10. FEJEZET

Kivételkezelés, alprogramok, triggerek

Elméleti összefoglaló

Kivételkezelés (EXCEPTION)

A PL/SQL-ben a kivétel egy azonosítóval ellátott esemény, amelynek hatására a blokk végrehajtása félbeszakad. A kivételkezelő rész a blokk azon része, amelyre kivétel esetén a vezérlés adódik. (Vázlatosan már utaltunk rá a 8. fejezetben.) A kivételkezelő résszel együtt a teljes blokk alakja:

```
DECLARE
  ...
BEGIN
  ...
EXCEPTION
  ...
END;
```

A kivételek fajtái

A FELHASZNÁLÓ ÁLTAL DEFINIÁLT KIVÉTELEK

A kivételt deklarálni kell (`EXCEPTION` típussal), majd a `RAISE` utasítás kiváltja az adott nevű kivételt. Ekkor a vezérlés a blokk kivételkezelő részére ugrik, ahol a hibához tartozó utasításrész végrehajtódik („lekezeli a hibát"), melynek során tipikusan felhasználói üzenet is kiíródik.

Alakja:

```
RAISE [kivételnév];
```

Tipikus használata:

```
DECLARE
  Kivételnév  EXCEPTION;
...
```

```
BEGIN
...
  IF LogikaiKifejezés
  THEN
    RAISE Kivételnév;
  ...
  END IF

EXCEPTION
  WHEN Kivételnév
    THEN utasítások;
END;
```

A felhasználó által definiált kivételeket társíthatjuk a rendszerkivételekhez az alprogram (*lásd alább*) deklaráló részében:

```
pragma EXCEPTION_INIT (kivételnév, hibakód);
```

ahol:

- *kivételnév* egy előzőleg deklarált kivétel,
- *hibakód* egy már létező Oracle-hiba sorszáma.

A `pragma` a fordítónak szóló direktíva (a futtatható kódba *nem* kerülő utasítás).

A PL/SQL ELŐREDEFINIÁLT AUTOMATIKUS RENDSZERKIVÉTELEI

Ezeket a futtatórendszer váltja ki, és a `STANDARD` csomagban találhatók, melyet a PL/SQL használ.

Példa (részlet)

```
EXCEPTION
  WHEN No_Data_Found  -- a SELECT INTO nem ad vissza sorokat
   THEN üzenet;
  WHEN Too_Many_Rows  -- a SELECT INTO több sorral tér vissza
   THEN üzenet;
  WHEN OTHERS         -- minden más meg nem nevezett hiba esetén
   THEN üzenet;       -- SQLCODE és SQLEERRM hibaüzenettel és kóddal
END;                  -- tér vissza
```

Alprogramok

A névvel ellátott és paraméterezhető blokkot alprogramnak nevezzük. Az alprogramok lehetnek eljárások és függvények.

Az eljárás szintaktikája

```
PROCEDURE név [(paraméter [, paraméter]...)]
IS
  [lokális deklarációk (a DECLARE kulcsszó nélkül)]
BEGIN
  végrehajtható utasítások
[EXCEPTION
  kivételkezelés]
END [név];
```

A függvény szintaktikája

```
FUNCTION név [(paraméter [, paraméter]...)]
RETURN adattípus
IS
  [lokális deklarációk (a DECLARE kulcsszó nélkül)]
BEGIN
  végrehajtható utasítások
  RETURN visszatérési érték
[EXCEPTION
  kivételkezelés]
END [név];
```

A *paraméter szintaktikája* (mindkét esetben):

paraméter_név [IN | OUT | IN OUT] *adattípus* [{DEFAULT | :=} *kifejezés*]

- IN — bemenő paramétert jelöl (az értéke nem változtatható meg az alprogramban, ez az alapértelmezés, ezért az IN kulcsszó elhagyható),
- OUT — kimenő paramétert jelöl (az alprogramban csak kaphat, de nem adhat át értéket),
- IN OUT — be- és kimenő paramétert jelöl (az alprogramban kaphat és adhat is értéket),
- *kifejezés* — kezdőérték megadására szolgál (csak az IN, vagy az IN OUT paraméterek esetén használható).

Paraméterek adattípusai

Az alprogramokban a már megismert PL/SQL-típusok használhatók azzal a megkötéssel, hogy nem tartalmazhatnak korlátozásokat (például a hosszúságra vagy a pontosságra vonatkozóan). Ennek megfelelően használhatók a NUMBER, az INTEGER, a REAL, a CHAR, a VARCHAR2 stb. (így, ebben a formában!). Természetesen használhatók a hivatkozott (%TYPE, %ROWTYPE) típusok is. (Részletesen *lásd* [15] és [16].)

Példa (részlet)

```
PROCEDURE Bolygók(sugár    IN     INTEGER,
                  térfogat OUT    REAL,
                  sorrend  IN OUT INTEGER)
```

Megjegyzés

A függvényeknél lehetőleg ne használjunk `OUT` és `IN OUT` paramétereket, mert mellékhatást okozhatnak. A függvény használatának célja egy visszatérési érték meghatározása.

10.1. példa

Mivel közeledik az év vége, ezért néhány dolgozó jutalmat kap a tulajdonostól. A jutalom a dolgozó fizetésének 10%-a, feltéve, hogy a fizetése 3000 USD alatt van, egyébként csak egy „`Boldog Karácsonyt!`" üdvözletet kap. Írjon erre egy PL/SQL programot, amely eljárást és kivételkezelést is tartalmaz. A jutalmat az `emp` táblából létrehozott `dolgozó` tábla jutalék (`comm`) értékéhez adja hozzá.

Megoldás

```
SET serveroutput ON
DROP TABLE dolgozó;
CREATE TABLE dolgozó
AS  SELECT * FROM emp;

-- JutalmazóProgram
ACCEPT neve  PROMPT 'Adja meg a jutalmazandó dolgozó nevét: '

-- A főprogram deklarációs szegmense
DECLARE
  v_neve        dolgozó.ename%TYPE;
  v_jutalma     dolgozó.sal%TYPE;
  NincsJutalom  EXCEPTION;

-- Az alprogram deklaráció
PROCEDURE Jutalmazás (p_neve     IN   dolgozó.ename%TYPE,
                      p_jutalma  OUT  dolgozó.sal%TYPE)
IS
-- Lokális deklaráció
  p_fizetése  dolgozó.sal%TYPE;
BEGIN
  SELECT sal
    INTO p_fizetése
    FROM dolgozó
    WHERE UPPER(ename) = UPPER(p_neve);
```

```
    IF p_fizetése >= 3000
    THEN
      RAISE NincsJutalom;
    ELSE
      p_jutalma := 0.1 * p_fizetése;
      DBMS_OUTPUT.PUT_LINE('>>');
      DBMS_OUTPUT.PUT_LINE('>> ' || p_neve ||' jutalma: '||
                                    p_jutalma ||' USD');
    END IF;

  EXCEPTION
    WHEN NO_DATA_FOUND
      THEN
        DBMS_OUTPUT.PUT_LINE('>>');
        DBMS_OUTPUT.PUT_LINE('>> NINCS ilyen nevű dolgozó...');
    WHEN NincsJutalom
      THEN
        DBMS_OUTPUT.PUT_LINE('>>');
        DBMS_OUTPUT.PUT_LINE('>> Boldog Karácsonyt Kedves ' ||
                             p_neve || '!');
  END Jutalmazás;
  -- Az alprogram deklarációjának vége

  -- Főprogram blokkja
  BEGIN
    v_neve := '&neve';
    Jutalmazás(v_neve, v_jutalma);
    UPDATE dolgozó
      SET comm = NVL(comm,0) + v_jutalma
      WHERE UPPER(ename) = UPPER(v_neve);
  END; -- JutalmazóProgram
  /

  -- PROMPT Listázás:
  SELECT ename        AS "Név",
         job          AS "Munkakör",
         sal          AS "Fizetés",
         NVL(comm,0)  AS "Jutalék+Jutalom"
    FROM dolgozó
    WHERE UPPER(ename) = UPPER('&neve');
```

1. futtatás eredménye

```
A tábla eldobva.
A tábla létrejött.
```

```
Adja meg a jutalmazandó dolgozó nevét: Smith
>>
>> Smith jutalma: 80 USD

A PL/SQL eljárás sikeresen befejeződött.

Név        Munkakör     Fizetés Jutalék+Jutalom
---------- --------- ---------- ---------------
SMITH      CLERK            800              80
```

2. futtatás eredménye

```
Adja meg a jutalmazandó dolgozó nevét: Mikulás
>>
>> NINCS ilyen nevű dolgozó...

A PL/SQL eljárás sikeresen befejeződött.

nincsenek kijelölve sorok
```

3. futtatás eredménye

```
Adja meg a jutalmazandó dolgozó nevét: Ford
>>
>> Boldog Karácsonyt Kedves Ford!

A PL/SQL eljárás sikeresen befejeződött.

Név        Munkakör     Fizetés Jutalék+Jutalom
---------- --------- ---------- ---------------
FORD       ANALYST          3000               0
```

Tárolt alprogramok

Az alprogramok eltárolhatók, ezzel lehetőség van újrafelhasználásra, saját vagy más felhasználó számára (ha erre van joga).

Megjegyezzük, hogy a SHOW ERRORS SQL*Plus-utasítás segítségével listázhatjuk az alprogramok fordításánál bekövetkezett hibákat.

Tárolt alprogramok létrehozása

```
CREATE OR REPLACE PROCEDURE eljárásnév [(paraméter [, paraméter]...)]
{IS | AS}
   [lokális deklarációk (a DECLARE kulcsszó nélkül)]
   blokk
```

```
CREATE OR REPLACE FUNCTION függvénynév [(paraméter [, paraméter]...)]
RETURN adattípus
{IS | AS}
  [lokális deklarációk (a DECLARE kulcsszó nélkül)]
  blokk
```

Tárolt alprogramok törlése

```
DROP PROCEDURE eljárásnév;
DROP FUNCTION függvénynév;
```

Tárolt alprogramok hibakezelése

A RAISE_APPLICATION_ERROR eljárás a tárolt alprogramban a felhasználó által definiált, és előredefiniált kivételek üzenetét segíti megjeleníteni. Használatával megtudhatjuk, hogy az alprogramok közül melyikben történt a kivétel, és azt mi okozta. Ez az eljárás bárhonnan hívható. Szintaktikája:

```
RAISE_APPLICATION_ERROR(hibakód, üzenet)
```

ahol:

- a *hibakód* értékének engedélyezett tartománya: -20000 ... -20999,
- az *üzenet* egy karaktersorozat, és rövidebb 2048 bájtnál.

Megjegyzés
A RAISE_APPLICATION_ERROR eljárás felfüggeszti az őt tartalmazó tárolt alprogramban a DBMS_OUTPUT.PUT_LINE kiíratás eljárások hatását. Az ezzel történő kiíratások zárolását, vagyis e zárolt memóriaterületről a szöveg képernyőn való megjelenését csak egy névtelen blokkban történő kiíratás szabadítja fel. (*Lásd* még a 10.1. feladat megoldásánál tett megjegyzést.)

Példa (részlet)

```
    hiba_1  EXCEPTION;
    hiba_2  EXCEPTION;
  BEGIN
    ...
    IF NOT alprogfeltétel1(aktuális paraméter lista)
    THEN
      RAISE hiba_1;
    END IF;
    IF NOT alprogfeltétel2(aktuális paraméter lista)
    THEN
      RAISE hiba_2;
    END IF;
```

```
  ...
-- A kivételkezelő rész
  EXCEPTION
    WHEN hiba_1 THEN
      RAISE_APPLICATION_ERROR(-20101, 'Kivétel az 1.alprogramban!');
    WHEN hiba_2 THEN
      RAISE_APPLICATION_ERROR(-20102, 'Kivétel a 2.alprogramban!');
END;
```

Tárolt alprogramok lekérdezése

A már eltárolt alprogramok (és triggerek, *lásd* alább) lekérdezhetők az adatszótár nézeteiből a `user_source` nézettábla segítségével, melynek szerkezete:

```
SET linesize 65
DESC user_source
SET linesize 400
```

Eredmény

```
Név                              Üres?    Típus
-------------------------------- -------- ----------------------
NAME                                      VARCHAR2(30)
TYPE                                      VARCHAR2(12)
LINE                                      NUMBER
TEXT                                      VARCHAR2(4000)
```

10.2. példa

Készítsen egy tárolt függvényt, mely `TRUE` értékkel tér vissza, ha az `ÚjFizetés` bemenő paraméter értéke nagyobb a `RégiFizetés` bemenő paraméternél, egyébként `FALSE` értékkel. A tárolt függvény forráskódját kérdezze le.

Megoldás

1. lépés (A tárolt függvény elkészítése)

```
CREATE OR REPLACE FUNCTION
  Fizetés (ÚjFizetés    IN NUMBER,
           RégiFizetés  IN NUMBER)
RETURN BOOLEAN IS
BEGIN
  IF ÚjFizetés > RégiFizetés
  THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
```

```
  END IF;
END;
/
SHOW ERRORS
```

Eredmény

```
A függvény létrejött.
Nincsenek hibák.
```

2. lépés (A forráskód lekérdezése)

```
COLUMN text  FORMAT A50
COLUMN line  FORMAT 9999
SELECT line, text
  FROM user_source
  WHERE UPPER(type)='FUNCTION' AND
        UPPER(name)='FIZETÉS';
CLEAR COLUMNS
```

Eredmény

```
LINE TEXT
----- --------------------------------------------------
    1 FUNCTION
    2   Fizetés (ÚjFizetés    IN NUMBER,
    3            RégiFizetés  IN NUMBER)
    4 RETURN BOOLEAN IS
    5 BEGIN
    6   IF ÚjFizetés > RégiFizetés
    7   THEN
    8     RETURN TRUE;
    9   ELSE
   10     RETURN FALSE;
   11   END IF;
   12 END;

12 sor kijelölve.
```

Tárolt eljárás futtatása

A már eltárolt alprogramokat jellemzően SQL*Plus szkript programbeli PL/SQL blokkból való hívással szoktuk meghívni, azonban lehetőség van arra is, hogy egy tárolt eljárást közvetlenül az SQL*Plus-környezetből hívjunk meg az

```
EXECUTE [felhasználó.]TároltEljárásNeve
```

SQL*Plus-utasítás segítségével, ahol a felhasználó megadása elmaradhat.

10.3. példa

Készítsen egy tárolt eljárást, mely a paramétereként adott két egész számot összeadja, és az eredményt kiírja a képernyőre. A kiíratás előtt írjon ki egy üres sort, és az eredménysor tabulálással kezdődjön. Az eljárást futtassa le közvetlenül az SQL*Plus-környezetből.

Megoldás

```
CREATE OR REPLACE PROCEDURE
  Összead (Egyik  IN NUMBER,
           Másik  IN NUMBER)
IS
  Eredmény  NUMBER(3);
  Kiírás    VARCHAR2(20);
BEGIN
  Eredmény := Egyik + Másik;
  Kiírás := CHR(10)          || /* soremelés (LF)  */
            CHR(9)           || /* tabulálás (TAB) */
            TO_CHAR(Egyik)   || ' + ' ||
            TO_CHAR(Másik)   || ' = ' ||
            TO_CHAR(Eredmény);
  DBMS_OUTPUT.PUT_LINE(Kiírás);
END;
/

SHOW ERRORS

SET serveroutput ON
EXECUTE Összead(3,4)
```

Eredmény

```
Az eljárás létrejött.

Nincsenek hibák.

        3 + 4 = 7

A PL/SQL eljárás sikeresen befejeződött.
```

Megjegyzés
Figyeljünk fel a vezérlőkarakterek használatára!