------------------------
NAME
--------------------------------------------------------------------------------
TYPE                                      HELD      REQ
------------------------------------------ --------- ---------
       870
SYS
DBMS_SYSTEM
Table/Procedure/Type                      Null      None


       SID
----------
OWNER
--------------------------------------------------------------------------------
NAME
--------------------------------------------------------------------------------
TYPE                                      HELD      REQ
------------------------------------------ --------- ---------
       870
SYS
IDGEN1$
Table/Procedure/Type                      Null      None
```

Slika 19: Raščlanjiva DDL brava nad procedurom



Slika 20: "Lomljenje" raščlanjive DDL brave

4.1.3. Sistemske brave

Kod Oracle-a postoji nekoliko tipova sistemskih brava koje služe da održe internu strukturu baze podataka i memorije i korisnici nemaju kontrolu nad ovim bravama. Tu spadaju lečevi, muteksi i interne brave.

Leč je jednostavan mehanizam serializacije na niskom nivou koji upravlja višekorisničkim pristupom deljenim strukturama podataka, objektima i datotekama [7]. Štiti resurse zajedničke memorije od oštećenja kada im se pristupa iz više procesa, tačnije u sledećim situacijama:

- istovremena modifikacija u više sesija,
- jedna sesija čita, a druga modifikuje podatke,
- brisanje (starenje) memorije tokom pristupa

Jedan leč često štiti više objekata. Za razliku od brava na koje sesije čekaju (npr. TX brava) lečevi ne dozvoljavaju da sesije čekaju u redu. Kada leč postane dostupan, prva sesija koja ga zahteva dobija ekskluzivni pristup. Oracle obično brzo pribavlja leč dok upravlja ili čita strukturu podataka. Na primer, dok obrađuje ažuriranje zarade jednog zaposlenog, baza podataka može da pribavi ili oslobodi na hiljade lečeva. Imeplementacija lečeva zavisi od operativnog sistema, posebno u pogledu toga da li i koliko dugo proces može čeka na leč. Što se više lečeva koristi, to je manje konkurentnosti. Pogled *V$LATCH* sadrži detaljne statistike upotrebe za svaki leč, uključujući broj zahteva i čekanja na svaki od njih.

Čekanje na leč može da bude skupa operacija. Ako nije dostupan odmah, a proces je spreman, tada se na multi-CPU mašini sesija “vrti”, pokušavajući iznova i iznovada pribavi leč u petlji [4]. Razlog je to što je promena konteksta procesura (oslobađanje i ponovo dobijanje

CPU-a skupa operacija. Dakle, ako proces ne može odmah da dobije leč, on i dalje drži CPU i pokušava odmah ponovo da pribavi leč, umesto da "spava", tj. otpusti CPU i čeka sledeći vremenski slot u kom će mu CPU opet biti dodeljen. Ovo se izvršava jer su lečevi napravljeni tako da ih procesi drže kratko, pa se očekuje da će se leč eventualno pribaviti u petlji. Ako to nije slučaj, dolazi do spavanja leča koje se javlja kada proces oslobodi CPU pre obnavljanja zahteva za leč. Ova akcija spavanja obično je rezultat istovremenog izvršavanja mnogih sesija koje zahtevaju isti leč (ne drži jedna sesija dugo, već veliki broj sesija želi istovremeno leč i otpuštaju ga za kratko vreme).

4.1.4. Muteksi

Objekat uzajamnog isključivanja ili muteks (eng. *mutual exclusion object*, *mutex*) je mehanizam na niskom nivou koji sprečava starenje ili oštećenje objekta u memoriji kada mu pristupaju konkurentni procesi [7]. Muteks je sličan leču, ali može da štiti samo jedan objekat, dok leč može više. Iako su slični, muteks i leč su nezavisni mehanizmi, pa proces može istovremeno da ima oba. Korišćenje muteksa ima sledeće prednosti:

- smanjuje se mogućnost pojave konflikta – pošto jedan leč može da štiti više objekata, može da postane usko grlo kada procesi istovremeno pokušavaju da pristupe bilo kom od tih objekata. Muteks, s druge strane, se odnosi samo na jedan objekat, pa je dostupnost veća.
- Jednostavniji je od leča i zahteva manje memorije od njega.

Muteks ima ekskluzivni i deljeni režim rada, a u deljenom režimu više sesija može pristupiti istom muteksu. Broj sesija koje pristupaju datom muteksu poznat je kao broj referenci i ta informacija čuva se u samom muteksu [9]. Kad je u ekskluzivnom režimu rada, broj referenci muteksa je 1. Pored toga, objekat u kešu ne može da ostari sve dok mu broj referenci ne postane 0. Informacije o muteksima dostupne su iz pogleda *V$MUTEX_SLEEP* i *V$MUTEX_SLEEP_HISTORY*. Pogled V$MUTEX_SLEEP sadrži podatke o tipu leča, lokaciji u kodu u kojoj proces "spava" čekajući na leč, vreme čekanja na leč i sl. a V$MUTEX_SLEEP_HISTORY pored ovih ima dodatne informacije o muteksima.

4.1.5. Interne brave

Interne brave su složeni mehanizmi za zaključavanje, na višem nivou od muteksa i lečeva. Postoji nekoliko tipova internih brava, uključujući brave keša rečnika podataka (eng. *dictionary cache lock*), brave nad datotekama i logovima (eng. *file and log management* lock) i brave nad prostorom tabela i segmentima za poništavanje (eng. *tablespace and undosegments lock*).

Brave keša rečnika podataka su obično kratkotrajne i pribavljaju se nad unosima u kešu rečnika kad se unosi menjaju ili koriste [7]. Keš rečnika podataka služi za čuvanje informacije o objektima u bazi podataka (metapodaci, imena tabelam kolona) i često se referencira prilikom preprocesiranja SQL naredbi. Zaključavanjem se osigurava da naredbe koje se parsuju ne vide

nekonzistentne definicije objekata. Mogu da budu u ekskluzivnom ili deljivom režimu. Deljive se oslobađaju po završetku parsiranja, a ekskluzivne kad se izvrši DDL naredba.

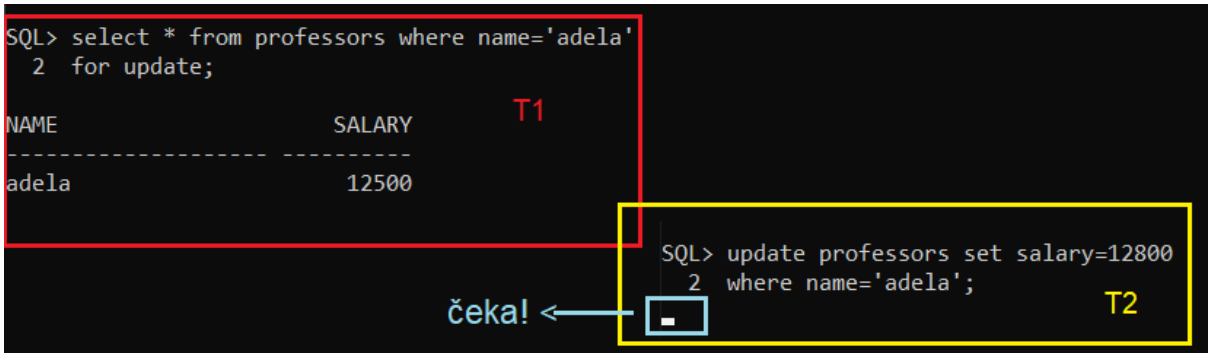
Brave nad datotekama i logovima služe za zaštitu različiih datoteka, poput kontrolnih, redo log datoteka, datoteka sa podacima. Pošto brave datoteka i logova ukazuju na status datoteka, obično se zadržavaju duži vremenski period [7].

Poslednji tip internih brava štiti prostor tabela i segmente poništavanja. Na primer, ako postoji više instanci koje pristupaju bazi podataka, sve moraju da se slože oko toga da li je prostor tabela na mreži ili ne. Segementi za poništavanje se zaključavaju da bi u jednom trenutku samo jedna instanca mogla da upisuje u segment [7].

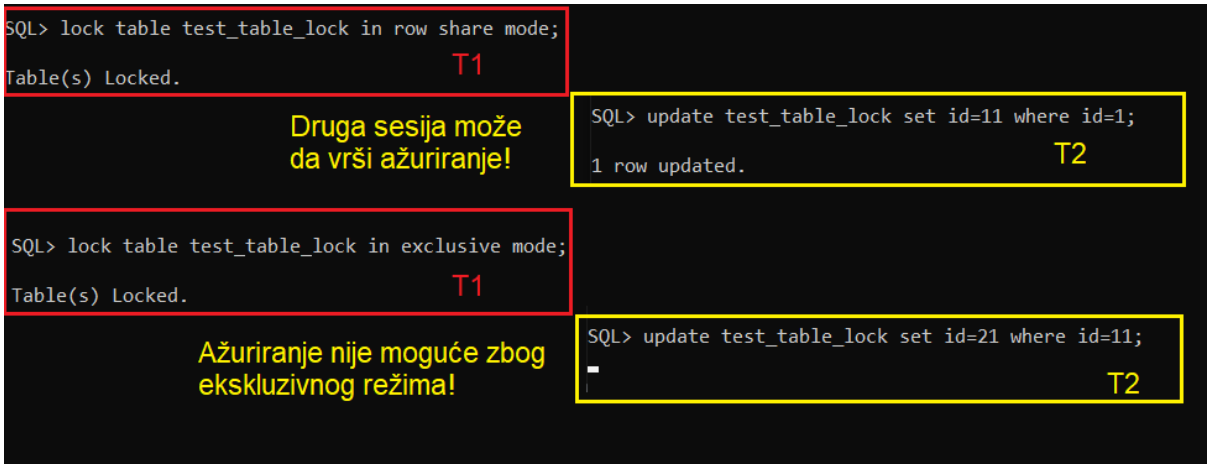
4.2. Manuelne i korisnički definisane brave

Manuelne brave nastaju predefinisanjem (eng. *override*) Oracle-ovog podrazumevanog mehanizma zaključavanja. To može da bude pogodno u situacijama kad aplikacije zahtevaju konzistentnost čitanja na nivou transakcije ili ponavljajuća čitanja. Tada upiti moraju da vraćaju konzistentne podatke tokom trajanja transakcije, bez uticaja promena koje su izvršile druge transakcije. Takođe, ako aplikacije zahtevaju da transakcija ima ekskluzivni pristup resursu, a da ne sme čeka da se druge transakcije izvrše, manuelne brave mogu da budu korisne. U ostalim slučajevima trebalo bi izbegavati njihovo korišćenje.

Preklapanje automatskog zaključavanja može da se izvrši na nivou sesije pomoću naredbe *ALTER SESSION* ili na nivou transakcije naredbama *LOCK TABLE* i *SELECT ... FOR UPDATE* [7]. Treba voditi računa da integritet podataka ostane zagarantovan, da konzistentnost podataka bude prihvatljiva, a da do uzajamnog blokiranja ne dolazi (ili se rešava na odgovarajući način). Naredbom *SELECT ... FOR UPDATE* vrši se manuelno zaključavanje na nivou vrste, tako da nijedna druga transakcija ne može da modifikuje tu vrstu dok se ne izvrši *commit* (primer na slici 21). Pomoću *LOCK TABLE* zaključava se tabela u nekom od režima: *ROW SHARE*, *ROW EXCLUSIVE*, *SHARE*, *SHARE ROW EXCLUSIVE EXCLUSIVE* (DML brave, poglavlje 3.1.1). Primer je dat na slici 22.



Slika 21: SELECT FOR UPDATE naredba



Slika 22: LOCK TABLE

Korisnici mogu da definišu svoje brave kada im je to potrebno u aplikaciji. Korisnički definisane brave su Oracle servisi za upravljanje zaključavanjem i mogu da se koriste u aplikacijama pozivanjem potprograma iz paketa *DBMS_LOCK*. U okviru PL/SQL blokova mogu da se pozovu naredbe koje zahtevaju bravu određenog tipa, daju joj jedinstveno ime, menjaju joj tip ili je oslobađaju. Korisničke brave su kao i Oracle-ove, pa imaju mehanizam za detekciju uzajmnog blokiranja. Do konflikta između korisnički definisanih i Oracle-ovih brava ne dolazi do konflikta jer se korisničke prepoznaju po prefiksu *UL*.

5. Zaključak

Kod višekorsničkih sistema potrebno je osigurati da baza podataka ostane u konzistentnom stanju. U tu svrhu koriste se transakcije i prevode bazu podataka iz jednog konzistentnog stanja u drugo. Imaju ACID svojstva i predstavljaju je skup naredbi takav da se ili izvršavaju sve naredbe iz skupa ili se ne izvršava nijedna. Ukoliko je transakcija autonomna, ona se poziva iz glavne transakcije i potpuno je nezavisna od nje. Kod distribuiranih baza podataka koriste se distribuirane transakcije i kod njih važi da se ili cela transakcija izvršava na svim čvorovima u mreži ili ni na jednom. Transakcijama je moguće postaviti nivo izolacije. Različiti nivoi izolacije određuju različite izlaze za iste ulazne podatke. Kod Oracle-a postoji tri nivoa izolacije: *READ COMMITTED* (podrazumevani), *SERIALIZABLE* i *READ ONLY* nivo.

Da bi podaci u višekorisničkom sistemu ostali konzistentni, potrebno je regulisati situacije u kojima se istovremeno pristupa resursu i vrši njegova modifikacija. Za to služi Oracle-ov mehanizam zaključavanja i brave. Brava osigurava da u jednom trenutku samo jedan korisnik može da ažurira neki resurs. Moguće je zaključavanje na nivou vrste ili na nivou tabele. Oracle brave pribavlja automatski, zavisno od toga koju naredbu izvršava. Pored toga, moguće je predefinisanje brava, kao i kreiranje korisničkih.

6. Literatura

[1] *Transakcija (baze podataka)*, https://en.wikipedia.org/wiki/Database_transaction, (maj 2021)

[2] Oracle, (2021), *Transactions*, Transactions (oracle.com), (maj 2021)

[3] Oracle, (2021), *V$TRANSACTION*, V$TRANSACTION (oracle.com), (maj 2021)

[4] Thomas Kyte, Darul Khun, (2014), *Oracle Database Transactions and Locking Revealed*, (maj 2021)

[5] Oracle, (2021), *Undo Segments*, Logical Storage Structures (oracle.com), (maj 2021)

[6] Oracle, (2021), *Data Concurrency and Consistency : Overview of Transaction Isolation Leveles*, Data Concurrency and Consistency (oracle.com), (jun 2021)

[7] Oracle, (2021), *Data Concurrency and Consistency : Overview of Oracle Locking Mechanism*, Data Concurrency and Consistency (oracle.com) (jun 2021)

[8] Oracle, (2021), *V$LOCK*, V$LOCK (oracle.com), (jun 2021)

[9] David Fitzjarrel, (2017), *Oracle mutexes*, https://www.databasejournal.com/features/oracle/oracle-mutexes.html, (jun 2021)