TESTIRANJE KONKURENTNIH TRANSAKCIJA

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

Neki izabrani (stručni) radovi

- **HrOUG 2015a: Povratak u Prolog (ili Mrav i med na valjku)**
- HrOUG 2015b: Kada Oracle naredba nije serijabilna?
- HrOUG 2014: Nasljeđivanje je dobro, naročito višestruko - Eiffel, C++, Scala, Java 8
- HrOUG 2013: Transakcije i Oracle - baza, Forms, ADF
- HrOUG 2012: Ima neka loša veza (priča o in-doubt distribuiranim transakcijama)
- HrOUG 2011: Kriptografija u Oracle bazi
- HrOUG 2010: Biometrijski sustavi - greške i ranjivosti
- HrOUG 2009: Kako spriječiti pojavu petlje u hijerarhijskoj strukturi podataka

Neki izabrani (stručni) radovi

- CASE 2018a: Primjena statistike u Oracle bazi podataka
- CASE 2018b: GDPR i baze podataka – sigurnost Oracle DBMS-a
- CASE 2017: Paralelno programiranje u Javi
- CASE 2016: Java JDBC Cached Row Set
- CASE 2015: Višestruko nasljeđivanje – san ili Java 8?
- CASE 2014: Trebaju li nam distribuirane baze u vrijeme oblaka?
- CASE 2013: Što poslije Pascala? Pa … Scala!
- CASE 2012: Utjecaj razvoja mikroprocesora na programiranje
- JavaCro 2016: Java paralelizacija 2
- JavaCro 2015: Java paralelizacija 1
- JavaCro 2014: Da li postoji samo jedna "ispravna" arhitektura web poslovnih aplikacija

Uvod

- Često baza podataka sadrži ne samo podatke, već i pakete / triggere, koji na deklarativni način osiguravaju integritet podataka.
- Rad tih paketa / triggera treba testirati, i to u višekorisničkom radu.
- Za testiranje rada npr. sto konkurentnih transakcija, nećemo koristiti 100 testera (ljudi), već odgovarajuće alate za testiranje.
- Možemo koristiti specijalne alate za tu namjenu, ili možemo sami napraviti testove na relativno jednostavan način.
- U radu će se prikazati kako testirati jedan primjer poslovnog pravila na bazi (koji smo prikazali na HROUG-u 2009. godine: "Kako spriječiti pojavu petlje u hijerarhijskoj strukturi podataka")
- To ćemo raditi pomoću Java Executora ili (za one koji ne znaju raditi s Javom) pomoću DBMS_SCHEDULER paketa.

Teme

- O SQL hijerarhijskim upitima – Oracle i ANSI varijanta; Connor McDonald: Old Dog, New Tricks 1 & 2 (Oracle Magazine January/February 2018 & September/October 2018).

- Prezentacija sa HrOUG 2009. "Kako spriječiti pojavu petlje u hijerarhijskoj strukturi podataka" (programski kod objavljen 10.08.2003 na otn.oracle.com/oramag/code kao PL/SQL tip tjedna).

- Testiranje konkurentnih transakcija pomoću Java executora.

- Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER paketa.

SQL hijerarhijski upit – Oracle varijanta (postoji od početka, nadograđivana je)

```
-- varijanta koja puca kod petlje
SELECT empno, ename, mgr, LEVEL
  FROM emp
 START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr;

-- varijanta koja NE puca kod petlje
SELECT empno, ename, mgr, LEVEL,
       CONNECT_BY_ISCYCLE iscycle
  FROM emp
 START WITH mgr IS NULL
CONNECT BY NOCYCLE PRIOR empno = mgr;
```

SQL hijerarhijski upit – ANSI varijanta

- ANSI standard za hijerarhijski upit koristi tzv. **recursive common table expressions** (recursive CTE).
- Oracle baza podržava recursive CTE od 11.2 – od tada WITH klauzula (u SELECT naredbi) može biti rekurzivna.

```
WITH each_level (empno, ename, mgr, rlevel) AS
  (SELECT empno, ename, mgr, 1 rlevel -- kao START WITH
     FROM emp
    WHERE mgr IS NULL
   UNION ALL -- kao CONNECT BY
   SELECT emp.empno, emp.ename, emp.mgr, rlevel + 1
     FROM emp, each_level
    WHERE emp.mgr = each_level.empno
  )
SELECT * FROM each_level;
```

SQL hijerarhijski upit – ANSI varijanta

- ANSI standard za hijerarhijski upit – varijanta koja ne puca kod petlje:

```
WITH each_level (empno, ename, mgr, rlevel) AS
  (SELECT empno, ename, mgr, 1 rlevel -- kao START WITH
     FROM emp
    WHERE mgr IS NULL
   UNION ALL -- kao CONNECT BY
   SELECT emp.empno, emp.ename, emp.mgr, rlevel + 1
     FROM emp, each_level
    WHERE emp.mgr = each_level.empno
  )
 CYCLE mgr SET iscycle TO 'y' DEFAULT 'n'
SELECT * FROM each_level; -- prikazuje i stupac iscycle
```

Primjer sa HrOUG 2009 (početak prezentacije sa HrOUG 2009)

- Često želimo spriječiti pojavu (zatvorene) petlje u podacima koji imaju višestruku stablastu strukturu (gdje svaki čvor stabla može imati više čvorova-djece i najviše jedan čvor-roditelj) ili jednostruku stablastu strukturu (gdje točno jedan čvor nema roditelja, a svi ostali čvorovi imaju točno jednog roditelja).
- Npr. u poznatoj Oracle tablici EMP želimo spriječiti da jedan djelatnik bude šef drugom djelatniku, a da istovremeno taj drugi bude (direktno ili indirektno) šef prvome.
- Želimo rješenje koje će raditi i **u višekorisničkom radu** i to rješenje koje će biti **u potpunosti na strani baze**, tj. rješenje koje ne traži "suradnju" klijenta (ili aplikacijskog servera) s bazom podataka.

Višeslojna arhitektura IS-a i poslovna pravila

- Krajem 70-tih godina pojavili su se opisi slojevite arhitekture informacijskih sustava, tzv. troslojne arhitekture (three-tier architecture); danas "tier" češće označava fizički čvor (node), dok se logički sloj obično naziva "layer“; troslojna arhitektura poprimila je veliku popularnost tek 90-tih godina, zahvaljujući promociji koju je napravila Gartner grupa.
- Izvorni opis navodio je 3 sloja: sloj korisničkog sučelja (User Interface), sloj aplikacijske logike (Application Logic) i podatkovni sloj (Storage); tijekom vremena se broj slojeva povećavao; npr. izvorna J2EE specifikacija navodi 4 sloja.
- U svakom slučaju, poslovna pravila (business rules) čine značajan dio aplikacijske logike (poslovne logike).

Definicija poslovnih pravila

- Definicija (i klasifikacija) iz Oracle CDM (Custom Development Method) metodike:

  "Poslovna pravila su ograničenja koja se primjenjuju na stanje sustava ili na promjenu stanja sustava (Constraint Rules), autorizacijska pravila (Authorization Rules), ili akcije koje se automatski pokreću nakon promjene stanja sustava (Change Event Rules)".
- Poslovno pravilo čije rješavanje ovdje prikazujemo spada po Oracle klasifikaciji u tip Constraint Rules, podtip Entity Rules i vrstu Other Entity Rules.

Zašto rješavati poslovna pravila u bazi, iako to nije lako

- Od početaka razvoja relacijskih sustava proteklo je oko 45 godina, ali područje koje je (nažalost) do sada od proizvođača RSUBP sustava relativno zanemarivano jesu poslovna pravila, drugačije rečeno - integritetna ograničenja u bazi.
- Proizvođači RSUBP-a su ugradili određena deklarativna pravila, npr. realizaciju primarnog ključa (PK), jedinstvenog ključa (UK), vanjskog ključa (FK) i check constraints-a (CK), ali deklarativna provjera složenijih pravila je izostala.
- Nepodržavanje poslovnih pravila u bazi rezultira time da je integritet podataka u bazi ovisan o aplikaciji; no, jedna aplikacija se može pridržavati poslovnih pravila, a da pritom druga aplikacija to ne radi, najčešće nenamjerno, ali može biti i zlonamjerno.

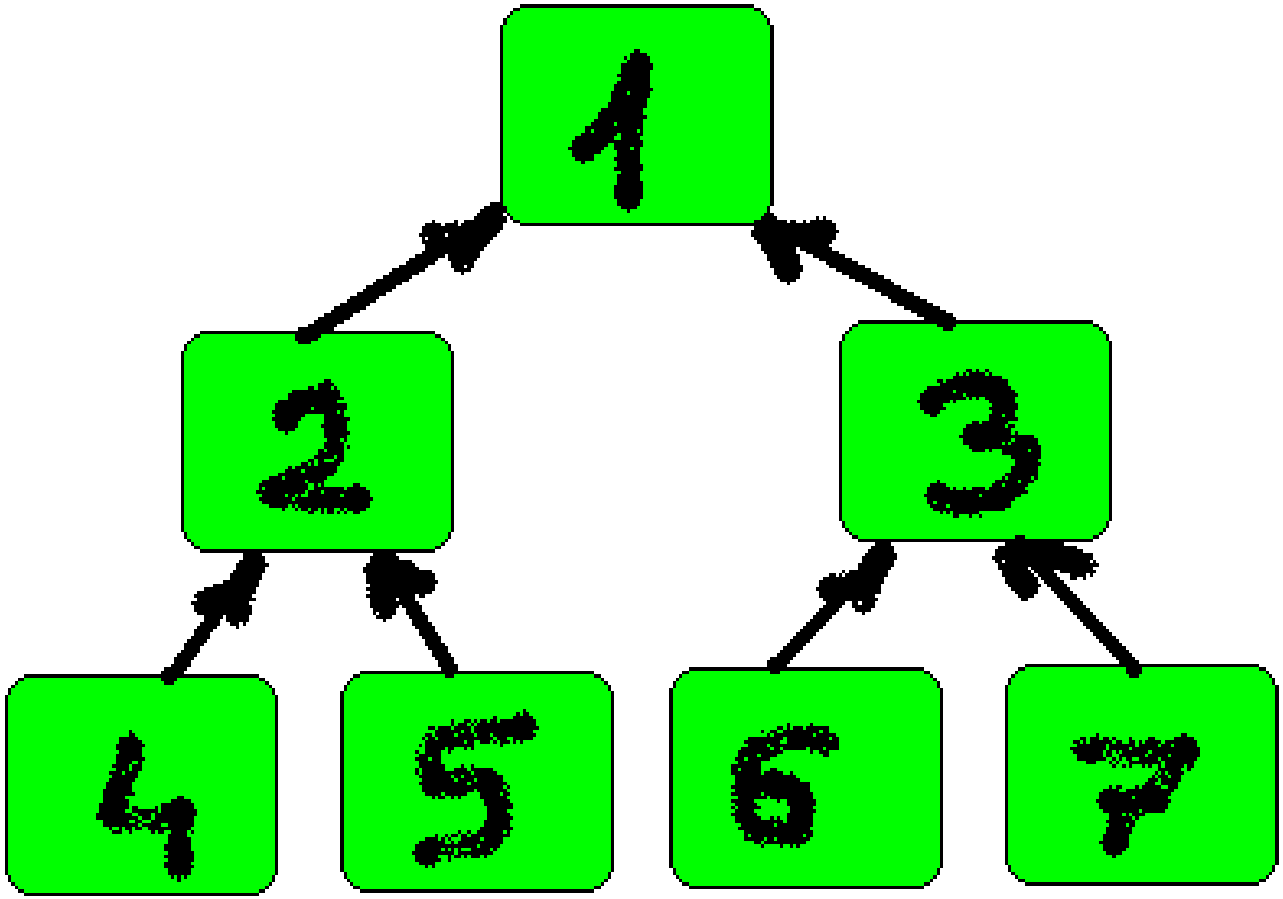
Prikaz tablice EMP sa testnim podacima

- Najjednostavniji opis tablice EMP:

```
CREATE TABLE emp (
   empno NUMBER (4),
   ename VARCHAR2 (20),
   mgr   NUMBER (4),
   CONSTRAINT emp_uk PRIMARY KEY (empno) USING INDEX,
   CONSTRAINT emp_mgr_fk FOREIGN KEY (mgr)
     REFERENCES emp (empno)
);
CREATE INDEX emp_mgr_fk_i ON emp (mgr);
```

- Napunit ćemo tablicu "emp" s 7 redaka; djelatnik s brojem 1 bit će "glavni šef", djelatnici s brojevima 2 i 3 bit će "šefovi" (podređeni "glavnom šefu"), djelatnici 4 i 5, odnosno 6 i 7, bit će podređeni "šefu 2", odnosno "šefu 3".

Grafički prikaz početnih podataka u tablici EMP



Mala digresija: kako je nekadašnji Homo sapiens gledao na stablo

Kako današnji Homo sapiens (podvrsta Homo informaticus) gleda na stablo

Sprečavanje petlje u jednokorisničkom radu

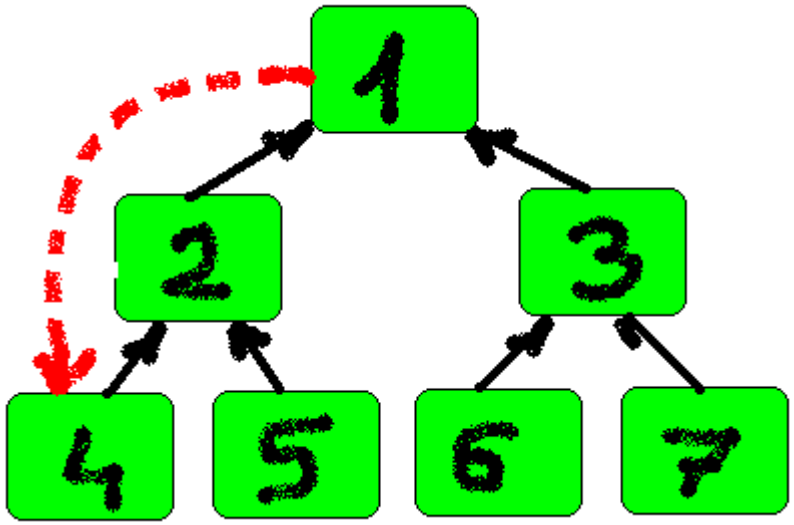
- Rješenje u jednokorisničkom radu bilo bi vrlo jednostavno kad ne bi dolazilo do mutirajućih tablica (mutating tables).
- Mutirajuća tablica je ona tablica koja se trenutačno modificira pomoću DML naredbi, ili ona tablica koja bi trebala biti ažurirana zbog efekta DELETE CASCADE.
- Oracle ne dozvoljava da se mutirajuće tablice čitaju (niti ažuriraju) u "row" okidačima, jer bismo kao rezultat čitanja mogli dobiti neku neočekivanu vrijednost.
- Međutim, čitanje se može raditi u "statement" okidačima; rješenje problema mutirajućih tablica jeste da se u "row" okidaču zapamti, npr. u PL/SQL memorijsku tablicu, koji su redovi ažurirani, a onda se u "after statement" okidaču čita PL/SQL tablica i radi se provjera poslovnih pravila nad redovima koji su u njoj zapamćeni.

Glavna procedura "test“ (ovu verziju ćemo nadograđivati, zato nema SELECT...CONNECT BY)

```
PROCEDURE test IS ...
BEGIN
  FOR i IN 1..m_rows LOOP
    l_empno := m_plsql_tab (i).empno;
    l_mgr   := m_plsql_tab (i).mgr;
    WHILE l_mgr IS NOT NULL LOOP
      SELECT mgr INTO l_mgr
        FROM emp
       WHERE empno = l_mgr;
      IF l_mgr = l_empno THEN
        RAISE_APPLICATION_ERROR (-20003, 'Petlja!');
      END IF;
    END LOOP; -- WHILE
  END LOOP; -- FOR
END;
```

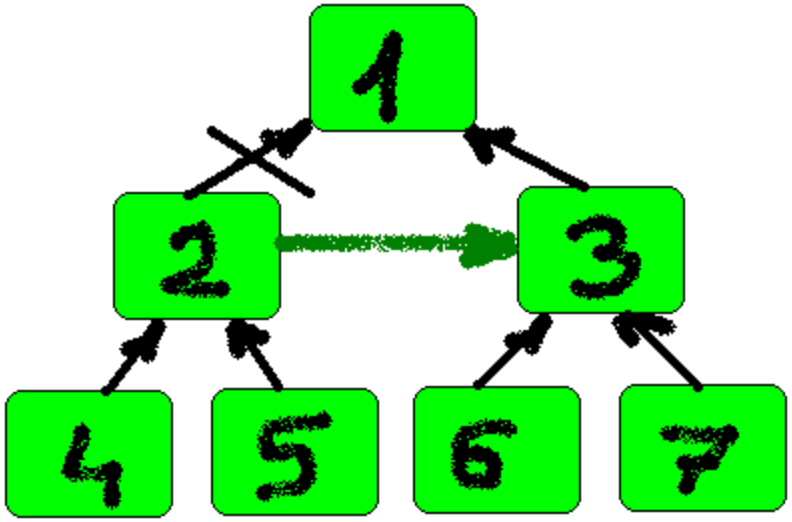
Test rješenja u jednokorisničkom radu: radi dobro

```
UPDATE emp SET mgr = 4 WHERE empno = 1;
ERROR at line 1: ORA-20003: Petlja
```



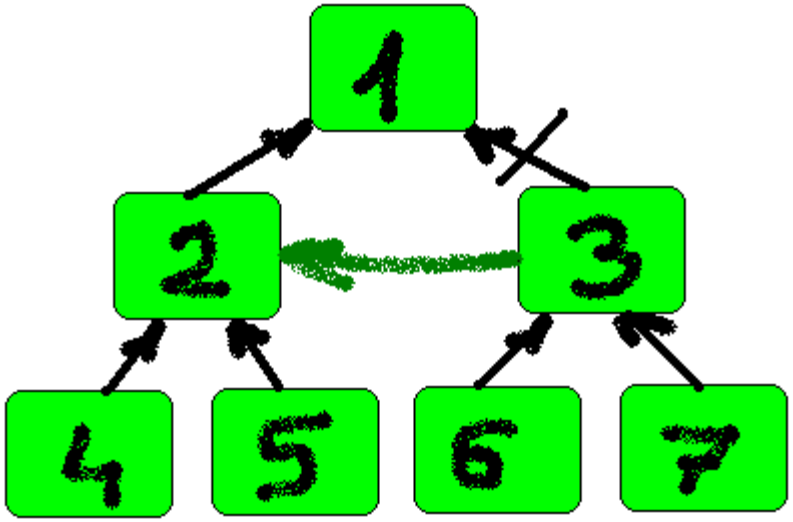
Test rješenja u višekorisničkom radu: 1. sesija uspijeva

```
UPDATE emp SET mgr = 3 WHERE empno = 2;
```



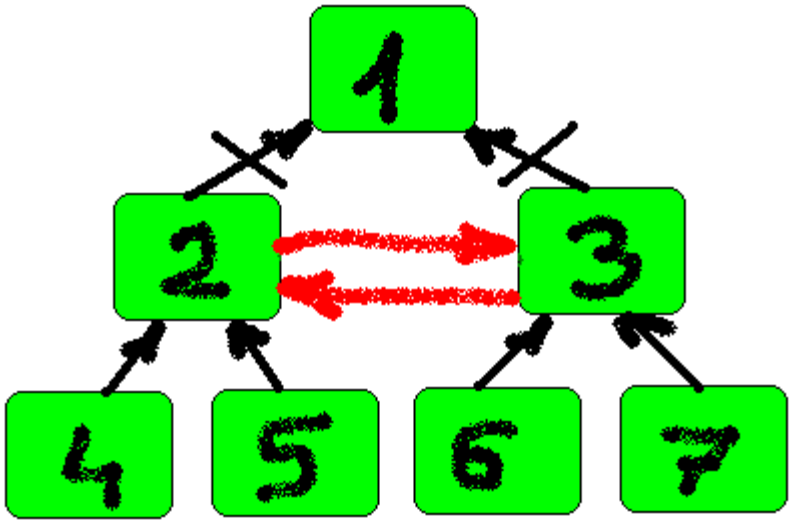
Test rješenja u višekorisničkom radu: i 2. sesija uspijeva (napomena: 1. nije dala COMMIT)

```
UPDATE emp SET mgr = 2 WHERE empno = 3;
```



Test rješenja u višekorisničkom radu: ne radi dobro, nastala je petlja

- Nakon što obje sesije daju COMMIT, u bazi ostaje petlja:



Jedno jednostavno rješenje u višekorisničkom radu

- Postoji vrlo jednostavno rješenje.
- Na početku svaka sesija zaključa cijelu tablicu EMP, pa je otključa na kraju transakcije; na taj način druge sesije ne mogu raditi dok prva ne završi, pa ne može doći do petlje.
- Međutim, takvo je rješenje ponekad neprihvatljivo, jer poništava mogućnost višekorisničkog ažuriranja.
- Takvo rješenje može biti dobro samo ako vrlo mali broj korisnika ažurira tablicu EMP ili/i ako je vjerojatnost istovremenog rada mala.
- Dakle, pokušajmo naći neko drugo rješenje.

Pokušaj sprečavanja petlje pomoću autonomne transakcije

- Glavna ideja je da, dok provjeravamo da li je došlo do petlje, gledamo da li je tekući redak (koji provjeravamo) zaključan; ako je, pretpostavljamo da bi moglo doći do petlje.
- Kako provjeriti da li je redak zaključan; ako koristimo SELECT FOR UPDATE, zaključat ćemo redak sve do kraja transakcije, zato što u okidaču Oracle baze ne možemo koristiti naredbu ROLLBACK TO SAVEPOINT (ovo ograničenje nije mana Oracle baze, već prednost); međutim, nije dobro da cijeli lanac redaka (do vrha) ostane zaključan sve do kraja transakcije, jer to sprečava druge da rade s njima.
- Od verzije 8i Oracle baza podržava autonomne transakcije, pa možemo razmišljati da ih primijenimo; u autonomnoj transakciji možemo koristiti ROLLBACK (zapravo, takva transakcija i mora na kraju imati ROLLBACK ili COMMIT).

Nova autonomna procedura "test_lock"

```
PROCEDURE test_lock (p_mgr emp.mgr%TYPE) IS
  PRAGMA AUTONOMOUS_TRANSACTION;
  l_dummy NUMBER;
BEGIN
  SELECT 1 INTO l_dummy
    FROM emp
   WHERE empno = p_mgr FOR UPDATE NOWAIT;
  ROLLBACK; -- ako smo uspješno zaključali, otključamo
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE = -54 THEN
      RAISE_APPLICATION_ERROR (-20004, 'Moguća petlja');
    ELSE
      RAISE;
    END IF;
END;
```

Autonomnu proceduru "test_lock" pozivamo iz glavne procedure "test"

```
PROCEDURE test IS ...
BEGIN
  FOR i IN 1..m_rows LOOP ...
    WHILE l_mgr IS NOT NULL LOOP
      test_lock (l_mgr);
      SELECT mgr INTO l_mgr
        FROM emp
       WHERE empno = l_mgr;

      IF l_mgr = l_empno THEN
        RAISE_APPLICATION_ERROR (-20003, 'Petlja!');
      END IF;
    END LOOP;
  END LOOP;
END;
```

Nažalost, autonomna transakcija je previše restriktivna

- Naredbe koje su prije uzrokovale grešku sad neće uspjeti, jer će baza upozoriti da bi moglo doći do petlje.
- Nažalost, rješenje općenito ne radi dobro, zato što su autonomnoj transakciji (baš zato što je autonomna, tj. nezavisna od "glavne" transakcije) zaključani oni redovi koje je zaključala "glavna" transakcija.
- Evo primjera s dvije UPDATE naredbe **u istoj sesiji**:

```
UPDATE emp SET mgr = 2 WHERE empno = 6;
UPDATE emp SET mgr = 6 WHERE empno = 7;
ERROR at line 1: ORA-20004: Moguća petlja
```

- Iako bi bilo sasvim u redu da djelatnik 6 postane nadređen djelatniku 7, druga naredba javlja grešku zato jer autonomna procedura "test_lock" nalazi da je djelatnik 6 zaključan (njega je zaključala "glavna" transakcija, kroz prvi UPDATE).

Simulacija SAVEPOINT / ROLLBACK TO SAVEPOINT ponašanja u okidaču Oracle baze

- Kako smo već rekli, SAVEPOINT / ROLLBACK TO SAVEPOINT ne možemo koristiti u okidaču Oracle baze, jer se javlja greška:
  ORA-04092: cannot SET SAVEPOINT in a trigger
- Međutim, našli smo da **možemo simulirati S / RTS ponašanje pomoću kvazi-udaljene procedure.**
- Trik (objavljen 06.2002 na www.quest-pipelines.com kao PL/SQL tip mjeseca) je da koristimo ovu osobinu Oracle baze: **ako pozivamo udaljenu proceduru (preko database link-a) i ako se u njoj desi neobrađena greška, njeni se efekti u cijelosti poništavaju, uključujući i zaključavanje redaka.**
- Nama ne treba udaljena procedura, ali možemo proceduru "test_lock" pozivati kao kvazi-udaljenu proceduru, koristeći "lokalni" database link (link baze na sebe samu).

Sprečavanje petlje u višekorisničkom radu pomoću simulacije SAVEPOINT / ROLLBACK TO SAVEPOINT

- Prvo ćemo napraviti "lokalni" database link:

```
CREATE DATABASE LINK local_db_link
  CONNECT TO scott IDENTIFIED BY tiger
  USING 'local_alias'; -- alias na lokalnu bazu
```

- Procedura "test_lock" sad mora biti navedena u specifikaciji paketa, jer će se pozivati preko db linka:

```
CREATE OR REPLACE PACKAGE emp_closed_loop IS ...
  PROCEDURE test;
  PROCEDURE test_lock (p_mgr emp.mgr%TYPE);
END emp_closed_loop;
```

Promijenjena procedura "test_lock" (nije više autonomna procedura)

```
PROCEDURE test_lock (p_mgr emp.mgr%TYPE) IS
  l_dummy NUMBER;
BEGIN
  SELECT 1 INTO l_dummy
    FROM emp
   WHERE empno = p_mgr FOR UPDATE NOWAIT;
  -- nije više ROLLBACK, ali efekat je isti !!!
  RAISE_APPLICATION_ERROR (-20999, 'Simulac.ROLLBACK');
EXCEPTION
  WHEN OTHERS THEN
    IF SQLCODE = -54 THEN
      RAISE_APPLICATION_ERROR (-20004, 'Moguća petlja');
    ELSE
      RAISE;
    END IF;
END;
```

Promijenjena glavna procedura "test"

```
PROCEDURE test IS ...
BEGIN
  FOR i IN 1..m_rows LOOP ...
    WHILE l_mgr IS NOT NULL LOOP
      BEGIN
        emp_closed_loop.test_lock@local_db_link(l_mgr);
      EXCEPTION
        WHEN OTHERS THEN
          IF SQLCODE = -20999 THEN NULL; -- "ROLLBACK"
          ELSE RAISE;
          END IF;
      END;
      ...
    END LOOP; -- WHILE
  END LOOP; -- FOR
END;
```

No, rješenje nije svemoguće

- Kao i kod (pokušaja) rješenja s autonomnom transakcijom, naredbe koje bi uzrokovale petlju neće uspjeti; dodatno, rješenje će raditi dobro i u slučaju u kojem autonomna transakcija javlja očitu "lažnu uzbunu“ (u okviru iste sesije).
- No, i ovo rješenje može javiti "lažnu uzbunu", tj. javiti da bi moglo doći do petlje (iako do toga ne bi došlo), kao u sljedećem primjeru:

```
-- 1.sesija
UPDATE emp SET mgr = 6 WHERE empno = 2;
-- 2.sesija
UPDATE emp SET mgr = 5 WHERE empno = 7;
ERROR at line 1: ORA-20004: Moguća petlja
```

- **Nažalost, "lažnu uzbunu" ne možemo spriječiti, jer sesija baze ne može točno "znati" što rade druge sesije baze.**

Da li se isplati sav taj trud?

- Prikazano rješenje je strogo vezano za Oracle bazu; ako bi o njemu govorili kao o predlošku ili uzorku (pattern), mogli bismo reći da ono ne spada u arhitekturne predloške ili dizajnerske predloške (architectural pattern, design pattern), već u tzv. idiome (idioms), tj. rješenja koja su ovisna o određenom programskom jeziku.
- **Vidjeli smo da sprečavanje pojave petlje** (u podacima koji imaju stablastu strukturu) isključivo na strani (Oracle) baze, tj. bez pomoći programa na klijentu ili aplikacijskom serveru, **nije jednostavno.**
- Nekome bi to bio dovoljan razlog da odustane od realizacije poslovnih pravila na bazi i da, umjesto toga, realizaciju poslovnih pravila napravi na klijentu ili (još bolje) na aplikacijskom serveru.

Mi mislimo da se isplati! (kraj prezentacije sa HrOUG 2009)

- Naše je mišljenje da trebamo pokušati realizirati poslovna pravila na strani baze, jer na taj način osiguravamo da su podaci u bazi konzistentni, neovisno od "vanjskih" programa (na klijentu ili aplikacijskom serveru).
- Takvo mišljenje zastupamo onda kad aplikacija radi samo s jednim RDBMS sustavom (npr. Oracle); ukoliko aplikacija mora raditi s više različitih sustava, tada je vjerojatno bolje sloj poslovnih pravila implementirati na aplikacijskom serveru, **jer je teško napisati rješenja (u bazi) koja bi bila primjenljiva na različite RDBMS sustave.**
- Naime, suprotno od vjerovanja da različiti RDBMS sustavi rade vrlo slično, istina je da su među njima razlike jako velike; nije riječ samo o različitim dijalektima SQL jezika, već npr. o fundamentalnim razlikama kod rada s transakcijama.

Testiranje konkurentnih transakcija – punjenje 1000 redaka u EMP

- Tablicu EMP, ćemo napuniti sa 1000 redaka:
  - 1 korijenski redak (šifra 0)
  - njegovih 9 podređenih
  - njihovih 90 podređenih (po 10 za svakog)
  - njihovih 900 podređenih (po 10 za svakog):

```
-- 0
insert into emp (empno, ename, mgr) values (0, 'EMP 0', null);
-- 1-9
declare
  ename varchar2(20);
begin
  for j in 1..9 loop
    ename := 'EMP ' || j;
    insert into emp (empno, ename, mgr) values (j, ename, 0);
  end loop;
end;
/
```

Testiranje konkurentnih transakcija – punjenje 1000 redaka u EMP

```
-- 10-99
declare
  empno number(4);
  ename varchar2(20);
begin
  for i in 1..9 loop
    for j in 0..9 loop
      empno := i * 10 + j;
      ename := 'EMP ' || empno;
      insert into emp (empno, ename, mgr) values (empno, ename, i);
    end loop;
  end loop;
end;
/
```

Testiranje konkurentnih transakcija – punjenje 1000 redaka u EMP

```
-- 100-999
declare
  empno number(4);
  ename varchar2(20);
begin
  for i in 10..99 loop
    for j in 0..9 loop
      empno := i * 10 + j;
      ename := 'EMP ' || empno;
      insert into emp (empno, ename, mgr) values (empno, ename, i);
    end loop;
  end loop;
end;
/

commit;
```

Testiranje konkurentnih transakcija pomoću Java executora

- Java program (u nastavku) ima ulazne parametre:
  - **brDretvi**: broj Java dretvi (default je 10)
  - **cekanje**: vrijeme namjernog čekanja u pojedinoj dretvi (default je 1 sekunda)
  - **brIteracija**: broj ponavljanja testa (default je 1).
- Glavna metoda **main** poziva metodu **testiraj**, u kojoj se kreira objekt (**connectionTask**) anonimne klase (podklasa od Runnable). U (nadjačanoj) metodi **run** dvaput se poziva metoda **slucajni_broj**, koja generira slučajne emp i mgr (brojeve između 200 i 300). Na temelju toga se radi UPDATE jednog retka i COMMIT.
- Kreira se fiskni broj executora sa **newFixedThreadPool(brDretvi)** i svakom se daje njegov zadatak sa **executorService.submit(connectionTask).**

Testiranje konkurentnih transakcija pomoću Java executora

```
import java.sql.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import static java.util.concurrent.TimeUnit.MINUTES;

public class TestEmp {
  static int brDretvi = 10;
  static int cekanje = 1; // 1 sekunda
  static int brIteracija = 1;

  // ... metode u nastavku
};
```

Testiranje konkurentnih transakcija pomoću Java executora

```
public static void main(String[] args)  {
  if (args.length > 0) {
    brDretvi = Integer.parseInt(args[0]);
  };
  if (args.length > 1) {
    cekanje = Integer.parseInt(args[1]) * 1000;
  };
  if (args.length > 2) {
    brIteracija = Integer.parseInt(args[2]);
  };
  for (int i = 1; i <= brIteracija; i++) {
    testiraj();
  };
};
```

Testiranje konkurentnih transakcija pomoću Java executora

```
private static void testiraj() {
  String url = "jdbc:oracle:thin:emp/emp@localhost:1521:ORCL";
  // kreira se objekt anonimne klase, koja je podklasa od Runnable
  Runnable connectionTask = new Runnable() {
    public void run() {
      try (Connection con = DriverManager.getConnection(url);) {
        // ... kod na sljedećem slajdu
      } catch (Exception e) {System.out.println(e.toString());}
    }
  };
  try {
    ExecutorService executorService =
      Executors.newFixedThreadPool(brDretvi);
    for (int j = 1; j <= brDretvi; j++) {
      executorService.submit(connectionTask);
    }
    executorService.shutdown();
    executorService.awaitTermination(30, MINUTES);
  } catch (Exception e) {System.out.println(e.toString());}
};
```

Testiranje konkurentnih transakcija pomoću Java executora

```
// ... kod koji nije prikazan na prethodnom slajdu
con.setAutoCommit(false);
String query;
PreparedStatement stm;
// ORA-01436: CONNECT BY loop in user data
query = "select distinct 0 from emp connect by prior empno=mgr";
stm = con.prepareStatement(query);
stm.executeQuery();

int mgr   = slucajni_broj (200, 300);
int empno = slucajni_broj (200, 300);
query = "update emp set mgr = ? where empno = ?";
stm = con.prepareStatement(query);
stm.setInt(1, mgr);
stm.setInt(2, empno);
stm.execute();

Thread.sleep(cekanje);
con.commit();
System.out.println("Empno:" + empno + " Mgr:" + mgr);
```

Testiranje konkurentnih transakcija pomoću Java executora

```
private static int slucajni_broj (int min_p, int max_p) {
  /*
  Generira slučajni cijeli broj
  od min (uključujući) do max (uključujući)
  Math.random vraća vrijednost
  od 0 (uključujući) do 1 (isključujući, tj. nikad ne vraća 1).
  Zato se množi s max_p i zbraja min_p. Vraća se u cijeli broj.
  */

  int broj = min_p + (int) (Math.random() * (max_p - min_p + 1));
  return broj;
}
```

Testiranje konkurentnih transakcija pomoću Java executora

- Testiranje možemo raditi npr. ovako –
  pokreće se 100 Java dretvi (time i 100 paralelnih transakcija),
  sa vremenom čekanja od 2 sekunde u transakciji,
  u 5 iteracija testiranja:

```
java -cp "*"; TestEmp 100 2 5
```

- Pokazuje se da nesigurna varijanta paketa emp_closed_loop vrlo brzo dovodi do greške.
- Sigurna (vjerojatno) varijanta tog paketa ne dovodi do greške niti nakon puno ponavljanja. Naravno, to nije matematički dokaz da je to zaista sigurna varijanta. Kao i uvijek, testiranjem se može dokazati da program ne radi dobro, ali se ne može dokazati da program (uvijek) radi dobro.

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

- PL/SQL paket (u nastavku) ima javne procedure:
  - **pokreni**: parametar je broj jobova (default je 10)
  - **zadatak**: procedura je javna zato jer se poziva iz joba, pa mora biti navedena u specifikaciji paketa (inače bi se javilo: ORA-06576: not a valid function or procedure name).
- U proceduri pokreni kreira se i pokreće zadani broj jobova. Kod poziva **DBMS_SCHEDULER.RUN_JOB** parametar **use_current_session** postavlja se na FALSE (default je TRUE), kako bi svaki job radio u posebnoj sesiji baze.
- Procedura **zadatak** (slično ekvivalentnom dijelu koda u Java programu) postavlja upit kojim utvrđuje da li je došlo do petlje, a onda (ako nije došlo do petlje) slučajno generira šifru za mgr i empno, radi UPDATE, čeka jednu sekundu, i daje COMMIT.

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

```
-- kao SYS i sl.
GRANT CREATE JOB TO emp
/

-- kao EMP

CREATE OR REPLACE PACKAGE test_emp IS
  PROCEDURE pokreni (br_jobova_p NUMBER DEFAULT 10);
  PROCEDURE zadatak;
END;
/

CREATE OR REPLACE PACKAGE BODY test_emp IS
  PROCEDURE pokreni (br_jobova_p NUMBER DEFAULT 10) IS ...

  PROCEDURE zadatak IS ...
END;
```

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

```
PROCEDURE pokreni (br_jobova_p NUMBER DEFAULT 10) IS
BEGIN
  FOR j IN 1 .. br_jobova_p LOOP
    DBMS_SCHEDULER.CREATE_JOB (
      job_name   => 'TEST_EMP_' || j,
      job_type   => 'STORED_PROCEDURE',
      job_action => 'TEST_EMP.ZADATAK',
      enabled    => TRUE,
      auto_drop  => TRUE -- default
    );

    DBMS_SCHEDULER.RUN_JOB (
      job_name => 'TEST_EMP_' || j,
      use_current_session => FALSE
    );
  END LOOP;
END;
```

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

```
PROCEDURE zadatak IS
  mgr_1   NUMBER (4);
  empno_1 NUMBER (4);
  dummy_1 NUMBER (1);
BEGIN
   -- može se desiti ORA-01436: CONNECT BY loop in user data
   SELECT DISTINCT 0 INTO dummy_1
     FROM emp
   CONNECT BY PRIOR empno = mgr;

   mgr_1   := TRUNC (DBMS_RANDOM.VALUE (200, 301));
   empno_1 := TRUNC (DBMS_RANDOM.VALUE (200, 301));

   UPDATE emp
      SET mgr = mgr_1
    WHERE empno = empno_1;

   DBMS_LOCK.SLEEP (1);
   COMMIT;
END;
```

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

```
-- pokretanje (npr.) 50 jobova;
-- greške se ne prikazuju, jer jobovi rade u drugim sesijama!
exec test_emp.pokreni (50)

-- prikaz jobova koji (još) rade
select job_name
  from user_scheduler_jobs
 order by 1
/

-- da li se desila petlja; ako je, sljedeći upit daje
-- ORA-01436: CONNECT BY loop in user data
select distinct 0
  from emp.emp
connect by prior empno = mgr;

-- vraćanje početnih podataka (iz pomoćne tablice)
delete emp.emp;
insert into emp.emp select * from emp.emp_save;
commit;
```

Testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER-a

```
-- prikaz grešaka iz loga za jobove;
-- prvo je pokrenuto nad lošom varijantom tijela paketa emp_closed_loop
select error#, count(*)
  from USER_SCHEDULER_JOB_RUN_DETAILS
 where job_name like 'TEST_EMP_%'
 group by error#
 order by 1;
    ERROR#   COUNT(*)
---------- ----------
         0        355
      1436        143 -- greške ORA-01436
     20002          2

-- nakon (naknadnog) pokretanje nad ispravnom varijantom tijela paketa
    ERROR#   COUNT(*)
---------- ----------
         0        436
      1436        143 -- više se nije javila nova greška ORA-01436
     20002          2
     20004         19
```

Zaključak

- Prvo smo se ukratko podsjetili na dvije varijante SQL hijerarhijskih upita – Oracle varijantu (koju je Oracle baza imala od početka) i ANSI varijantu (moguća od baze 11.2).
- Zatim smo ponovili prezentaciju sa HrOUG 2009.
- Na primjeru toga prikazali smo testiranje konkurentnih transakcija pomoću Java executora.
- Zatim smo prikazali testiranje konkurentnih transakcija pomoću DBMS_SCHEDULER paketa.
- Naravno, postoje i bolji načini testiranja konkurentnih transakcija – pomoću gotovih alata namijenjenih tome, od kojih su neki (možda) i besplatni. No, nije loše, barem iz edukativnih razloga, probati testiranje (i) pomoću vlastitih, jednostavnih programa.

Literatura (dio)

- Boyarsky, J., Selikoff, S. (2015): OCP: Oracle Certified Professional Java SE 8 Programmer II Study Guide: Exam 1Z0-809, Sybex
- Kyte, T. (2009): Expert Oracle Database Architecture, Apress
- Oracle priručnik (2010): Oracle Database Administrator's Guide 11g Release 2
- Oracle priručnik (2010): Oracle Database Advanced Application Developer's Guide 11g Release 2
- Oracle priručnik (2013): Oracle Database Development Guide 12c Release 1
- Oracle priručnik (2013): Oracle Database JDBC Developer’s Guide 12c Release 1
- Sierra K., Bates B. (2015): OCA/OCP Java SE 7 Programmer I & II Study Guide, McGraw-Hill Education