Proměnné, výrazy a typy v PL/SQL

Deklarace proměnných

- Proměnné je nutné před prvním použitím vždy deklarovat:
  - V anonymním bloku a triggeru za klíčovým slovem **DECLARE**,
  - ve funkcích a procedurách za hlavičkou – slovem **AS** nebo **IS**.
- Během deklarace je možné proměnnou inicializovat, případně určit, že nesmí nabývat hodnoty NULL. Pokud je proměnná NOT NULL, musí být inicializována na výchozí hodnotu pomocí DEFAULT nebo přiřazení (operátoru :=).
- Syntaxe: `<název proměnné> <typ> [{NOT NULL DEFAULT <výchozí hodnota> | NOT NULL := <výchozí hodnota> | := <výchozí hodnota>}];`

Deklarace proměnných – příklad

```
DECLARE
  v_promenna1 NUMBER(2);
  v_promenna2 NUMBER NOT NULL DEFAULT 50;
  v_promenna3 NUMBER := 45;
  v_promenna4 VARCHAR2(10 CHAR) NOT NULL := 'Ahoj';
BEGIN
  v_promenna1 := 33;
  DBMS_OUTPUT.PUT_LINE(v_promenna1);
  DBMS_OUTPUT.PUT_LINE('Hodnota proměnné 2 je ' ||
    v_promenna2);
  DBMS_OUTPUT.PUT_LINE('Hodnota proměnné 4 je ' ||
    v_promenna4);
END;
```

Typy proměnných deklarované podle jiných typů (%TYPE)

- Typ proměnné podle typu jiné proměnné:

```
DECLARE
   v_text VARCHAR2(255);
   v_text2 v_text%TYPE;
```

- Typ proměnné podle typu sloupce tabulky:

```
DECLARE
   v_jmeno zamestnanec.jmeno%TYPE;
```

SUBTYPE

- Pokud nemůžete využít typ proměnné na základě typu sloupce tabulky, pak je best practice definovat subtype (vlastní datový typ na základě stávajícího) a použít ten.
- Například místo tohoto:

```
DECLARE
   v_jmeno VARCHAR2(20);
   v_prijmeni VARCHAR2(20);
```

- Je best practice udělat:

```
DECLARE
   SUBTYPE JMENO IS VARCHAR2(20);
   v_jmeno JMENO;
   v_prijmeni JMENO;
```

Deklarace konstant

- Konstanta se rovněž deklaruje v deklarační sekci, kde se musí konstantě zároveň přiřadit hodnota (v sekci BEGIN již nelze hodnotu měnit).
- Rovněž lze využít %TYPE.
- Syntaxe:

```
<název konstanty> CONSTANT <typ> := <hodnota>;
```

Vnořené bloky a proměnné

- V PL/SQL lze do těla jednoho bloku zanořit další blok:

```
DECLARE
  v_cislo NUMBER(10);
  v_cislo2 NUMBER(10);
BEGIN
  v_cislo := 40;
  v_cislo2 := 80;
  DECLARE
    v_cislo NUMBER(10);
  BEGIN
    v_cislo := 20;
    DBMS_OUTPUT.PUT_LINE(v_cislo);
    DBMS_OUTPUT.PUT_LINE(v_cislo2);
  END;
  DBMS_OUTPUT.PUT_LINE(v_cislo);
  DBMS_OUTPUT.PUT_LINE(v_cislo2);
END;
-- Výsledkem jsou čísla 20, 80, 40, 80
```

Vnořený blok může používat proměnné z vnějšího bloku. Může zastínit proměnné ve vnějším bloku vlastními proměnnými.

Výrazy v PL/SQL

- Skládají se z operandů a případných operátorů:
  - Operandy jsou proměnné, konstanty, literály (konkrétní čísla, řetězce, hodnota NULL), nebo volání funkcí (funkce vrátí určitou hodnotu).

Přehled operátorů

- Přehled použitelných operátorů z jazyka SQL (pro Oracle), v PL/SQL máme navíc operátor **:

Operátor	**Popis**
+, -, *, /	sčítání/kladné číslo, odčítání/záporné číslo, násobení, dělení
||	spojování řetězců
AND, OR, NOT	logické operátory (log. součin, součet, negace)
=, <, >, <=, >=, <>, !=, IS NULL, IS NOT NULL, LIKE, BETWEEN, IN	relační operátory (rovno, menší, větší, menší rovno, větší rovno, nerovno, nerovno 2. způsobem, je/není NULL, porovnání řetězce se vzorem (se zástupnými znaky _, %), je v uzavřeném intervalu, je v množině)
()	závorky pro změnu priority operátorů
**	umocňování (např. 2 ** 3), funguje jen v PL/SQL, nikoliv v SQL.

Priorita operátorů

- Operátory seřazené podle priority (od největší po nejmenší):

  **

  +, -

  *, /

  +, -, ||

  =, <, >, <=, >=, <>, !=, IS [NOT] NULL, LIKE, BETWEEN, IN

  NOT

  AND

  OR

Datové typy PL/SQL a SQL

- PL/SQL využívá stejné datové typy databáze Oracle, jaké jsou k dispozici v SQL:
  - **Skalární** – NUMBER, VARCHAR2, ...
  - **Kolekce** – TABLE, VARRAY, RECORD
  - **Referenční** – REF CURSOR, REF obj_type,
  - **LOB typy** – BFILE, BLOB, CLOB, NCLOB.
- **Rozdíly oproti SQL:**
  - Existuje datový typ BOOLEAN
  - VARCHAR2 má max. 32 767 bytů (oproti 4000 bytů u SQL – do Oracle 12c)

Kolekce I.

- **VARRAY – statické pole** – množina prvků stejného typu, s indexy (od 1). Pole má určitou velikost – aktuální počet prvků v poli, musí být specifikována max. velikost.

```
TYPE pole_cisel IS VARRAY(6) OF NUMBER(3,0);
v_pole_cisel pole_cisel;
```

- Příklad:

```
declare
  type text_varray is varray(3) of varchar2(255);
  pole text_varray := text_varray('a', 'b', 'c');
begin
  FOR i IN 1..pole.count LOOP
    DBMS_OUTPUT.PUT_LINE('[' || i || '] = ' || pole(i));
  END LOOP;
end;
```

Další metody:https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/collection_method.htm

Kolekce II.

- **TABLE – indexovaný list** – množina prvků o stejném typu (oproti poli ale nemá omezenou kapacitu):

```
TYPE name_list IS TABLE OF VARCHAR2(255);
```

- Příklad:

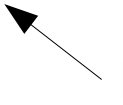
```
declare
  type varchars_table is table of varchar2(255);
  jmena varchars_table;
begin
  jmena := varchars_table('Jirka', 'Michal', 'Ales');
  jmena.extend();
  jmena(jmena.count) := 'Xavier';
  for i in jmena.first..jmena.last loop
    DBMS_OUTPUT.PUT_LINE(jmena(i));
  end loop;
end;
```

Další metody:https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/collection_method.htm

Kolekce III.

- **TABLE OF NUMBER INDEX BY – hash mapa** – množina dvojic klíč-hodnota:


```
TYPE countries_table IS TABLE OF NUMBER INDEX BY VARCHAR2(255);
```



- Příklad:

```
declare
  type countries_table is table of number index by varchar2(255);
  countries countries_table;
begin
  countries('EU') := 580;
  countries('USA') := 320;
  countries('Russian Federation') := 144;
  DBMS_OUTPUT.PUT_LINE('USA population: '
                       || countries('USA') || ' mil.');
end;
```

Kolekce IV.

- Jak vypsat všechny klíče a hodnoty hash mapy?

```
declare
  type countries_table is table of number index by varchar2(255);
  countries countries_table;
  c_idx varchar2(255);
begin
  countries('EU') := 580;
  countries('USA') := 320;
  countries('Russian Federation') := 144;
  c_idx := countries.first;
  while c_idx is not null loop
    dbms_output.put_line(c_idx || ' population: ' || countries(c_idx));
    c_idx := countries.next(c_idx);
  end loop;
end;
```

Kolekce & množinové operátory I.

- V SQL jsou množinové operátory UNION (ALL), INTERSECT a MINUS. Při práci s kolekcemi je možné použít tyto operátory:
  - SET(kolekce): odstraní z kolekce duplicity.
  - CARDINALITY(kolekce): vrátí počet záznamů v kolekci. Užitečné ve spojení se SET (abychom získali unikátní záznamy):
    - Použití: CARDINALITY(SET(kolekce))
    - Stejné jako: SET(kolekce).count s tím rozdílem, že když je kolekce null, pak CARDINALITY() vrací také null. Count vyhodí chybu.

```
declare
  type varchar_list is table of varchar2(255);
  list varchar_list := varchar_list('a', 'a', 'b');
begin
  dbms_output.put_line(cardinality(set(list)));
end;
```

Kolekce & množinové operátory II.

- Kolekce IS [NOT] EMPTY: Vrací jestli je kolekce prázdná nebo ne.

```
declare
  type varchar_list is table of varchar2(255);
  list varchar_list := varchar_list('a', 'b', 'c');
begin
  if list is empty then
    dbms_output.put_line('list is empty');
  else
    dbms_output.put_line('list is NOT empty');
  end if;
end;
```

Kolekce & množinové operátory III.

- Element IS MEMBER OF kolekce: Vrací jestli je element součástí kolekce nebo ne.

```
declare
  type varchar_list is table of varchar2(255);
  list varchar_list := varchar_list('a', 'b', 'c');
  item varchar(255) := 'a';
begin
  if item member of list then
    dbms_output.put_line('"' || item || '" is in list');
  else
    dbms_output.put_line('"' || item || '" is NOT in list');
  end if;
end;
```

Kolekce & množinové operátory IV.

- Další operátory:
  - Kolekce_1 MULTISET EXCEPT kolekce_2: to samé jako UNION MINUS.
  - Kolekce_1 MULTISET INTERSECT kolekce_2: to samé jako SQL INTERSECT.
  - Kolekce_1 MULTISET UNION kolekce_2: to samé jako SQL UNION ALL. Pro odebrání duplicit (docílení stejné funkcionality jako SQL UNION) se použije: Kolekce_1 MULTISET UNION DISTINCT kolekce_2
  - Kolekce_1 SUBMULTISET OF kolekce_2: vrací jestli je kolekce_1 podmnožinou kolekce_2.

RECORD

- **RECORD – záznam**, typ složený z nadefinovaných složek různých typů (více v přednášce o kurzorech):

```
TYPE TimeRec IS RECORD (minutes INTEGER, hours INTEGER);
```

Globálně definované typy

- Vlastní často používaný datový typ lze vytvořit také jako globální pomocí příkazu CREATE [OR REPLACE] TYPE. Typ je pak použitelný v nejrůznějších procedurách a funkcích:

```
CREATE OR REPLACE TYPE MYSCHEMA.StringList
  AS TABLE OF VARCHAR2(2000);
```