Összefoglaló

PL/SQL alapok: Minden utasítást pontos vesszővel zárunk le.

1. Változó deklaráció:

   <variable_name> <data_type> [:=<default_value>];

2. Paraméter megadása (adattípusra itt nem lehet megszorítás!):

```
<variable_name> [IN|OUT|IN OUT] <data_type>
                    [:=<default_value>]
```

3. Paraméter módok:
   a. `IN`: Az eljárás vagy a függvény ugyan hivatkozhat az értékre, de nem változtathatja meg azt.
   b. `OUT`: Az eljárás vagy a függvény ugyan megváltoztathatja az értékét, de nem hivatkozhat rá.
   c. `IN OUT`: Mind a kettő teljesül.

4. `GOTO` utasítás:

```
      . . .
<label_name>: <statements>
      . . .
GOTO <label_name>;
      . . .
```

5. Értékadás:

```
<variable> :=<expression>;
```

6. Blokkok:

```
DECLARE
      <deklarációs rész>
BEGIN
      <statements>
[EXCEPTION
    <kivételkezelő rész>]
END;
```

CURSOR

1. Deklaráció:
   a. Egyszerű deklaráció:

```
CURSOR <cursor_name> IS <SELECT_statement>;
```

   b. Paraméteres deklaráció:

```
CURSOR <cursor_name>(<parameter_list>)
         IS <SELECT_statement>;
```

   c. `RETURN` opcióval:

```
CURSOR <cursor_name> RETURN field%ROWTYPE
        IS <SELECT_statement>;
```

d. Kurzor típusú változók (a változó egy kurzorra mutat, az OPEN utasításkor fogjuk megadni a lekérdezést) Lehet pl. alprogram paramétere is.

Kurzor típus definiálása

```
TYPE  cursor_tipus  IS  REF CURSOR [RETURN rekordtípus]
```

A rekordtípus megadása történhet %TYPE, %ROWTYPE, vagy saját rekordtípussal. Van egy generikus rekordtípus is, amikor nem adjuk meg a RETURN részt.

Változó deklarálása

```
c_változó  cursor_tipus
```

Kurzor megnyitása, olvasása, lezárása

```
OPEN c_változó FOR SELECT ...
FETCH c_változó INTO ...
CLOSE c_változó
```

Ha a változót paraméterül adjuk át egy alprogramnak, amelyik megnyitja vagy lezárja a kurzort, akkor a paraméter IN OUT kell hogy legyen. Ha a paraméterül kapott kurzor változó nem megfelelő típusú akkor a ROWTYPE_MISMATCH hibát generálja a rendszer.

Package-ben nem deklarálhatunk REF CURSOR típusú változót, mert ezek a változók nem maradnak életben a session egész időtartama alatt, ellentétben a többi típusú változóval.

2. Kezelés:
   - **a.** Megnyitás: `OPEN <cursor_name>;`
   - **b.** Sor bekérése: `FETCH <cursor_name> INTO <list of variables>;`
   - **c.** Lezárás: `CLOSE <cursor_name>;`
3. Attribútumok:
   - **a.** `%ISOPEN`: Az értéke `TRUE`, ha a '`cursor`' meg van nyitva, és `FALSE`, ha nem.
   - **b.** `%FOUND`: Értéke `INVALID_CURSOR`, ha deklarálva van a '`cursor`', de nincs megnyitva. Értéke `NULL`, ha még nem volt '`fetch`'. Egyébként, ha van újabb sor a '`fetch`' hatására, akkor `TRUE`. Ha nincs több sor, akkor `FALSE`.
   - **c.** `%NOTFOUND`: Értéke `INVALID_CURSOR`, ha deklarálva van a '`cursor`', de nincs megnyitva. Értéke `NULL`, ha még nem volt '`fetch`'. Egyébként, ha van újabb sor a '`fetch`' hatására, akkor `FALSE`. Ha nincs több sor, akkor `TRUE`.
   - **d.** `%ROWCOUNT`: Értéke `INVALID_CURSOR`, ha deklarálva van a '`cursor`', de nincs megnyitva. Egyébként az eddig bekért sorok számával tér vissza.

Ha a SELECT utasításban szerepel FOR UPDATE utasításrész, akkor az aktív halmaz sorai zárolódnak, vagyis más felhasználó nem tudja módosítani azokat.

Az ezt követő INSERT, DELETE, UPDATE utasításban a WHERE feltétel helyett a CURRENT OF záradék használandó, ahol az utolsó FETCH utasítás értéke módosul.

ELÁGAZÁSOK

1. IF-THEN elágazás:

```
IF <condition> THEN
      <statements>
END IF;
```

2. IF-THEN-ELSE elágazás:

```
IF <condition> THEN
      <statements>
ELSE
      <statements>
END IF;
```

3. IF-ELSEIF elágazás:

```
IF <condition> THEN
      <statements>
ELSIF <condition> THEN
      <statements>
   . . .
ELSE
      <statements>
END IF;
```

CIKLUSOK

1. LOOP ciklus:

```
LOOP
      <statements>
END LOOP;
```

2. WHILE ciklus:

```
WHILE <condition>
LOOP
      <statements>
END LOOP;
```

3. FOR ciklusok:

   a. Sima változat:

```
FOR <loop_counter> IN [REVERSE]
      <lowest_number> .. <highest_number>
```

```
LOOP
      <statements>
END LOOP;
```

b. `CURSOR` változat:

```
FOR <record_index> IN <cursor_name>
LOOP
      <statements>
END LOOP;
```

4. `REPEAT` ciklusok:

```
LOOP
      <statements>
      EXIT WHEN <boolean_condition>;
END LOOP;
```

KIVÉTELEK

1. Legfontosabb kivételek:
   a. `DUP_VAL_ON_INDEX (ORA-00001)`: Olyan `UPDATE` vagy `INSERT` végrehajtása történt, amely megsérti valamelyik kulcs vagy egyediségi megszorítást.
   b. `TIMEOUT_ON_RESOURCE (ORA-00051)`: Valamilyen erőforrásra várakozás túllépte az időlimitet.
   c. `TRANSACTION_BACKED_OUT (ORA-00061)`: Egy távoli tranzakció egy része vissza lett vonva.
   d. `INVALID_CURSOR (ORA-01001)`: Nem létező kurzorra történt hivatkozás.
   e. `NOT_LOGGED_ON (ORA-01012)`: Nem jelentkezett be a felhasználó, és Oracle hívást próbált intézni.
   f. `LOGIN_DENIED (ORA-01017)`: Hibás bejelentkezés.
   g. `NO_DATA_FOUND (ORA-01403)`: Vagy SELECT INTO történt, és nem volt eredmény sor. Vagy a tábla még nem inicializált sorára történt hivatkozás. Vagy az UTL_FILE csomag használatával a fájl vége után is olvasás történt.
   h. `TOO_MANY_ROWS (ORA-01422)`: SELECT INTO történt, és több eredmény sor is volt.
   i. `ZERO_DEVIDE (ORA-01476)`: Nullával történt osztás.
   j. `INVALID_NUMBER (ORA-01722)`: Egy olyan SQL utasítást próbáltunk végrehajtani, amely egy karakterláncot akart számmá alakítani, de sikertelenül.
   k. `STORAGE_ERROR (ORA-06500)`: A memória hiány vagy egyéb memória probléma.
   l. `PROGRAM_ERROR (ORA-06501)`: Valamilyen belső hiba történt.
   m. `VALUE_ERROR (ORA-06502)`: Valamilyen értékhiba történt a végrehajtás során.
   n. `CURSOR_ALREADY_OPEN (ORA-06511)`: Egy megnyitott kurzort próbáltunk újra megnyitni.
2. Felhasználó által definiált kivételek:

```
name_of_exception EXCEPTION;
```

3. **`SQLCODE és SQLERRM függvények:`**

A felhasználó által definiált hibára +1-et ad vissza az SQLCODE, a belső hibákra pedig negatív számot. (kivétel +100 -> NO_DATA_FOUND)

Milyen hibaüzenetei vannak a rendszernek?

```
FOR err_num IN 1..9999 LOOP
  dbms_output.put_line(SQLERRM(-err_num));
END LOOP;
```

A fenti két fv. nem használható közvetlenül SQL utasításban (pl. VALUES(SQLCODE)), értékükek lokális változóba kell tenni. (err_num := SQLCODE)

A le nem kezelt hibák esetén a rendszer különbözően viselkedik a futtató környezettől függően. Pl. C-be ágyazott program ROLLBACK-el, alprogram nem, és az OUT tipusú változóinak sem ad értéket.

TÍPUSOK, TÖMBÖK

- TYPE név IS RECORD(…);
- TYPE név IS TABLE OF …%type INDEX BY binary_integer;

Kifejezés	**Jelentés**
táblanév.EXISTS(n)	létezik-e az n-edik elem
tábla_név.COUNT	hány eleme van a táblának
tábla_név.FIRST és tábla_név.LAST	a legkisebb és legnagyobb index (NULL ha a tábla üres)
táblanév.PRIOR(n) és táblanév.NEXT(n)	az n index előtti index (NULL ha nincs előtte már). Az indexeknek nem kell egymás utáni számoknak lenniük (!)
táblanév.DELETE(n)	az n-edik elemet törli (felszabadítja az erőforrást is)
táblanév.DELETE(m, n)	m-től n-ig törli
táblanév.DELETE	az egész táblát törli

- **%TYPE:**
  Két legfőbb előnye:
  - Nem kell ismernem az objektum típusát
  - Ha később változik a típus a kód maradhat

- **%ROWTYPE:**
  A deklarációban nem szerepelhet inicializáció, de rekordok közötti értékadás megengedett

FÜGGVÉNYEK

1. Szintaxis:

```
CREATE [OR REPLACE] FUNCTION function_name
    [ (parameter [,parameter]) ]
 RETURN return_datatype
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

2. Példák:

   a. **Bemenet**: két dátum karakterlánc, alakja -> <hónap>/<nap_intervallum>/<év>, ahol a hónap és az év egy vagy két számjegyből állhat, illetve a nap intervallum (alak -> <nap>-<nap>) egy vagy két darab egy vagy két számjeggyel reprezentált érték

   Kimenet: Melyik dátum volt előbb? 'E' = egyenlő; 'G' = az első később volt; 'L' = a második volt később

```
CREATE OR REPLACE FUNCTION compare_battle_date
            (date_1 IN VARCHAR2, date_2 IN VARCHAR2)
    RETURN CHAR
  IS
    date_1_from VARCHAR2(8);
    date_2_from VARCHAR2(8);
  BEGIN
      /* Az első dátum első fele. */
      IF date_1 LIKE '__/__-%' OR date_1 LIKE '__/__/%' THEN
            date_1_from := substr(date_1,1,5);
      ELSIF date_1 LIKE '__/_-%' OR date_1 LIKE '__/_/%' OR
              date_1 LIKE '_/__-%' OR date_1 LIKE '_/__/%' THEN
            date_1_from := substr(date_1,1,4);
      ELSIF date_1 LIKE '_/_-%' OR date_1 LIKE '_/_/%' THEN
            date_1_from := substr(date_1,1,3);
      END IF;
      /* Az második dátum első fele. */
      IF date_2 LIKE '__/__-%' OR date_2 LIKE '__/__/%' THEN
            date_2_from := substr(date_2,1,5);
      ELSIF date_2 LIKE '__/_-%' OR date_2 LIKE '__/_/%' OR
              date_2 LIKE '_/__-%' OR date_2 LIKE '_/__/%' THEN
            date_2_from := substr(date_2,1,4);
      ELSIF date_2 LIKE '_/_-%' OR date_2 LIKE '_/_/%' THEN
            date_2_from := substr(date_2,1,3);
      END IF;
      /* Az első dátum második fele. */
      IF date_1 LIKE '%/__' THEN
            date_1_from := date_1_from ||
                  substr(date_1,length(date_1)-2,3);
      ELSIF date_1 LIKE '%/_' THEN
            date_1_from := date_1_from ||
                  substr(date_1,length(date_1)-1,2);
      END IF;
      /* Az második dátum második fele. */
      IF date_2 LIKE '%/__' THEN
            date_2_from := date_2_from ||
                  substr(date_2,length(date_2)-2,3);
      ELSIF date_2 LIKE '%/_' THEN
            date_2_from := date_2_from ||
                  substr(date_2,length(date_2)-1,2);
      END IF;
      /* Összehasonlítás elvégzése. */
      IF TO_DATE(date_1_from,'MM/DD/YY') <
            TO_DATE(date_2_from,'MM/DD/YY') THEN
            RETURN 'L';
      ELSIF TO_DATE(date_1_from,'MM/DD/YY') >
            TO_DATE(date_2_from,'MM/DD/YY') THEN
            RETURN 'G';
      ELSE
            RETURN 'E';
```

```
    END IF;
END;
```

b. **Bemenet**: dolgozó azonosító

Kimenet: az adott dolgozó főnökeinek száma

```
CREATE OR REPLACE FUNCTION rank_of_emp(emp_id IN NUMBER)
    RETURN NUMBER
  IS
    emp NUMBER:= emp_id;
    boss_id NUMBER;
    return_value NUMBER:=0;
  BEGIN
    LOOP
      SELECT menedzser_id INTO boss_id FROM dolgozok_zh
                  WHERE dolgozok_zh.id = emp;
      EXIT WHEN boss_id IS NULL;
      return_value := return_value + 1;
      emp := boss_id;
    END LOOP;

    RETURN(return_value);
  END;
```

PACKAGE

A `package`-ben lehetnek procedúrák, függvények, típus definíciók, változó deklarációk, konstansok, kivételek, kurzorok. Két része a specifikációs rész és a törzs (`body`). A specifikációs részben vannak a publikus deklarációk. Ennek létrehozása (`SQL` utasítással):

```
CREATE OR REPLACE PACKAGE p_név IS
   publikus típus és objektum deklarációk
   alprogram specifikációk
END;
```

A `body`-ban vannak az alprogramok és a kurzorok implementációi. Csak ezeknek van implementációs része, így ha a `package` csak más objektumokat tartalmaz (változók, típusok, kivételek ... stb.) akkor nem is kell, hogy `body`-ja is legyen.

A kurzorok kétféleképpen is megadhatók.

1. Vagy a specifikációban adjuk meg őket a szokásos módon, ekkor nem is szerepelnek az implementációs részben.
2. A specifikációs részben csak a nevét és a sortípusát adjuk meg (`CURSOR C1 RETURN <sortípus>`) és az implementációs részben adjuk meg a `SELECT`-et.

```
CREATE OR REPLACE PACKAGE BODY p_név IS
     privát típus és objektum deklarációk
     alprogramok törzse (PROCEDURE ... IS ...)
     kurzorok (CURSOR C1 RETURN <sortípus> IS SELECT ...)
[BEGIN  inicializáló utasítások ]
END;
```

A `body`-ban vannak az implementációk és lehet neki inicializációs része is (`BEGIN ... END` között), ami csak egyszer fut le, amikor a `package`-re először hivatkoznak.

A package specifikációs részében szereplő objektumok lokálisak az adatbázissémára nézve és globálisak a package-re nézve. Hivatkozás package-beli objektumokra: p_név.obj (a STANDARD package-beli objektumokra hivatkozhatunk a p_név nélkül).

Lehet azonos a neve két package-ben levő alprogramnak, amelyeknek más a paraméterezése. Ilyenkor híváskor derül ki, hogy melyik fog futni a formális és aktuális paraméterek egyeztetésekor (overloading). Például a STANDARD package-ben van több verzió is a TO_CHAR fv-re.

A package-ek legfontosabb előnyei:

- Modularítás
- Információ elrejtés
- Egészben töltődik be a memóriába minden objektuma az első hivatkozáskor.

A package-ben deklarált változók és kurzorok a session végéig léteznek, így közösen használhatják azokat a többi programok. (Kivétel a REF CURSOR, ami package-ben nem deklarálható.) Túlterhelt alprogramok írhatók (a lokális alprogramok is túlterhelhetők, csak a tároltak nem)

A package-ek forrásszövege a DBA_SOURCE táblában megnézhető.

A legfontosabb package-ek:

STANDARD	Beépített függvények és alprogramok ebben vannak
DBMS_SQL	DDL és dinamikus SQL végrehajtására
DBMS_OUTPUT	pl. put_line()
DBMS_STANDARD	az alkalmazás és az Oracle közötti interakciót segíti
UTL_FILE	op. rendszer fájlok írása, olvasása

PL/SQL package példa:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE PACKAGE pack1 IS
  user_name VARCHAR2(30);
  datum DATE;
  FUNCTION rossz_fv RETURN number;
  FUNCTION fv_plusz_1(szam number) RETURN number;
  PROCEDURE pr_plusz_1(szam number);

--  PRAGMA RESTRICT_REFERENCES(rossz_fv, WNDS);    -- nem lehet
becsapni
END pack1;
/
SHOW ERRORS


CREATE OR REPLACE PACKAGE BODY pack1 IS

 FUNCTION rossz_fv RETURN number IS              -- itt fogja
észrevenni ha becsapjuk
 BEGIN
   INSERT INTO emp(ename) VALUES('kiss');
   RETURN 11;
 END;
```

```
  FUNCTION fv_plusz_1(szam number) RETURN number IS
    lokalis_valtozo NUMBER(6);
  BEGIN
    lokalis_valtozo := szam + 1;
    RETURN(lokalis_valtozo);
  END;

  PROCEDURE pr_plusz_1(szam number) IS
    lokalis_valtozo NUMBER(6);
  BEGIN
    lokalis_valtozo := szam + 1;
    dbms_output.put_line(TO_CHAR(lokalis_valtozo));
  END;

BEGIN   -- a package első meghívásakor fut le minden sessionben
egyszer
  SELECT user, sysdate INTO user_name, datum FROM dual;
END pack1;
/
SHOW ERRORS

DECLARE
  v NUMBER;
BEGIN
  pack1.pr_plusz_1(100);
  dbms_output.put_line(pack1.user_name||' -- '||TO_CHAR(pack1.datum,
'yyyy-mm-dd:hh24:mi:ss'));
-- select pack1.rossz_fv INTO v FROM dual;
-- ha meg sem hívjuk, akkor nincs hiba
END;
/
```