PL/SQL - Procedure, Function, Trigger;

7. apríla 2015

PL/SQL - Procedural Language

- nepomenované bloky príkazov
- pomenované bloky príkazov - procedúry, funkcie
- triggre

Deklarácia premenných

- premenná je daného typu

```
Premenna          typ;
```

- deklarácia a inicializácia

```
Premenna          typ := init_hodnota;
```

- záznam má položky rovnakého typu a názvu ako tabuľka

```
Premenna_record   tabulka%ROWTYPE;
```

- premenná je rovnakého typu ako príslušný stĺpec tabuľky

```
Premenna          tabulka.stlpec%TYPE;
```

Príkazy

1. Príkaz priradenia

   ```
   Premenna := vyraz;
   ```

2. Prázdny príkaz

   ```
   NULL;
   ```

Vetvenie príkazom IF

```
IF podmienka THEN
   Prikazy;
 [ ELSE ]
   Prikazy;
END IF;
```

```
IF podmienka THEN
  Prikazy;
ELSIF podmienka2 THEN
  Prikazy;
[ ELSE
   Prikazy; ]
END IF;
```

```
IF podmienka THEN
  Prikazy;
ELSE IF podmienka2 THEN
    Prikazy;
  [  ELSE
       Prikazy;]
  END IF;
END IF;
```

Vetvenie príkazom CASE

```
CASE vyraz
  WHEN hodnota1 THEN prikazy1;
  WHEN hodnota2 THEN prikazy2;
  ...
  [ ELSE prikazy; ]
END CASE;
```

```
CASE
  WHEN podmienka1 THEN prikazy1;
  WHEN podmienka2 THEN prikazy2;
  ...
  [ ELSE prikazy; ]
END CASE;
```

Cykly

- LOOP
- WHILE
- FOR

- Nekonečný cyklus:

```
LOOP
  Prikazy;
END LOOP;
```

- Ukončenie nekonečného cyklu pomocou EXIT:

```
LOOP
  ...
  IF podmienka THEN
    EXIT;
  END IF;
  ...
END LOOP;
```

```
LOOP
  ...
  EXIT WHEN podmienka;
  ...
END LOOP;
```

Cyklus WHILE

```
WHILE podmienka LOOP
  Prikazy;
END LOOP;
```

Cyklus FOR

```
FOR premenna IN min..max LOOP
   Prikazy;
END LOOP;
```

```
FOR premenna IN REVERSE min..max LOOP
   Prikazy;
END LOOP;
```

Syntax nepomenovaného bloku

```
DECLARE
  [ nazov_premennej  typ[:= init_hodnota]; ]
BEGIN
  Prikazy;
  [ EXCEPTION
      WHEN typ_vynimky THEN
         Prikazy;
      [WHEN ...]
  ]
END;
/
```

Syntax procedúry

```
CREATE [OR REPLACE] PROCEDURE nazov_procedury
 [( parameter1 [ mode1] datatype1,
    parameter2 [ mode2] datatype2, . . .)]
IS|AS
  [ nazov_premennej  typ[:= init_hodnota]; ]
BEGIN
  Prikazy;
  [ EXCEPTION
      WHEN typ_vynimky THEN
         Prikazy;
      [WHEN ...]
  ]
END  [nazov_procedury];
/
```

Syntax funkcie

```
CREATE [OR REPLACE] FUNCTION nazov_funkcie
 [( parameter1 [ mode1] datatype1,
    parameter2 [ mode2] datatype2, . . .)]
RETURN datatype
IS|AS
  [ nazov_premennej  typ[:= init_hodnota]; ]
BEGIN
  Prikazy;
  RETURN vyraz;
[ EXCEPTION
      WHEN typ_vynimky THEN
         Prikazy;
      [WHEN ...]
  ]
END  [nazov_funkcie];
/
```

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 procedúry

Majme procedúru `query_stud`, ktorá na základe vstupného parametra (osobné číslo), vráti celé meno študenta a jeho štúdijnú skupinu vo výstupných parametroch.

1.
```
CREATE OR REPLACE PROCEDURE query_stud
  (v_oc       IN  student.os_cislo%TYPE,
   v_meno     OUT VARCHAR2,
   v_skupina  OUT student.st_skupina%TYPE)
IS
BEGIN
  SELECT meno||' '|| priezvisko, st_skupina
    INTO v_meno, v_skupina
  FROM student st JOIN  os_udaje ou USING (rod_cislo)
  WHERE st.os_cislo = v_oc;
END query_stud;
/
```

2.
```
-- deklarácia premenných v sqlplus
SQL> VARIABLE p_meno VARCHAR2(30)
SQL> VARIABLE p_skupina CHAR(5)

-- spustenie procedúry v sqlplus - pozor na :premenna
SQL> EXECUTE query_stud( 1512, :p_meno, :p_skupina);

PL/SQL procedure successfully completed.
```

3. Výpis hodnoty premennej (bez :)

```
SQL> PRINT p_meno p_skupina
```

P_MENO
Peter Novak

P_SKUPINA
5Z012

Aby bolo možné vypísať text na konzolu pomocou metódy `dbms_output.put_line`, je najprv nutné zadať nasledovný príkaz:

```
SQL> set serveroutput on
```

```
declare
  --- deklarácia premenných v bloku príkazov
  p_meno VARCHAR2(30);
  p_skupina CHAR(5);
begin
  --- spustenie procedúry v bloku
  query_stud( 1512, p_meno, p_skupina);

  --- vypis hodnoty premennej
  dbms_output.put_line ( 'Meno:     '|| p_meno);
  dbms_output.put_line ( 'Skupina : '|| p_skupina);
end;
/

 --- na konzole bude výpis:
  Meno:    Peter Novak
  Skupina: 5Z012
```

Príklad funkcie

1.
```
CREATE OR REPLACE FUNCTION pocet_kreditov (p_oc IN student.os_cislo%TYPE)
   RETURN NUMBER
IS
   p_kredity zap_predmety.ects%TYPE :=0;
BEGIN
  SELECT sum(nvl(ects,0)) INTO p_kredity
    FROM zap_predmety
    WHERE os_cislo = p_oc
      AND vysledok in ('A','B','C','D','E');

   RETURN p_kredity;

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RETURN 0;
END pocet_kreditov;
/
```

2. deklarácia premenných v sqlplus

```
SQL> VARIABLE kredity NUMBER
SQL> VARIABLE oc NUMBER
SQL> EXECUTE :oc := 500438
```

3. spustenie funkcie v sqlplus - pozor na :premenna

```
SQL> EXECUTE :kredity := pocet_kreditov(:oc)
```

4. výpis hodnoty - pozor bez :

```
SQL> PRINT kredity

   KREDITY
----------
       104
```

Použitie funkcie v bloku príkazov

```
declare
  kredity number;
  oc    number := 500438;
begin
   kredity := pocet_kreditov(oc);       --- vykonaj funkciu
  dbms_output.put_line(kredity); --- vypis vysledok
end;
/
```

Použitie funkcie v príkaze Select

```
SQL> SELECT st.os_cislo, ou.meno, ou.priezvisko, pocet_kreditov(st.os_cislo)
     FROM student st JOIN os_udaje ou USING (rod_cislo);
```

Poznámky

- **Parametre v príkazovom riadku**
  - Pri spustení metódy z príkazového riadku môžeme hodnotu vstupného parametra (IN) zadať priamo konštantou, alebo premennou.
  - Hodnotu vstupno-výstupného parametra (IN OUT) je nutné zadať pomocou premennej.
  - Pre získanie výstupnej hodnoty z výstupného parametra (OUT) procedúry je nutné zadať premennú.
  - Premennú je potrebné deklarovať pomocou príkazu VARIABLE (len názov premennej a dátový typ).
  - Pri použití premennej pri spustení metódy je nutné zadať pred meno znak ":"(priamo pred názov premennej).
  - Pri zisťovaní hodnoty (vstupno-)výstupnej premennej príkazom PRINT už sa nepoužíva znak ":"

- **Funkcie**
  - Rozdiel medzi funkciami a procedúrami je minimálny - funkcia svojim menom vracia hodnotu.
  - Funkcie je možné priamo využiť v príkazoch Select, pokiaľ návratová hodnota funkcie je typu, ktorý korešponduje s SQL typmi.(t.j. napr. nesmie vracať BOOLEAN hodnotu).
  - Návratovú hodnotu funkcie nie je možné "zahodiť", ale je nutné výsledok funkcie využiť v príkaze Select, alebo výsledok funkcie uložiť do premennej, alebo použiť ako vstup inej metódy, prípadne výrazu.

Upozornenie:
Názov parametra alebo premennej, ktoré chcete používať v príkaze Select **NESMIE** mať rovnaký názov ako stĺpec tabuľky (pohľadu), z ktorej vyberáme. Toto je závažná chyba, ktorú kompilátor nerozpozná, ale pravdepodobne bude viesť k nesprávnym výsledkom.

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.

Spôsob odovzdávania parametrov - príklad

1. ```
   CREATE OR REPLACE PROCEDURE add_dept
   ( v_name  IN dept.dname%TYPE  DEFAULT 'unknown',
     v_loc   IN dept.loc%TYPE    DEFAULT 'unknown') ...
   ```

2. ```
   begin
   add_dept;                          -- ... použitie defaultných
   add_dept( 'TRAINING', 'NEW YORK'); -- ...pozíciou
   add_dept( v_loc => 'DALLAS', v_name =>'EDUCATION'); --
   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.
```

Rozdiel medzi nepomenovaným a pomenovaným blokom

- Nepomenovaný blok sa vykonáva hneď po kompilácii zdrojového kódu a neukladá sa v databáze.
- Nepomenovaný blok sa pri každom spustení musí kompilovať a pri jeho spustení musíte mať priamo k dispozícii zdrojový kód daného bloku.
- Pomenovaný blok príkazov (metóda) sa najprv skompiluje a jeho zdrojový aj skompilovaný kód sa uloží do databázového priestoru.
- Teda metóda môže byť zavolaná z ľubovoľnej časti aplikácie a aj keď vymažeme súbor so zdrojovým kódom, pokiaľ nezmažeme samotnú metódu z DBS, bude ju možné naďalej používať.

Select v PL/SQL

Priamo v PL/SQL bloku je možné využiť len SELECT-INTO, alebo použiť kurzor pre spracovanie výsledkov selectu.

Syntax

```
SELECT zoznam_stlpcov INTO zoznam_premennych
  FROM zoznam_tabuliek
  ...
```

Podmienky pre aplikovanie konštruktu SELECT-INTO

1. SELECT-INTO musí byť select, ktorý vráti práve jeden riadok.
2. Za INTO musí byť uvedený zoznam premenných, ktoré sú predtým deklarované.
3. Premenné musia v rovnakom poradí ako stĺpce selectu a rovnakého typu ako očakávame výsledok. Na to môžeme využiť kopírovanie typov(pomocou %TYPE).

Príklad - Select into

```
declare
  p_meno   os_udaje.meno%TYPE;
  p_priezv os_udaje.priezvisko%TYPE;
  pocet    integer;
begin
  select meno, priezvisko, count(*)
     INTO p_meno, p_priezv, pocet
  from os_udaje ou join student st using ( rod_cislo )
       left join zap_predmety using (os_cislo )
  where os_cislo = 550807
  group by meno, priezvisko;

  dbms_output.put_line('Pocet predmetov studenta - '
            || p_meno || ' '||p_priezv||' je '|| pocet );

end;
/
```

- Štandardné výnimky

```
BEGIN
   prikazy bloku
EXCEPTION
  WHEN nazov_vynimky1 THEN
    priakzy1
  WHEN nazov_vynimky2 THEN
    priakzy2
END; -- ukoncenie bloku prikazov
```

- Užívateľom definované výnimky:

1. Pomocou funkcie

```
RAISE_APPLICATION_ERROR(cislo_chyby, text_chyby);
```

Číslo chyby musí byť z intervalu $< -20000, -29999 >$. Presne túto chybu nie je možné odchytiť v časti EXCEPTION, len medzi ostanými chybami (WHEN OTHERS THEN ...)

2. Pomocou premennej

Deklarácia chybovej premennej

```
chybova_premenna  EXCEPTION;
```

Vyvolanie výnimky

```
RAISE chybova_premenna;
```

Odchytenie výnimky

```
WHEN chybova_premenna THEN
  prikazy...
```

Spracovanie štandardnej výnimky.

```
declare
  p_meno       os_udaje.meno%TYPE;
  p_priezv     os_udaje.priezvisko%TYPE;
BEGIN
  SELECT meno, priezvisko
       INTO p_meno, p_priezv
     from os_udaje
     where rod_cislo = '3';

  dbms_output.put_line( p_meno || ' '|| p_priezv);

EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line ('Nebol najdeny');
  WHEN others THEN
    dbms_output.put_line ('Ina chyba');
END;
/
```

Vyvolanie užívateľskej výnimky.

```
create or replace procedure zapis_predmet
   ( oc  student.os_cislo%TYPE,
     cp  predmet.cis_predm%TYPE,
     rok zap_predmety.skrok%TYPE
   )
AS
  pocet   integer;
BEGIN
  SELECT count(*) INTO pocet
    from student
    where os_cislo = oc;

  IF (pocet = 0) THEN
    RAISE_APPLICATION_ERROR(-20000, 'Not existing student');
  END IF;

  SELECT count(*) INTO pocet
    from predmet
    where cis_predm = cp;

  IF (pocet = 0) THEN
    RAISE_APPLICATION_ERROR(-20001, 'Not existing subject');
  END IF;

  SELECT count(*) INTO pocet
    from predmet_bod
    where cis_predm = cp
      and skrok = rok;
```

Vyvolanie užívateľskej výnimky - pokračovanie.

```
  IF (pocet = 0) THEN
    RAISE_APPLICATION_ERROR(-20002, 'Not existing subject for the year');
  END IF;

  INSERT INTO zap_predmety (os_cislo, cis_predm, skrok, prednasajuci, ects )
    SELECT oc, cp, rok, garant, ects
      FROM predmet_bod
      WHERE cis_predm = cp
        AND skrok = rok;
END;
/
```

Vyvolanie a spracovanie užívateľskej výnimky.

```
create or replace procedure zapis_predmet(
     oc  student.os_cislo%TYPE,
     cp  predmet.cis_predm%TYPE,
     rok zap_predmety.skrok%TYPE
   )
AS
  err1    EXCEPTION;
  err2    EXCEPTION;
  err3    EXCEPTION;

  pocet   integer;
BEGIN
  SELECT count(*) INTO pocet
    from student
    where os_cislo = oc;

  IF ( pocet = 0 ) THEN
     RAISE err1;
  END IF;

  SELECT count(*) INTO pocet
    from predmet
    where cis_predm = cp;

  IF ( pocet = 0 ) THEN
     RAISE err2;
  END IF;

  SELECT count(*) INTO pocet
    from predmet_bod
    where cis_predm = cp
      and skrok = rok;
```

Vyvolanie a spracovanie užívateľskej výnimky - pokračovanie.

```
  IF ( pocet = 0 ) THEN
     RAISE err3;
  END IF;

  INSERT INTO zap_predmety (os_cislo, cis_predm, skrok, prednasajuci, ects )
    SELECT oc, cp, rok, garant, ects
      FROM predmet_bod
      WHERE cis_predm = cp
        AND skrok = rok;

EXCEPTION
  WHEN err1 THEN
    dbms_output.put_line('Not existing student');
  WHEN err2 THEN
    dbms_output.put_line('Not existing subject');
  WHEN err3 THEN
    dbms_output.put_line('Not existing subject for the year');
  WHEN others THEN
    dbms_output.put_line('Other error');
END;
/

SQL> execute zapis_predmet ( 501555, 'BI06', 2015 );
Not existing subject for the year
```

- 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

Trigger

Syntax

```
CREATE  [OR REPLACE]  TRIGGER [schema.] trigger
 BEFORE | AFTER
  DELETE | INSERT | UPDATE [ OF stlpec1 [, stlpec2 [,...] ] ]
   [ OR DELETE | INSERT | UPDATE [ OF stlpec1 [, stlpec2 [,...] ] ] ] [...]
|
 INSTEAD OF DELETE | INSERT | UPDATE
ON [schema.] tabulka
[ REFERENCING  OLD [AS] stary | NEW [AS] novy]
[ FOR EACH ROW ]
[ WHEN (podmienka)]
Telo triggra
```

- 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 položkami alebo záznamami je vždy potrebné dávať :

```
(:new.os_cislo)
```

Príklad - rozdiel medzi s FOR EACH ROW a bez FOR EACH ROW

1. Príprava tabuľky.

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

2. Vytvorenie triggra na logovanie.

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

b)
```
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;
```

3.
```
SQL>update zap_predmety
  2  set    skrok = 2002
  3  where  skrok = 2000;
7 rows updated
```

4. 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, vypnutie a zrušenie 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:

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

```
SELECT trigger_name FROM user_triggers;
```

Definovanie AUTOINCREMENT stĺpca

1. Vytvorenie SEQUENCE

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

2. Vytvorenie tabuľky

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

3.
```
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;
```

1. Default hodnota

```
create table T1
(
   id    integer not null primary key,
   cislo integer  DEFAULT 3
);
```

2.
```
insert into T1 values ( 1, 5 );
insert into T1(id) values (2);
insert into T1 values ( 3, null );
```

3.
```
SQL> select * from t1;

        ID      CISLO
---------- ----------
         1          5
         2          3
         3
```

1.
```
create table T1
(
   id    integer not null primary key,
   cislo integer
);
```

2.
```
create or replace trigger T1_default
 before insert on T1
 referencing NEW as novy
 for each row
begin
  if (:novy.cislo IS NULL) then
      :novy.cislo := 3;
  end if;
end;
/
```

3.
```
insert into T1 values ( 1, 5 );
insert into T1(id) values (2);
insert into T1 values ( 3, null );
```

4.
```
SQL> select * from t1;

        ID      CISLO
---------- ----------
         1          5
         2          3
         3          3
```

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;
/
```

Zabránenie niektorým užívateľom meniť hodnoty primárneho kľúča

```
create or replace trigger st_oc
  before UPDATE OF 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 triggra.

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

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

  delete from zap_predmety
  where os_cislo = :old.os_cislo;
end;
/
```