Triggre.

Triggre sú podobné uloženým podprogramom, až na nasledujúce výnimky:

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

Triggre sú to najpodstatnejšie pri vývoji produktívnych dátovo-orientovaných systémov, pretože zaisťujú nasledujúce úlohy:

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

Poznámka:

Pretože sa triggre vyvolávajú implicitne, nepoužívajte pri implementácii rekurzívne volania, t.j. triggre volajúce iné triggre.

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

kde

OR REPLACE – predefinovanie triggra, ak už existuje

BEFORE / AFTER – kedy sa má trigger spúšťať pred operáciou , alebo po špecifikovanej DML operácii

INSTEAD OF - trigger, ktorý má preddefinovať operácie INSERT, DELETE, UPDATE. Využíva sa hlavne pri pohľadoch.

REFERENCING – definovanie premennej reprezentujúcej záznam pre nový, príp. pôvodný riadok.

FOR EACH ROW – ak operácia pracuje s viacerými riadkami relácie, trigger sa vykoná pre každý riadok zvlášť

WHEN - dodatočná podmienka spustenia triggra

Obmedzenia pri vytváraní triggra:

Pre sémantiku triggra platia nasledujúce obmedzenia:

- 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* ;

Príklady:

- ## DEFINOVANIE „AUTOINCREMENT STĹPCA“ POMOCU SEQUENCE V TABUĽKE

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);
```

Vytvorenie triggra pre automatické vloženie nasledujúcej hodnoty sequence do stĺpca id.

```
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;
```

- ## LOGOVANIE V RÁMCI JEDNEJ TABUĽKY

Najprv pridajte do tabuľky zap_predmety stĺpce, kde budete evidovať, kto naposledy menil údaje daného riadku.

```
ALTER TABLE zap_predmety
ADD ( uziv      varchar2(15),
      datum_zm  date);
```

Definovanie triggra, ktorý sa bude spúšťať pred operáciami INSERT a UPDATE. Je dôležité si všimnúť, že napriek tomu, že ak sa užívateľ snaží napísať tam niekoho iného, alebo prípadne iný dátum modifikácie, nemá šancu. Bude zaznamenaný užívateľ, ktorý je prihlásený a systémový dátum.

Pomocou selectu zmeníme hodnoty uziv a datum_zm pred operáciou.

```
CREATE OR REPLACE TRIGGER zap_predmety_log
  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 KTO MENIL, VKLADAL DÁTA DO POMOCNEJ TABUĽKY

Niečo podobné ako predchádzajúci príklad, ale tento krát si vytvoríme tabuľku, kde budeme zaznamenávať, kto a kedy posledne updatoval riadky v tabuľke zap_predmety.

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

Definujeme trigger, ale tentokrát nemáme klauzulu FOR EACH ROW a vkladáme informáciu do tabuľky log_table_zp. (V tomto prípade je to jedno, či AFTER, alebo BEFORE)

```
CREATE OR REPLACE TRIGGER t_log_zp
AFTER UPDATE ON zap_predmety
begin
  INSERT INTO log_table_zp
  VALUES (USER, SYSDATE);
END;
```

Ak zmeníme jedným príkazom viac riadkov tabuľky zap_predmety, napr. :

```
SQL>UPDATE zap_predmety
  2  SET zp_skrok = 2002
  3  WHERE zp_skrok = 2000;

7 rows updated.
```

Potom zistíme, že do tabuľky log_table_zp bol **vložený LEN JEDEN RIADOK**, napriek tomu, že príkaz UPDATE **zmenil VIAC RIADKOV**. Dôvodom je práve vynechanie klauzuly FOR EACH ROW.

```
SQL> SELECT * FROM log_table_zp;
```

- ## LOGOVANIE KTO A KOHO MENIL, VKLADAL, VYMAZÁVAL

Ďalší variant logu, do log_table uchovávať aj informácie, kto (študent s akým osobným číslom) bol vkladaný, prípadne vymazaný z tabuľky študent.

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

Trigger pre operáciu insert.

```
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 pre operáciu delete.

```
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;
```

- ## 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;
```

Trigger, ktorý odloží vymazávaný riadok do pomocnej tabuľky.

```
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;
```

- ## ZABRÁNENIE NIEKTORÝM UŽÍVATEĽOM MENIŤ HODNOTY PRIMÁRNEHO KĽÚČA

  Len užívateľ vajsova bude mať právo meniť hodnotu osobného čísla študenta. Iným užívateľom vyvolá takýto pokus výnimku.

```
CREATE OR REPLACE TRIGGER st_oc
  BEFORE UPDATE OF st_os_cislo ON student
  FOR EACH ROW
  WHEN (USER NOT IN 'VAJSOVA')
BEGIN
  RAISE_APPLICATION_ERROR(-20000,'ERROR - NEMOZES MENIT OS_CISLO');
END;
```

- ## DEFINOVANIE KASKÁDY PRE DELETE POMOCOU TRIGGRU

Definícia triggra, ktorý autormaticky vymaže referencované riadky v tabuľke zap_predmety, pri vymazaní z tabuľky študent. Počet vymazaných riadkov sa zobrazí na konzole, pomocou dbms_ouptup.put_line.

```
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 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;
```

Aby sa „hláška“ skutočne aj objavila na konzole, je nevyhnutné nasledovné nastavenie: (Stačí jeden krát počas prihlásenia)

```
SQL>SET SERVEROUTPUT  ON
```

Pohľady.

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]

kde

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

- VYTVORENIE JEDNODUCHÉHO POHĽADU

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

Nie je možná operácia INSERT, pretože tento pohľad neobsahuje primárny kľúč tabuľky os_udaje.

UPDATE, DELETE – tak ako nad tabuľkou so stĺpcami ou_meno, ou_priezvisko.

- PREDEFINOVANIE EXISTUJÚCEHO POHĽADU

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

Je možná operácia INSERT, pretože tento pohľad obsahuje primárny kľúč a všetky not null stĺpce tabuľky os_udaje.

- POHĽAD S PREMENOVANÍM 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

```
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%';
```

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');
```

- ODSTRÁNENIE PROBLÉMU INSERTU – POHĽAD S PODMIENKOU

```
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 CHECK OPTION;
```

Tento insert už nefunguje

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

Tento insert je v poriadku

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

- ZAKÁZANIE INSERT, DELETE A UPDATE NAD POHĽADOM

```
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

```
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, 100, 101);
//  100,101 …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

Definujte Trigger, ktorý zabezpečí vymazanie z oboch tabuliek

```
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 pohl4
WHERE rod_cislo = '0512224/0000';
```

- 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;
```

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');
```

- POUŽITIE POHĽADU V TRIGGROCH

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

Trigger pre insert

```
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 – CASCADE A LOCAL CHECK OPTION (SQL92)

```
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;


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;
```

Podprogramy: procedúra a funkcia.

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;

Replace voľba určuje, že procedúra bude zrušená v prípade, ak už existuje a bude nahradená novou verziou definovanou príkazom

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
Hodnota odovzdávaná do podprogramu	Hodnota vrátená do volaného prostredia	Hodonota odovzdaná do podprogramu a vrátená do volaného prostredia
Formálny parameter sa chova ako konštanta	Neinicializovaná premenná	Inicializovaná premenná
Parameter môže byť literal, vyraz, konštanta alebo inicializovaná premenná	Musí byť premenná	Musí byť premenná

Príklad:

```
SQL> CREATE OR REPLACE PROCEDURE QUERY_EMP
1 (V_ID IN EMP.EMPNO%TYPE,
2 V_NAME OUT EMP.ENAME%TYPE,
3 V_SALARY OUT EMP.SAL%TYPE,
4 V_COMM OUT EMP.COMM%TYPE)
5 IS
6 BEGIN
7 SELECT ENAME, SAL, COMM
8 INTO V_NAME, V_SALARY, V_COMM
9 FROM EMP
10 WHERE EMPNO = V_ID;
11 END QUERY_EMP;
12 /

SQL> VARIABLE G_NAME VARCHAR2(15)
SQL> VARIABLE G_SAL NUMBER
SQL> VARIABLE G_COMM NUMBER

SQL> EXECUTE QUERY_EMP(7654,:G_NAME,:G_SAL,:G_COMM);
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

SQL> PRINT G_NAME
G_NAME
---------------
MARTIN
```

Príklad:

```
SQL> CREATE OR REPLACE PROCEDURE FORMAT_PHONE
2 (V_PHONE_NO IN OUT VARCHAR2)
3 IS
4 BEGIN
5 V_PHONE_NO := '(' || SUBSTR(V_PHONE_NO,1,2) ||
6 ')' || SUBSTR(V_PHONE_NO,3,3) ||
7 '-' || SUBSTR(V_PHONE_NO,6);
8 END FORMAT_PHONE;
9 /

SQL> VARIABLE G_PHONE_NO VARCHAR2(15)
SQL> BEGIN :G_PHONE_NO := '41633057'; END;
2 /
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

SQL> PRINT G_PHONE_NO
G_PHONE_NO
---------------
41633057

SQL> EXECUTE FORMAT_PHONE (:G_PHONE_NO)
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

SQL> PRINT G_PHONE_NO
G_PHONE_NO
---------------
(41)633-057
```

Príklad pre funkciu:

```
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 /

SQL> VARIABLE G_SALARY NUMBER

SQL> EXECUTE :G_SALARY := GET_SAL(7934)
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

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

Príklad:

```
SQL> CREATE OR REPLACE PROCEDURE ADD_DEPT
1 (V_NAME IN DEPT.DNAME%TYPE DEFAULT 'UNKNOWN',
2 V_LOC IN DEPT.LOC%TYPE DEFAULT 'UNKNOWN')
3 IS
4 BEGIN
5 INSERT INTO DEPT
6 VALUES (DEPT_DEPTNO.NEXTVAL, V_NAME, V_LOC);
7 END ADD_DEPT;
8 /

SQL> BEGIN
2 ADD_DEPT;
3 ADD_DEPT ( 'TRAINING', 'NEW YORK');
4 ADD_DEPT ( V_LOC => 'DALLAS', V_NAME =>'EDUCATION');
5 ADD_DEPT ( V_LOC => 'BOSTON') ;
6 END;
7 /
PL/SQL PROCEDURE SUCCESSFULLY COMPLETED.

SQL> SELECT * FROM DEPT;
DEPTNO DNAME LOC
------ -------------- -------------
... ... ...
41 UNKNOWN UNKNOWN
42 TRAINING NEW YORK
43 EDUCATION DALLAS
44 UNKNOWN BOSTON
```

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:

Príklad:

```
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;
```