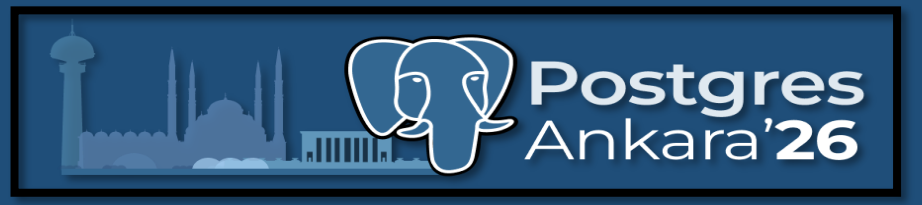

PostgreSQL'de Fonksiyon, Trigger ve Stored Procedure kullanımı ve bu nesneler üzerinden Audit Log oluşturulması

Yasin TATAR

İçerik

- **Pg/PgSQL**
- **Stored Procedure**
- **Procedure**
- **Trigger**
- **Uygulama Örneği**

PL/pgSQL

- Procedural Languages (yordamsal dil)
- Oracle PL/SQL diline yazılım kuralları açısından çok benzer. Bu geçişkenliği artırır.
- PostgreSQL veritabanı üzerinde kullanılabilen yordamsal dillerinden biridir.
- Yüklenebilir bir dildir. (varsayılan olarak gelir, 9.0 ve üzeri)
- Yordamsal diller, SQL sorgulama dilin yetersiz kaldığı işlemler için kullanılmaktadır.
- SQL sorgularının ve programatik ifadeleri kullanır

PL/pgSQL Avantajları

- PostgreSQL'in tüm veri tiplerine destekler
- Program kontrol deyimleri kullanıla bilir
- Tanımlı tüm fonksiyonlarına ulaşabilirsiniz
- Veritabanının üzerinde çalıştığı için network yükünüzü azaltır
- Blok bazlı yapıdadır
- Alt blok (sub-blocks) ları destekler
- Kolay anlaşılır ve yazılır

PL/pgSQL Dil Yapısı

- Basit yapı
- Kodun her bir bölümü bir fonksiyon olarak çalışması için tasarlanmıştır.
- C diline benzer yapı
- Kodun fonksiyonlarla yazılması
- Değişkenlerin kullanılmadan önce declare edilmesi
- Büyük-küçük harfe duyarlı değildir.
- '' işareti ile kullanılırsa büyük küçük harf duyarlı olur.

PL/pgSQL Dil Yapısı

CREATE FUNCTION *foksiyon_ad* (*parametreler*) RETURNS *type* AS '
DECLARE
  tanımlar;
[...]
BEGIN
 kodlar;
[...]
END;
' LANGUAGE 'plpgsql';

PL/pgSQL Değişkenler

- CONSTANT anahtar kelimesi
  - Sabit değer belirtir.
- NOT NULL anahtar kelimesi
  - Değişkenin NULL olamayacağını belirtir.
- DEFAULT anahtar kelimesi
  - Değişkene ön tanımlı bir değer atar.

```
DECLARE
    değişken adı [ CONSTANT ] veri_tipi [ NOT NULL ] [{ DEFAULT | := } değer ];
BEGIN

Ornek_Degistken     CONSTANT INTEGER := 1;
```

PL/pgSQL'de Yorumlar

- Tek Satır Yorumlar (comments)
  - Tek satırlı yorumlar iki çizgi ile (--) başlarlar.
  - Satır sonu karakteri barındırmazlar.

```
-- test yorumu
```

- Blok yorumlar
  - Blok yorumlar /* ile başlarlar.
  - Bir veya birden fazla satır içerirler.
  - */ ile biterler.

```
/*
   blok yorum
*/
```

Dinamik sorgular;

EXECUTE anahtar kelimesi ileçalıştırılan SQL sorguları dışında, bir fonksiyondaki tüm PL/pgSQL expressionları PostgreSQL backendinin yaşama süresi içinde sadece bir kez prepare edilir.

Kontrol Yapıları

- IF … THEN
- IF … THEN … ELSE
- IF … THEN …. ELSIF … THEN … ELSE
- CASE … WHEN … THEN … ELSE … END CASE
- CASE WHEN … THEN … ELSE … END CASE

```
BEGIN
IF degisken > 1 THEN
      LOOP
            EXIT WHEN degisken2> 100; -- degisken2 > 100 ise LOOP tan çık
            CONTINUE degisken2 < 80; -- degisken2 < 80 olduğu sürece LOOP a dön
            -- Şartlar sağlandığı sürece çalışacak kodlar
      END LOOP
      ELSIF ((degisken <=1)  and (derger3 =20) )THEN
        --- deger3 = 20 ise çalışacak komutlar
      ELSE
       --Şartta Uymayan
END IF;
RETURN True;
END;
```

IF-THEN

IF ***boolean-expression*** THEN

Statements

END IF;

- Koşul doğru ise THEN ve END IF arasındaki ifadeler işletilir. Aksi takdirde bu blok atlanır.

ÖRN:

```
IF kullanici_id <> 0 THEN
   UPDATE tb_kullanici SET email = mvrprime@gmail.com
WHERE kullanici_id=5;
END IF;
```

IF-THEN-ELSE

IF ***boolean-expression*** THEN

    statements

ELSE

    statements

END IF;

- Koşul doğru ise THEN ve ELSE arasındaki ifadeler işletilir. Koşul doğru değilse ELSE ve END IF arasındaki ifadeler işletilir.

ÖRN:

```
IF adet > 0 THEN
      INSERT INTO tb_kullanici_adet (adet)
      VALUES (71);
   RETURN 't';
ELSE
   RETURN 'f';
END IF;
```

IF-THEN-ELSIF

IF ***boolean-expression*** THEN

Statements

[ELSIF ***boolean-expression*** THEN

Statements

[ELSIF ***boolean-expression*** THEN

statements

...]]

[ELSE ***statements***]

END IF;

ÖRN:

```
IF sayi = 0 THEN
        sonuc := 'sıfır';
ELSIF sayi > 0 THEN
        sonuc := 'pozitif';
ELSIF sayi < 0 THEN
        sonuc := 'negatif';
ELSE
        sonuc := 'NULL';
END IF;
```

CASE ... WHEN ... THEN ... ELSE ... END CASE

- Eşitliğe dayalı koşullu yürütme sağlar.
- Direk değer döndürür.
- Belirli durumlara göre farklı işlemler yapılmak istendiğinde bu yapı kullanılır.
- WHEN ifadeleri art arda değerlendirilir.

Bir eşleşme bulunduğunda gelinen ifade çalışır. Eğer eşleşme yoksa en son ELSE ifadesi çalışır. Eğer ELSE ifadesi de bir değer döndürmüyorsa CASE_NOT_FOUND hatası ortaya çıkar.

CASE ***search-expression***

WHEN ***durum1*** [, ***expression*** [...]] THEN ***İşletilecek_kod_bloğu***

[WHEN ***durum2*** [, ***expression*** [...]] THEN ***İşletilecek_kod_bloğu***

...]

[ELSE ***İşletilecek_kod_bloğu***]

END CASE;

ÖRN: NOT örneği ile değiştirilebilir (90-> AA, 80 -> BB)

```
CASE x
        WHEN 1,2 THEN
        msg := ' x değeri 1 veya 2 dir.';
        WHEN 3,4 THEN
        msg := ' x değeri 3 veya 4 arasındadır.';
END CASE;
```

LOOPS (DÖNGÜLER)

LOOP

Bir veya birden fazla işlem satırını, bir koşula bağlı olarak, belirli sayıda veya bir koşul sağlandığı sürece tekrarlayarak çalıştıran kalıplara döngü adı verilir.

[<<***label***>>]

LOOP

statements

END LOOP [***label***];

EXIT

Loop ile oluşturulan döngüyü koşulsuz olarak sonlandırmaya yarar.

EXIT [***label***]

[WHEN ***boolean-expression***];

EXIT kullanarak, koşullu deyimin ve gövdenin geri kalanındaki kodun atlanması sağlanabilir. Böylece döngü acil bir şekilde bitirilmiş olur.

Döngü içinde bir EXIT ile karşılaşıldığında döngü bitirilir ve kontrol döngüden sonra gelen ifadeye geçer.

CONTINUE

Hiçbir etiket belirtilmemişse, bir sonraki yenileme en içteki döngüden başlar.

CONTINUE [***label***]

[WHEN ***boolean-expression***];

```
ÖRN:

LOOP
        EXIT WHEN count > 100;
        CONTINUE WHEN count < 50;
END LOOP;
```

WHILE

Mantıksal ifadesi true olduğu sürece kod bloğu çalışır.

[<<***label***>>]
    WHILE ***boolean-expression***
    LOOP ***statements***
END LOOP [***label***];

FOR

[<<***label***>>]

FOR ***name*** IN [REVERSE] ***expression*** .. ***expression*** [BY ***expression***]

LOOP ***statements***

END LOOP [***label***];

Örnek EXECUTE

```
l_sql =‘Update tb_ad Set = PostgreSQL Wher id <=100’;
EXECUTE l_sql;

GET DIAGNOSTICS l_count = ROW_COUNT;

 IF l_count > 0  THEN
        raise notice ‘Etkilenen Satır Sayısı ’, l_count;
 END IF;
```

Örnek FOUND

```
Update tb_firma ad= ‘PostgreSQL’ Where id = 1;
        IF NOT FOUND THEN
            Insert Into tb_firma (id,ad) values (1,’PostgreSQL’)
        END IF;

Select 1 from tb_firma Where id = 1;
        IF FOUND THEN
            delete From tb_firma Where id = 1;
END IF;
```

Stored Procedure

CREATE [**OR REPLACE**] **FUNCTION** fonksiyon_adı (parametre tipi) RETURNS dönüs_turu AS $$degisken_Adi

DECLARE tanımlamalar;

BEGIN komutlar;

[RETURN] [çıktı değeri;]

END;

$$ LANGUAGE plpgsql;

```
CREATE FUNCTION topla(integer,integer) RETURNS
integer AS $$
DECLARE
        l_sonuç Integer;
BEGIN
        l_sonuc:= $1 + $2;
        RETURN l_sonuc;
END;
$$ LANGUAGE plpgsql;
```

- Fonksiyonu çağırmak için:

SELECT fonksiyon_adi (parametre değerleri);

PERFORM fonksiyon_adi (parametre değerleri);

İpucu: PERFORM ile kullanımda fonksiyon bir değer döndürmez.

```
CREATE FUNCTION topla(integer,integer) RETURNS integer AS
$body$
DECLARE
    l_sonuc INTEGER;
BEGIN
    l_sonuc:= $1 + $2;
    return l_sonuc;
END;
$body$
LANGUAGE 'plpgsql';
--------------------------------------------------------------------------------
prod=# SELECT topla(5,7);
topla
------
12
(1 row)
```

```
CREATE OR REPLACE FUNCTION fatura.sp_get_donem_bilgisi (
  g_donem integer,
  out o_yaziyla text
)
RETURNS text AS
$body$
DECLARE
  aylar text [ ];
BEGIN
  aylar =  ARRAY['Ocak','Şubat','Mart','Nisan','Mayıs','Haziran','Temmuz','Ağustos','Eylül','Ekim','Kasım','Aralık'];
  o_yaziyla = aylar[substring(g_donem::text from 5 for 2)::INTEGER]|| '-' || substring(g_donem::text from 1 for 4) ;
  RETURN ;
END;
$body$
LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
COST 100;
```

```
prod=# Select fatura.sp_get_donem_bilgisi (202612) ;

sp_get_donem_bilgisi
------
Aralık- 2026

(1 row)
```

Yorumlar (Comments)

Programlama dillerinin yorum yapısına benzemektedir. Bunun için pl/pgsql de 2 yol vardır.

Tek satırlı yorumlar

- İki çizgi ile (--) başlarlar, satır sonu karakteri barındırmazlar.

Blok yorumlar (çok satırlı yorumlar)

- Blok yorumlar /* ile başlarlar ve birden fazla satır içerirler. */ ile biter.

DECLARE

Değişken_adı [CONSTANT] *veri_tipi* [NOT NULL] [{ DEFAULT | := } *değer*];

BEGIN

ÖRN:

quantity integer DEFAULT 06;

url varchar := 'http://mvrprime.com';

user_id CONSTANT integer := 17;

```
DECLARE
        kayit RECORD;
  BEGIN
        SELECT INTO kayit * FROM kullanici WHERE
kullanici_id = 7;
    IF kayit.homepage IS NULL THEN
    RETURN 'http://mvrprime.com';
    END IF;
END;
```

Fonksiyon Parametreleri (Function Parameters)

- IN
- OUT
- INOUT
- VARIADIC

InOut Parametre tipi, Hem veri girişi hem veri çıkışında kullanılır. Değeri alt program içerisinde set edilebilir ve program sonlandığında sahip olduğu son değeri dışarıya döndürür.

ÖRN:

```
CREATE OR REPLACE FUNCTION kare(INOUT a integer ) AS $$
BEGIN
        a:= a + a;
END;
$$ LANGUAGE plpgsql;
```

VARIADIC parametre tipi, Aynı veri türüne sahip tüm argümanları bir dizi (array) olarak fonksiyonda kullanmaya yarar.

Bir fonksiyonu sonucunda tek bir değer dönmek zorunda değildir. Birden fazla dönüş olacaksa output parametre tanımı kullanılmalıdır. Bunun dışında fonksiyonlar,

- Basit tipte bir veri dönebilir;
- Record tipinde composit bir data dönebilir;
- Sonuç tablosunun pointer'ı gibi tek bir instance dönebilir.
- Hatta bazen hiç değer dönmeyebilir. Hiç bir değer dönmüyor ise fonksiyonun sonunda sadece "return" ya da "return void" denilebilir .

```
Select s.hizmet_durumu_id,
       s.hizmet_id,
       s.hizmet_turu_id,
       MH.bakiye, mh.musteri_id
from fatura.sp_bagli_hizmetler(1201) s
    left join tb_musteri_hizmet mh on mh.id = s.hizmet_id
```

```
Select s.hizmet_durumu_id,
       s.hizmet_id,
       s.hizmet_turu_id,
       MH.bakiye
from tb_musteri_hizmet mh
left join fatura.sp_bagli_hizmetler(mh.id) s on mh.id = s.hizmet_id
 Where mh.musteri_id = 140
```

```
CREATE OR REPLACE FUNCTION fatura.sp_bagli_hizmetler (
  g_hizmet_id bigint,
  out hizmet_id bigint,
  out hizmet_turu_id bigint,
  out hizmet_durumu_id bigint
)
RETURNS SETOF record AS
$body$
DECLARE
  l_satir RECORD;
```

```
BEGIN
  SELECT
      mh.id, mh.hizmet_turu_id, mh.hizmet_durumu_id
    INTO
      hizmet_id, hizmet_turu_id, hizmet_durumu_id
    FROM tb_musteri_hizmet mh WHERE mh.id = g_hizmet_id;
  RETURN NEXT;
```

```
FOR l_satir IN
   SELECT
      mh.id,
      mh.hizmet_turu_id,
      mh.hizmet_durumu_id
    FROM ktv.tb_musteri_hizmet mh
    WHERE
      mh.bagli_hizmet_id = g_hizmet_id
  LOOP
```

```
 hizmet_id = l_satir.id;
    hizmet_turu_id = l_satir.hizmet_turu_id;
    hizmet_durumu_id = l_satir.hizmet_durumu_id;
    RETURN NEXT;
    RETURN QUERY SELECT * FROM fatura.sp_bagli_hizmetler(hizmet_id);
  END LOOP;
END;
$body$
LANGUAGE 'plpgsql‘;
```

Fonksiyonu düşürmek için:

DROP FUNCTION fonksiyon_adi (parametre değerleri);

Alt bloklar (sub-blocks)

- PL/pgSQL kod bloğu sınırsız sayıda alt blok (sub-block) içerebilir. Alt bloklar, normal bloklarla aynı şekilde okunur ve yorumlanırlar, dolayısıyla onlar da kendi içlerinde alt blok içerebilirler.

- Alt bloklar büyük PL/pgSQL fonksiyonlarının düzenlenmesinde yardımcı olmaktadır. Tüm alt bloklar normal blok yapısını kullanırlar; yani DECLARE anahtar sözcüğü ile başlayıp BEGIN ile devam ederler; ifadelerin olduğu kısımdan sonra da END anahtar kelimesi ile biterler.

NULL Kontrolü

CALLED ON NULL INPUT

Input değerlerinde NULL varsa çalış

RETURNS NULL ON NULL INPUT

Input değerlerinde NULL varsa direk sonuç NULL döner

Güvenlik

SECURITY INVOKER

Çalışması için Foksiyon içindelki neşelere **Çağıran** kullanıcının yetkisi olası gerekir.

SECURITY DEFINER

Çalışması için Foksiyon içindeki neşelere **Oluşturan** kullanıcının yetkisi olası gerekir.

```
CREATE OR REPLACE FUNCTION public.sp_ogrenci_notler_getir (
)
RETURNS SETOF tb_ogrenci_notlar
LANGUAGE 'plpgsql‘
CALLED ON NULL INPUT
SECURITY INVOKER
AS'
BEGIN
 RETURN QUERY Select * from tb_ogrenci_notlar;
END;  ';
```

```
CREATE OR REPLACE FUNCTION public.sp_ogrenci_notler_getir (
)
RETURNS SETOF tb_ogrenci_notlar
LANGUAGE 'plpgsql‘
RETURNS NULL ON NULL INPUT
SECURITY DEFINER
AS'
BEGIN
 RETURN QUERY Select * from tb_ogrenci_notlar;
END;  ';
```

Procedure

CREATE [OR REPLACE] PROCEDURE
name ([[*argmode*] [*argname*] *argtype* [{ DEFAULT | = } *default_expr*] [, ...]])
{ LANGUAGE *lang_name*
| TRANSFORM { FOR TYPE *type_name* } [, ...]
| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER
| SET *configuration_parameter* { TO *value* | = *value* | FROM CURRENT }
| AS '*definition*'
| AS '*obj_file*', '*link_symbol*'
| *sql_body*
} ...

```
CREATE OR REPLACE PROCEDURE public.p_deneme (
)
LANGUAGE 'plpgsql'
SECURITY INVOKER
AS
$body$
Declare
        l_test Integer;
BEGIN
        Update tb_dersler set ders_kodu ='MAT-1' Where ders_kodu = 'MAT1';
        Commit;
        l_test= ' Sayısal Deger';
END;
$body$;
```

```
call p_deneme();
---------------------------------------------------------------------
HATA:  integer tipi için geçersiz giriş sözdizimi: " Sayısal Deger "
LINE 1: l_test= ' Sayısal Deger';
```

TRIGGER (TETİKLEYİCİ)

Trigger’lar, ilişkilendirildiği tablo, view veya foreign key de meydana gelen olaylarda, belirtilen işlemleri gerçekleştirmek için kullanılmaktadır.

Trigger’lar bir satır üzerinde işlem yapılmaya çalışılmadan önce (kısıtlar sınanmadan ve INSERT, UPDATE, DELETE veya

TRUNCATE yapılmadan önce) ya da işlem tamamlandıktan sonra (kısıtlar sınandıktan ve INSERT, UPDATE, DELETE, TRUNCATE tamamlandıktan sonra) çalışabilir.

Kullanımı şu şekildedir:

CREATE [CONSTRAINT] TRIGGER ***name***
{ BEFORE | AFTER | INSTEAD OF }
{ ***event*** [OR ...] }

Eğer trigger olaydan önce çalışırsa, geçerli satır için işlemi atlayabilir ya da veri girilen satır değişebilir (sadece INSERT ve UPDATE işlemleri için).

Eğer trigger olaydan sonra çalışırsa, tüm değişiklikler, son insert, update veya delete işlemi trigger'a geçerli olur.

FOR EACH ROW seçili bir trigger, işlemi değiştiren her satır için bir defa çağrılır.

Örneğin, 10 satırı etkileyen bir DELETE, her satır silinişinde bir kere olmak üzere 10 ayrı defa ON DELETE tetiğinin çağrılmasına sebep olur.

Buna karşılık, FOR EACH STATEMENT seçili bir trigger belirtilen bir işlem için işlemin kaç satırı etkilediğinden bağımsız olarak, sadece bir defa çalıştırılır (hatta, işlem hiçbir satırı değiştirmese bile trigger yine de çalıştırılacaktır).

Eğer aynı olay için aynı türden çok sayıda trigger tanımlanmışsa, bunlar isimlerine göre alfabetik sırayla çalıştırılırlar.

SELECT herhangi bir satırı
değiştirmediğinden, SELECT trigger’ları oluşturulamaz.

Rules ve views böyle durumlarda daha uygundur.

Parametreler

- **Name :** Yeni trigger’ı belirtecek isim. Bu isim, aynı tablodaki diğer trigger isimlerinden farklı olmalıdır.
- **Before/ After/ Instead Of :** İşlevin olaydan önce mi sonra mı çağrılacağını belirler.
- **Event :** INSERT, UPDATE, DELETE veya TRUNCATE’den biri; trigger’ı çalıştıracak olayı belirtmek için kullanılır. OR kullanarak çok sayıda olay belirtilebilir.
- **OF** : Kolon bazlı kontrol için atama yapar

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/ DELETE	Tables and foreign tables	Tables, views, and foreign tables
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/ DELETE	Tables and foreign tables	Tables, views, and foreign tables
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/ DELETE	Views	—
	TRUNCATE	—	—

- **table_name :** Trigger içerecek tablonun ismi (şema nitelemeli olabilir), view veya foreign table.
- **referenced_table_name :** Bu seçenek foreign key (yabancı anahtar) constraints için kullanılır ve genel kullanım için tavsiye edilmez. Bu kullanım, yalnızca constraint trigger için belirtilebilir.
- **function_name :** Tetikle çalıştırılan, kullanıcı tarafından argümansız ve geri dönüş türü trigger olarak bildirilmiş ve tanımlanmış bir işlev.

ÖRN:

```
CREATE TRIGGER check_update
        BEFORE UPDATE OF Columns ON accounts
        FOR EACH ROW
EXECUTE PROCEDURE check_account_update();
```

```
CREATE OR REPLACE FUNCTION public.sp_ortalama_hesapla (
)
RETURNS trigger LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
AS'
BEGIN
    New.ortalama =      ((New.not_1 + New.Not_2)/2 *0.3) +      (New.Not_3 * 0.7);
     New.sonuc = New.Ortalama >=50;
    RETURN New;
END;';
```

- CREATE TRIGGER tb_ogrenci_notlar_tr
- BEFORE UPDATE
- ON tb_ogrenci_notlar
-
- FOR EACH ROW
- EXECUTE PROCEDURE public.sp_ortalama_hesapla();

- **Audit Log**

```
Create EXTENSION hstore;
CREATE SCHEMA loglar AUTHORIZATION postgres;

CREATE TABLE loglar.tb_log (
  log_id BIGSERIAL,
  schema_name TEXT NOT NULL,
  table_name TEXT NOT NULL,
  user_name TEXT NOT NULL,
  user_id INTEGER,
  action_timestamp TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  action TEXT NOT NULL,
  old_values public.hstore,
  new_values public.hstore,
  record_id BIGINT NOT NULL,
  CONSTRAINT audit_action_check CHECK (action = ANY (ARRAY['i'::text, 'd'::text, 'u'::text]))   ) ;
```

```
CREATE TABLE loglar.log_tb_dersler (
    ) INHERITS (loglar.tb_log)
    WITH (oids = false);
```

```
CREATE OR REPLACE FUNCTION loglar.sp_set_log ( )
RETURNS trigger LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
PARALLEL UNSAFE
COST 100
AS
$body$
DECLARE
  l_user_id integer;
  l_id BIGINT = 0;
  l_val HSTORE;
  l_exp HSTORE;
BEGIN
```

```
BEGIN
    IF current_query() LIKE '/*%' THEN
      l_user_id = (string_to_array(current_query(), ' '))[2];
    END IF;
  EXCEPTION WHEN invalid_text_representation THEN
    l_user_id = NULL;
  END;


  IF tg_op = 'INSERT' THEN
      l_val = hstore(NEW);
      l_id = new.id;
  END IF;
```

```
IF tg_op = 'UPDATE' THEN
    IF new.* <> old.* THEN
            l_exp = hstore(new.*) - hstore(old.*);

        INSERT INTO loglar.tb_log(schema_name,table_name, user_name, user_id, action,
                                  old_values, new_values, record_id)
            VALUES (TG_TABLE_SCHEMA::TEXT,tg_table_name::text,  current_user::text, l_user_id,
                    'u', hstore(old . *) - hstore(new . *),  hstore(new . *) - hstore(old . *), l_id );

    END IF;
    RETURN new;
```

```
ELSIF tg_op = 'DELETE' THEN
    INSERT INTO loglar.tb_log(schema_name,table_name, user_name, user_id, action,
old_values, record_id)
    VALUES (TG_TABLE_SCHEMA::TEXT,tg_table_name::text, current_user::text, l_user_id, 'd',
hstore(old . *), l_id);
    RETURN old;
```

```
 ELSIF tg_op = 'INSERT' THEN
    INSERT INTO loglar.tb_log(schema_name,table_name, user_name, user_id,
                                                    action, new_values, record_id)
    VALUES (TG_TABLE_SCHEMA::TEXT,tg_table_name::text, current_user::text,
                    l_user_id, 'i',  hstore(new . *),l_id);
    RETURN new;
  end if;
end;
$body$;
```

```
CREATE TRIGGER tb_ogrenci_tr
 BEFORE INSERT OR UPDATE OR DELETE
 ON   tb_ogrenci

FOR EACH ROW
 EXECUTE PROCEDURE loglar.sp_set_log();
```

```
Update ab2024.tb_ogrenci Set ad='Ahmet Mehmet' Where id = 11;

/*% 234234 %*/ INSERT INTO  ab2024.tb_ogrenci  (id, ad, soyad, kimlik_no, ogrenci_no, bolum)
                    VALUES (  35,  'Veli',  'GÜRBÜZ',  123456789,  9876,  'Mühendis');

/*% 758 %*/ Delete From ab2024.tb_ogrenci Where id = 35;
```

- Select * from loglar.tb_log ;
- Select * from loglar.log_tb_ogrenci;

log_id	schema_name	table_name	user_name	user_id	action_timestamp	action	old_values	new_values	record_id
1	ab2024	tb_ogrenci	postgres	Null	12.05.2026 03:52:15	u	"ad"=>"Ahmet"	"ad"=>"Ahmet-1"	0
3	ab2024	tb_ogrenci	postgres	Null	12.05.2026 03:53:45	i	Null	"ad"=>"Hakkı", "id"=>"31", "bolum"=>"Deneme ", "soyad"=>"Bulut",	31
4	ab2024	tb_ogrenci	postgres	Null	12.05.2026 03:56:39	u	"ad"=>"Cem"	"ad"=>"Ahmet Mehmet"	0
5	ab2024	tb_ogrenci	postgres	Null	12.05.2026 03:57:42	i	Null	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis", "soyad"=>"GÜRBÜZ",	35
6	ab2024	tb_ogrenci	postgres	Null	12.05.2026 03:58:08	d	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis",	Null	0
7	ab2024	tb_ogrenci	postgres	234.234	12.05.2026 03:59:21	i	Null	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis", "soyad"=>"GÜRBÜZ",	35
9	ab2024	tb_ogrenci	postgres	758	12.05.2026 04:00:01	d	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis",	Null	0

Select user_id,old_values, new_values from loglar.tb_log

Where old_values @> '"ad"=>"Cem"'

user_id	old_values	new_values
0	"ad"=>"Cem"	"ad"=>"Ahmet Mehmet"

Select table_name, action_timestamp, old_values from loglar.tb_log

Where action ='d' and user_id= 758;

table_name	action_timestamp	old_values
tb_ogrenci	12.05.2026 04:00:01	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis", "soyad"=>"GÜRBÜZ", "kimlik_no"=>"123456789", "ogrenci_no"=>"9876"

```
CREATE OR REPLACE FUNCTION public.delete_kotnrol ( )
RETURNS trigger LANGUAGE 'plpgsql'
AS
$body$
BEGIN
  RETURN NULL;
END;
$body$;
-----------------------------------------------------------------------------------------------
CREATE TRIGGER tb_log_tr
  BEFORE DELETE
  ON loglar.tb_log

FOR EACH ROW
  EXECUTE PROCEDURE public.delete_kotnrol();
```

- Delete From loglar.tb_log Where log_id = 9 ;

  Query OK, 0 rows affected (execution time: 0 ms; total time: 0 ms)

- Select * From loglar.tb_log Where log_id = 9 ;

log_id	schema_name	table_name	user_name	user_id	action_timestamp	action	old_values
9	ab2024	tb_ogrenci	postgres	758	12.05.2026 04:00:01	d	"ad"=>"Veli", "id"=>"35", "bolum"=>"Mühendis", "soyad"=>"GÜRBÜZ", "kimlik_no"=>"123456789", "ogrenci_no"=>"9876"

Sorular ……

Dinlediğiniz İçin Teşekkürler