***Baze de date-Anul 3 (semestrul 1)***
***Laborator 5 PL/SQL***

## Triggeri (declanşatori)

Ø Un trigger
- este un bloc PL/SQL asociat unui tabel, view, scheme sau unei baze de date.
- trigger-ul se executa implicit ori de câte ori are loc un anumit eveniment
- pot fi de următoarele tipuri:
  - trigger-i la nivel de aplicație: se declanşează odată un un anumit eveniment din aplicație;
  - trigger-i la nivel de bază de date: se declanşează atunci când un eveniment asupra datelor (de ex, LMD) sau un eveniment sistem (logon, shutdown) apare asupra unei scheme sau asupra bazei de date.

Ø Instrucțiunea pentru crearea trigger-ilor LMD conține următoarele informații:
- o timpul declanşării trigger-ului în raport cu evenimentul:
  - pentru tabele: BEFORE, AFTER
  - pentru view-uri nemodificabile: INSTEAD OF
- o evenimentul declanşator: INSERT, UPDATE, DELETE
- o numele tabelului
- o tipul trigger-ului – precizează de câte ori se execută corpul acestuia; trigger-ul poate fi la nivel de:
  - instrucțiune (statement): corpul triggerului se execută o singură dată pentru evenimentul declanşator. Un astfel de trigger se declanşează chiar dacă nici o linie nu este afectată.
  - linie (row): corpul triggerului se declanşează o dată pentru fiecare linie afectată de către evenimentul declanşator. Un astfel de trigger nu se execută dacă evenimentul declanşator nu afectează nici o linie.
- o clauza WHEN – precizează o condiție restrictivă
- o corpul trigger-ului (blocul PL/SQL)

Ø Sintaxa comenzii de creare a unui trigger LMD este:

*CREATE* [*OR REPLACE*] *TRIGGER* [*schema.*]*nume_trigger*
{*BEFORE* | *AFTER*}
[*INSTEAD OF*]
{*DELETE* | *INSERT* | *UPDATE* [*OF coloana*[, *coloana …*] ] }
[*OR* {*DELETE* | *INSERT* | *UPDATE* [*OF coloana*[, *coloana …*] ] …}
*ON* [*schema.*]*nume_tabel*
[*REFERENCING* {*OLD* [*AS*] *vechi NEW* [*AS*] *nou*
| *NEW* [*AS*] *nou OLD* [*AS*] *vechi* } ]
[*FOR EACH ROW*]
[*WHEN* (*condiție*) ]
*corp_trigger*;

Ø Informații despre triggeri se găsesc în următoarele vizualizări ale dicționarului datelor: *USER_TRIGGERS*, *USER_TRIGGER_COL*, *ALL_TRIGGERS*, *DBA_TRIGGERS.*

Ø *Modificarea unui declanşator constă din redenumirea, recompilarea, activarea sau dezactivarea acestuia şi se realizează prin comenzi de forma:*

*ALTER TRIGGER nume_trigger ENABLE;*
*ALTER TRIGGER nume_trigger DISABLE;*
*ALTER TRIGGER nume_trigger COMPILE;*
*ALTER TRIGGER nume_trigger RENAME TO nume_nou;*

Activarea şi dezactivarea tuturor triggerilor asociați unui tabel se realizează prin comenzile:

*ALTER TABLE nume_tabel*
*DISABLE ALL TRIGGERS;*
*ALTER TABLE nume_tabel*
*ENABLE ALL TRIGGERS;*

Eliminarea unui declanşator se face prin
*DROP TRIGGER nume_trigger;*

Ø Sintaxa pentru crearea unui declanşator sistem este următoarea

*CREATE* [*OR REPLACE*] *TRIGGER* [*schema.*]*nume_declanşator*
{*BEFORE* | *AFTER*}
{*lista_evenimente_LDD* | *lista_evenimente_bază*}
*ON* {*DATABASE* | *SCHEMA*}
[*WHEN* (*condiție*) ]
*corp_declanşator*;

unde: *lista_evenimente_LDD - CREATE*, *DROP*, *ALTER*)
*lista_evenimente_bază - STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SERVERERROR*, *SUSPEND*)

## Exerciții

**1.** Să se creeze un trigger care asigură ca inserarea de angajați în tabelul EMP_PNU se poate realiza numai în zilele lucrătoare, între orele 8-18.

***Obs:***Trigger-ul nu are legătură directă cu datele => este un ***trigger la nivel de instrucțiune.***

```
CREATE OR REPLACE TRIGGER b_i_emp_pnu
    BEFORE INSERT ON emp_pnu
    BEGIN
        IF (TO_CHAR(SYSDATE, 'dy') IN ('sat', 'sun')) OR
            (TO_CHAR(SYSDATE, 'HH24:MI')
                NOT BETWEEN '08:00' AND '18:00')
        THEN
            RAISE_APPLICATION_ERROR (-20500, 'Nu se pot introduce
                inregistrari decat in timpul orelor de lucru');
        END IF;
END;
/
```

Testați trigger-ul:

```
INSERT INTO emp_pnu (employee_id, last_name, first_name, email, hire_date,
        job_id, salary, department_id)
VALUES (300, 'Smith', 'Robert', 'rsmith', SYSDATE, 'IT_PROG', 4500, 60);
```

**2.** Modificați trigger-ul anterior, astfel încât să fie generate erori cu mesaje diferite pentru inserare, actualizare, actualizarea salariului, ştergere.

```
CREATE OR REPLACE TRIGGER b_i_emp_pnu
    BEFORE INSERT OR UPDATE OR DELETE ON emp_pnu
    BEGIN
    IF (TO_CHAR(SYSDATE, 'dy') IN ('sat', 'sun')) OR
        (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00')
    THEN
        IF DELETING THEN
```

```
                RAISE_APPLICATION_ERROR (-20501, 'Nu se pot
                        sterge inregistrari decat in timpul orelor de luru');
            ELSIF INSERTING THEN
                RAISE_APPLICATION_ERROR (-20500, 'Nu se pot
                        adauga inregistrari decat in timpul orelor de lucru');
            ELSIF UPDATING ('SALARY') THEN
                RAISE_APPLICATION_ERROR (-20502, 'Nu se poate
                        actualiza campul SALARY decat in timpul orelor de
                        lucru');
            ELSE
                RAISE_APPLICATION_ERROR (-20503, 'Nu se pot
                        actualiza inregistrari decat in timpul orelor de
                        lucru');
            END IF;
        END IF;
END;
/
```

**3.** Să se creeze un trigger care să permită ca numai salariații având codul job-ului AD_PRES sau AD_VP să poată câştiga mai mult de 15000.

***Obs:*** Trigger-ul se declanşează de un număr de ori = nr de înregistrări inserate sau al căror câmp salary este modificat (deci are legătură cu datele din tabel) => este un ***trigger la nivel de linie***.

```
CREATE OR REPLACE TRIIGER b_i_u_emp_pnu
    BEFORE INSERT OR UPDATE OF salary ON emp_pnu
    FOR EACH ROW
    BEGIN
        IF NOT (:NEW>job_id IN ('AD_PRES', 'AD_VP'))
                AND :NEW.salary > 15000
        THEN
                RAISE_APPLICTION_ERROR (-20202, 'Angajatul nu poate
    castiga aceasta suma');
        END IF;
END;
/
```

**4.** Să se implementeze cu ajutorul unui declanşator constrângerea că valorile salariilor nu pot fi reduse (trei variante). După testare, suprimați trigger-ii creați.

*Varianta 1*:

```
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi micşorată');
END;
/
Update emp_pnu
Set salary = salary/2;
Drop trigger verifica_salariu_pnu;
```

*Varianta 2*:

```
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
BEGIN
  IF (:NEW.salary < :OLD.salary) THEN
    RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi micsorata');
  END IF;
END;
/
```

*Varianta 3*:

```
CREATE OR REPLACE PROCEDURE p4l6_pnu IS
   BEGIN
         RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi
                                   micsorata');
END;
/
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
  CALL p4l6_pnu;
```

**5.** Să se creeze un trigger care calculează comisionul unui angajat 'SA_REP' atunci când este adăugată o linie tabelului emp_pnu sau când este modificat salariul.

***Obs:*** Dacă se doreşte atribuirea de valori coloanelor utilizând NEW, trebuie creați triggeri BEFORE ROW. Dacă se încearcă scrierea unui trigger AFTER ROW, atunci se va obține o eroare la compilare.

```
CREATE OR REPLACE TRIGGER b_i_u__sal_emp_pnu
      BEFORE INSERT OR UPDATE OF salary ON emp_pnu
      FOR EACH ROW
      WHEN (NEW.job_id = 'SA_REP')
BEGIN
      IF INSERTING
          THEN :NEW.commission_pct := 0;
      ELSIF :OLD.commission_pct IS NULL
          THEN :NEW.commission_pct := 0;
      ELSE :NEW.commission_pct := :OLD.commission_pct * (:NEW.salary/:OLD.salary);
      END IF;
END;
/
```

**6.** Să se implementeze cu ajutorul unui declanşator constrângerea că, dacă salariul minim şi cel maxim al unui job s-ar modifica, orice angajat având job-ul respectiv trebuie să aibă salariul între noile limite .

```
CREATE OR REPLACE TRIGGER verifica_sal_job_pnu
         BEFORE UPDATE OF min_salary, max_salary ON jobs_pnu
         FOR EACH ROW
DECLARE
         v_min_sal  emp_pnu.salary%TYPE;
         v_max_sal  emp_pnu.salary%TYPE;
```

```
        e_invalid  EXCEPTION;
BEGIN
        SELECT MIN(salary), MAX(salary)
          INTO   v_min_sal, v_max_sal
          FROM   emp_pnu
          WHERE  job_id = :NEW.job_id;
        IF (v_min_sal < :NEW.min_salary) OR
           (v_max_sal > :NEW.max_salary) THEN
                RAISE e_invalid;
        END IF;
EXCEPTION
        WHEN e_invalid THEN
                RAISE_APPLICATION_ERROR (-20567, 'Exista angajati avand salariul in afara
                        domeniului permis pentru job-ul corespunzator');
END verifica_sal_job_pnu;
/
```

**7.** Să se creeze un trigger check_sal_pnu care garantează ca, ori de câte ori un angajat nou este introdus în tabelul EMPLOYEES sau atunci când este modificat salariul sau codul job-ului unui angajat, salariul se încadrează între minimul şi maximul salariior corespunzătoare job-ului respectiv. Se vor exclude angajatii AD_PRES.

```
CREATE OR REPLACE TRIGGER check_sal_pnu
        BEFORE INSERT OR UPDATE OF salary, job_id
        ON emp_pnu
        FOR EACH ROW
        WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
        v_min employees.salary %TYPE;
        v_max employees.salary %TYPE;
BEGIN
        SELECT MIN(salary), MAX(salary)
           INTO v_min, v_max
          FROM emp_pnu         -- FROM copie_emp_pnu
          WHERE job_id = :NEW.job_id;
        IF :NEW.salary < v_min OR
           :NEW.salary > v_max THEN
                RAISE_APPLICATION_ERROR (-20505, 'In afara domeniului');
        END IF;
END;
/
```

Testați trigger-ul anterior:

```
UPDATE emp_pnu
SET salary = 3500
WHERE last_name= 'Stiles';
```

Ce se obține şi de ce? Modificați trigger-ul astfel încât să funcționeze corect.

***Obs:*** Tabelul este mutating. Pentru ca trigger-ul să funcționeze, utilizați o copie a tabelului emp_pnu în instrucțiunea SELECT din corpul trigger-ului. (aceasta este doar una dintre solutii, se vor vedea ulterior si altele).

***8.*** a) Se presupune că în tabelul *dept_pnu* se păstrează (într-o coloană numită *total_sal*) valoarea totală a salariilor angajaților în departamentul respectiv. Introduceți această coloană în tabel şi actualizați conținutul.

```
ALTER TABLE dept_pnu
ADD (total_sal NUMBER(11, 2));

UPDATE dept_pnu
SET    total_sal =
         (SELECT  SUM(salary)
          FROM    emp_pnu
          WHERE   emp_pnu.department_id = dept_pnu.department_id);
```

b) Creați un trigger care permite reactualizarea automată a acestui câmp.

```
CREATE OR REPLACE PROCEDURE creste_total_pnu
        (v_cod_dep  IN dept_pnu.department_id%TYPE,
         v_sal      IN dept_pnu.total_sal%TYPE) AS
BEGIN
  UPDATE dept_pnu
  SET     total_sal = NVL (total_sal, 0) + v_sal
  WHERE   department_id = v_cod_dep;
END creste_total_pnu;
/
CREATE OR REPLACE TRIGGER calcul_total_pnu
  AFTER INSERT OR DELETE OR UPDATE OF salary ON emp_pnu
  FOR EACH ROW
BEGIN
  IF DELETING THEN
    creste_total_pnu (:OLD.department_id, -1*:OLD.salary);
  ELSIF UPDATING THEN
    creste_total_pnu (:NEW.department_id, :NEW.salary - :OLD.salary);
  ELSE /* inserting */
    Creste_total_pnu (:NEW.department_id, :NEW.salary);
  END IF;
END;
/
```

**9.** Să se creeze două tabele noi new_emp_pnu şi new_dept_pnu pe baza tabelelor employees şi departments. Să se creeze un view view_emp_pnu, care selectează codul, numele, salariul, codul departamentului, email-ul, codul job-ului, numele departamentului şi codul locației pentru fiecare angajat.
Să se creeze un ***trigger de tip INSTEAD OF*** care, în locul inserării unei linii direct în view, adaugă înregistrări corespunzătoare în tabelele new_emp_pnu şi new_dept_pnu. Similar, atunci când o linie este modificată sau ştearsă prin intermediul vizualizării, liniile corespunzătoare din tabelele new_emp_pnu şi new_dept_pnu sunt afectate.

```
CREATE TABLE new_emp_pnu AS
        SELECT employee_id, last_name, salary, department_id, email,
                        job_id, hire_date
        FROM employees;

CREATE TABLE new_dept_pnu AS
        SELECT d.department_id, d.department_name, d.location_id,
                SUM(e.salary) total_dept_sal
        FROM employees e, departments d
        WHERE e.department_id = d.department_id
```

```
        GROUP BY d.department_id, d.department_name, d.location_id;

    CREATE VIEW view_emp_pnu AS
        SELECT e.employee_id, e.last_name, e.salary, e.department_id, e.email,
            e.job_id, d.department_name, d.location_id
        FROM employees e, departments d
        WHERE e.department_id = d.department_id;

CREATE OR REPLACE TRIGGER new_emp_dept_pnu
INSTEAD OF INSERT OR UPDATE OR DELETE ON view_emp_pnu
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emp_pnu
        VALUES(:NEW.employee_id, :NEW.last_name, :NEW.salary,
            :NEW.department_id, :NEW.email, :NEW.job_id, SYSDATE);
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emp_pnu
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal - :OLD.salary
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('salary') THEN
        UPDATE new_emp_pnu
        SET salary = :NEW.salary
        WHERE employee_id = :NEW.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + (:NEW.salary - :OLD.salary)
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('department_id') THEN
        UPDATE new_emp_pnu
        SET department_id = :NEW.department_id
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal  - :OLD.salary
        WHERE department_id = :OLD.department_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    END IF;
    END;
    /
```

**10.** Să se implementeze cu ajutorul unui declanşator restricția că într-un departament pot lucra maximum 50 de angajați.

```
CREATE OR REPLACE TRIGGER TrLimitaDep_pnu
  BEFORE INSERT ON emp_pnu
  FOR EACH ROW
```

```
DECLARE
  v_Max_emp CONSTANT NUMBER := 50;
  v_emp_curent      NUMBER;
BEGIN
  SELECT  COUNT(*) INTO v_emp_curent
  FROM    emp_pnu
  WHERE   department_id = :NEW.department_id;
  IF v_emp_curent + 1 > v_Max_emp THEN
    RAISE_APPLICATION_ERROR(-20000,'Prea multi angajati in departamentul
                              avand codul ' || :NEW.department_id);
  END IF;
END TrLimitaDep_pnu;
/
```

Testați trigger-ul.

```
INSERT INTO emp_pnu (employee_id, …, department_id)
VALUES (emp_seq_pnu.nextval, …, 30);  --se va executa de mai multe ori astfel încât să existe mai mult de 50 de angajați în departamentul 30.
```

***Obs:*** Declanşatorul *TrLimitaDep_pnu* consultă chiar tabelul (*emp_pnu*) la care este asociat declanşatorul (*mutating*).

Tabelul *emp_pnu* este *mutating* doar pentru un declanşator la nivel de linie.

O solutie pentru acestă problemă este crearea a doi declanşatori, unul la nivel de linie şi altul la nivel de instrucțiune :

- în declanşatorul la nivel de linie se înregistrează valoarea lui *:NEW.department_id*, dar nu va fi interogat tabelul *emp_pnu*.
- interogarea va fi făcută în declanşatorul la nivel de instrucțiune şi va folosi valoarea înregistrată în declanşatorul la nivel de linie.

O modalitate pentru a înregistra valoarea lui :*NEW.department_id* este ***utilizarea unui tablou indexat în interiorul unui pachet***.

```
CREATE OR REPLACE PACKAGE PdepDate_pnu AS
  TYPE t_cod IS TABLE OF dept_pnu.department_id%TYPE
             INDEX BY BINARY_INTEGER;
  v_cod_dep   t_cod;
  v_NrIntrari  BINARY_INTEGER := 0;
END PdepDate_pnu;
/
CREATE OR REPLACE TRIGGER TrLLimitaDep_pnu
  BEFORE INSERT ON dept_pnu
  FOR EACH ROW
BEGIN
  PDepDate_pnu.v_NrIntrari := PDepDate_pnu.v_NrIntrari + 1;
  PdepDate_pnu.v_cod_dep (PdepDate_pnu.v_NrIntrari) :=
                              :NEW.department_id;
END TrLLimitaDep_pnu;
/
CREATE OR REPLACE TRIGGER TrILimitaDep_pnu
  BEFORE INSERT ON emp_pnu
DECLARE
  v_Max_emp      CONSTANT NUMBER := 50;
  v_emp_curent  NUMBER;
  v_cod_dep       dep_pnu.department_id%TYPE;
```

```
BEGIN
  /* Parcurge fiecare departament inserat sau actualizat si
     verifica daca se incadreaza in limita stabilita */
  FOR v_LoopIndex IN 1..PdepDate_pnu.v_NrIntrari LOOP
    v_cod_dep := PdepDate_pnu.v_cod_dep(v_LoopIndex);
    SELECT  COUNT(*)
    INTO    v_emp_curent
    FROM    emp_pnu
    WHERE   department_id = v_cod_dep;
    IF v_emp_curent > v_Max_emp THEN
      RAISE_APPLICATION_ERROR(-20000, 'Prea multi angajati
              in departamentul avand codul: ' || v_cod_sala);
    END IF;
  END LOOP;
  /* Reseteaza contorul deoarece urmatoarea executie
     va folosi date noi */
  PdepDate_pnu.v_NrIntrari := 0;
END TrILimitaopere;
/
```

***Obs:*** Această soluție funcționează pentru departamentele nou introduse.
Pentru testare:
- introduceți un nou departament
-introduceti mai mult de 50 de angajați în departamentul inserat anterior (eventual, cu o comandă de tip INSERT INTO ... (SELECT ...)).

**11.** Să se creeze un declanşator care:

a) dacă este eliminat un departament, va şterge toți angajații care lucrează în departamentul respectiv;

b) dacă se schimbă codul unui departament, va modifica această valoare pentru fiecare angajat care lucrează în departamentul respectiv.

```
CREATE OR REPLACE TRIGGER dep_cascada_pnu
  BEFORE DELETE OR UPDATE OF department_id ON dept_pnu
  FOR EACH ROW
BEGIN
  IF DELETING THEN
    DELETE FROM emp_pnu
    WHERE  department_id = :OLD.department_id;
  END IF;
  IF UPDATING AND :OLD. department_id != :NEW. department_id THEN
    UPDATE emp_pnu
    SET    department_id = :NEW. department_id
    WHERE  department_id = :OLD. department_id;
  END IF;
END dep_cascada_pnu;
```

***Obs:*** Declanşatorul anterior realizează constrângerea de integritate *UPDATE* sau *ON DELETE CASCADE*, adică ştergerea sau modificarea cheii primare a unui tabel „părinte“ se va reflecta şi asupra înregistrărilor corespunzătoare din tabelul „copil“.
Testați trigger-ul:

```
DELETE FROM  dept_pnu
WHERE  department_id = 10;
```

```
UPDATE  dept_pnu
SET     department_id = 12
WHERE   department_id = 10;
```
***Obs :*** Se presupune că asupra tabelului *emp_pnu* există o constrângere de integritate:
FOREIGN KEY (department_id) REFERENCES dept_pnu(department_id)

În acest caz sistemul *Oracle* va afişa un mesaj de eroare prin care se precizează că tabelul *dept_pnu* este *mutating*, iar constrângerea definită mai sus nu poate fi verificată.
*ORA-04091*: *table MASTER.DEPT_PNU is mutating*, *trigger/function may not see it*

**12.** Să se creeze un ***declanşator la nivelul bazei de date*** prin care să nu se permită ştergerea informațiilor din tabelul emp_pnu de către utilizatorul curent. Dezactivați, iar apoi activați trigger-ul creat. Testați, iar apoi suprimați acest trigger.

```
CREATE OR REPLACE TRIGGER p13l6_pnu
  BEFORE DELETE ON emp_pnu
 BEGIN
  IF USER= UPPER('g231') THEN
    RAISE_APPLICATION_ERROR(-20900,'Nu este permisa stergerea de catre ' ||USER);
  END IF;
 END;
/

ALTER TRIGGER p13l6_pnu DISABLE;

DELETE FROM emp_pnu WHERE employee_id = 100;

ALTER TRIGGER p13l6_pnu DISABLE;

DELETE FROM emp_pnu WHERE employee_id = 100;

DROP TRIGGER p13l6_pnu;
```

**13.** Să se creeze un tabel care conține următoarele câmpuri: user_id, nume_bd, eveniment_sis, nume_obj, data. Să se creeze un **trigger sistem** (la nivel de schemă) care să introducă date în acest tabel după ce utilizatorul a folosit o comandă LDD. Testați, iar apoi suprimați trigger-ul.

```
CREATE TABLE log_pnu
        (user_id        VARCHAR2(30),
          nume_bd     VARCHAR2(50),
          eveniment_sis  VARCHAR2(20),
          nume_obj          VARCHAR2(30),
          data              DATE);
CREATE OR REPLACE TRIGGER p14l6_pnu
  AFTER CREATE OR DROP OR ALTER ON SCHEMA
BEGIN
  INSERT INTO info_pnu
  VALUES (SYS.LOGIN_USER, SYS.DATABASE_NAME, SYS.SYSEVENT,
                SYS.DICTIONARY_OBJ_NAME, SYSDATE);
END;
/
CREATE VIEW v_test_pnu AS SELECT *  FROM jobs;

DROP      VIEW v_test_pnu;

SELECT * FROM log_pnu;

DROP TRIGGER p14l6_pnu;
```