Příkazy jazyka PL/SQL

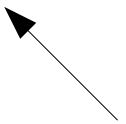
Příkazy v PL/SQL

- Lze používat:
  - Všechny možné příkazy jazyka SQL, ovšem příkaz SELECT má speciální syntaxi (SELECT ... INTO).
  - Přiřazovací příkaz (proměnná := výraz;).
  - Příkazy pro větvení programu.
  - Příkazy pro cykly.
- Příkazy ukončovat středníkem.

Komentáře v PL/SQL

```
-- jednořádkový komentář

/* víceřádkový
komentář */
```



V SQL Developer
zmáčkněte `CTRL + /`
pro (od)komentování

Řízení toku programu – větvení

```
IF podmínka1 THEN
   posloupnost_příkazů1
ELSIF podmínka2 THEN
   posloupnost_příkazů2
ELSE
   posloupnost_příkazů3
END IF;
-- větve elseif (může jich být více) a else
-- jsou nepovinné
```

Pozor!!! Opravdu ELSIF, nikoli ELSEIF

Řízení toku programu – větvení

```
CASE
  WHEN podmínka1 THEN  posloupnost_příkazů1;
  WHEN podmínka2 THEN  posloupnost_příkazů2;
  ..
  WHEN podmínkaN THEN  posloupnost_příkazůN;
  [ ELSE posloupnost_příkazůN+1; ]
END CASE;
```

- V podmínce může být např. v_promenna BETWEEN 1 AND 5 pro vyčlenění intervalu hodnot nějaké proměnné.

Řízení toku programu – cykly

- Jednoduchý cyklus LOOP (nekonečný cyklus), příkazem EXIT lze na základě určité podmínky cyklus opustit:

```
LOOP
  posloupnost_příkazů
  IF podmínka THEN
    EXIT;
  END IF;
END LOOP;

  nebo...

LOOP
  posloupnost_příkazů
  EXIT WHEN podmínka; -- cyklus s podmínkou na konci
END LOOP;
```

Poznámka: Od Oracle 11g také existují příkazy CONTINUE a CONTINUE WHEN:
http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/continue_statement.htm

Řízení toku programu – cykly

- Cyklus FOR s čítačem:

```
FOR počítadlo IN [REVERSE] nejnižší hodnota..nejvyšší
LOOP
  posloupnost_příkazů
END LOOP;
```

např.:

```
FOR v_citac IN 1..10
LOOP
  DBMS_OUTPUT.PUT_LINE('v_citac = ' || v_citac);
END LOOP;
-- Průchod v opačném pořadí od 10 do 1:
FOR v_citac IN REVERSE 1..10
LOOP
  DBMS_OUTPUT.PUT_LINE('v_citac = ' || v_citac);
END LOOP;
```

Řízení toku programu – cykly

- Cyklus WHILE, s podmínkou na začátku:

```
WHILE podmínka
LOOP
  posloupnost_příkazů
END LOOP;
```

Conditional Compilation I.

- Zejména při vývoji a debugování se používá conditional compilation. Dejme tomu, že chceme vytvořit tento kód:

```
BEGIN
  dbms_output.put_line('vlozim zaznam do destinace');
  insert into destinace (nazev) values (:nazev);
  dbms_output.put_line('zaznam uspesne vlozen');
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
  dbms_output.put_line('CHYBA! Destinace jiz existuje!');
END;
```

- Tento přístup má tu nevýhodu, že se budou logovací informace vykonávat neustále, i na produkci … což není dobře!

Conditional Compilation II.

- Část předchozího příkladu přepsána s pomocí conditional compilation vypadá takto:

```
$if $$debug_enabled $then
  dbms_output.put_line('vlozim zaznam do destinace');
$end
insert into destinace (nazev) values (:nazev);
$if $$debug_enabled $then
  dbms_output.put_line('zaznam uspesne vlozen');
$end
```

- Pro zapnutí conditional compilation je nutné spustit:

```
alter session set plsql_ccflags = 'debug_enabled:true';
```

Conditional Compilation III.

- Při conditional compilation se používají klíčová slova: `$if $then $elsif $end`. Jejich význam je stejný jako jejich alternativy bez `$`.
- `$$debug_enabled` je tzv. compilation flag. Prakticky to je proměnná typu boolean, která může nabývat hodnot `true` nebo `false`.
- Když chceme aktivovat conditional compilation uvnitř procedury / funkce apod., pak musíme zavolat:

```
alter session set plsql_ccflags = 'debug_enabled:true';
alter procedure NAZEV_PROCEDURY compile;
```

- A když chceme conditional compilation vypnout (až je kód odladěný a běží na produkci), pak zavoláme:

```
alter session set plsql_ccflags = 'debug_enabled:false';
alter procedure NAZEV_PROCEDURY compile;
```

Naplnění proměnných SELECTem

- V PL/SQL nelze používat klasickou syntaxi SELECTu, pouze SELECT s klauzulí INTO.
- Příkaz SELECT ... INTO:

```
SELECT [* | seznam_atributů]
INTO [seznam_proměnných nebo proměnná typu záznam]
FROM název_tabulky
WHERE podmínky_výběru
```

- Je potřeba, aby tento SELECT vracel právě jeden záznam, jinak by se vyvolala výjimka.
- Také lze použít SELECT vybírající více řádků, ovšem pouze v kombinaci s kurzory pro procházení vrácených záznamů.

Naplnění proměnných SELECTem – příklad

```
DECLARE
  v_jmeno zamestnanec.jmeno%TYPE;
  v_id zamestnanec.id%TYPE;
BEGIN
  SELECT jmeno, id INTO v_jmeno, v_id
  FROM zamestnanec WHERE id = 6;

  DBMS_OUTPUT.PUT_LINE('Jméno: ' || v_jmeno);
  DBMS_OUTPUT.PUT_LINE('Id: ' || v_id);
END;
```

Naplnění proměnných s ošetřením výjimek

```
DECLARE
  v_jmeno zamestnanec.jmeno%TYPE;
  v_id zamestnanec.id%TYPE;
BEGIN
  SELECT jmeno, id INTO v_jmeno, v_id
  FROM zamestnanec WHERE id = 6;
  DBMS_OUTPUT.PUT_LINE('Jméno: ' || v_jmeno);
  DBMS_OUTPUT.PUT_LINE('Id: ' || v_id);
EXCEPTION
  -- ošetření výjimky při nenalezení dat
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Data nenalezena');
  -- ošetření výjimky při nalezení více řádků
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE(
      'SELECT vybral mnoho řádků, ne jeden!');
END;
```