Trigger

Rozdiel medzi triggrom a uloženým podprogramom:

- triggre sa spúšťajú implicitne pri modifikácii tabuľky – nezávisle na užívateľovi modifikujúcom tabuľku alebo aplikáciu, ktorá modifikuje tabuľku.
- triggre sa definujú len pre databázové tabuľky príp. pohľady
- triggre neprijímajú argumenty
- trigger sa dá spustiť len pri týchto DML príkazoch: UPDATE, INSERT a DELETE.

Použitie triggrov:

- Nepovolia neplatné dátové transakcie
- Zaisťujú komplexnú bezpečnosť
- Zaisťujú referenčnú integritu (RI) cez všetky uzly v distribuovanej databáze
- Vytvárajú strategické a komplexné aplikačné pravidlá
- Zaisťujú sledovanie (audit)
- Spravujú synchronizáciu tabuliek
- Zaznamenávajú štatistiku často modifikovaných tabuliek

Syntax

CREATE [OR REPLACE] **TRIGGER** [*schema.*] *trigger*

{ {BEFORE | AFTER } ***...pred alebo po operácii***

{DELETE | INSERT | UPDATE [OF *stlpec1* [, *stlpec2* [,…]]] }

[OR {DELETE | INSERT | UPDATE [OF *stlpec1* [, *stlpec2* [,…]]] }] […]

|

INSTEAD OF {DELETE | INSERT | UPDATE }} ***...používa sa pri pohľadoch***

ON [*schema.*] *tabulka*

[REFERENCING { OLD [AS] *stary* | NEW [AS] *novy*}]

[FOR EACH ROW]

[WHEN (*podmienka*)] ***...dodatočná podmienka***

Telo triggra

Referencing

- Referencing je možné použiť len pri triggri, ktorý je spúšťaný pre každý riadok zadanej operácie (t.j.) For each row
- Záznam NEW je možné použiť pri operácii INSERT, alebo UPDATE
- Záznam OLD je možné použiť pri operácii UPDATE, alebo DELETE
- Záznamy NEW a OLD majú vždy rovnakú štruktúru ako tabuľka, na ktorej je definovaný trigger
- Pred záznamami je vždy potrebné dávať :

  (:new.os_cislo)

For each row

...trigger

```
create or replace trigger
t_log_zp
after update on zap_predmety
for each row
begin
  insert into log_table_zp
  values (user, sysdate);
end;
```

...operácia spúšta trigger

```
SQL>update zap_predmety
  2  set zp_skrok = 2002
  3  where zp_skrok = 2000;

7 rows updated.
```

bez FOR EACH ROW

1 riadok vložený do log_table_zp

s FOR EACH ROW

7 riadkov vložených do log_table_zp

Obmedzenia pri vytváraní triggra:

- Telo môže obsahovať DML SQL príkazy, ale SELECT príkazy musia byť príkazy typu **SELECT ... INTO**, alebo sa musia nachádzať v deklaráciách kurzora.
- **DDL deklarácie** nie sú povolené v tele triggra.
- Nie sú povolené žiadne príkazy riadiacie transakciu (**COMMIT, SAVEPOINT**, alebo **ROLLBACK** príkaz).
- Vo volanom uloženom podprograme taktiež nie sú povolené žiadne príkazy riadiacich transakcií, pretože sa vykonávajú v rozsahu daného triggra .
- Premenné typu LONG a LONG RAW nemôžu byť použité ako :OLD alebo :NEW hodnoty.

Zapnutie a vypnutie triggra:

Zapnutie a vypnutie vykonávania ***konkrétneho triggra***:

ALTER TRIGGER [*schema*.] *trigger*
{ENABLE | DISABLE};

Zapnutie a vypnutie vykonávania ***všetkých triggrov*** pre určitú tabuľku:

ALTER TABLE [*schema*.] *tabuľka*
{ENABLE | DISABLE} ALL TRIGGERS;

Zrušenie triggra:

SELECT trigger_name FROM user_triggers;

DROP TRIGGER [*schema*.] *trigger* ;

DEFINOVANIE „AUTOINCREMENTU STĹPCA“

Vytvorenie SEQUENCE

```
CREATE SEQUENCE SEKV_ID
INCREMENT BY 1 START WITH 1;
```

Vytvorenie tabuľky

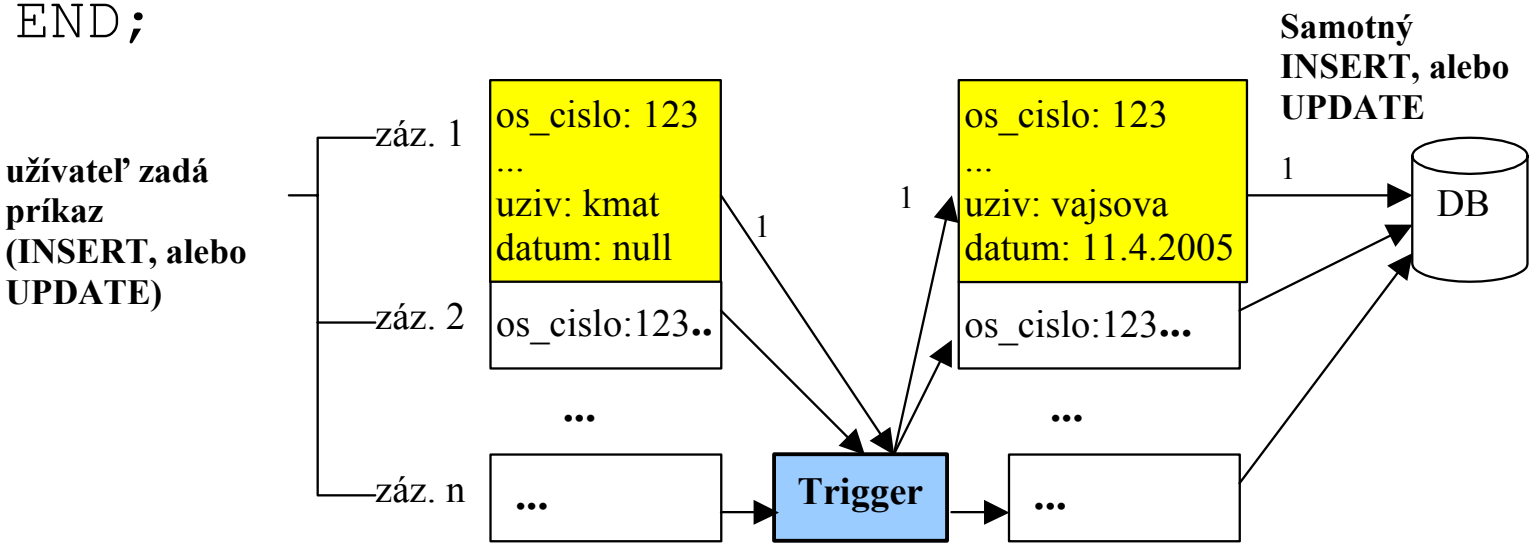
```
CREATE TABLE tab_seq
(id integer, popis varchar2(10));

ALTER TABLE tab_seq
ADD PRIMARY KEY (id);
```

DEFINOVANIE TRIGGRA PRE „AUTOINCREMENT STĹPCA“

```
CREATE OR REPLACE TRIGGER
  tab_seq_ins
   BEFORE INSERT ON tab_seq
   REFERENCING NEW AS novy
   FOR EACH ROW
BEGIN
   SELECT sekv_id.NEXTVAL INTO
   :novy.id FROM dual;
END;
```

Monitorovanie zmien tabuľky

```
create or replace trigger zap_predmety_mon
   BEFORE INSERT OR UPDATE ON zap_predmety
   REFERENCING new as novy
   FOR EACH ROW
BEGIN
   select user, sysdate into :novy.uziv,
   :novy.datum_zm from dual;
END;
```



Logovanie do pomocnej tabuľky

Príprava logovacej tabuľky

```
CREATE TABLE log_table_zp (
  user_name varchar2(20),
  datum date);
```

Logovanie do pomocnej tabuľky

```
create or replace trigger t_log_zp
before    update on zap_predmety
begin
   insert into log_table_zp
   values (user, sysdate);
end;
```

LOGOVANIE KTO A KOHO MENIL, VKLADAL, VYMAZÁVAL

Ďalší variant logu:

```
CREATE TABLE log_table (
   user_name varchar2(20),
   datum date,
   operacia char(1),
   table_name varchar2(20),
   os_cislo number(38));
```

Trigger
LOGOVANIE KTO A KOHO **VKLADAL**

```
CREATE OR REPLACE TRIGGER st_ins
  BEFORE INSERT ON student
  REFERENCING NEW AS novy
  FOR EACH ROW
  BEGIN
    INSERT INTO log_table
      (user_name, datum, operacia,
       table_name, os_cislo)
    SELECT USER, SYSDATE, 'I',
      'student',:novy.st_os_cislo
    FROM dual;
END;
```

Trigger
LOGOVANIE KTO A KOHO **VYMAZÁVAL**

```
CREATE OR REPLACE TRIGGER st_del
  BEFORE DELETE ON student
  REFERENCING OLD AS stary
  FOR EACH ROW
  BEGIN
    INSERT INTO log_table
      (user_name, datum, operacia,
        table_name, os_cislo)
    SELECT USER, SYSDATE, 'D',
       'student',:stary.st_os_cislo
    FROM dual;
END;
```

Zabránenie niektorým užívateľom meniť hodnoty primárneho kľúča

```
create or replace trigger st_oc
  before update of st_os_cislo on student
  for each row
  when (user not in 'vajsova')
begin
  --vyvolaj vlastnu vynimku
  raise_application_error(-20000,
  'ERROR - NEMOZES MENIT OS_CISLO');
end;
```

Definovanie Kaskády pre delete pomocou triggru

```
create or replace trigger st_del_cascade
before delete on student
for each row
declare
  pocet integer;          ...deklarácia premennej
begin
  select count(*) into pocet from zap_predmety
  where zp_st_os_cislo = :old.st_os_cislo;

  dbms_output.put_line ('Bolo vymazanych
  '||pocet||' zaznamov zo zap_predmety');

  delete from zap_predmety
  where zp_st_os_cislo = :old.st_os_cislo;
end;
```

ZÁLOHA VYMAZÁVANÝCH RIADKOV

Vytvoríme tabuľku, kam budeme odkladať vymazávané riadky.

```
CREATE TABLE zp_del
AS
SELECT * FROM zap_predmety
WHERE zp_st_os_cislo IS NULL;
```

ZÁLOHA VYMAZÁVANÝCH RIADKOV

```
CREATE OR REPLACE TRIGGER zp_del
  BEFORE DELETE ON zap_predmety
  REFERENCING OLD AS old
  FOR EACH ROW
BEGIN
  INSERT INTO ZP_DEL (ZP_ZAPOCET,ZP_SKROK,ZP_TERMIN,
    ZP_KREDITY,ZP_VYSLEDOK, VYSLEDOK, ZP_PREDNASAJUCI,
    ZP_DATUM_SK,ZP_ST_OS_CISLO ,ZP_PR_CIS_PREDM,
    ZP_UC_OS_CIS,ZP_UZIV , ZP_DATUM_ZM  )
  VALUES (:OLD.ZP_ZAPOCET, :OLD.ZP_SKROK,
    :OLD.ZP_TERMIN, :OLD.ZP_KREDITY,
    :OLD.ZP_VYSLEDOK,:OLD.ZP_PREDNASAJUCI,
    :OLD.ZP_DATUM_SK, :OLD.ZP_ST_OS_CISLO,
    :OLD.ZP_PR_CIS_PREDM, :OLD.ZP_UC_OS_CIS,
    USER, SYSDATE);
END;
```

Procedure, function

Syntax pre procedúru:

CREATE [OR REPLACE] PROCEDURE
procedure_name [(parameter1 [mode1] datatype1, parameter2 [mode2] datatype2, . . .)]
IS|AS
PL/SQL Block;

Syntax pre funkciu:

CREATE [OR REPLACE] FUNCTION
function_name [(parameter1 [mode1] datatype1,parameter2 [mode2] datatype2, . . .)]
RETURN datatype
IS|AS
PL/SQL Block;

PL/SQL blok

```
Create or replace ....
IS
   [... deklarácia premenných]
BEGIN
   ... príkazy
   [RETURN návratová_hodnota;]
   [EXCEPTION
      … časť na odchytenie výnimiek]
END;
/
```

Mode – Typ argumentu:

IN (default) – vstupný. Odovzdáva sa hodnota z volaného prostredia do procedúry ako konštanta. Pri pokuse o zmenu hodnoty argumentu, ktorý je definovaný ako IN, nastane chyba.

OUT – výstupný. Odovzdáva sa hodnota argumentu do prostredia, odkiaľ bola procedúra volaná.

IN OUT – vstupno-výstupný. Odovzdáva sa hodnota argumentu z prostredia a zmenená hodnota môže byť pomocou toho istého argumentu odovzdaná do prostredia, odkiaľ bola procedúra volaná.

IN	OUT	IN OUT
default	Musí byť špecifikovaný	Musí byť špecifikovaný
Formálny parameter sa chová ako konštanta	Neinicializovaná premenná	Inicializovaná premenná
Parameter môže byť literál, výraz, konštanta alebo inicializovaná premenná	Musí byť premenná	Musí byť premenná

Príklad 1

```
...deklarácia procedúry
CREATE OR REPLACE PROCEDURE query_emp
 (v_id IN emp.empno%TYPE,
  v_name OUT emp.ename%TYPE,
  v_salary OUT emp.sal%TYPE,
  v_comm OUT emp.comm%TYPE)
IS
BEGIN
  SELECT ename, sal, comm INTO v_name, v_salary, v_comm
  FROM emp  WHERE empno = v_id;
END query_emp;
/
```

...v_id bude toho istého typu ako stĺpec empno z tabuľky emp

Príklad 1

```
... deklarácia premenných
SQL> VARIABLE g_name VARCHAR2(15)
SQL> VARIABLE g_sal NUMBER
SQL> VARIABLE g_comm NUMBER

... vykonanie procedúry
SQL> EXECUTE query_emp(7654,:g_name,:g_sal,:g_comm);
PL/SQL procedure successfully completed.

... výpis hodnoty výstupného parametra
SQL> PRINT g_name
G_NAME
---------------
MARTIN
```

Príklad 2

```
...deklarácia funkcie
SQL> CREATE OR REPLACE FUNCTION get_sal
2 (v_id IN emp.empno%TYPE)
3 RETURN NUMBER
4 IS
5 v_salary emp.sal%TYPE :=0;
6 BEGIN
7 SELECT sal
8 INTO v_salary
9 FROM emp
10 WHERE empno = v_id;
11 RETURN v_salary;
12 END get_sal;
13 /
```

Príklad 2

```
SQL> VARIABLE g_salary number
```

... vykonanie funkcie, návratová hodnota bude uložená do premennej g_salary

```
SQL> EXECUTE :g_salary := get_sal(7934)
PL/SQL procedure successfully completed.
```

... výpis hodnoty premennej g_salary

```
SQL> PRINT g_salary
G_SALARY
------------------
1300
```

Spôsob odovzdávania parametrov

- **pozíciou** – premenné odovzdané procedúre v takom istom poradí ako sú deklarované
- **názvom** – premenné odovzdané v ľubovoľnom poradí, každá hodnota je asociovaná s názvom premennej použitím syntaxe =>
- **kombinované** – prvé parametre odovzdané pozíciou, zbytok názvom

Odovzdávanie parametrov

```
CREATE OR REPLACE PROCEDURE add_dept
 (v_name IN dept.dname%TYPE DEFAULT 'unknown',
 v_loc IN dept.loc%TYPE DEFAULT 'unknown') ...

begin
add_dept; ... použitie defaultných hodnôt
add_dept( 'TRAINING', 'NEW YORK'); ...pozíciou
add_dept( v_loc => 'DALLAS', v_name
  =>'EDUCATION');                    ...názvom
add_dept( v_loc => 'BOSTON') ; ...názvom a default
end;
```

Zrušenie procedúry a funkcie

Syntax:

```
DROP PROCEDURE procedure_name;
DROP FUNCTION function_name;
```

Príklad:

```
SQL> DROP PROCEDURE raise_salary;
Procedure dropped.

SQL> DROP FUNCTION get_sal;
Function dropped.
```

Využitie funkcie v SQL

```
SQL> CREATE OR REPLACE FUNCTION tax
2 (v_value IN NUMBER)
3 RETURN NUMBER
4 IS
5 BEGIN
6 RETURN (v_value * .08);
7 END tax;
8 /
Function created.

SQL> SELECT empno, ename, sal, tax(sal)
2 FROM emp;
```

Odchytenie výnimky

```
create or replace procedure ins_predm
   (cp      IN predmet.cis_predm%TYPE,
    nazv    IN varchar2,   --bez dlzky stringu
    garant IN predmet.gestor%TYPE)
AS
BEGIN
     insert into predmet (cis_predm, nazov, gestor )
     values ( cp, nazv, garant);
     dbms_output.put_line ('Vlozene ...');
   EXCEPTION
    WHEN  DUP_VAL_ON_INDEX THEN
      dbms_output.put_line('Poruseny PK');
     WHEN OTHERS THEN
      dbms_output.put_line('Ina chyba');
END;
/
```

Poznámka:

Kvôli výpisu najprv nastaviť SET SERVEROUTPUT ON

Výnimky

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

SELECT...INTO

- V podprogramoch a triggroch nie je možné použiť priamo select - je nutné použiť buď cursor, alebo príkaz SELECT INTO
- SELECT INTO je možné použiť jedine vtedy, keď daný select vráti maximálne jeden riadok
- Výsledok selectu bude uložený do zoznamu premenných, t.j. v akom poradí sú vyberané stĺpce v takom poradí s príslušným typom sú uvádzané premenné a ich počet musí byť rovnaký

Príklad:

```
SELECT sysdate INTO datum
FROM dual;
```

Príkazy PL/SQL

- Jednoriadkový komentár `-- komentar`
- Príkaz priradenia
```
premenna:= hodnota;
```
- Vetvenie
```
IF (podmienka) THEN
        prikazy;
[ ELSE
        prikazy;]
END IF;
```
- Viacnásobné vetvenie
```
IF (podmienka) THEN
    prikazy;
ELSIF (podmienka) THEN
    prikazy;
[ ELSE prikazy;]
END IF;
```

Cykly v PL/SQL

```
LOOP
   prikazy;
  EXIT WHEN podmienka;
END LOOP;
```

…podmienka ukončenia cyklu

```
done := FALSE;
WHILE NOT done LOOP
  prikazy;
  done := podmienka;
END LOOP;
```

```
FOR i IN [REVERSE] od_hodnota..do_hodnota LOOP
  prikazy;
END LOOP;
```

…ak je REVERSE, potom od>do

View

(Pohľady)

Je to predpripravený select, ktorý sa vykoná keď sa odkáže na view.

Syntax

CREATE [OR REPLACE] [FORCE | NOFORCE]

VIEW [*schema*.] *pohľad* [(*alias_stlpca* [,…])]

AS Select-prikaz

[WITH [READ ONLY | CHECK OPTION
[CONSTRAINT *obmedzenie*]]]

|

CREATE VIEW [*schema*.] *pohľad* [(*alias_stlpca* [,…])]

AS Select-prikaz

[WITH [CASCADED | LOCAL] CHECK OPTION]

- **schema** – názov schémy, v ktorej sa má pohľad nachádzať
- **OR REPLACE** – predefinovanie pohľadu, ak už bol definovaný
- **FORCE** – tento druh pohľadu je možné vytvoriť aj vtedy, ak tabuľky (objekty), z ktorých má byť pohľad odvodený neexistujú, alebo užívateľ, ktorý vytvára pohľad nemá na ne práva
- **NOFORCE** – implicitne – pohľad je možné vytvoriť len vtedy, ak bázické tabuľky (objekty) existujú a užívateľ má na ne práva.
- **READ ONLY** – nedovolí operácie INSERT, UPDATE ani DELETE nad pohľadom.
- **CHECK OPTION** – kontroluje dodržiavanie podmienky WHERE pri operáciách INSERT, UPDATE, DELETE do pohľadu
- **CONSTRAINT** – pomenovanie obmedzenia
- **CASCADED** - kontrola podmienok v odvodených pohľadoch
- **LOCAL** – obmedzenie kontroly podmienok len na podmienku definovanú v danom pohľade

Jednoduchý pohľad

```
CREATE OR REPLACE VIEW pohl1
AS
SELECT ou_meno, ou_priezvisko
  FROM os_udaje;
```

-- vytvorenie, alebo predefinovanie pohľadu

-- nie je možný INSERT

Jednoduchý pohľad-upravený

```
CREATE OR REPLACE VIEW pohl1a
AS
SELECT
  ou_rod_cislo,ou_meno,
  ou_priezvisko
FROM os_udaje;
```

-- vytvorenie, alebo predefinovanie pohľadu

-- už je možný INSERT, ak všetky atribúty sú NOT NULL

Pohľad s predefinovaním stĺpcov

```
CREATE OR REPLACE VIEW pohl1 (meno,
  priezvisko, rod_cislo)
AS
SELECT ou_meno, ou_priezvisko,
  ou_rod_cislo FROM os_udaje;
```

alebo

```
CREATE OR REPLACE VIEW pohl1
AS
SELECT ou_meno meno, ou_priezvisko
  priezvisko, ou_rod_cislo rod_cislo
  FROM os_udaje;
```

Pohľad s podmienkou

Odstránenie problému

```
CREATE VIEW OR REPLACE pohl2 (meno, priezvisko,
   rod_cislo)
AS
SELECT ou_meno, ou_priezvisko, ou_rod_cislo FROM
   os_udaje
WHERE ou_meno LIKE 'K%'
WITH CHECK OPTION;
```

Pozor! Je možné vložiť do pohľadu pohl2 aj údaje, ktoré pri selecte nebudete vidieť, ale dáta budú vložené do zdrojovej tabuľky

```
INSERT INTO pohl2 (meno, priezvisko,rod_cislo)
VALUES ('Martinko','Klingacik','0512224/0000');
```

Tento insert už neprebehne, lebo nevyhovuje podmienke

Zakázanie deštruktívnych operácií

```
CREATE OR REPLACE VIEW pohl2
  (meno, priezvisko, rod_cislo)
AS
SELECT ou_meno, ou_priezvisko,
  ou_rod_cislo FROM os_udaje
WHERE ou_meno LIKE 'K%'
WITH READ ONLY;
```

Pohľad s použitím funkcií

```
CREATE OR REPLACE VIEW pohl3 (meno,
  priezvisko, priemer)
AS
SELECT ou_meno, ou_priezvisko,
  avg(nvl(zp_vysledok,4))
FROM os_udaje, student, zap_predmety
WHERE ou_rod_cislo = st_ou_rod_cislo
  AND st_os_cislo = zp_st_os_cislo
group by ou_meno, ou_priezvisko,
  st_os_cislo
WITH READ ONLY;
```

Pohľad z viacerých tabuliek

```
CREATE OR REPLACE VIEW pohl4 (meno,
  priezvisko, rocnik, skupina, rod_cislo,
  os_cislo)
AS
SELECT ou_meno, ou_priezvisko, st_rocnik,
  st_st_skupina, ou_rod_cislo,
  st_os_cislo
FROM os_udaje, student
WHERE ou_rod_cislo = st_ou_rod_cislo;
```

Insert pre pohľad z viacerých tabuliek

```
INSERT INTO pohl4 (meno,
  priezvisko, rocnik, skupina,
  rod_cislo, os_cislo)
VALUES ('Peter','Novy',1,'5Z011',
  '841231/1212',55);
```

Tento insert nefunguje, aby fungoval je **potrebné definovať trigger** namiesto Insertu

Trigger pre predefinovanie insertu

```
create or replace trigger pohl4_ins
  instead of insert on pohl4
referencing new as novy
  begin
    insert into os_udaje (ou_meno, ou_priezvisko,
       ou_rod_cislo)
    values (:novy.meno,:novy.priezvisko, :novy.rod_cislo);

    insert into student
     (st_ou_rod_cislo, st_os_cislo, st_st_skupina, st_rocnik,
      st_so_st_odbor, st_so_st_zameranie)
    values (:novy.rod_cislo, :novy.os_cislo,:novy.skupina,
     :novy.rocnik, 0, 0);

--  0, 0 …je to potrebne, aby boli dodržané pravidlá
   referenčnej integrity, IRS – bez zamerania
 end;
```

Delete z pohľadu z viacerých tabuliek

```
DELETE FROM pohl4
WHERE os_cislo = 55;
```

POZOR!!! Tento delete funguje “záhadne“ -
z pohľadu síce riadok zmizne, ale v tabuľke os_udaje zostanú údaje o študentovi - Peter Novy

Trigger pre predefinovanie deletu

```
create or replace trigger pohl4_del
  instead of delete on pohl4
referencing old as stary
  begin
    delete from student
    where st_os_cislo = :stary.os_cislo;

    delete from os_udaje
    where ou_rod_cislo=:stary.rod_cislo;
 end;
```

Pohľad z pohľadu - DELETE

```
CREATE OR REPLACE VIEW pohl5
AS
SELECT meno, priezvisko, rod_cislo
FROM pohl4;
```

POZOR!!! Insert nebude fungovať, lebo nemáte všetky potrebné údaje.

Ale nasledovný DELETE vymaže nielen z tabuľky os_udaje, ako by sa zdalo, ale aj z tabuľky študent

```
DELETE FROM pohl5
WHERE rod_cislo = '0512224/0000';
```

POUŽITIE POHĽADU V TRIGGROCH

```
CREATE OR REPLACE VIEW pohl8 (pr_meno, rod_cislo)
AS
SELECT TRIM(ou_priezvisko)||' '||TRIM(ou_meno),
   ou_rod_cislo
FROM os_udaje;
```

Trigger na miesto insertu

```
CREATE OR REPLACE TRIGGER pohl8
 INSTEAD OF INSERT ON pohl8
 REFERENCING NEW AS NEW
  BEGIN
    INSERT INTO os_udaje (ou_meno, ou_priezvisko,
      ou_rod_cislo)
  VALUES
     (SUBSTR(:new.pr_meno,1,INSTR(:new.pr_meno,' ')-1),
      SUBSTR(:new.pr_meno,INSTR(:new.pr_meno,' ')+1) ,
      :new.rod_cislo);
 END;
```

Pohľad z pohľadu - CHECK OPTION

```
CREATE OR REPLACE VIEW pohl6 (meno,
  priezvisko,rod_cislo)
AS
SELECT ou_meno, ou_priezvisko, ou_rod_cislo
FROM os_udaje
where ou_meno like 'S%';

CREATE OR REPLACE VIEW pohl7
AS
SELECT * FROM pohl6
where rod_cislo like '79%'
WITH CHECK OPTION;
```

Pohľad z pohľadu - CHECK OPTION 2

Nasledovný insert funguje

```
INSERT INTO pohl6
VALUES ( 'Karol', 'Novy', '790502/1212');
```

Tento insert nefunguje. Klausula WITH CHECK OPTION kontroluje aj zdedené podmienky.

```
INSERT INTO pohl7
VALUES ( 'Karol', 'Novy', '790502/1212');
```

CASCADE CHECK OPTION

```
CREATE VIEW pohl9 (meno, priezvisko,rod_cislo)
AS
SELECT ou_meno, ou_priezvisko, ou_rod_cislo
FROM os_udaje
WHERE ou_meno LIKE 'S%'
WITH CHECK OPTION;
```

alebo

```
CREATE VIEW pohl9 (meno, priezvisko,rod_cislo)
AS
SELECT ou_meno, ou_priezvisko, ou_rod_cislo
FROM os_udaje
WHERE ou_meno LIKE 'S%'
WITH CASCADE CHECK OPTION;
```

LOCAL CHECK OPTION

```
CREATE VIEW pohl10
AS
SELECT * FROM pohl6
WHERE rod_cislo LIKE '79%'

CREATE VIEW pohl11
AS
SELECT * FROM pohl10
WHERE priezvisko LIKE 'M%'
WITH LOCAL CHECK OPTION;
```

Zhrnutie

- ak SELECT pohľadu obsahuje PK a všetky ostatné NOT NULL stĺpce tabuľky, potom je možné vykonať INSERT
- ak je pohľad definovaný ako READ ONLY, nie sú možné operácie INSERT, DELETE a UPDATE
- ak je pohľad definovaný ako WITH CHECK OPTION, potom do pohľadu je možné vložiť len riadky, ktoré zodpovedajú podmienkam SELECTu a zdedeným podmienkam
- ak je SELECT z viacerých tabuliek, pre operácie INSERT, DELETE a UPDATE je nutné definovať triggre INSTEAD OF