

Lab-7

PL/pgSQL Alias, Record/Cursor ve Trigger Tanımları.

Örnek – 1 : geçen haftadan.

CREATE or REPLACE FUNCTION ornek1 (num1 NUMERIC, num2 NUMERIC)

RETURNS numeric AS \$\$

DECLARE

toplam NUMERIC;

BEGIN


toplam := num1 + num2;

RAISE NOTICE 'sayi1:% , sayi2:%', num1, num2;

RETURN toplam;

END;

\$\$ LANGUAGE 'plpgsql';



Tek tırnak işaretini (')
burada kullanırsak hata
verir.
Çünkü fonksiyon
gövdesinde ' kullanılmış.

RAISE

- Bilgilendirme ve hata mesajı yazdırmak için kullanılır.

RAISE mesaj_türü **MESAJ**;

Örnek:

RAISE NOTICE 'Bilgilendirme';

RAISE EXCEPTION 'Hata Mesajı';

RAISE NOTICE 'Salary here is %', sal_variable;

Mesaj Türleri:

DEBUG,

LOG,

INFO,

NOTICE,

WARNING,

EXCEPTION

PL/pgSQL Record -Tür tanımlama

- PL/pgSQL fonksiyonları, sadece tek bir değer döndürmek zorunda değildir. **Karmaşık sonuçları** veya bir **tabloyu** da döndürebiliriz.
- Bu tarz **composit veri tiplerini** döndürmek için **RECORD** tanımları kullanılmaktadır.

CREATE TYPE tür_ismi **AS** (isim1 tür1, isim2 tür2, ...);

Örnek:

CREATE TYPE urunler

AS (miktar1 **INTEGER**, miktar2 **INTEGER**);

Bu türde değişken tanımlaması şu şekildedir:

depo **URUNLER**;

Yardımcı örnek

- '123456789' ssn'i olan çalışanın ismini, çalıştığı departmanın ismini ve maaşını bulunuz.

```
SELECT      fname, dname, salary
FROM        employee e, department d
WHERE       e.dno = d.dnumber
           AND
           e.ssn = '123456789';
```

Örnek – 2

- SSN'i parametre olarak verilen çalışanın ismini, çalıştığı departmanın ismini ve maaşını ekrana yazdıran PL/pgSQL bloğunu yazın. Bir ssn vererek fonksiyonu çağırınız.

```
CREATE TYPE yeni_tur AS (isim VARCHAR(15), dep_isim VARCHAR(25), maas INTEGER);
```

```
CREATE or REPLACE FUNCTION ornek2 (eno employee.ssn%type)
```

```
RETURNS yeni_tur AS $$
```

```
DECLARE
```

```
    bilgi yeni_tur;
```

```
BEGIN
```

```
SELECT  fname, dname, salary INTO bilgi
FROM    employee e, department d
WHERE   e.dno = d.dnumber AND
        e.ssn = eno;
```

```
RAISE NOTICE 'Calisan ismi: %, departmanin ismi: %, maasi: % TLdir. ',
              bilgi.isim, bilgi.dep_isim, bilgi.maas ;
```

```
RETURN bilgi;
```

```
END;
```

```
$$ LANGUAGE 'plpgsql';
```

```
SELECT ornek2('123456789');
```

```
DROP FUNCTION ornek2 (employee.ssn%type);
```

CURSOR

- Eğer bir **tablo döndürmek istiyorsak CURSOR** tanımlayarak sonucu bunun üzerinden döndürebiliriz.

```
cursor_ismi CURSOR FOR sql_query;
```

Örnek:

```
curs_all CURSOR FOR    SELECT * FROM employee;
```


Örnek – 3

- Numarası verilen bir departmandaki çalışanların isimlerini bulan bir fonksiyon yazınız. Bir departman numarası vererek fonksiyonu çağırınız.

```
CREATE or REPLACE FUNCTION ornek3 (dnum NUMERIC)
```

```
RETURNS void AS $$
```

```
DECLARE
```

```
    yeni_cur CURSOR FOR SELECT fname, lname
```

```
                                FROM   employee
```

```
                                WHERE  dno = dnum;
```

```
BEGIN
```

```
    FOR satir IN yeni_cur LOOP
```

```
        RAISE INFO 'Employee name is % %', satir .fname, satir .lname;
```

```
    END LOOP;
```

```
END;
```

```
$$ LANGUAGE 'plpgsql';
```

```
SELECT ornek3(6);
```

```
DROP FUNCTION ornek3(numeric);
```

Örnek – 4

- Departman numarası verilen bir departmandaki çalışanların toplam maaşını (SUM() fonksiyonundan yararlanmadan) bulan bir fonksiyon yazınız.

```
CREATE FUNCTION ornek4 (dnum NUMERIC)
```

```
RETURNS NUMERIC AS $$
```

```
DECLARE
```

```
    toplam_maas NUMERIC;
```

```
    curs CURSOR FOR SELECT salary FROM employee WHERE dno = dnum;
```

```
BEGIN
```

```
    toplam_maas := 0;
```

```
    FOR satir IN curs LOOP
```

```
        toplam_maas := toplam_maas + satir.salary;
```

```
    END LOOP;
```

```
    RETURN toplam_maas;
```

```
END;
```

```
$$ LANGUAGE 'plpgsql';
```

```
SELECT ornek4(6);
```

```
DROP FUNCTION ornek4(numeric);
```

Örnek – 4 (OUT ile çözümü)

- Departman numarası verilen bir departmandaki çalışanların toplam maaşını (SUM() fonksiyonundan yararlanmadan) bulan ve **OUT değişkeni üzerinden geri döndüren** bir fonksiyon yazınız.

(SELECT sum(salary) FROM employee WHERE dno = X;)

```
CREATE OR REPLACE FUNCTION dep_sum_salary(dnum numeric, OUT sum_sal numeric)
```

```
AS '
```

```
DECLARE
```

```
    emp_cursor CURSOR FOR SELECT salary FROM employee WHERE dno = dnum;
```

```
BEGIN
```

```
    sum_sal := 0;
```

```
    FOR emp_record IN emp_cursor LOOP
```

```
        sum_sal := sum_sal + emp_record.salary;
```

```
    END LOOP;
```

```
END;
```

```
' LANGUAGE 'plpgsql';
```

```
SELECT dep_sum_salary(6);
```

```
DROP FUNCTION dep_sum_salary(numeric);
```

Örnek – 5

Numarası verilen bir projede çalışanların maaşları verilen bir değere tam bölünebiliyorsa, o kişilerin ad, soyad ve maaş bilgilerini HAVING fonksiyonu kullanmadan listeleyen ve geri döndüren fonksiyonu yazınız.

```
CREATE TYPE calisan AS (isim varchar(15), soyisim varchar(15), maas integer);
```

```
CREATE OR REPLACE FUNCTION calisan_listele(pnum project.pnumber%TYPE, bolen integer)
```

```
RETURNS calisan[] AS '
```

```
DECLARE
```

```
    emp_cursor CURSOR FOR SELECT fname, lname, salary FROM employee, works_on WHERE ssn = essn AND pno = pnum;
```

```
    cal calisan[];
```

```
    i integer;
```

```
BEGIN
```

```
    i := 1;
```

```
    FOR emp_record IN emp_cursor LOOP
```

```
        IF emp_record.salary % bolen = 0 THEN
```

```
            cal[i] = emp_record;
```

```
            i := i + 1;
```

```
        END IF;
```

```
    END LOOP;
```

```
    RETURN cal;
```

```
END;
```

```
' LANGUAGE 'plpgsql';
```

note: an array of n elements starts with array[1] and ends with array[n].

```
SELECT calisan_listele('61',16);
```

```
DROP FUNCTION calisan_listele(project.pnumber%TYPE, integer);
```

TRIGGERS (tetikleyiciler)

1. Fonksiyonlar gibi veri tabanına kaydedilirler.
2. VTYS tarafından trigger'ın şartları oluştuğunda **otomatik** olarak çağrılırlar.
3. Tablolar üzerinde değişiklik yapılmak istendiğinde çalışırlar.

INSERT, UPDATE, DELETE

```
CREATE TRIGGER trigger_isim
```

```
{ BEFORE | AFTER } { events }
```

```
ON tablo_adi
```

```
FOR EACH ROW EXECUTE PROCEDURE trigger_fonk_adi();
```

```
CREATE OR REPLACE FUNCTION trig_fonk( )
```

```
RETURNS TRIGGER AS '
```

```
BEGIN
```

```
    Statements;
```

```
    [ RETURN [NULL | OLD | NEW]; ]
```

```
END;
```

```
' LANGUAGE 'plpgsql';
```

Trigger fonksiyonları:

Parametre almazlar
Trigger döndürürler.

1. Tablo ile trigger fonksiyonu bağlanır:

CREATE TRIGGER trig_isim

2. Trigger fonksiyonu yazılır.

CREATE or REPLACE FUNCTION trig_fonk_isim()

INSERT = Sadece **NEW** kullanılır.

UPDATE = **OLD** ve **NEW** kullanılır.

DELETE = Sadece **OLD** kullanılır.

Part	Description	Possible Values
Trigger timing	Trigger'ın harekete geçtiği an	Before / After
Trigger event	Trigger'ı tetikleyen DML	Insert / Update / Delete
Trigger type	Trigger body'nin çalışma sayısı	Statement / Row

Trigger tipi, trigger fonksiyonunun, bir SQL sorgusu için sadece bir kez mi, yoksa trigger olayından etkilenen her bir satır için mi çalışacağını belirler. Varsayılanı "FOR EACH STATEMENT"tır.

NEW: Tetikleyici prosedürün/fonksiyonun body bloğunda kullanılır. Row-level tetikleyiciler için insert/update olaylarında yeni eklenen satırın değerini tutan record yapısındaki değişkendir. Statement-level tetikleyicilerde ve Delete işlemlerinde NEW değişkeni NULL'dır.

OLD: Tetikleyici prosedürün/fonksiyonun body bloğunda kullanılır. Row-level tetikleyiciler için update/delete olaylarında, değişen/silinen eski satırın değerini tutan record yapısındaki değişkendir. Statement-level tetikleyicilerde ve Insert işlemlerinde OLD değişkeni NULL'dır.

Trigger düşürülmesi: **DROP TRIGGER trigger_fonk_adi ON tablo_adi [CASCADE | RESTRICT]**

CASCADE: Tetikleyiciye bağlı olan nesneleri de otomatik olarak düşürür.

RESTRICT: Eğer tetikleyiciye bağlı nesneler varsa tetikleyici düşürülmez. Varsayılanı budur.

Örnek – 6

- Sadece tatil günleri dışında ve mesai saatleri içinde employee tablosuna insert yapılmasına izin veren trigger'ı yazınız.

```
CREATE TRIGGER t_ornek6
```

```
BEFORE INSERT
```

```
ON employee
```

```
FOR EACH ROW EXECUTE PROCEDURE trig_fonk_ornek6();
```

```
CREATE FUNCTION trig_fonk_ornek6()  
RETURNS TRIGGER AS $$  
BEGIN  
    IF ( to_char(now(), 'DY') in ('SAT', 'SUN') OR to_char(now(), 'HH24') not between '08' and '18') THEN  
        RAISE EXCEPTION 'Sadece mesai günlerinde ve mesai saatlerinde insert yapabilirsiniz.' ;  
        RETURN null;  
    ELSE  
        RETURN new;  
    END IF;  
  
END;  
$$ LANGUAGE 'plpgsql';
```

Tetiklenmesi:

```
INSERT INTO employee VALUES('Vladimir', 'S', 'Putin', '666666666', '1952-10-07', '8975  
Rusya', 'M', '125000', '333445555', '5');
```

Düşürülmesi:

Önce:

```
DROP TRIGGER t_ornek6 on employee;
```

Sonra:

```
DROP FUNCTION trig_fonk_ornek6();
```

Örnek – 7

- Departman tablosunda dnumber kolonundaki değer değişince employee tablosunda da dno'nun aynı şekilde değişmesini sağlayan trigger'ı yazınız. (Öncelikle departman tablosundaki yabancı anahtar olma kısıtlarını kaldırmalıyız. Department tablosundaki 'dnumber' sütununa referans veren 3 tablo bulunmaktadır:)
 - **ALTER TABLE** project **DROP CONSTRAINT** project_dnum_fkey;
 - **ALTER TABLE** dept_locations **DROP CONSTRAINT** dept_locations_dnumber_fkey;
 - **ALTER TABLE** employee **DROP CONSTRAINT** foreign_key_const;

```
CREATE TRIGGER t_ornek7
```

```
AFTER UPDATE
```

```
ON department
```

```
FOR EACH ROW EXECUTE PROCEDURE trig_fonk_ornek7();
```

```
CREATE FUNCTION trig_fonk_ornek7()  
RETURNS TRIGGER AS $$  
BEGIN  
  
    UPDATE      employee  
    SET dno = new.dnumber  
    WHERE      dno = old.dnumber;  
  
    RETURN new;  
END;  
$$ LANGUAGE 'plpgsql';
```

Tetiklenmesi:

```
UPDATE department  
SET dnumber = 2  
WHERE dnumber = 5;
```

Örnek – 8

- Maaş inişine ve %10'dan fazla maaş artışına izin vermeyen trigger'ı yazınız.

```
CREATE TRIGGER t_ornek8
```

```
BEFORE UPDATE
```

```
ON employee
```

```
FOR EACH ROW EXECUTE PROCEDURE trig_fonk_ornek8();
```

```
CREATE FUNCTION trig_fonk_ornek8()
```

```
RETURNS TRIGGER AS $$
```

```
BEGIN
```

```
    IF(    old.salary > new.salary OR new.salary>1.1*old.salary) THEN
```

```
        RAISE EXCEPTION 'Maasi dusuremezsiniz ve %%10dan fazla zam yapamazsiniz.';
```

```
        RETURN old;
```

```
    ELSE
```

```
        RETURN new;
```

```
    END IF;
```

```
END;
```

```
$$ LANGUAGE 'plpgsql';
```

Tetiklenmesi:

```
UPDATE employee SET salary = salary*1.12;
```

Düşürülmesi:

Önce:

```
DROP TRIGGER t_ornek8 ON employee;
```

Sonra:

```
DROP FUNCTION trig_fonk_ornek8();
```


Örnek – 9

- Departman tablonuza salary ile aynı tipte total_salary kolonu ekleyin. Employee tablosunda maaş sütununda değişiklik olduğunda department tablosundaki total_salary kolonunda gerekli güncellemeyi yapacak trigger'ı yazınız.

```
ALTER TABLE department ADD COLUMN total_salary INTEGER default 0;
```

```
UPDATE department
```

```
SET total_salary = (SELECT SUM(salary) FROM employee WHERE dno = dnumber);
```

SORU : Yazılacak triggerda insert? update? delete? hangisi veya hangileri olmalı?

```
CREATE TRIGGER t_ornek9  
AFTER INSERT or UPDATE or DELETE  
ON employee  
FOR EACH ROW EXECUTE PROCEDURE trig_fonk_ornek9();
```

```
CREATE FUNCTION trig_fonk_ornek9()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        update    department
        set       total_salary=total_salary-old.salary
        where     dnumber=old.dno;
    ELSIF (TG_OP = 'UPDATE') THEN
        update    department
        set       total_salary=total_salary-old.salary+new.salary
        where     dnumber=old.dno;
    ELSE
        update    department
        set       total_salary=total_salary+new.salary
        where     dnumber=new.dno;
    END IF;
    RETURN new;
END;
$$ LANGUAGE 'plpgsql';
```

Tetiklenmesi 1:

```
INSERT INTO employee VALUES('Vladimir', 'S', 'Putin', '666666667',  
'1952-10-07', '8975 Rusya', 'M', '100000000', '333445555', '1');
```

Tetiklenmesi 2:

```
UPDATE employee SET salary = salary*1.07 WHERE dno = 1;
```

Tetiklenmesi 3:

```
DELETE FROM employee WHERE ssn = '111111103';
```

SON