Základy PL/SQL

PL/SQL

PL/SQL je rozšíření jazyka SQL o procedurální rysy. Je specifické pro produkty firmy Oracle, procedurální rozšíření SQL produktů jiných firem se zpravidla navzájem liší. Výjimkou je ŠRBD DB2 společnosti IBM, který podporuje jak vlastní procedurální jazyk SQL PL, tak je plně kompatibilní s jazykem PL/SQL včetně datových typů. Základním stavebním kamenem PL/SQL je tzv. **PL/SQL blok**, který může být buď tělem triggeru, procedury a funkce, nebo samostatný. Struktura PL/SQL bloku je následující:

DECLARE

```
deklarace konstant a proměnných;
```

BEGIN

```
výkonné příkazy bloku;
```

EXCEPTION

```
výkonné příkazy obsluh přerušení;
```

END;

Sekce `EXCEPTION` je učena pro ošetření výjimečných situací, zeleně zvýrazněné sekce jsou nepovinné. Pro lepší představu si prohlédněte následující ukázku:

```
DECLARE
  numerator   NUMBER;
  denominator NUMBER;
  the_ratio   NUMBER;
  lower_limit CONSTANT NUMBER := 0.72;
  samp_num    CONSTANT NUMBER := 132;

BEGIN
  SELECT x, y INTO numerator, denominator
    FROM result_table
   WHERE  sample_id = samp_num;

  the_ratio := numerator / denominator;

  IF the_ratio > lower_limit THEN
    INSERT INTO ratio VALUES (samp_num, the_ratio);
  ELSE
    INSERT INTO ratio VALUES (samp_num, -1);
  END IF;

  COMMIT;

EXCEPTION
  WHEN ZERO_DIVIDE THEN
    INSERT INTO ratio VALUES (samp_num, 0);
    COMMIT;

  WHEN OTHERS THEN
    ROLLBACK;
END;
/
```

Tento příklad ukazuje tzv. **anonymní PL/SQL blok**. Při použití SQL konzole je PL/SQL blok spuštěn teprve po zadání lomítka na začátku samostatného řádku.

Datové typy podporované v jazyce PL/SQL

V jazyce PL/SQL lze použít kromě datových typů jazyka SQL též další speciální datové typy. Následující tabulka ukazuje některé z nich:

Datové typy z SQL	Speciální PL/SQL typy	Složené datové typy
NUMBER	BINARY_INTEGER	RECORD
REAL	PLS_INTEGER	TABLE
FLOAT	BOOLEAN	VARRAY
CHAR		
VARCHAR2		
DATE		
BLOB		

Řídící struktury programu

Větvení programu - konstrukce `IF-THEN-ELSE`

Základní zápis

```
IF podmínka THEN
   příkaz_1;
   příkaz_2;
END IF;
```

Ekvivalentní zápis

```
IF podmínka THEN
   příkaz_1;
ELSE
   příkaz_2;
END IF;
```

```
IF podmínka_1 THEN
   příkaz_1;
ELSIF podmínka_2 THEN
   příkaz_2;
ELSIF podmínka_3 THEN
   příkaz_3;
ELSE
   příkaz_4;
END IF;
```

```
IF podmínka_1 THEN
  příkaz_1;
ELSE
  IF podmínka_2 THEN
    příkaz_2;
  ELSE
    IF podmínka_3 THEN
      příkaz_3;
    ELSE
      příkaz_4;
    END IF;
  END IF;
END IF;
```

Větvení programu - konstrukce `CASE`

Konstrukce IF

```
IF podmínka_1 THEN
   příkaz_1;
ELSIF podmínka_2 THEN
   příkaz_2;
ELSIF podmínka_3 THEN
   příkaz_3;
ELSE
   příkaz_4;
END IF;
```

Konstrukce CASE

```
CASE
  WHEN podmínka_1 THEN příkaz_1;
  WHEN podmínka_2 THEN příkaz_2;
  WHEN podmínka_3 THEN příkaz_3;
  ELSE příkaz_4;
END CASE;
```

Iterace a smyčky - konstrukce `LOOP`

Základní zápis

```
LOOP
  .
  .
  .
  IF podminka_1 THEN
    EXIT;
  END IF;
  .
  .
  .
  IF podminka_2 THEN
    CONTINUE;
  END IF;
  .
  .
  .
END LOOP;
```

Ekvivalentní zápis

```
LOOP
  .
  .
  .
  EXIT WHEN podminka_1;
  .
  .
  .
  CONTINUE WHEN podminka_2;
  .
  .
  .
END LOOP;
```

Iterace a smyčky - konstrukce `WHILE-LOOP`

```
WHILE podmínka LOOP
  .
  .
  příkaz;
  .
  .
END LOOP;
```

Iterace a smyčky - konstrukce `FOR-LOOP`

```
FOR i IN 1..10 LOOP
  příkazy cyklu;
END LOOP;

FOR i IN REVERSE 1..10 LOOP
  příkazy cyklu;
END LOOP;

SELECT COUNT(os_cislo) INTO pocet FROM osoby;
FOR i IN 1..pocet LOOP
```

```
  ...
END LOOP;
```

Skoky - konstrukce `GOTO`

```
BEGIN
  ...
  GOTO vloz_zaznam;
  ...
  <<vloz_zaznam>>;
  INSERT INTO osoby VALUES ...
  ...
END;
```

Prázdný příkaz - konstrukce `NULL`

```
BEGIN
  ...
  IF a > 10 THEN
    -- toto je komentář
    NULL;
  ELSE
    x := a - 10;
  END IF;
  ...
END;
```

Strukturované datové typy

V PL/SQL blocích se velmi často pracuje s objekty typu `RECORD`, které obsahují např. celé záznamy uložené v tabulce. Pro získání odpovídající hodnoty je použit příkaz `SELECT` doplněný o specifickou klauzuli `INTO`.

```
DECLARE
  TYPE TZam IS RECORD (
    os_cislo NUMBER(5),
    prijmeni VARCHAR2(30),
    jmeno    VARCHAR2(30)
  );

  zam1 TZam;
  zam2 TZam;

BEGIN
  SELECT os_cislo, prijmeni, jmeno INTO zam1
    FROM zamestnanci
   WHERE os_cislo = 123;

  zam2 := zam1;
  zam2.os_cislo := zam1.os_cislo + 1;
  zam2.prijmeni := 'MARTAN';

  INSERT INTO zamestnanci (os_cislo, prijmeni, jmeno)
  VALUES (zam2.os_cislo, zam2.prijmeni, zam2.jmeno);

  COMMIT;
```

```
END;
```

Deklarace z výše uvedeného příkladu lze napsat i takto:

```
DECLARE
  TYPE TZam IS RECORD (
    os_cislo  zamestnanci.os_cislo%TYPE,
    prijmeni  zamestnanci.prijmeni%TYPE,
    jmeno     zamestnanci.jmeno%TYPE
  );

  zam1 TZam;
  zam2 TZam;
...
```

Pseudoproměnná `%TYPE` představuje datový typ sloupce tabulky. V deklaracích lze použít i pseudoproměnnou `%ROWTYPE` spojenou s tabulkou nebo pohledem. Představuje strukturovaný datový typ `RECORD` se strukturou shodnou s řádkem dané tabulky. Výše uvedené deklarace lze tedy zjednodušit takto:

```
DECLARE
  zam1 zamestnanci%ROWTYPE;
  zam2 zamestnanci%ROWTYPE;
...
```

Zpracování výjimek - sekce `EXCEPTION`

Při zpracování PL/SQL bloku může nastat výjimečná situace. SŘBD Oracle má definovánu velkou množinu výjimečných situací, které sám obsluhuje, ale mi jeho obsluhu můžeme doplnit o své příkazy. Následující příklad ukazuje odchycení a obsluhu některých systémových výjimek.

```
DECLARE
  TYPE TZam IS RECORD (
    prijmeni  osoby.prijmeni%TYPE;
    jmeno     osoby.jmeno%TYPE;
  );

  zam Tzam;

BEGIN
  SELECT prijmeni, jmeno INTO zam FROM osoby WHERE os_cislo = 123;
...
  INSERT INTO osoby (os_cislo, prijmeni, jmeno)
  VALUES (123, 'PESICKA', 'Ladislav');
...
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      dbms_output.put_line('Osoba nebyla nalezena');

    WHEN DUP_VAL_ON_INDEX THEN
      dbms_output.put_line('Duplicitni osobni cislo');
...
END;
```

Kromě systémových výjimečných situací můžeme definovat též uživatelské výjimečné situace, které je možné též v sekci `EXCEPTION` obsloužit. Uživatelská výjimka je datového typu `EXCEPTION`.

```
DECLARE
  zam_plat   NUMBER(5,2);
  novy_plat  NUMBER(5,2);
  nema_plat  EXCEPTION;
```

```
BEGIN
  SELECT plat INTO zam_plat FROM osoby WHERE os_cislo = 123;
  IF zam_plat IS NULL THEN
    RAISE nema_plat;
  END IF;
...
  EXCEPTION
    WHEN nema_plat THEN
      UPDATE osoby SET plat = novy_plat WHERE os_cislo = 123;
      COMMIT;
...
END;
```

Chceme-li odchytit všechny (popř. ostatní) výjimky, použijeme konstrukci `EXCEPTION WHEN OTHERS THEN`. Bližší informace o chybě jsou uloženy do pseudoproměnných `SQLCODE` a `SQLERRM`, které obsahují číselný kód a popis poslední chyby:

```
DECLARE
...
BEGIN
...
EXCEPTION WHEN OTHERS THEN
   dbms_output.put_line('Nastala chyba ! Blizsi info o chybe :');
   dbms_output.put_line('Cislo chyby : '||to_char(SQLCODE));
   dbms_output.put_line('Popis chyby : '||SQLERRM);

END;
```

Poznámka:

- při ladění lze využít standardního balíku DBMS_OUTPUT pro ladicí výpisy. Výstup na konzoli se však musí povolit příkazem `SET SERVEROUTPUT ON`. Jak provádět ladící výpisy je vidět z uvedených příkazů.
- **Pozor !** Výpis je proveden až po skončení PL/SQL bloku.

Voláním standardní procedury `raise_application_error` lze vypropagovat výjimku přímo do aplikace volající anonymní PL/SQL blok, proceduru nebo funkci. Voláním této funkce je ukončeno zpracování PL/SQL bloku, procedury nebo funkce. Pro tyto výjimky jsou rezervována čísla chyb od -20000 do -20999:

```
DECLARE
...
BEGIN
...
  IF plat IS NULL THEN
    raise_application_error(-20005, 'Plat neni vyplnen !', TRUE);
  END IF;
...
END;
```

Uložené procedury a funkce

Všechny výše uvedené anonymní PL/SQL bloky lze natrvalo uložit do databáze, a to buď ve tvaru procedury nebo funkce. Způsob vytvoření uložené procedury ukazuje následující příklad:

```
CREATE OR REPLACE PROCEDURE kopiruj_zamestnance AS
  zam1 zamestnanci%ROWTYPE;
  zam2 zamestnanci%ROWTYPE;
```

```
BEGIN
  SELECT os_cislo, prijmeni, jmeno INTO zam1
    FROM zamestnanci
   WHERE os_cislo = 123;

  zam2 := zam1;
  zam2.os_cislo := zam1.os_cislo + 1;
  zam2.prijmeni := 'MARTAN';

  INSERT INTO zamestnanci (os_cislo, prijmeni, jmeno)
  VALUES (zam2.os_cislo, zam2.prijmeni, zam2.jmeno);
  COMMIT;
END;
```

Pokud uložení procedury nebo funkce neproběhlo bez chyb, nelze ji používat a je nutné ji opravit. Ke zjištění, jaká/é chyba/y se vyskytla/y slouží v programu SQL*Plus příkaz `SHOW ERRORS`. Ten vypíše popis poslední chyby, na kterou při ukládání (kompilaci) narazil. Proto je vhodné tento postup opakovat tak dlouho, dokud proces ukládání proběhne bez chyb.

Proceduru můžete zavolat (spustit) různými způsoby:

- pomocí direktivy EXEC

  ```
  EXEC kopiruj_zamestnance;
  ```

- v těle jiného PL/SQL bloku

  ```
  BEGIN
    ...
    kopiruj_zamestnance;
    ...
  END;
  /
  ```

Uložené procedury a funkce mohou mít parametry, které mohou být:

- `IN` - vstupní
- `OUT` - výstupní
- `IN OUT` - vstupní i výstupní

Procedura se vstupními parametry:

```
CREATE OR REPLACE PROCEDURE nastav_plat (id IN NUMBER, novy_plat IN NUMBER) AS
  zam_plat  NUMBER(5,2);
  nema_plat EXCEPTION;
BEGIN
  SELECT plat INTO zam_plat FROM osoby WHERE os_cislo = id;

  IF zam_plat IS NULL THEN
    RAISE nema_plat;
  END IF;

EXCEPTION
  WHEN nema_plat THEN
    UPDATE osoby SET plat = novy_plat WHERE os_cislo = id;
    COMMIT;
END;
```

Volání procedury s parametry:

```
EXEC nastav_plat(123, 10000);
```

Procedura se vstupně-výstupním parametrem:

```
CREATE OR REPLACE PROCEDURE inc(a IN OUT INTEGER) AS
BEGIN
  a := a + 1;
END;
```

Volání procedury s vstupně-výstupním parametrem:

```
DECLARE
  a INTEGER;
BEGIN
  a := 9;
  inc(a);
  dbms_output.put_line('Vysledek : '||a);
END;
```

Poznámka: Před vykonáním tohoto PL/SQL bloku je nutné zadat `SET SERVEROUTPUT ON`.

Způsob vytvoření uložené funkce (s parametry) ukazuje následující příklad:

```
CREATE OR REPLACE FUNCTION secti(a IN INTEGER, b IN INTEGER) RETURN INTEGER AS
BEGIN
  RETURN (a + b);
END;
```

Stejně jako proceduru lze funkci volat dvěma různými způsoby:

- Volání funkce v příkazu SELECT

  ```
  SELECT secti(2, 3) FROM dual;
  ```

- Volání v těle jiného PL/SQL bloku

  ```
  DECLARE
    a INTEGER;
    b INTEGER;
  BEGIN
    a := 5;
    b := secti(a, 2);
    dbms_output.put_line('Vysledek : '||b);
  END;
  ```

Následující příklad ukazuje dva různé způsoby přiřazení skutečných parametrů formálním parametrům při volání procedur a funkcí:

```
CREATE OR REPLACE FUNCTION spoj(a IN VARCHAR2, b IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN (a||' a '||b);
END;
```

V prvním volání funkce SPOJ je přiřazení provedeno na základě pozice, v druhém volání na základě jména:

```
DECLARE
  tmp  VARCHAR2(100);
BEGIN
  tmp := spoj('Prvni', 'Druhy');
  dbms_output.put_line('Vysledek : '||tmp);

  tmp := spoj(b => 'Prvni', a => 'Druhy');
  dbms_output.put_line('Vysledek : '||tmp);
END;
```

Pro zrušení (smazání) uložené procedury nebo funkce použijeme odpovídající příkaz `DROP PROCEDURE`

nebo `DROP FUNCTION`:

```
DROP PROCEDURE kopiruj_zamestnance;

DROP FUNCTION secti;
```

Uložené balíky procedur a funkcí

Uložené balíky procedur a funkcí slouží ke sdružení logicky spolu souvisejících procedur a funkcí. Mohou obsahovat i globální proměnné, jejichž platnost je omezena délkou aktuálního spojení s databází. Definice balíku představuje definici rozhraní balíku a těla balíku:

Rozhraní

```
CREATE OR REPLACE PACKAGE arithmetic AS
  usage INTEGER;

  FUNCTION  add(a IN INTEGER, b IN INTEGER) RETURN INTEGER;
  FUNCTION  sub(a IN INTEGER, b IN INTEGER) RETURN INTEGER;
  PROCEDURE inc(a IN OUT INTEGER);
END;
/
```

Tělo

```
CREATE OR REPLACE PACKAGE BODY arithmetic AS

  FUNCTION  add(a IN INTEGER, b IN INTEGER) RETURN INTEGER IS
  BEGIN
    usage := usage + 1;
    RETURN (a + b);
  END;

  FUNCTION  sub(a IN INTEGER, b IN INTEGER) RETURN INTEGER IS
  BEGIN
    usage := usage + 1;
    RETURN (a - b);
  END;

  PROCEDURE inc(a IN OUT INTEGER) IS
  BEGIN
    usage := usage + 1;
    a := a + 1;
  END;

BEGIN
  usage := 0;
END;
/
```

Příklad použití balíku:

```
DECLARE
  a INTEGER;
  b INTEGER;
BEGIN
```

```
  a := 6;
  b := arithmetic.add(a, 3);
  arithmetic.inc(b);
  dbms_output.put_line('Vysledek : '||b);
  dbms_output.put_line('Pouzito '||arithmetic.usage||' krat.');
END;
```

Zrušení balíku procedur a funkcí je v jazyce PL/SQL reprezentován dvojicí příkazů. `DROP PACKAGE` pro zrušení celého balíku (tj. rozhraní a tělo) a `DROP PACKAGE BODY` pro zrušení těla balíku. Rozhraní balíku může existovat bez těla.

```
DROP PACKAGE BODY arithmetic;

DROP PACKAGE arithmetic;
```

Řízení přístupu k uloženým procedurám, funkcím a balíkům

Stejně jako u tabulek a pohledů lze řídit přístup k uloženým procedurám, funkcím a balíkům pomocí příkazu GRANT. Přidělované/odebírané právo je jen jediné, a to `EXECUTE`:

```
GRANT EXECUTE ON secti TO PUBLIC;

REVOKE EXECUTE ON spoj FROM jarda;

GRANT EXECUTE ON arithmetic TO PUBLIC;
```

Poznámka: Nelze řídit přístup k jednotlivým procedurám a funkcím v balíku.

Pokud chceme, aby se procedura spouštěla s právy volajícího, použijeme klauzule `AUTHID CURRENT_USER`, jinak se spouští s právy toho, kdo proceduru vytvořil (implicitní klausule `AUTHID DEFINER`):

```
CREATE OR REPLACE PROCEDURE nove_heslo(pwd IN VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
  EXECUTE IMMEDIATE 'ALTER USER '||user||' IDENTIFIED BY '||pwd;
END;
```

a takto přidělíme právo spuštění každému uživateli v databázi ...

```
GRANT EXECUTE ON nove_heslo TO PUBLIC;
```

... a takto si můžeme změnit svoje heslo:

```
EXEC ZIMA.nove_heslo ('heslo');
```

Kontrolní úloha

Vytvořte tabulku `POKUS`, která bude mít sloupce:

- `kod` (celé číslo, primární klíč)
- `hodnota` (textová položka max. délky 50 znaků)

Vytvořte proceduru s parametrem (vstupní textový parametr), která do tabulky `POKUS` zapíše do sloupce `hodnota` text zadaný v parametru procedury a do sloupce `kod` vygeneruje pořadové číslo (řada 1, 2, ...).

Nápověda: Vložení prvního záznamu tabulky realizujte obsluhou uživatelem definované výjimečné

situace.