Veritabanı uygulama dersi - 8: PL/pgSQL Record/Cursor ve Trigger Tanımları:

PL/Pqsql fonksiyonlarını tanımlarken create function dedikten sonra (‘) işareti kullanılır. Bu durum fonksiyonun body bloğunda (') işaretini kullanmak istediğimizde sıkıntı yaratır. Ya body bloğunda (') işaretini kullanacağımız yerde tek tırnak yerine peşpeşe 2 tane tek tırnak koyarız ya da başka bir delimeter ile fonksiyonun bloğunu başlatırız.
Kullanılan diğer delimeter alternatifi $$ işaretidir. Body bloğu bittikten sonra yine bu işareti kullanırız. Ancak body kısmında da $$ karakterlerinin geçmesi gerekiyorsa bla_bla şeklinde $ işaretleri arasına bir tag koyabiliriz. Fonksiyon bloğunu bitirince de aynı tag'le kapatırız. Tag'ler case sensitive'dir. Fonksiyonu, TAG ile açıp; tag ile kapatamayız.

Record / Cursor Tanımları

- PL/pgSQL, fonksiyonları sonucunda tek bir değer ve basit tipte olan veriler dönmek zorunda değildir.
- Fonksiyondan dönen değerler basit tipte değil de record'lar gibi composit veri tipinde veya bir sonuç tablosu formatında olabilir.
- Bu durumlarda dönen veriyi record olarak tanımlarız. Lokal olarak kullanacağımız değişkenler içinde de record tipte değişkenler bulunacaksa bunları da record olarak tanımlayabiliriz.
- Fonksiyondan bir sonuç tablosu dönecekse, **CURSOR** tanımlayarak sonucu cursor yardımıyla döndürebiliriz.
- Tüm sorguyu bir kerede çalıştırmak yerine sorgu sonucunu kısaltarak alan **cursor**'ları kullanabiliriz. Böylece sorgu sonucu tek seferde okunmaz, her seferinde birkaç satır okunur. Böylece çok satırdan oluşan sorguların sonucunda memory yetmezliği gibi durumlarla da karşılaşılmamış olur.

Record tanımı:	**CURSOR tanımı:**
CREATE TYPE my_record_type as (field1 type1, field2 type2, …); **Ör:** CREATE TYPE sum_prod AS (sum int, product int) Bu tipi, lokal değişken ya da dönüş tipi olarak fonksiyonlarda kullanabiliriz artık: my_record_variable my_record_type;	cursor_name CURSOR FOR sql_query; **Ör:** my_cur CURSOR FOR select * from emp;

Ekrana Bilgilendirme Mesajları ve Hata Mesajları Yazdırmak:

Ekrana mesaj yazdırabilmek için RAISE ifadesi kullanılır:

Raise [level] ‘Mesaj’;

"Level" seçeneği, ne tür mesaj yazdırmak istediğimizi söyler: DEBUG, LOG, INFO, NOTICE, WARNING ve EXCEPTION değerlerini alabilir. Default'u EXCEPTION'dır. Ekrana bilgi vermek amacıyla NOTICE veya INFO kullanılabilir.

Ör: RAISE NOTICE 'Salary here is %', sal_variable;

Örnekler

1. SSN'i parametre olarak verilen çalışanın ismini, çalıştığı departmanın ismini ve maaşını ekrana yazdıran PL/pgSQL bloğunu yazın. Bir ssn vererek fonksiyonu çağırınız.
2. Numarası verilen bir departmandaki çalışanların isimlerini bulan bir fonksiyon yazınız. Bir departman numarası vererek fonksiyonu çağırınız.
3. Departman numarası verilen bir departmandaki çalışanların toplam maaşını (SUM() fonksiyonundan yararlanmadan) bulan bir fonksiyon yazınız.

Cevaplar

1. Bunu gerçekleştirmek için fonksiyon tanımından önce çalışan ismi, departman ismi ve maaş alanlarını içeren bir record oluşturun:

```
CREATE TYPE my_record AS (isim varchar(20), dep_isim varchar(20), maas numeric);
CREATE OR REPLACE FUNCTION ornek1 (eno employee.ssn%type) RETURNS my_record AS $$
DECLARE
emprec my_record;
BEGIN
        select fname, dname, salary into emprec from employee e, department d where ssn = eno and e.dno = d.dnumber;
        raise notice 'Calisan ismi: %, departmanin ismi: %, maasi: % TLdir. ', emprec.isim, emprec.dep_isim, emprec.maas ;
        return emprec;
END;
$$ LANGUAGE  'plpgsql';
```

Çağırılması: select ornek1('123456789'); **Düşürme:** DROP FUNCTION ornek1 (employee.ssn%type); drop type my_record;

2.
```
CREATE OR REPLACE FUNCTION ornek2 (dnum numeric) RETURNS void AS $$
DECLARE
        emp_cur CURSOR FOR select fname, lname from employee where dno = dnum;
BEGIN
    FOR emp_rec IN emp_cur  LOOP
        RAISE INFO  'Employee name is % %', emp_rec.fname, emp_rec.lname;
    END LOOP;
END;
$$ LANGUAGE  'plpgsql';
```

Çağırılması: select ornek2(6); **Düşürme:** DROP FUNCTION ornek2(numeric);
NOT: Döngü içinde "RAISE INFO mesaj" yerine; "RAISE NOTICE mesaj;" da yazılabilirdi.

3.
```
CREATE OR REPLACE FUNCTION ornek3 (dnum numeric) RETURNS numeric AS $$
DECLARE
        toplam_maas numeric;
        emp_cur CURSOR FOR select salary from employee where dno = dnum;
BEGIN
        toplam_maas := 0;
        FOR emp_rec IN emp_cur  LOOP
                toplam_maas := toplam_maas+emp_rec.salary;
        END LOOP;
        RETURN toplam_maas;
END;
$$ LANGUAGE  'plpgsql';
```

Çağırılması: select ornek3 (6); **Düşürme:** DROP FUNCTION ornek3(numeric);

TRIGGER TANIMI

Tetiklemeler de yordamlar gibi veritabanına kaydedilirler ama program kodu içerisinden çağırılmazlar, veritabanı tarafından otomatik olarak başlatılırlar. INSERT, UPDATE, DELETE gibi DML komutları, trigger'ları başlatır. PL/psSQL'de tetikleyici tanımları aşağıdaki gibidir:

CREATE TRIGGER *trigger_isim* { **BEFORE** | **AFTER** } { ***event1*** [OR ***event2*** OR ...] } **ON** *tablo_adı* **[FOR [EACH] { ROW | STATEMENT }]**
EXECUTE PROCEDURE *trigger_fonk_adi(arguments*);

Trigger fonksiyonunun dönüş tipi TRIGGER olmalıdır. Örnek:

```
CREATE OR REPLACE FUNCTION trig_fonk() RETURNS TRIGGER AS '
BEGIN
    Statements;
    [ RETURN [NULL | OLD | NEW]; ]
END;
' LANGUAGE 'plpgsql';
```

Trigger fonksiyonunun içerisinde hata mesajı yazdırmak için:

```
RAISE EXCEPTION 'Hata mesajı: şu tabloya şu işlemi yapamazsınız, vs...';
```

Part	Description	Possible values
Trigger timing	Trigger event'inin harekete geçtiği an	**Before/After**
Trigger event	Trigger'ı tetikleyen DML statement'ı	**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. insert/update olaylarında yeni eklenen satırın değerini tutan record yapısındaki değişkendir. Delete işlemlerinde NEW değişkeni NULL'dır.

OLD: Tetikleyici prosedürün/fonksiyonun body bloğunda kullanılır. update/delete olaylarında, değişen/silinen eski satırın değerini tutan record yapısındaki değişkendir. Insert işlemlerinde OLD değişkeni NULL'dır.

Trigger düşürülmesi:
DROP TRIGGER *trigger_fonk_adi* **ON** *tablo_adı*

ÖRNEKLER

1. Sadece tatil günleri dışında ve mesai saatleri içinde employee tablosuna insert yapılmasına izin veren trigger'ı yazınız.
2. Departman tablosunda dnumber kolonu değişince employee tablosunda da dno'nun aynı şekilde değişmesini sağlayan trigger'ı yazınız.
3. Maaş inişine ve %10'dan fazla maaş artışına izin vermeyen trigger'ı yazınız.
4. Departman tablonuza salary ile aynı tipte total_salary kolonu ekleyin. Employee tablosunda bir işçinin maaşında maaş değişikliği olduğunda departman tablonuzdaki total_salary kolonunda da gerekli güncellemeyi yapacak trigger'ı yazınız.

CEVAPLAR

1.
```
CREATE OR REPLACE FUNCTION trig_fonk_ornek1() 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 gunlerinde ve mesai saatlerinde insert yapabilirsiniz.';
        RETURN NULL;
    else RETURN NEW;
    end if;
```

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

CREATE TRIGGER **ornek1** BEFORE insert ON employee FOR EACH ROW EXECUTE PROCEDURE **trig_fonk_ornek1();**

Trigger ve fonk'u düşürmek:
Önce: DROP TRIGGER ornek1 on employee; **Sonra:** DROP FUNCTION trig_fonk_ornek1();

```
2. CREATE OR REPLACE FUNCTION trig_fonk_ornek2() RETURNS TRIGGER AS $$
BEGIN
        update employee set dno = new.dnumber where dno = old.dnumber;
        RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER ornek2 AFTER update ON department
FOR EACH ROW EXECUTE PROCEDURE trig_fonk_ornek2();
```

Trigger ve fonk'u düşürmek:
Önce: DROP TRIGGER ornek2 on department; **Sonra:** DROP FUNCTION trig_fonk_ornek2();

```
3. CREATE OR REPLACE FUNCTION trig_fonk_ornek3() RETURNS TRIGGER AS $$
BEGIN
        if (old.salary > new.salary or new.salary>1.1*old.salary) then
                Raise exception 'Maasi dusuremezsiniz ve %%10"dan fazla zam yapamazsınız.';
                Return old;
        Else    Return new;
        end if;
END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER ornek3 BEFORE update ON employee FOR EACH ROW EXECUTE
PROCEDURE trig_fonk_ornek3();
```

Önce: DROP TRIGGER ornek3 on employee; **Sonra:** DROP FUNCTION trig_fonk_ornek3();

```
4. ALTER TABLE department ADD COLUMN total_salary INTEGER default 0;
CREATE OR REPLACE FUNCTION ornek4() RETURNS TRIGGER AS $ornek4$
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;
$ornek4$ LANGUAGE 'plpgsql';

CREATE TRIGGER ornek4 AFTER insert or update or delete ON employee FOR EACH ROW
EXECUTE PROCEDURE ornek4();
```

Önce: DROP TRIGGER ornek4 on employee; **Sonra:** DROP FUNCTION trig_fonk_ornek4();
Department'i eski haline getirelim: ALTER TABLE department **DROP COLUMN** total_salary;