PL/SQL

ORACLE DATABASE DOCUMENTATION 10g

PL/SQL User's Guide and Reference alapján

(Kósa Balázs .ppt bemutatói felhasználásával)

Jelölések

- Az SQL, PL/SQL utasítások, nevek stb. (terminálisok) nagybetűvel szerepelnek, pl.: SELECT.
- A nemterminálisok kisbetűvel, pl.: eredmeny.
- Az alternatívákat függőleges vonal választja el egymástól (|).
- Az opcionális elemek szögletes zárójelben állnak ([ ]),

  pl.: [NOT NULL].
- A kötelezően megadandó alternatívákat kapcsos zárójelek fogják közre ({ }), pl.: {kifejezes | feltetel}.
- Az iteráció jelölésére három pont szolgál (...),

  pl.: utasitas [utasitas]...

A példákhoz használt adatbázis

- automata (azon, penz, reszl_azon, varos)
- ugyfel (azon, nev, szuletes, keret)
- szamla (azon, ugyf_azon, reszl_azon, osszeg, lejarat)
- reszleg (azon, varos, tartalek)
- tranzakcio (azon, aut_azon, szaml_azon, ugyf_azon, datum, osszeg)

- Itt az *automata* tábla *reszl_azon* attribútuma azt adja meg, hogy melyik bank részleghez tartozik az adott automata, a *penz* pedig az automatán rendelkezésre álló pénzösszeget tárolja.
- A *tranzakcio* tábla *osszeg* attribútumának egy-egy értéke, ha negatív előjelű, pénzlevételt, ha pozitív, számlafeltöltést jelöl.
- A példa adatbázisban a Zseton Bank részlegeiről van szó.

Blokk

- A PL/SQL alapvető programegysége a blokk. Bárhol megjelenhet, ahol végrehajtható utasítás állhat.
- Felépítése:

```
[DECLARE deklaraciok]
BEGIN utasitas [utasitas]...
[EXCEPTION kivetelkezelo]
END;
```

- Megjegyzés: A címkék és GOTO utasítás használatához lásd Oracle dokumentáció.

Feltételes kifejezés

- A legtöbb kifejezés az SQL tárgyalásánál szerepelt.
- Feltételes kifejezés alakja:

```
CASE szelektor
        WHEN kifejezes THEN eredmeny
        [WHEN kifejezes THEN eredmeny]...
        [ELSE eredmeny]
END
```

Példa feltételes kifejezésre

```
DECLARE
        datum  DATE := DATE '1983-05-29';
        evszak VARCHAR2(6);
BEGIN
        evszak :=
        CASE
            WHEN TO_CHAR(datum, 'MM') IN ('01', '02', '12')
                        THEN 'tel'
            WHEN TO_CHAR(datum, 'MM') IN ('03', '04', '05')
                        THEN 'tavasz'
            WHEN TO_CHAR(datum, 'MM') IN ('06', '07', '08')
                        THEN 'nyar'
            WHEN TO_CHAR(datum, 'MM') IN ('09', '10', '11')
                        THEN 'osz'
            ELSE 'Hiba'
        END;
        DBMS_OUTPUT.PUT_LINE (evszak);
END;
```

Üres utasítás, feltételes utasítás

- Üres utasítás használható általában mindenütt, ahol végrehajtható utasítás állhat. Alkalmazásakor a vezérlés a következő utasításra kerül. Használható pl. rekurzív függvények „zárásaként”.
- Alakja: NULL;

- Feltételes utasítás alakja:

```
IF feltetel THEN utasitas [utasitas]....
[ELSIF feltetel THEN utasitas [utasitas]....]...
[ELSE utasitas [utasitas]....]
END IF;
```

Példa feltételes utasításra

```
DECLARE
        szam NUMBER := -5;
        absz NUMBER;
BEGIN
    IF szam >= 0 THEN absz := szam;
    ELSE absz:= -1 * szam;
    END IF;
    DBMS_OUTPUT.PUT_LINE (szam || ' abszolút értéke: '|| absz);
END;
```

CASE utasítás

- A CASE utasítás alakja:

```
CASE [szelektor_kifejezes]
        WHEN {kifejezes | feltetel} THEN utasitas [utasitas]...
        [WHEN {kifejezes | feltetel} THEN utasitas [utasitas]...]...
        [ELSE utasitas [utasitas]...]
END CASE;
```

Példa CASE utasításra

```
DECLARE
        osztalyzat NUMBER(1) := 4;

    BEGIN
        CASE osztalyzat
            WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('elégtelen');
            WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('elégséges');
            WHEN 3 THEN DBMS_OUTPUT.PUT_LINE('közepes');
            WHEN 4 THEN DBMS_OUTPUT.PUT_LINE('jó');
            WHEN 5 THEN DBMS_OUTPUT.PUT_LINE('jeles');
            ELSE DBMS_OUTPUT.PUT_LINE('Nincs ilyen jegy.');
        END CASE;
    END;
```

Alapciklus

- Alakja:

  LOOP utasitas [utasitas]... END LOOP;

- Példa:

```
DECLARE
        fakt  NUMBER:=1;
        i     PLS_INTEGER:=2;
BEGIN
        LOOP
            fakt:=fakt*i;
            EXIT WHEN fakt > 1000;
            i:=i+1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('A faktorialis: ' || fakt);
END;
```

WHILE ciklus

- Alakja:
  WHILE feltétel
  LOOP utasitas [utasitas]...
  END LOOP;

- Példa:

```
DECLARE
        ossz  NUMBER :=1;
BEGIN
        WHILE ossz < 10000 LOOP
                ossz := 2*ossz;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE (ossz);
END;
```

FOR ciklus

- Alakja:

  FOR ciklusvaltozo IN [REVERSE] also_hatar..felso_hatar
  LOOP utasitas [utasitas]...
  END LOOP;

- A *ciklusvaltozo* PLS_INTEGER típusú változó lesz, hatásköre a ciklusmag. Rendre felveszi az *also_hatar*, *felso_hatar* egészértékű kifejezések által definiált tartomány minden értékét.

- Az EXIT utasítással itt is bármikor kiléphetünk a ciklusból.

Példa FOR ciklusra

```
DECLARE
        ossz NUMBER := 0;

BEGIN
        FOR i in 1..100 LOOP
              ossz := ossz + i;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Az első száz szám
                                        összege: ' || ossz);
END;
```

A SELECT INTO utasítás I.

- Példa I.:

```
DECLARE
        reszl_varos        reszleg.varos%TYPE;

BEGIN
        SELECT varos
        INTO reszl_varos
        FROM reszleg
        WHERE azon = 'B01';
        DBMS_OUTPUT.PUT_LINE('A varos: ' || reszl_varos);
END;
```

A SELECT INTO utasítás II.

- Példa II.:

```
DECLARE
        sor         reszleg%ROWTYPE;
BEGIN
        SELECT *
        INTO sor
        FROM reszleg
        WHERE azon = 'B01';
END;
```

- A SELECT, FROM záradékok ugyanúgy viselkednek, mint a „normális” SQL lekérdezések esetében.
- Gondoskodni kell, hogy a lekérdezés végeredménye megfeleljen az INTO záradékban megadott változó, rekord típusának. A lekérdezés nem adhat vissza több sort végeredményként ebben az esetben.

Kurzorok

ORACLE DATABASE DOCUMENTATION 10g

PL/SQL User's Guide and Reference alapján

(Kósa Balázs .ppt bemutatói felhasználásával)

Mi az a kurzor?

- Egy SQL-utasítás feldolgozásához az Oracle egy speciális területet használ, amit környezeti területnek nevezünk.
- A környezeti terület a feldolgozott sorokról ad meg információt, lekérdezés esetén tartalmazza az eredmény sorait (aktív halmaz), valamint tartalmaz egy mutatót az utasítás belső reprezentációjára.
- A kurzor olyan eszköz, amellyel megnevezhetjük a környezeti területet, segítségével hozzáférhetünk az ott elhelyezett információkhoz, az aktív halmaz sorait egyenként elérhetjük és feldolgozhatjuk.
- Az Oracle minden DML utasításhoz felépít egy implicit kurzort, ennek neve: SQL.
- Az explicit kurzor kezelésének lépései:
  - kurzor deklarálása;
  - kurzor megnyitása;
  - sorok betöltése PL/SQL változókba;
  - kurzor lezárása.

Kurzor deklarálása

- Alakja:

  CURSOR nev [(parameter[, parameter]...)]

  [RETURN sortipus] IS select_utasitas;
- A parameter a kurzor formális paramétere. Alakja:

  parameter_nev [IN] tipus [{:= | DEFAULT} kifejezes]
- Megjegyzés: az IN érték szerinti paraméter-átadás. A select_utasitasban bárhol használható, ahol konstans szerepelhet, de a paraméternek nem adható ott érték. Valójában a formális paraméter referencia típusú, csak az aktuális paraméter értékének címét adja meg, de ez nem írható felül.
- A sortipus a kurzor által szolgáltatott érték típusa. Alakja:

  {{ab_tabla_nev | kurzor_nev | kurzor_valtozo_nev}%ROWTYPE | rekord_nev%TYPE | rekordtipus_nev}
- A select_utasitas egy INTO utasításrészt nem tartalmazó SELECT utasítás.

Kurzor deklarálása, példák:

```
CURSOR cur_ugyfel (dat tranzakcio.datum%TYPE DEFAULT SYSDATE)
    RETURN ugyfel%ROWTYPE IS
    SELECT DISTINCT u.azon, nev, szuletes, keret
    FROM ugyfel u, tranzakcio t
     WHERE u.azon = ugyf_azon AND dat = t.datum;

CURSOR cur_ossz_nap (dat DATE) IS
    SELECT u.azon ugyf_azon, SUM(t.osszeg), SUM(sz.osszeg)
    FROM ugyfel u, tranzakcio t, szamla sz
    WHERE u.azon = t.ugyf_azon AND dat = t.datum AND
          sz.ugyf_azon = u.azon
    GROUP BY u.azon;
```

Kurzor megnyitása

- A kurzor megnyitásánál lefut a kurzorhoz rendelt utasításrész, meghatározódik az aktív halmaz és az ehhez rendelt kurzormutató az első sorra áll.
- A kurzor megnyitásának alakja:

  OPEN kurzor_nev [(aktualis_parameter_lista)];

- Az aktuális paraméter típusának kompatibilisnek kell lennie a formális paraméter típusával.
- A pozíció szerinti egymáshoz rendelés esetén a paraméterek sorrendje a döntő.
- A név szerinti egymáshoz rendelésnél a sorrend lényegtelen, először megadjuk a formális paraméter nevét, majd egy => jelkombináció után az aktuális paramétert.
- Csak fix paraméterű kurzorok írhatók.
- Ha egy kurzort már megnyitottunk, és újból próbálkozunk, CURSOR_ALREADY_OPEN kivétel váltódik ki.

Sorok betöltése

- Alakja:

  FETCH {kurzor_nev | kurzorvaltozo_nev}

  INTO {rekord_nev | valtozo_nev[, valtozo_nev]... };

- Az utasítás a kurzormutató által megcímzett sort betölti a megadott rekordba vagy skalárváltozókba. A típusoknak rendre kompatibiliseknek kell lenniük természetesen.
- Ha a kurzort vagy kurzorváltozót nem nyitottuk meg, INVALID_CURSOR kivétel váltódik ki.
- Ha a FETCH utasítást az utolsó sor feldolgozása után adjuk ki, akkor a rekord vagy a változók előző értéke marad meg. E helyzet ellenőrzéséhez használhatók a %FOUND és %NOTFOUND attribútumok.

Kurzor lezárása

- Az utasítás érvényteleníti a kurzor vagy kurzorváltozó és az aktív halmaz közötti kapcsolatot és megszünteti a kurzormutatót.

- Alakja:

  CLOSE {kurzor_nev | kurzorvaltozo_nev};

- Lásd 1. példa
  http://people.inf.elte.hu/sila/lab3_plsql/Kurzor_pelda.html

Kurzorattribútumok

- %FOUND: kurzor vagy kurzorváltozó megnyitása előtt NULL az értéke. Ha sikerül betölteni egy újabb sort, akkor értéke TRUE, ha nem FALSE.
- %NOTFOUND: a %FOUND attribútum logikai ellentetje.
- %ISOPEN: értéke TRUE, ha a kurzort vagy kurzorváltozót már megnyitottuk, egyébként FALSE.
- %ROWCOUNT: értéke az első sorbetöltés előtt és megnyitás után 0. Ez az érték minden egyes sikeres sorbetöltés után 1-gyel nő.
- Ha a kurzort vagy kurzorváltozót nem nyitottuk meg és valamelyik attribútumot használjuk, INVALID_CURSOR kivétel váltódik ki.

Az implicit kurzor attribútumai

- Az implicit kurzor attribútumai az INSERT, DELETE, UPDATE, SELECT INTO utasításáról adnak információt.
- %FOUND: az utasítás előtt NULL az értéke. Ha az utasítás legalább egy értéket érintett, értéke TRUE, ha nem FALSE.
- %NOTFOUND: a %FOUND attribútum logikai ellentetje.
- %ISOPEN: az Oracle automatikusan lezárja az implicit kurzort az utasítás végrehajtása után, ezért értéke mindig FALSE.
- %ROWCOUNT: értéke az INSERT, DELETE, UPDATE utasításoknál a „kezelt” sorok száma, SELECT INTO esetén, ha nincs visszaadott sor, értéke 0, egy sor esetén 1, különben TOO_MANY_ROWS kivétel váltódik ki, az attribútum tehát nem a ténylegesen visszaadott sorok számát tartalmazza.

Példa

```
DECLARE
    i NUMBER(1);
BEGIN
    SELECT 1 INTO i FROM dual;
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT: ' ||
            SQL%ROWCOUNT);
    DELETE FROM ugyfel WHERE azon = 'U09';
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT: ' ||
            SQL%ROWCOUNT);
END;
/
```

- Megjegyzés: az implicit kurzor attribútumai által visszaadott értékek mindig az időben utolsó SQL műveletre vonatkoznak.

FOR UPDATE utasításrész

- 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.
- Lásd 3. példa http://people.inf.elte.hu/sila/lab3_plsql/Kurzor_pelda.html

Kurzorváltozók

- A kurzor statikus, azaz egyetlen, a fordításkor már ismert SELECT utasításhoz kötődik, addig a kurzorváltozó dinamikus, amelyhez futásidőben bármely típuskompatibilis kérdés hozzákapcsolható.

- A kurzorváltozó lényegében egy referencia típusú változó, amely mindig a hivatkozott sor címét adja vissza.

Kurzorváltozók deklarálása

- 1. lépésben a REF CURSOR saját típust kell létrehozni:

  TYPE nev IS REF CURSOR [RETURN
  {{ab_tabla_nev | kurzor_nev | kurzorvaltozo_nev}%ROWTYPE |
  rekord_nev%TYPE | rekordtipus_nev | kurzorrefenciatipus_nev}]

  - ab_tabla_nev: egy létező tábla, nézettábla neve;
  - kurzor_nev, kurzorvaltozo_nev: egy már korábban deklarált kurzor, kurzorváltozó;
  - rekord_nev: egy korábban definiált rekord neve stb.

- A 2. lépésben aztán a létrehozott típus a szokásos módon használható.

Kurzorváltozók deklarálása, példa

```
DECLARE

    TYPE pelda_ref_cursor IS REF CURSOR RETURN
          ugyfel%ROWTYPE;

    v_ref_cursor            pelda_ref_cursor;

BEGIN ...
```

Kurzorváltozó megnyitása

- Alakja:

  OPEN kurzorvaltozo_nev FOR select_utasitas;

- Egy adott kurzorváltozó bármennyi OPEN-FOR utasításban szerepelhet, egy újabb megnyitás előtt nem kell lezárni. Egy „újramegnyitás” azt jelenti, hogy egy új aktív halmaz jön létre, amelyre a kurzorváltozó hivatkozni fog, az előző pedig értelemszerűen törlődik.
- A FETCH és CLOSE utasítások ugyanúgy működnek.
- A kapcsolt *select_utasítas* nem tartalmazhat FOR UPDATE utasításrészt.
- Lásd 4. példa http://people.inf.elte.hu/sila/lab3_plsql/Kurzor_pelda.html

Kurzor FOR ciklus

- Az explicit kurzor használatát a kurzor FOR ciklus használatával egyszerűsíthetjük. Ez implicit módon %ROWTYPE típusúnak deklarálja a ciklusváltozót, megnyitja a kurzort, rendre betölti az aktív halmaz elemeit, majd lezárja a kurzort.
- Alakja:

  FOR ciklusvaltozo IN {kurzor_nev [(parameterek)] | (select_utasitas)}
  LOOP utasitas [utasitas]… END LOOP;

- Lásd 5. példa
  http://people.inf.elte.hu/sila/lab3_plsql/Kurzor_pelda.html

Alprogramok, tárolt alprogramok, csomagok

ORACLE DATABASE DOCUMENTATION 10g

PL/SQL User's Guide and Reference alapján

(Kósa Balázs .ppt bemutatói felhasználásával)

Alprogramok

- Az alprogramok paraméterezhetők és meghívhatók. Alapértelmezés szerint rekurzívan is meghívhatják egymást. Két fő részük van: a specifikáció és a törzs.
- Alakjuk szinte teljesen megegyezik a blokkok alakjával (nincs DECLARE utasítás):

```
specifikacio
{IS | AS}
[deklaraciok]
BEGIN utasitas [utasitas]...
[EXCEPTION kivetelkezelo]
END;
```

Eljárások

- Specifikáció alakja:

PROCEDURE nev [(formalis_parameter [, formalis_parameter]...)]

- Formális paraméter formája:

nev [{IN | {OUT | INOUT} [NOCOPY]}] tipus [{:= | DEFAULT} kifejezes]

- A formális paraméter után a paraméter-átadás módját lehet meghatározni:
  - IN esetén (alapeset) érték szerinti az átadás, az alprogram törzsében nevesített konstansként viselkedik, vagyis nem adható neki érték. Igaziból ekkor a formális paraméter az aktuális paraméter értékének a címét kapja meg, az aktuális paraméter értéke azonban nem írható felül.
  - OUT esetén eredmény szerinti az átadás, az alprogram törzsében változóként viselkedik, automatikus kezdőértéke NULL (emiatt nem lehet NOT NULL megszorítású altípus pl.: NATURALN).
  - INOUT esetén érték szerinti az átadás, az alprogram törzsében inicializált változóként viselkedik.
  - A NOCOPY azt ajánlja a fordítónak, hogy az OUT és az INOUT esetén se legyen érték szerinti a másolás.

Függvények

- A függvény specifikációjának alakja:

FUNCTION nev [(formalis_parameter [, formalis_parameter]...)]
  RETURN tipus

- A függvény törzsében legalább egy RETURN utasításnak szerepelnie kell, ellenkező esetben PROGRAM_ERROR kivétel váltódik ki.
- Függvény törzsében a RETURN utasítás alakja:

  RETURN [( ] kifejezes [ )];

- Lásd 1. példa
  http://people.inf.elte.hu/sila/lab3_plsql/Alprog_pelda.html

Tárolt eljárások

- Eddig lokális alprogramokkal foglalkoztunk, lehetőség van azonban arra, hogy az alprogramokat adatbázis-objektumként kezeljük.
- Ekkor az SQL szokásos létrehozó, törlő és módosító DDL-utasításait használjuk.
- A tárolt alprogramok p-kódban tárolódnak. Ha egy tárolt alprogramot hívunk meg, az betöltődik az adatbázisba és ott kerül interpretálásra.
- A tárolt alprogramokról infomációkat a következő adatszótárnézetekből nyerhetünk: USER_OBJECTS, USER_SOURCE (a forráskkódot tartalmazza), USER_ERRORS (a fordítási hibák találhatók itt).
- SQL*Plus-ban a fordítási hibákat a SHOW ERRORS paranccsal tekinthetjük meg.

Tárolt eljárások I.

CREATE [OR REPLACE] eljaras_fej

[AUTHID {DEFINER | CURRENT USER}]

eljaras_torzs

- Az eljaras_torzsre a lokális alprogramoknál elmondottak vonatkoznak.
- Az AUTHID segítségével megadható, hogy az eljárás létrehozójának (DEFINER, ez az alapértelmezés), vagy aktuális hívójának (CURRENT_USER) jogosultságai érvényesek-e a hívásnál.

Tárolt eljárások II.

ALTER PROCEDURE eljaras_nev COMPILE [DEBUG]

- A parancs segítségével újrafordíthatunk egy tárolt eljárást. A DEBUG opció azt írja elő, hogy a PL/SQL fordító használja a nyomkövetőt a p-kód generálásánál.

DROP PROCEDURE eljaras_nev;

- A parancs segítségével törlünk egy tárolt eljárást.

Tárolt függvények I.

Csomagok

- A csomag olyan PL/SQL programegység, amely adatbázis-objektum és a PL/SQL programozási eszközök egy gyűjteményének tekinthető.
- Két részből áll: specifikációból és törzsből. A törzs nem kötelező.
- A specifikációban típusdefiníciók, nevesített konstansok deklarációja, változó-deklarációk, kurzor-specifikációk, alprogram-specifikációk helyezhetők el.
- Ha a specifikációban szerepel kurzor-specifikáció illetve alprogram-specifikáció, akkor a törzs kötelező, és abban az adott eszközök teljes deklarációját meg kell adni.
- Az eszközök felsorolása tetszőleges a specifikációban, kivéve azt, hogy az alprogramoknak kötelezően a többi eszköz deklarációja után kell állniuk, mögöttük már csak pragmák következhetnek.
- A specifikáció deklarációi nyilvánosak, az itt deklarált eszközök tetszés szerint használhatók a séma bármely alkalmazásában. Szintaxisa: *csomagnev.eszkoznev*.
- A törzs deklarációi nem nyilvánosak.

Csomagok specifikációja

CREATE [OR REPLACE] PACKAGE csomagnev
[AUTHID {DEFINER | CURRENT USER}]
{IS | AS}
{tipus_definicio | nevesitett_konstans_deklaracio | valtozo_deklaracio | kivetel_deklaracio | kurzor_specifikacio | alprogram_specifikacio | pragma}
END [csomagnev];

- Az AUTHID segítségével megadható, hogy az eljárás létrehozójának (DEFINER, ez az alapértelmezés), avagy az aktuális hívójának (CURRENT_USER) jogosultságai érvényesek-e a hívásnál.

Csomagok törzse

```
CREATE [OR REPLACE] PACKAGE BODY csomagnev
{IS | AS}
deklaraciok
[BEGIN
    vegrehajthato_utasitasok]
END [csomagnev];
```

- A csomag alprogramjainak nevei túlterhelhetőek.
- A PL/SQL fordító p-kódra fordítja le a specifikációt és a törzset.

Csomag megváltoztasása

ALTER PACKAGE csomagnev COMPILE [DEBUG]
[PACKAGE | SPECIFICATION | BODY];

- A DEBUG előírja, hogy a PL/SQL fordító használja a nyomkövetőjét.
- PACKAGE (ez az alapértelmezés) esetén az egész csomag, míg a SPECIFICATION illetve BODY esetén csak a specifikáció vagy a törzs fordítódik újra.
- Megjegyzés: egy csomagban (is) használt táblának pl. megváltozik a sémája, akkor a csomagot a fenti utasítással újra kell fordítanunk.

Triggerek

ORACLE DATABASE DOCUMENTATION 10g

PL/SQL User's Guide and Reference alapján

(Kósa Balázs .ppt bemutatói felhasználásával)

Alapok I.

- A triggerek olyan tevékenységeket definiálnak, amelyek automatikusan végrehajtódnak, ha egy tábla vagy nézet módosul vagy egyéb felhasználói vagy rendszeresemény történik.
- Ilyen események lehetnek:
  - egy táblán vagy nézeten DML utasítás,
  - egyes DDL utasítások,
  - szerverhibák,
  - felhasználói be- és kijelentkezés,
  - adatbázis elindítása és leállítása.
- A trigger adatbázis-objektum.
- A kód megírható C, Java és PL/SQL nyelveken is.

Triggerek típusai

- Sorszintű trigger: mindannyiszor lefut, ahányszor a tábla adatai módosulnak.
- Utasítás szintű trigger: függetlenül az érintett sorok számától csak egyszer fut le. Akkor is lefut, ha egyetlen sort sem kezeltünk.
- BEFORE és AFTER triggerek: egyaránt lehetnek sor és utasítás szintűek. Csak táblához kapcsolhatók, nézethez nem, ám egy alaptáblához kapcsolt trigger lefut a nézeten végrehajtott DML utasítás esetén is. A BEFORE trigger a hozzákapcsolt utasítás előtt, az AFTER utána fut le. Több is megadható belőlük ugyanarra az eseményre.
- INSTEAD OF trigger: a hozzákapcsolt utasítás helyett fut le. Csak sor szintű lehet és csak nézethez kapcsolható. Általában akkor használjuk, ha egy nézetet módosítani szeretnénk, de ezt nem tehetjük közvetlen DML utasítással.
- Rendszertriggerek: bővebben lásd Oracle dokumentáció.

Triggerek létrehozása I.

```
CREATE [OR REPLACE] TRIGGER [sema.]triggernev
    {BEFORE | AFTER | INSTEAD OF}
    {dml_trigger | {ddl_esemeny [OR ddl_esemeny]… |
          ab_esemeny [OR ab_esemeny]…}
    ON {DATABASE | [sema.]SCHEMA}
    [WHEN (feltétel)] {plsql_blokk | eljarashivas}
```

- A *sema* a triggert tartalmazó séma neve. Ha hiányzik, a parancsot kiadó felhasználó sémájában jön létre a trigger.
- *ddl_esemeny* lehet: ALTER, DROP, GRANT, RENAME, REVOKE stb.
- Az *ab_esemeny*ekhez lásd a dokumentációt.
- Ez a két utóbbi esemény az adatbázishoz (DATABASE) vagy a *sema* nevű sémához köthető. Csak BEFORE és AFTER triggerek esetében adható meg.

Triggerek létrehozása II. *(dml_trigger)*

{INSERT | DELETE | UPDATE [OF oszlop [, oszlop]…]}

[OR {INSERT | DELETE | UPDATE [OF oszlop [, oszlop]…]}] …

ON [sema.]tabla | [sema.]nezet

[REFERENCING {OLD [AS] regi | NEW [AS] uj}

    [{OLD [AS] regi | NEW [AS] uj] …

[FOR EACH ROW]

- Az INSERT, DELETE, UPDATE definiálja azt az SQL-utasítást, amelyre a trigger lefut.
- Az ON után szerepel az az adatbázis-objektum, amelyre a triggert létrehoztuk.
- A REFERENCING utasításrész korrelációs neveket határoz meg sorszintű triggerek esetén. Az OLD az aktuális sor módosítás előtti, a NEW a módosítás utáni neveket adja meg. Az alapértelmezett korrelációs nevek: :OLD, :NEW.
- A FOR EACH ROW sorszintű triggert hoz létre.

Triggerek működése

- A triggerek végrehajtásánál az Oracle a következő végrehajtási modellt követi:
  - Végrehajtja az összes utasításszintű BEFORE triggert.
  - A DML-utasítás által érintett minden sorra ciklikusan:
    - végrehajtja a sorszintű BEFORE triggereket,
    - zárolja és megváltoztatja a sort és ellenőrzi az integritási megszorításokat. A zár csak a tranzakció végeztével oldódik fel,
    - végrehajtja a sorszintű AFTER triggereket.
  - Ellenőrzi a késleltetett integritási megszorításokat.
  - Végrehajtja az utasítás szintű AFTER triggereket.
- Ha egy trigger futása közben kivétel keletkezik, és azt nem kezeljük le, akkor az összes tevékenység visszagörgetésre kerül.
- A végrehajtási modell rekurzív, egy trigger működése közben újabb triggerek indulhatnak el.

A trigger törzse

- A törzsben nem szerepelhetnek tranzakció-vezérlő utasítások, s a törzsből meghívott alprogramok sem tartalmazhatnak ilyeneket (COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION).
- A DML-triggerek törzsében használható három logikai függvény, amelyek a triggert aktivizáló utasításról adnak információt. A függvények: INSERTING, DELETING, UPDATING rendre akkor adnak vissza igaz értéket, ha a DML-utasítás INSERT, DELETE illetve UPDATE volt.

Triggerek tárolása

- A trigerekkel kapcsolatos információk az adatszótárnézetekből érhetők el.
- A USER_TRIGGERS nézet tartalmazza a törzs és a WHEN feltétel kódját és a trigger típusát.
- Az ALL_TRIGGERS az aktuális felhasználó triggereiről, a DBA_TRIGGERS minden triggerről tartalmaz információt.

Triggerek megváltoztatása, kidobása

ALTER TRIGGER [sema.]triggernev

{ENABLE | DISABLE | RENAME to uj_nev |

COMPILE [DEBUG] [REUSE SETTINGS]];

- Az ENABLE engedélyezi, a DISABLE letiltja, a RENAME átnevezi a triggert.
- A COMPILE újrafordítja a triggert függetlenül attól, hogy az éppen letiltott-e vagy sem. A DEBUG arra utasítja a fordítót, hogy használja a PL/SQL nyomkövetőjét. Újrafordításra akkor van szükség, ha az érintett objektumok sémája változott.
- Az Oracle először újrafordít minden olyan objektumot, amelytől a trigger függ, ha azok érvénytelenek.
- Trigger törlésének alakja:

  DROP TRIGGER triggernev;