KAKO SPRIJEČITI POJAVU PETLJE U HIJERARHIJSKOJ STRUKTURI PODATAKA

Zlatko Sirotić, dipl.ing.
Istra informatički inženjering d.o.o.
Pula

Tema je rješavanje određenog tipa poslovnih pravila u Oracle bazi podataka

- 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 35 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

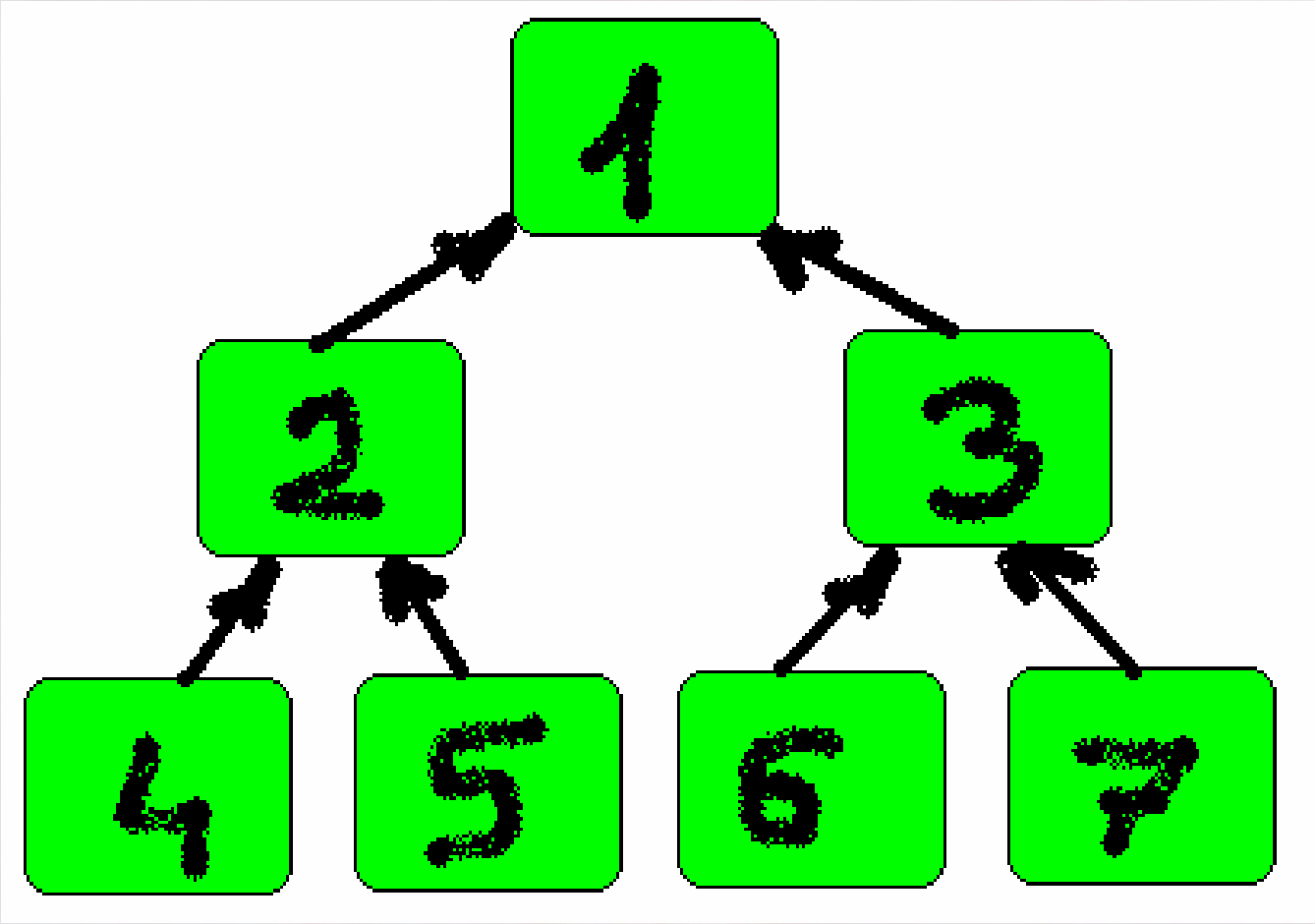
Prikaz tablice EMP s testnim podacima

- najjednostavniji opis tablice EMP:

```
CREATE OR REPLACE TABLE emp (
  empno NUMBER (4),    -- šifra
  ename VARCHAR2 (20) -- ime
  mgr   NUMBER (4)     -- šifra nadređenog
);
```

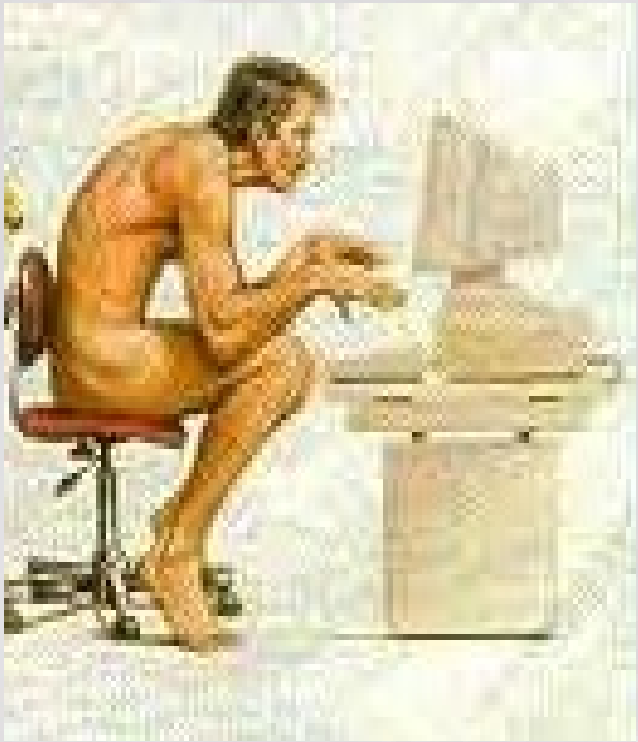
- napunit ćemo tablicu "emp" s 7 redaka; djelatnik s brojem 1 bit će "glavni šef", djelatnici s brojevima 2 i 3 bit će "šefovi" (podređeni "glavnom šefu"), djelatnici 4 i 5, odnosno 6 i 7, bit će podređeni "šefu 2", odnosno "šefu 3“

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

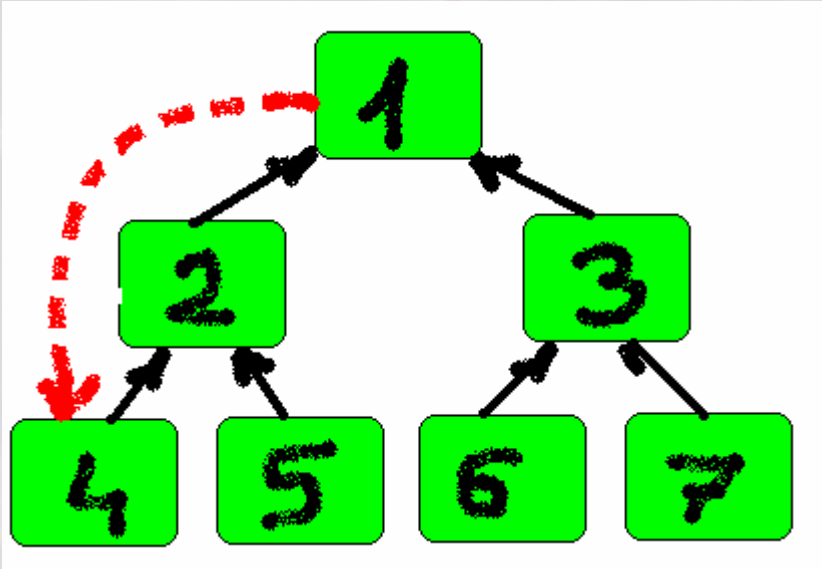
- rješenje u jednokorisničkom radu bilo bi vrlo jednostavno kad ne bi dolazilo do mutirajućih tablica (mutating tables)
- mutirajuća tablica je ona tablica koja se trenutačno modificira pomoću DML naredbi, ili ona tablica koja bi trebala biti ažurirana zbog efekta DELETE CASCADE
- Oracle ne dozvoljava 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;
```

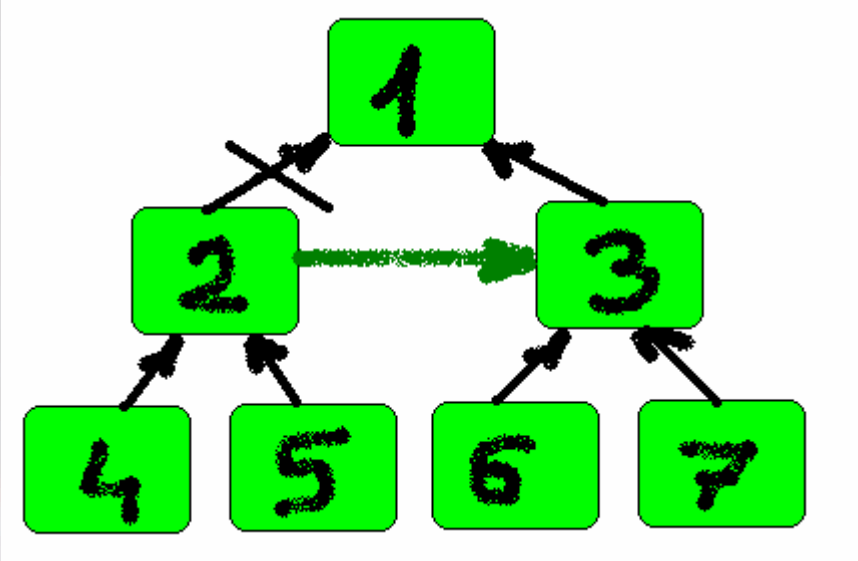
Test rješenja u jednokorisničkom radu: radi dobro

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



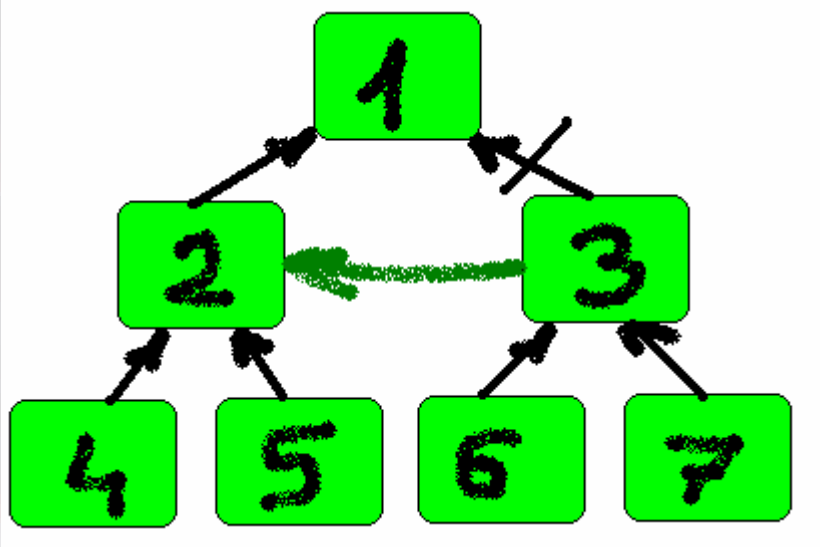
Test rješenja u višekorisničkom radu: 1. sesija uspijeva

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



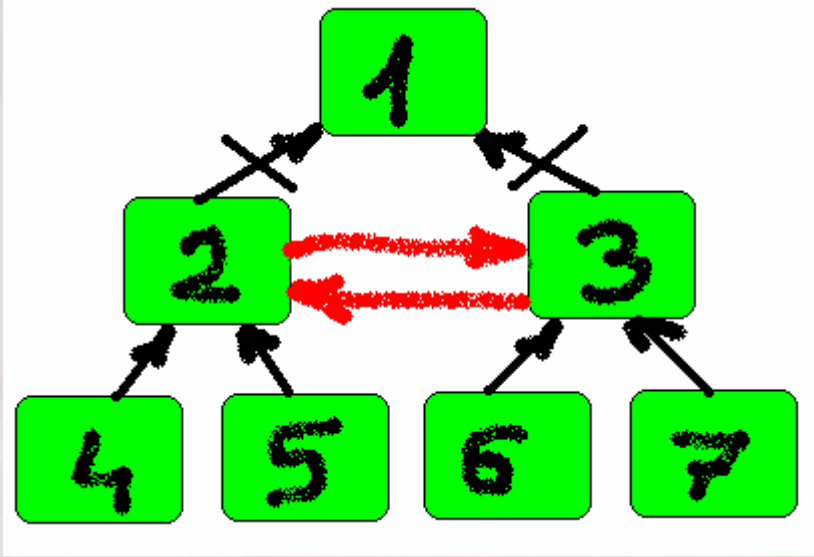
Test rješenja u višekorisničkom radu: i 2. sesija uspijeva (napomena: 1. nije dala COMMIT)

```
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 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!

- 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

Zaključak

- C.J.Date, autoritet na području relacijskih baza podataka, koji je popularizirao relacijski model (čiji je autor E.F.Codd), naglašava da su današnji RDBMS sustavi još poprilično daleko od teorijskog modela koji je postavio Codd (1969/70.)
- od tri glavne komponente relacijskog modela, komponenta integriteta podataka (Data Integrity) je realizirna vrlo manjkavo; komponenta za definiranje strukture podataka (Data Structures) i komponenta za manipulaciju podacima (Data Manipulation) su puno potpunije realizirane
- ako vjerujemo Dateovim vizijama, **možemo se nadati da će ubuduće proizvođači RDBMS sustava posvetiti veću pažnju komponenti integriteta podataka**; kad se to ostvari, nama koji radimo aplikacije nad bazama podataka bit će olakšana realizacija poslovnih pravila u bazi