

# Lab 6

PL/pgSQL Fonksiyon Tanımı.

# PostgreSQL Fonksiyonel Dilleri

■ PostgreSQL'de 4 farklı fonksiyonel (prosedürel) dil var.

1. PL/pgSQL
2. PL/TcL (C programlama dili)
3. PL/Perl (Perl Programlama Dili)
4. PL/Phyton (Phyton Programlama Dili)

# PL/pgSQL Avantajları:

1. Fonksiyonlar ve trigger'lar oluşturulabilir.
2. Döngüsel ve koşula bağlı işlem adımları daha kolay yapılabilir. (while, for, if)
3. Karmaşık sorgulamalar ve hesaplamalar yapılabilir.
4. Kullanıcının kendi amacına yönelik fonksiyon yazması sağlanabilir.
5. PL/pgSQL, SQL'in tüm veri tipi, operatör ve hazır fonksiyonlarını tanıır ve kullanabilir.

# PL/pgSQL Fonksiyon Dönüş Tipleri

1. PL/pgSQL **tek bir değer döndürmek zorunda değildir.**
2. Birden fazla dönüş yapılacaksa «**out**» anahtar sözlüğü kullanılır.
3. PL/pgSQL fonksiyonları **basit tipte** veri döndürebilecekleri gibi **birleşik (composit)** bir veri de döndürebilirler.
4. Ya da bir sonuç kümesinin (tablosunun) adresini gösteren bir **işaretçi (pointer)** döndürebilir.
5. Bütün fonksiyonlar da değer döndürmek zorunda değildir (**return, return void** ).

- Standart PostgreSQL'de prosedürel bir dili kullanmadan önce bu dili oluşturmalıyız:

***"CREATE LANGUAGE plpgsql "***

# PL/pgSQL Fonksiyonların Tanımlanması

```
CREATE FUNCTION fonksiyon_adı (parametre1 tipi, parametre2 tipi,..., [out] parametreN tipi )  
[RETURNS çıktının veri tipi] AS [$$] [']  
DECLARE  
    tanımlamalar;  
BEGIN  
    komutlar;  
[RETURN] [çıktı değeri;] ..  
EXCEPTION  
    kural dışı durumlar;  
END;  
[$$] ['] LANGUAGE plpgsql;
```

# DECLARE

DECLARE

```
isim tür := ilk_değer;
```

- user\_id     INTEGER;
- quantity    NUMERIC;
- url          VARCHAR(20);
- my\_var      tablo.sutunismi%TYPE;

# PL/pgSQL RETURN ve Dönüş Çeşitleri

- PL/pgSQL fonksiyonundan çıkış için «RETURN » anahtar sözcüğü kullanılır.
- Eğer **parametre döndürülmeyecekse** sadece «RETURN» yazılır.
- Veya **birden çok parametre döndürülecekse** «OUT» anahtar sözcüğü kullanılabilir.
- «OUT» veya «RETURN VOID» olmayan fonksiyonlarda «RETURN» ifadesi olmak zorundadır. Yoksa çalışma sırasında hata ile karşılaşılır.



# Örnek – 1

- Girdi olarak verilen 2 sayının toplamını bulan fonksiyonu yazınız ve (22,63) parametreleri için çalıştırınız.

```
CREATE FUNCTION ornek1 (num1 NUMERIC, num2 NUMERIC)
```

```
RETURNS numeric AS '
```

```
DECLARE
```

```
toplam NUMERIC;
```

```
BEGIN
```

```
    toplam := num1+num2;
```

```
    RETURN toplam;
```

```
END;
```

```
' LANGUAGE plpgsql;
```

# Örnek – 1

- Girdi olarak verilen 2 sayının toplamını bulan fonksiyonu yazınız ve (22,63) parametreleri için çalıştırınız.

```
CREATE OR REPLACE FUNCTION ornek1 (num1 NUMERIC, num2 NUMERIC)
```

```
RETURNS numeric AS '
```

```
DECLARE
```

```
toplam NUMERIC;
```

```
BEGIN
```


```
    toplam := num1 + num2;
```

```
    RETURN toplam;
```

```
END;
```

```
' LANGUAGE plpgsql;
```

**:= yerine sadece = kullanılabilir.**  
**toplam = num1 + num2;**



# Örnek – 1 (return'süz yazılımı)

- Girdi olarak verilen 2 sayının toplamını bulan fonksiyonu yazınız ve (22,63) parametreleri için çalıştırınız.

```
CREATE FUNCTION ornek1 (num1 NUMERIC, num2 NUMERIC,  
    OUT num3 NUMERIC)
```

```
AS ‘
```

```
BEGIN
```

```
    num3:=num1+num2;
```

```
END;
```

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

# Fonksiyonun çalıştırılması ve Silinmesi

**SELECT** *fonksiyon\_adı (parametre değerleri);*

**SELECT** ornek1(22,63);

**DROP FUNCTION** *fonksiyon\_adı (parametre tipleri);*

**DROP FUNCTION** ornek1 (NUMERIC, NUMERIC);

# Örnek – 2

- Adı verilen bir departmandaki çalışanların ortalama maaşını bulan bir fonksiyon yazınız.

```
CREATE or REPLACE FUNCTION ornek2 (depname department.dname%type)
```

```
RETURNS real AS '
```

```
DECLARE
```

```
    maas NUMERIC;
```

```
BEGIN
```

```
    SELECT      AVG(salary) INTO maas
    FROM        employee e, department d
    WHERE       e.dno = d.dnumber AND
               d.dname = depname;
    RETURN maas;
```

```
END;
```

```
' LANGUAGE plpgsql;
```

NOT: “real” veri tipi,  
virgülden sonra 7  
basamak tutar.

```
SELECT ornek2('Hardware')
```

```
DROP FUNCTION ornek2 (department.dname%type)
```

## Örnek – 3 : kullanıcıdan parametre almayan örnek

- Departman tablosundaki minimum ve maksimum departman numarasını bulup min\_deptno ve max\_deptno değişkenlerine atan fonksiyonu yazınız.

```
CREATE FUNCTION ornek3 (OUT min_deptno department.dnumber%type,  
                        OUT max_deptno department.dnumber%type)  
  
AS '  
  
BEGIN  
  
    SELECT  MIN(dnumber), MAX(dnumber) INTO min_deptno, max_deptno  
    FROM    department;  
  
END;  
  
' LANGUAGE 'plpgsql';
```

```
SELECT ornek3()  
DROP FUNCTION ornek3 ()
```

# Yardımcı örnek

- 6 no'lu departmanda çalışanların sayısını bulunuz

```
CREATE OR REPLACE FUNCTION ornek(OUT calisan_sayisi NUMERIC)
AS '
BEGIN
    SELECT count(*) into calisan_sayisi
    FROM employee
    WHERE dno=6;
END;
' LANGUAGE plpgsql;
```



## Örnek – 4 : parametre almayan, dönüş tipi void olan örnek

- 6 no'lu departmanda çalışanların sayısı 10'dan azsa departmandaki tüm çalışanların maaşına %5 zam yapın.

```
CREATE FUNCTION ornek4 ()
```

```
RETURNS void AS '
```

```
DECLARE
```

```
    num_worker NUMERIC(3) := 0;
```

```
BEGIN
```

```
    .....
```

```
END;
```

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

```
SELECT      COUNT(*) INTO num_worker  
FROM        employee  
WHERE       dno = 6
```

IF (num\_worker < 10) THEN

```
UPDATE employee  
SET          salary = salary*1.05  
WHERE dno=6;
```

END IF;

```
CREATE FUNCTION ornek4 ()  
RETURNS void AS '  
DECLARE  
    num_worker NUMERIC(3) := 0;  
BEGIN  
    SELECT      COUNT(*) INTO num_worker  
    FROM        employee  
    WHERE       dno = 6  
  
    IF (num_worker < 10) THEN  
        UPDATE employee  
        SET      salary = salary*1.05  
        WHERE    dno=6;  
    END IF;  
END;  
' LANGUAGE 'plpgsql';
```

```
SELECT ornek4()  
  
DROP FUNCTION ornek4 ()
```

# «IF» Koşulu Tanımı:

**IF** koşul **THEN**

yapılacaklar;

**[ELIF** koşul **THEN**

yapılacaklar;]

**[ELSE**

yapılacaklar;]

**END IF;**

# IF-ELSIF-ELSE örnek

IF number = 0 THEN

    result := 'zero';

ELSIF number > 0 THEN

    result := 'positive';

ELSIF number < 0 THEN

    result := 'negative';

ELSE

    result := 'NULL';

END IF;

# «CASE» Tanımı:

CASE secici

WHEN secici\_kosulu1 THEN yapılacaklar1;

WHEN secici\_kosulu2 THEN yapılacaklar2;

...

WHEN secici\_kosuluN THEN yapılacaklarN;

[ELSE secici\_kosulu(N+1)]

END CASE;

# CASE örnek

**CASE x**

*WHEN 1, 2 THEN*

*msg := 'one or two';*

*WHEN 3,4 THEN*

*msg := 'three or four';*

*ELSE*

*msg := 'other value';*

**END CASE;**

# CASE örnek

```
CREATE OR REPLACE FUNCTION myfunc1 (x integer) RETURNS text AS $$  
DECLARE  
msg text;  
BEGIN  
CASE x  
    WHEN 1,2 THEN  
        msg := 'one or two';  
    WHEN 3,4 THEN  
        msg := 'three or four' ;  
    ELSE  
        msg := 'other value';  
END CASE;  
RETURN msg;  
END;  
$$ LANGUAGE plpgsql
```



# «WHILE» Tanımı:

```
WHILE dongu_kosulu LOOP  
    yapılacaklar...  
END LOOP;
```

# For döngüsü

FOR sayac IN başlangıç..bitiş

*LOOP*

*Kodlar.....*

*END LOOP;*

FOR sayac IN REVERSE başlangıç..bitiş **BY** kaçır

*LOOP*

*Kodlar.....*

*END LOOP;*

FOR sayac IN başlangıç..bitiş **BY** kaçır

*LOOP*

*Kodlar.....*

*END LOOP;*

```
DO $$  
BEGIN  
    FOR x IN 1 .. 5  
        LOOP  
            RAISE NOTICE 'Sayı: %', x;  
        END LOOP;  
END; $$
```

```
DO $$  
BEGIN  
    FOR x IN REVERSE 5 .. 1  
        LOOP  
            RAISE NOTICE 'Sayı: %', x;  
        END LOOP;  
END; $$
```

```
DO $$  
BEGIN  
    for x IN REVERSE 5 .. 1 BY 2  
        LOOP  
            RAISE NOTICE 'Sayı: %', x;  
        END LOOP;  
END; $$
```

## Örnek – 5 : return ifadesinden sonuç döndüren örnek

- Verilen bir sayıyı 1 arttıran fonksiyonu yazınız.

```
CREATE FUNCTION increment(t INTEGER)
```

```
RETURNS INTEGER AS ‘
```

```
BEGIN
```

```
    RETURN t + 1;
```

```
END;
```

```
‘ LANGUAGE plpgsql;
```

## Örnek – 6

İsmi verilen bir departmanda çalışanların ortalama maaşı, verilen bir değerden düşük ve o departmandaki kadın çalışanların maaşlarının toplamı verilen bir limitin üstündeyse, o departmanda 1'den fazla projede çalışanların maaşlarına yine verilen bir oranda zam yapan fonksiyonu yazınız.

➤ `SELECT kosullu_zam_yap('Research', 50000, 20000, 5);`

CREATE OR REPLACE FUNCTION kosullu\_zam\_yap(bolum\_ismi department.dname%TYPE, ort\_maas real, f\_top\_maas employee.salary%TYPE, zam\_orani real) RETURNS VOID AS '

DECLARE

ger\_ort\_maas real;

kadin\_maaslari integer;

bolum\_no department.dnumber%TYPE;

BEGIN

SELECT dnumber INTO bolum\_no FROM department WHERE dname = bolum\_ismi;

SELECT AVG(salary) INTO ger\_ort\_maas FROM employee WHERE dno = bolum\_no;

SELECT SUM(salary) INTO kadin\_maaslari FROM employee WHERE dno = bolum\_no AND sex = "F";

IF ger\_ort\_maas < ort\_maas AND kadin\_maaslari > f\_top\_maas THEN

UPDATE employee SET salary = salary\*zam\_orani/100 + salary WHERE ssn IN (SELECT essn  
FROM employee, works\_on WHERE

ssn = essn AND dno = bolum\_no GROUP BY essn HAVING COUNT(\*) > 1);

END IF;

END;

'LANGUAGE plpgsql;

SON