KADA ORACLE NAREDBA NIJE SERIJABILNA?

Zlatko Sirotić, univ.spec.inf.
ISTRA TECH d.o.o.
Pula

Neki noviji radovi

- CASE 2015: Višestruko nasljeđivanje - san ili Java 8?
- JavaCro 2015: Java paralelizacija

- HrOUG 2014: Nasljeđivanje je dobro, naročito višestruko - Eiffel, C++, Scala, Java 8
- CASE 2014: Trebaju li nam distribuirane baze u vrijeme oblaka?
- JavaCro 2014: Da li postoji samo jedna "ispravna" arhitektura web poslovnih aplikacija

- HrOUG 2013: Transakcije i Oracle - baza, Forms, ADF
- CASE 2013: Što poslije Pascala? Pa … Scala!

Teme

- Oracle transakcije
- (Ne)serijabilnost Oracle transakcije
- Serijabilnost Oracle naredbe i izuzeci

Uvod

- Ispravno rukovanje transakcijama u bazama podataka vrlo je bitno za poslovne aplikacije.

- Razumijevanje transakcija postaje sve važnije, jer su one "ušle" u programske jezike, pa i u hardver, u vidu:

- softverskih transakcijskih memorija (STM);
- hardverskih TM (npr. procesor Intel Haswell);
- hibridnih TM.

Osobine Oracle transakcije

- Sesija BP **ne vidi promjene koje je napravila druga sesija**, dok druga sesija ne napravi COMMIT (ili ROLLBACK). Međutim, sesije nisu nezavisne, jer zaključavanje redaka u jednoj sesiji utječe na drugu sesiju koja pokušava ažurirati redak koji je zaključala prva sesija.
- Kada sesija izvršava DML naredbu, **automatski se zaključa redak**. Redak se može otključati tek na kraju transakcije (COMMIT ili ROLLBACK), ili pomoću ROLLBACK TO SAVEPOINT. Međutim, redci koji su otključani sa ROLLBACK TO SAVEPOINT ostaju zaključani za one sesije koje su već prije toga pokušale izvršiti DML na tim redcima.

Osobine Oracle transakcije

- Transakcija može ili u cijelosti uspjeti (COMMIT), ili biti u cijelosti poništena (ROLLBACK).
- Pritom se mogu desiti **tri tipa grešaka**:
  - narušen **uvjet na tip stupca**, npr. upis 100 u NUMBER (2)
  - narušeno **deklarativno integritetno ograničenje** (PK, UK, FK, CK, NOT NULL)
  - narušeno **proceduralno ograničenje** (okidači baze).
- Ako se desila greška na bazi koja nije obrađena, a početak je DML naredba, **poništavaju se svi efekti naredbe**.
- Ako se desila greška na bazi, a početak je poziv procedure sa strane klijenta (npr. Forms), ili poziv sa strane druge baze (udaljena procedura), **poništavaju se svi efekti procedure**.

Implicitno / eksplicitno zaključavanje; Deadlock

- **Zaključavanje se najčešće radi implicitno**, pomoću DML naredbi (INSERT, UPDATE, DELETE).
- Ponekad je potrebno koristiti **eksplicitno zaključavanje** pomoću SELECT ... FOR UPDATE kako bi se sačuvao integritet podataka.
- No, (posebno) tada se može desiti **deadlock** - ako npr. dvije transakcije pokušaju zaključati dva retka, ali u suprotnom redoslijedu.
- Oracle baza otkrit će da je došlo do deadlocka, te će jedna od dvije transakcije dobiti grešku **ORA-00060: deadlock detected while waiting for resource**. Nakon što ta transakcija (koja je dobila grešku) napravi ROLLBACK, druga transakcija će nastaviti sa radom.

SAVEPOINT / ROLLBACK TO SAVEPOINT i SELECT … FOR UPDATE

- Redci se mogu otključati i sa ROLLBACK TO SAVEPOINT, ali sesiji koja pokuša pristupiti zaključanim redcima prije nego druga sesija napravi ROLLBACK TO SAVEPOINT, ti redci ostaju zaključani.
- Sesija koja čeka na zaključane retke u pravilu **čeka neograničeno**, tj. dok joj druga sesija ne otključa retke.
- Postoje **dva izuzetka**:

1. SELECT ... FOR UPDATE **NOWAIT / WAIT timeout**;
2. ako sesija čeka na otključavanje redaka sa udaljene baze, onda je **čekanje definirano parametrom** DISTRIBUTED_LOCK_TIMEOUT (default = 60 sekundi)

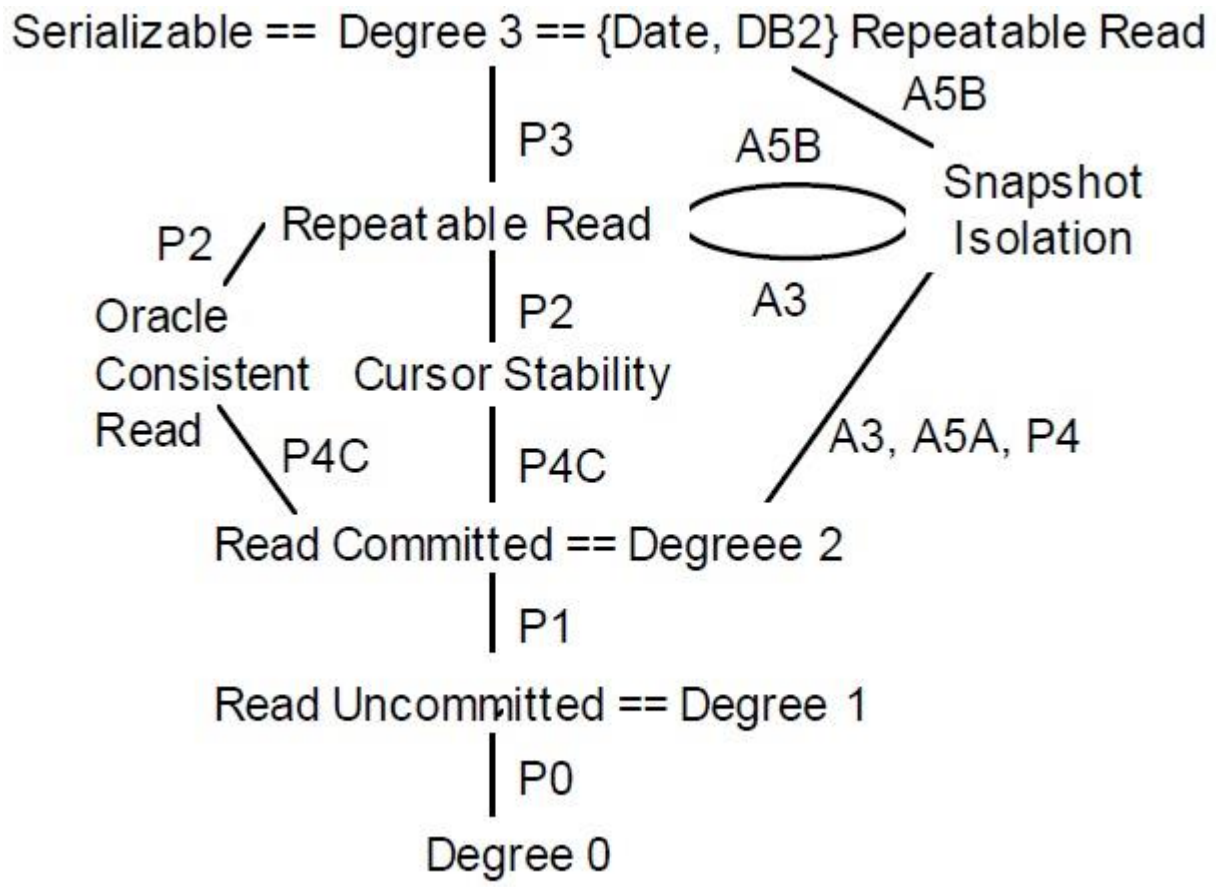
LOCK TABLE

- Ponekad želimo **zaključati cijelu tablicu** – to možemo izvesti **u djeljivom ili ekskluzivnom načinu** (modu).
- Više sesija može zaključati tablicu u djeljivom načinu:
  LOCK TABLE tablica **IN SHARE MODE** NOWAIT;
- Samo jedna sesija može zaključati tablicu u ekskluzivnom načinu:
  LOCK TABLE tablica **IN EXCLUSIVE MODE** NOWAIT;
- Zaključavanje tablice u djeljivom i ekskluzivnom načinu može se iskoristiti npr. za omogućavanje da **više sesija mijenja neku tablicu** dokumenata (npr. tablicu narudžbi), a da **samo jedna sesija može raditi obradu** narudžbi.

ACID svojstva (ACID property) i dvofazno zaključavanje (two-phase locking) transakcije

- Često se kaže da transakcija treba zadovoljavati **ACID svojstva** . Slova označavaju atomarnost (Atomicity), konzistentnost (Consistency), izoliranost (Isolation), trajnost (Durability).
- **Izoliranost** se često zove i **serijabilnost** (serializability). Misli se na to da bi transakcije trebale uvijek ostaviti efekt kao da se izvršavaju **serijski** (jedna za drugom), iako se izvršavaju konkurentno (paralelno ili kvazi-paralelno).
- Da bi se postigla serijabilnost, DBMS sustavi često primjenjuju **dvofazno zaključavanje (2PL)**, koje ne treba miješati sa dvofaznim commit protokolom kod distribuiranih BP. U **fazi širenja** (growing phase), transakcija zaključava retke, a poslije ih samo otključava, u **fazi stezanja** (shrinking phase). **Oracle radi drugačije.**

Oracle koristi Multi-version zaključavanje, varijantu Oracle Consistent Read - slika iz Berenson H. i drugi (1995): A Critique of ANSI SQL Isolation Levels



UNDO tablespace

- UNDO tablespace **služi za elimininiranje (iz tablica podataka) promjena koje nisu commitirane**, kod oporavka sustava nakon pada.
- **Služi i za omogućavanje višekorisničkog rada**. Budući da se često ne-commitirani podaci moraju spremati na disk, pitanje je otkuda sesija može čitati stare podatke (prije promjene) – oni se nalaze u UNDO tablespaceu.
- Postojanje UNDO tablespacea (ili neke slične strukture u nekom drugom RSUBP sustavu) omogućava da mijenjanje podataka ne utječe na čitanje podataka, tj. **mijenjanje podataka ne sprečava da se ti podaci istovremeno i čitaju**. U RSUBP sustavima koji nemaju nešto slično kao što je UNDO tablespace, mijenjanje podataka utječe na čitanje.

Udaljeni (remote) upit i DML naredba, distribuirani upit i DML naredba, distribuirana transakcija

- **Udaljeni upit**

```
SELECT * FROM neka_tablica@neki_link;
```

- **Distribuirani upit**

```
SELECT * FROM lokalna_tab,
   udaljena2@baza2, udaljena3@baza3 WHERE ...
```

- Osim udaljenog upita / distribuiranog upita, Oracle podržava i udaljene / distribuirane DML naredbe.
- **Distribuirana transakcija** je ona transakcija u toku koje se ažuriraju (tj. unose, mijenjanju ili brišu) redci iz barem dvije tablice koje se nalaze u različitim bazama podataka. Distribuirana transakcija ne mora sadržavati distribuiranu DML naredbu, niti distribuirani upit.

Oracle dvofazni commit protokol ima tri faze; 1. faza je faza pripreme (prepare phase)

- U **fazi pripreme**, globalni koordinator traži od ostalih baza, osim commit point site baze, da pređu u **pripravno stanje** (prepared state). Transakcija je od tada u **osjetljivom stanju**.
- Pripremljene baze stavljaju distribuirani lokot na sve svoje modificirane tablice. **Taj lokot sprečava čak i čitanje podataka (što Oracle baza inače ne radi), pa u slučaju problema podaci ostaju zaključani i za čitanje!**
- Inače, baza može odgovoriti na sljedeća tri načina:
  1. **Prepared**: baza javlja da je pripravna.
  2. **Read-only**: baza javlja da ona nije radila nikakav DML, pa ne treba ići u pripravno stanje.
  3. **Abort**: baza javlja da se ne može pripremiti, otključava sve svoje zaključane retke i radi lokalni ROLLBACK.

Faza potvrde (commit phase) i faza zaboravljanja (forget phase)

- Ako globalni koordinator od barem jedne baze dobije odgovor **Abort**, on šalje svim bazama ROLLBACK. Ako od svih baza dobije odgovor Prepared, **započinje fazu potvrde**:
  - Šalje commit point site bazi naredbu da izvrši COMMIT. Commit point site baza izvršava COMMIT i obavještava globalnog koordinatora.
  - Globalni koordinator i lokalni koordinatori šalju naredbe svim svojim podređenim bazama, tražeći od njih da naprave COMMIT. Ostale baze rade lokalni COMMIT.
- **U fazi zaboravljanja**, globalni koordinator kaže commit point site bazi da zaboravi transakciju, tj. da izbriše stanja o transakciji koja ona vodi kod sebe. Globalni koordinator nakon toga briše i kod sebe informacije o distribuiranoj transakciji.

In-doubt transakcija

- U slučaju da se u bilo kojoj fazi desi greška (preciznije, nakon što transakcija postane "osjetljiva", tj. nakon što barem jedna baza završi fazu pripreme), distribuirana transakcija postaje in-doubt. Greške mogu biti:
  - **Padne računalo** na kojem radi Oracle baza.
  - **Prekine se veza** između dvije ili više baza koje sudjeluju u distribuiranoj transakciji.
  - Desi se neki **softverski problem**.
- **RECO proces baze** najčešće **automatski rješava in-doubt** transakciju nakon što se podigne računalo, uspostavi veza, odnosno riješi softverski problem. No, dok RECO proces ne riješi problem, ostaju zaključani podaci modificiranih tablica, i to (kako je već rečeno) ne samo za pisanje, već i za čitanje. **Ponekad problem moramo riješiti vlastoručno.**

ANSI / ISO SQL
Transaction Isolation Levels

- Standardi definiraju četiri razine izoliranosti transakcije
- **Dirty read** - sesija čita ono što druga još nije commitirala.
- **Non-repeatable read** i **phantom read** - transakcija vidi tuđe promjene kojih nije bilo na njenom početku, ali ih je u međuvremenu druga sesija commitirala, pa ih sada vidi. Non-repeatable read se odnosi na UPDATE / DELETE, a phantom read na INSERT, pa je to u principu isto.

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

ANSI / ISO SQL
Transaction Isolation Levels

- Usporedba default rada Oracle baze i default rada drugih baza (Not Oracle = IBM DB2 i Microsoft SQL Server)

Isolation Level	Implementation	Writes Block Reads	Reads Block Writes	Deadlock-Sensitive Reads	Incorrect Query Results	Lost Updates	Lock Escalation or Limits
READ UNCOMMITTED	Not Oracle	No	No	No	Yes	Yes	Yes
READ COMMITTED	Not Oracle	Yes	No	No	Yes	Yes	Yes
READ COMMITTED	Oracle	No	No	No	No	No*	No
REPEATABLE READ	Not Oracle	Yes	Yes	Yes	No	No	Yes
SERIALIZABLE	Not Oracle	Yes	Yes	Yes	No	No	Yes
SERIALIZABLE	Oracle	No	No	No	No	No	No

* *With SELECT FOR UPDATE NOWAIT.*

ANSI / ISO SQL
Transaction Isolation Levels

- Oracle baza ne podržava **READ UNCOMMITTED**, što je dobro. Ne podržava niti **REPEATABLE READ**, ali ta razina i tako rješava samo UPDATE / DELETE, a ne i INSERT.
- Oracle baza kao default ima READ COMMITED, a eksplicitno se može odabrati i SERIALIZABLE:
  SET TRANSACTION ISOLATION LEVEL
  READ COMMITTED / SERIALIZABLE;
- Razina **SERIALIZABLE** je teoretski idealna, ali ona smanjuje skalabilnost. Ako pokušamo mijenjati redak koji je netko drugi već mijenjao i commitirao, javlja se greška:
  ORA-08177: can't serialize access for this transaction
- Oracle ima i dodatnu razinu **READ ONLY** (transakcija može samo čitati), koja je podvarijanta od SERIALIZABLE:
  SET TRANSACTION READ ONLY;

U Oracle bazi SERIALIZABLE transakcija nije uvijek "teoretski serijabilna"

- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; **ne znači da će transakcije uvijek ostaviti efekt kao da se izvršavaju serijski (jedna za drugom), iako se izvršavaju konkurentno (paralelno ili kvazi-paralelno).**
- Ovo je relativno malo poznata činjenica, koju navodi Tom Kyte u svojoj knjizi Expert Oracle Database Architecture (1. izdanje 2005., 2. izdanje 2010.):
  As a final point, be aware that SERIALIZABLE does not mean that all transactions executed by users will behave as if they were executed one right after another in a serial fashion. It does not imply that there is some serial ordering of the transactions that will result in the same outcome. The phenomena previously described by the SQL standard do not make this happen.

U Oracle bazi SERIALIZABLE transakcija nije uvijek "teoretski serijabilna"

- *This last point is a frequently misunderstood concept, and a small demonstration will clear it up.The following table represents two sessions performing work over time.*
  The database tables A and B start out empty and are created as follows:

  ops$tkyte@ORA11GR2> create table a (x int);
  Table created.

  ops$tkyte@ORA11GR2> create table b (x int);
  Table created.

  Now we have the series of events shown in Table 7-7.

U Oracle bazi SERIALIZABLE transakcija nije uvijek "teoretski serijabilna"

Table 7-7. SERIALIZABLE Transaction Example

Time	Session 1 Executes	Session 2 Executes
T1	`Alter session set isolation_level=serializable;`	`--`
T2	`--`	`Alter session set isolation_level=serializable;`
T3	`Insert into a select count(*) from b;`	`--`
T4	`--`	`Insert into b select count(*) from a;`
T5	`Commit;`	`--`
T6	`--`	`Commit;`

U Oracle bazi SERIALIZABLE transakcija nije uvijek "teoretski serijabilna"

- *Now, when this is all said and done, tables A and B will each have a row with the value 0 in it. If there were some serial ordering of the transactions, we could not possibly have both tables containing the value 0 in them. If session 1 executed in its entirety before session 2, then table B would have a row with the value 1 in it. If session 2 ... As executed here, however, both tables will have rows with a value of 0. They just executed as if they were the only transaction in the database at that point in time. No matter how many times session 1 queries table B and no matter the committed state of session 2, the count will be the count that was committed in the database at time T1. Likewise, no matter how many times session 2 queries table A, the count will be the same as it was at time T2.*

Serijabilnost naredbe i izuzeci

- Kako smo vidjeli, Oracle transakcija standardno nije serijabilna, jer je default ISOLATION LEVEL READ COMMITED i, ako želimo da transakcija bude serijabilna, moramo eksplicitno navesti SET TRANSACTION ISOLATION LEVEL SERIALIZABLE.

- Za razliku od transakcije, Oracle SELECT i DML naredbe standardno JESU serijabilne, tj. postoji statement-level read consistency. To znači da naredba ne vidi ništa što je nastalo nakon njenog početka, čak i kad je COMMIT-irano.

- **Međutim, postoje izuzeci!**

Serijabilnost naredbe i izuzeci
1. slučaj

- Prvi slučaj neserijabilnosti naredbe javlja se kada **u SELECT naredbi koristimo PL/SQL funkciju koja čita tablice iz baze** (što je u praksi često, jer želimo modularni kod; zanemarimo dodatni problem s performansama).
- Pripremimo dvije tablice (DEPT1 i EMP1) za demonstraciju:

```
Name          Null?    Type
------------- -------- ------------------
DEPTNO        NUMBER(2)
DNAME         NOT NULL VARCHAR2(20)

Name          Null?    Type
------------- -------- ------------------
EMPNO         NOT NULL NUMBER(4)
ENAME         NOT NULL VARCHAR2(20)
SALARY        NOT NULL NUMBER(8,2)
```

Serijabilnost naredbe i izuzeci
1. slučaj

❖ Pretpostavimo da tablice sadrže sljedeće podatke:

```
select * from dept1 order by deptno;
    DEPTNO DNAME
---------- --------------------
         1 D1
         2 D2
         3 D3

select * from emp1 order by empno;
     EMPNO ENAME                          SALARY
---------- -------------------- ----------
         1 A                             10000
         2 B                             20000
         3 C                             30000
```

Serijabilnost naredbe i izuzeci
1. slučaj

- Kreirajmo PL/SQL funkciju koja čita tablicu iz baze (funkcija ne radi ništa pametno, služi samo za demonstraciju):

```
create or replace function emp1_count return number is
  l_num number(8);
begin
  sys.dbms_lock.sleep(10); -- waits approx. 10 seconds
  select count(*) into l_num from emp1;
  return l_num;
end;
```

Serijabilnost naredbe i izuzeci
1. slučaj

```
-- 1.sesija - trajanje je otprilike 3 * 10 = 30 s
select deptno, dname, emp1_count
  from dept1 order by deptno;

-- 2.sesija - commit oko 15 s nakon starta 1.transakcije
insert into emp1(empno,ename,salary) values(4,'D',40000);
commit;

-- 1.sesija - SELECT pokazuje 3 i 4, umjesto samo 3!!!
DEPTNO DNAME                          EMP1_COUNT
---------- ---------------------- ----------
         1 D1                                3
         2 D2                                4
         3 D3                                4
```

Serijabilnost naredbe i izuzeci
2. slučaj

- Drugi slučaj neserijabilnosti naredbe javlja se kada **imamo DML naredbe i okidače (trigger) baze koji (direktno ili indirektno) čitaju tablice iz baze.**
- Tablice su iste kao prije, okidač je ovakav:

```
create or replace trigger aur_dept1
  after update on dept1
  for each row
declare
  l_num number(8);
begin
  sys.dbms_lock.sleep(20);
  select count(*) into l_num from emp1;
  dbms_output.put_line ('EMPs: ' || l_num);
end;
```

Serijabilnost naredbe i izuzeci
2. slučaj

```
-- 1.sesija
update dept1 set dname = dname where deptno = 1;

-- 2.sesija - radi commit prije nego istekne 20 sekundi
insert into emp1(empno,ename,salary) values(5,'E',50000);
commit;

-- 1.sesija - vidi 5, umjesto 4 !!!
EMPs: 5
1 row updated.
commit;
```

Serijabilnost naredbe i izuzeci - "popravak"

- **Možemo eliminirati prethodna dva izuzetka, korištenjem SET TRANSACTION ISOLATION LEVEL SERIALIZABLE.** Tada je transakcija serijabilna, pa je i naredba sigurno serijabilna.

- **Postoji i druga varijanta - možemo koristiti:**

  DBMS_Flashback.Enable_At_System_Change_Number

  DBMS_Flashback.Get_System_Change_Number

  DBMS_Flashback.Disable:

Serijabilnost naredbe i izuzeci - "popravak"

```
-- 1.sesija
begin
 DBMS_Flashback.Enable_At_System_Change_Number(
  DBMS_Flashback.Get_System_Change_Number());
end;

select deptno, dname, emp1_count
  from dept1
 order by deptno;

-- 2.sesija
insert into emp1 (empno, ename, salary)
  values (6, 'F', 60000);
commit;
```

Serijabilnost naredbe i izuzeci - "popravak"

```
-- 1.sesija - sada vidi ispravno - samo 5, ne 5 i 6 !!!

   DEPTNO DNAME                          EMP1_COUNT
--------- ------------------------------ ----------
         1 D1                                      5
         2 D2                                      5
         3 D3                                      5

begin
  DBMS_Flashback.Disable();
end;
```

I za kraj, slučaj neserijabilnosti kod distribuirane transakcije

- Administrator's Guide 11.2
  (35 Managing Read Consistency, str. 35-19)
- **An important restriction exists in the Oracle Database implementation of distributed read consistency.**
  The problem arises because each system has its own SCN *(System Change Number o.a.)*, which you can view as the database internal timestamp. The Oracle Database server uses the SCN to decide which version of data is returned from a query.
- The SCNs in a distributed transaction are synchronized at the end of each remote SQL statement and at the start and end of each transaction. Between two nodes that have heavy traffic and especially distributed updates, the synchronization is frequent.

I za kraj, slučaj neserijabilnosti kod distribuirane transakcije

- **Nevertheless, no practical way exists to keep SCNs in a distributed system absolutely synchronized:** a window always exists in which one node may have an SCN that is somewhat in the past with respect to the SCN of another node.
- Because of the SCN gap, you can execute a query that uses a slightly old snapshot, so that the most recent changes to the remote database are not seen. **In accordance with read consistency, a query can therefore retrieve consistent, but out-of-date data.** Note that all data retrieved by the query will be from the old SCN, so that if a locally executed update transaction updates two tables at a remote node, then data selected from both tables in the next remote access contain data prior to the update.

I za kraj, slučaj neserijabilnosti kod distribuirane transakcije

- **One consequence of the SCN gap is that two consecutive SELECT statements can retrieve different data even though no DML has been executed between the two statements.**

- For example, you can issue an update statement and then commit the update on the remote database. When you issue a SELECT statement on a view based on this remote table, the view does not show the update to the row. The next time that you issue the SELECT statement, the update is present.

I za kraj, slučaj neserijabilnosti kod distribuirane transakcije

- **You can use the following techniques to ensure that the SCNs of the two machines are synchronized just before a query:**

1. Because SCNs are synchronized at the end of a remote query, precede each remote query with a dummy remote query to the same site, for example, **SELECT * FROM DUAL@REMOTE.**

2. Because SCNs are synchronized at the start of every remote transaction, **commit or roll back the current transaction before issuing the remote query.**

Zaključak

- Sesija Oracle baze podataka ne vidi promjene koje je napravila druga sesija, dok druga sesija ne napravi COMMIT (ili ROLLBACK).
- Oracle baza kao default za Transaction Isolation Level ima READ COMMITED. Eksplicitno se može odabrati i SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- Razina SERIALIZABLE je teoretski idealna, ali ona smanjuje skalabilnost.
- Osim toga, u Oracle bazi SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ne znači da će transakcije uvijek ostaviti efekt kao da se izvršavaju serijski (jedna za drugom), iako se izvršavaju konkurentno (paralelno ili kvazi-paralelno).

Zaključak

- Za razliku od Oracle transakcije, Oracle naredba je (default) serijabilna, tj. postoji statement-level read consistency. To znači da naredba ne vidi ništa što je nastalo nakon njenog početka, čak i kad je COMMIT-irano.

- Međutim, postoje izuzeci:

- kada SQL naredba koristi PL/SQL funkciju koja čita bazu
- kada SQL naredba pokreće triggere baze koji čitaju ili mijenjaju bazu
- kod remote naredbe.

- **Ako imamo ovakve situacije, moramo voditi brigu o eventualnim posljedicama na konzistentnost transakcije.**

Literatura (dio)

- Berenson H. i drugi (1995): A Critique of ANSI SQL Isolation Levels, Microsoft Research Advanced Technology Division, Technical Report MSR-TR-95-51
- Bernstein, P.A., Newcomer, E. (2009): Principles of transaction processing (2. izdanje), Elsevier / Morgan Kaufmann Publishers
- Date, C.J. (2004): An introduction to Database Systems (8.izdanje), Addison-Wesley
- Kyte, T. (2009): Expert Oracle Database Architecture, Apress
- Oracle Database Administrator's Guide 11g Release 2, E17120-05, August 2010
- Oracle Database Development Guide 12c Release 1, E17620-11, June 2013