به نام خدا

دستور کار آزمایشگاه پایگاه داده

جلسه پنجم

توابع، رویه‌ها و آغازگرها

استاد درس

دکتر شهریاری

تابستان ۹۹

توابع

استفاده از توابع همانند زبان‌های برنامه نویسی کمک به عدم بازنویسی فرایندها می‌کند. در PostgresSQL استفاده از توابع باعث بوجود آمدن سرویس‌های آماده مختلفی می‌شود، که کاربر می‌تواند در هر زمانی از آن‌ها استفاده و پاسخ خود را دریافت کند. غالبا یک تابع شامل یکسری declarations ، expressions و statements می‌باشد. در PostgresSQL تعداد زیادی از توابع بصورت خودساخته وجود دارد که مناسب انواع ساختار های داده است. در این قسمت تمرکز بر روی توابع در PostgresSQL می‌باشد.

علاوه بر توابعی که در خود PostgresSQL وجود دارد، کاربران می‌توانند تابع دلخواه خودشان را نیز با استفاده از زبان‌های مختلف برنامه نویسی و پرس و جو بنویسند. که این زبان‌ها شامل C،SQL،PL/pgSQL،PL/Tcl،PL/Python،PL/Perl است. به غیر از زبان های C،SQL،PL/pgSQL که بصورت پیشفرض وجود دارد، برای بقیه زبان‌ها باید ابزار مناسب آن‌ها در دیتابیس نصب کنید.

بطور خاص می‌توان موارد زیر را به عنوان دلایل استفاده از پایگاه داده شمرد.

- پیاده سازی منطق‌های پیچیده در پایگاه داده
- بوجود آوردن اعمالی که پس از یا قبل از اجرا شدن کوری‌ها با استفاده از trigger ها اعمال شوند.
- مرتب سازی و پاک سازی کد با کاهش کدهای تکراری

بطور کلی می‌توان توابع را در سه نوع طبقه بندی کرد. بصورت پیشفرض نوع تابع در هنگام پیاده‌سازی بصورت volatile می‌باشد. و می‌توان دو نوع دیگر یعنی stable و immutable را برای تابع انتخاب کرد. انتخاب درست نوع تابع کمک شایانی در بهبود کارایی پایگاه داده می‌کند. در ادامه این سه نوع توضیح داده شده است.

Volatile : این نوع تابع می‌تواند در هر بار فراخوانی موفق، مقدار خروجی متفاوتی را بوجود آورد. حتی با فرض این که مقدار ورودی‌های آن یکسان باشد. همچنین می‌تواند اطلاعات درون پایگاه داده را تغییر دهد. به عنوان مثال تابع random() به این صورت است.

Stable and immutable : در این دو نوع تابع نمی‌تواند اطلاعات درون پایگاه داده را تغییر دهد، و تضمین می‌شود که مقدار خروجی با مقادیر ورودی یکسان، یکتا باشد. در توابع stable این تضمین در یک محدوده[1] می‌باشد ولی در immutable همه محدوده ها را دربر می‌گیرد. تابع round() مثالی از این دو نوع می‌باشد.

سوال: تابع now() از چه نوعی می باشد؟

---

[1] scope

توابع پایگاه داده را می‌توان از جنبه های دیگری نیز بررسی و دسته بندی کرد، که در ادامه در مورد هرکدام توضیحاتی آورده شده است.

توابع تجمعی[1]

این نوع از توابع بر روی مجموعه‌ای از داده‌ها عمل می‌کنند تنها یک مقدار را با محاسبه مقادیر یک ستون بر می‌گرداند. در لینک زیر لیستی از این توابع وجود دارد.

https://www.postgresql.org/docs/9.5/functions-aggregate.html

توابع مقیاسی[2]

این توابع برای هرکدام از رکوردها مقدار منحصر به فردی را بر می‌گرداند.

```
1.  SELECT
2.    UPPER(s.ContactName)
3.  FROM
4.    Customers s
```

در این مثال نام مشتریان به صورت حروف بزرگ نمایش داده می‌شود. تابع UPPER() بر روی تمام رکوردها عمل کرده و برای هرکدام از آها یک مقدار خروجی منحصر به فرد ایجاد کرده است.

> **سوال**: خروجی پرس و جوی زیر چیست؟

```
1.  SELECT
2.      SUBSTRING(ContactName, 1, CHARINDEX(' ', ContactName) - 1)
3.  FROM
4.      Customers
```

توابع رتبه بندی[3]

این نوع توابع برای هرکدام از سطر های موجود در یک محدوده مقداری را به عنوان امتیازبندی آن سطر ارائه می‌کند.

---

[1] Aggregate Functions
[2] Scalar Functions
[3] Ranking functions

توابع تعریف شده توسط کاربر[1]

پایگاه داده این اجازه را می‌دهد که کاربران بتوانند تابع تعریف کنند. برای این کار میتوان هم از طریق محیط گرافیکی انجام داد و هم از طریق یک پرس و جو. با استفاده از قطعه کد زیر می‌توان یک تابع را پیاده سازی و ذخیره کرد.

```
1.  create function get_film_count(len_from int, len_to int) returns int language plpgsql a
    s $ $ declare film_count integer;
2.  begin
3.  select
4.    count(*) into film_count
5.  from
6.    film
7.  where
8.    length between len_from
9.    and len_to;
10. return film_count;
11. end;
12. $ $;
```

- در خط اول بعد از create function می توان مشخص کرد که آیا این تابع باید جایگزین شود و یا خیر. و پس از آن نام تابع درج می‌شود.
- در قسمت بعدی باید مقادیر ورودی این تابع مشخص شود. این مقادیر می‌تواند خالی هم باشد.
- در خط دوم مشخص می‌شود که نوع خروجی این تابع چه چیزی می‌باشد.
- در ادامه باید مشخص شود که زبان مورد استفاده برای پیاده سازی این تابع چه چیزی می‌باشد.
- قسمت انتهای، بدنه اصلی تابع است که مقادیر تعریفی و منطق تابع در آن نوشته می‌شود.

مقادیر ورودی و خروجی

مقادیر ورودی در یک تابع می‌توانند سه نوع مختلف داشته باشند. که بصورت پیش فرض نوع اول برای مقادیری که نوعشان مشخص نشده است قرار می‌گیرد. و انواع آن‌ها به شرح زیر می‌باشد.

IN: این نواع به معنای ورودی می‌باشد. در این نوع می‌توان یک مقداری را برای تابع ارسال کرد و درون تابع از آن استفاده کرد.

OUT : این نوع درواقع خروجی تابع می‌باشد. و هنگام فراخوانی تابع لازم نیست مقدار دهی شود. هیچگاه نمی‌توان به صورت همزمان آرگمان ورودی با این نوع داشته باشیم و هم مقدار خروجی (return) در بدنه تابع اجرا کنیم.

INOUT : می‌تواند هم OUT باشد و هم IN .

```
1.  Create
2.  or replace function fun(n inout int) returns int as $ $ begin n := n + 1;
```

[1] User defined functions

```
end;
$ $ language 'plpgsql';
SELECT
  public.func(20)
```

نتیجه:

	func bigint
1	21

سوال: اگر در یک تابع تعداد آرگمان ها با نوع OUT یا INOUT بیشتر از یک باشد چه چیزی به عنوان خروجی تابع در نظر گرفته می شود؟

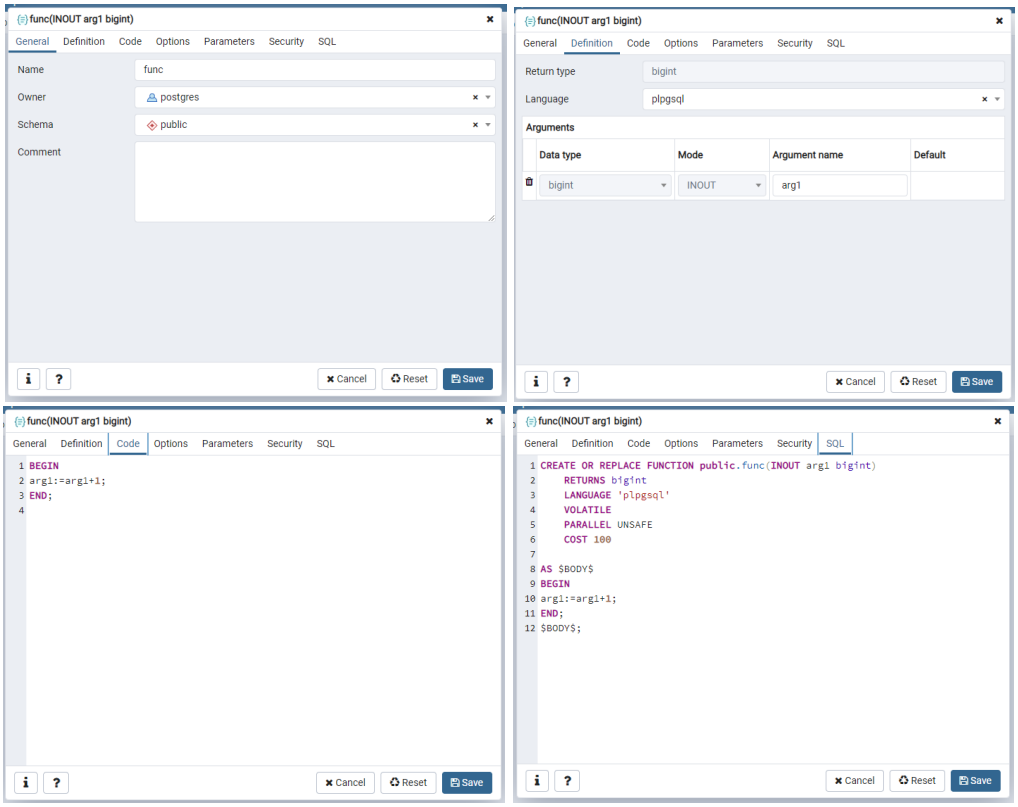
نوع مقدار خروجی در ابتدا باید مشخص شود. نوع خروجی می‌تواند یک جدول باشد. یعنی پس از پایان یافتن اجرا، تابع به عنوان خروجی یک جدول بر می‌گرداند. به عنوان مثال خروجی تابع زیر یک جدول است.

```
CREATE OR REPLACE FUNCTION public.get_film4(IN p_pattern character varying)
    RETURNS TABLE(film_title character varying, film_release_year integer)
    LANGUAGE 'plpgsql'

AS $BODY$
begin
    return query
        select
            title,
            release_year::integer
        from
            film
        where
            title ilike p_pattern;
end;
$BODY$;
```

ایجاد تابع با pgAdmin

علاوه بر نحوه ایجاد تابع بصورتی که گفته شد، می‌توان با استفاده از pgAdmin و رابط کاربری نیز این کار را انجام داد. برای این کار باید بر روی قسمت Functions راست کلیک کرده و بر روی گزینه create کلیک شود. و پس از آن صفحه‌ای مشابه زیر باز می‌شود. و اطلاعات لازم از قبیل نام تابع، مقادیر ورودی و خروجی و ... می‌توان با رابط کاربری انتخاب و وارد کرد. در قسمت code باید بدنه اصلی تابع نوشته شود و قسمت sql پرس و جوی نهایی که با استفاده از آن این تابع ساخته شده است را نشان می‌دهد.



فراخوانی توابع

برای فراخوانی توابع کافی است از نام آن تابع به همراه مقادیر ورودی آن استفاده کنید. مقادیر ورودی می‌تواند به ۳ صورت به تابع ارسال شود.

```
1.  SELECT public.func(10,20)//positional notation
2.  SELECT public.func(arg1=>10,arg2=>20)//named notation
3.  SELECT public.func(arg1:=10,arg2:=20)//named notation
4.  SELECT public.func(arg1:=10,20)//mixed notation
```

توجه : روش named notion زمانی مورد استفاده قرار می‌گیرد که تعداد مقادیر ورودی زیاد باشد.

حذف تابع

برای حذف تابع می‌توان از دستور Drop به شکل زیر استفاده کرد.

```
1.  DROP function [if exists] function_name(argument_list) [cascade | restrict]
```

چند نکته در حذف توابع وجود دارد. همانظور که مشخص است برای حذف یک تابع علاوه بر نام تابع، باید آرگمان های ورودی آن‌را نیز وارد کنید.دلیل این امر یکتا بودن توابع با داشتن نام و آرگمان های ورودی آن‌ها می‌باشد که به آن امضای تابع گفته می‌شود. نکته دیگر عبارت if exists است که جلوی خطا در هنگام نبود چنین تابعی را می‌گیرد. تابع ممکن است به یک موجودیت دیگر نیز وابسته باشد و در هنگام حذف دچار مشکل شود مثل trigger، لذا برای حذف کردن یک تابع به همراه همه موجودیت های وابسته می‌توان از cascade استفاده کرد.

تفاوت توابع و رویه های ذخیره شده در PostgresSQL

اصلی ترین تفاوت بین توابع و رویه‌ها در خروجی است که تولید می‌کنند. در توابع همیشه انتظار می‌رود که یک خروجی داشته باشد، اما در رویه‌ها خروجی وجود ندارد. درواقع هدف اصلی رویه ها صرفا انجام یکسری فعالیت و سپس پایان یافتن آن‌ها می‌باشد. در حالی که توابع قرار است پردازشی انجام داده و خروجی تولید کنند. در نسخه های قدیمی تر postgres، رویه‌ها همان توابعی بوده‌اند که صرفا خروجی برای آن‌ها تعبیه نشده بود ولی در نسخه های جدید تر بصورت خاص به آن‌ها پرداخته شده است. تفاوت دیگر امکان ایجاد تراکنش‌ها در یک رویه می‌باشد. یکی از معایب توابع عدم امکان ایجاد تراکنش در آن ها بوده است، که در نسخه‌های جدید تر این پایگاه داده امکان ایجاد رویه با پشتیبانه تراکنش‌ها به آن اضافه شده است.

رویه‌های ذخیره شده

نحوه ایجاد یک رویه مشابه ایجاد یک تابع است با درنظر گرفتن تفاوت‌های آن، یعنی عدم داشتن خروجی و امکان ایجاد تراکنش. البته امکان اجرای دستور return در یک رویه وجود دارد اما بدون مقدار و رویه در لحظه اجرای این دستور متوقف می‌شود.

در مثال زیر پرس و جویی است که یک رویه را ایجاد می‌کند. هدف این رویه، روالی است که بواسطه آن مقداری پول از حسابی به حساب دیگری منتقل می‌شود.

```
create
or replace procedure transfer(
    sender int,
    receiver int,
    amount dec
) language plpgsql as $ $ begin -- subtracting the amount from the sender's account
update
    accounts
set
    balance = balance - amount
where
    id = sender;
-- adding the amount to the receiver's account
update
    accounts
set
    balance = balance + amount
where
    id = receiver;
commit;
end;
$ $
```

برای اجرایی کردن این رویه باید آن را به شکل زیر فراخوانی کرد.

```
CALL public.transfer(1,2,1000);
```

حذف رویه‌ها

طریقه حذف رویه‌ها همانند حذف توابع با دستور Drop می‌باشد.

```
drop procedure [if exists] procedure_name (argument_list) [cascade | restrict]
```

آغازگرها[1]

آغازگرها در postgresSQL توابعی هستند که هنگام رخدادی مثل اضافه کردن به جدول،بروزرسانی جدول و حذف از جدول، بصورت خودکار فراخوانی می‌شود و می‌توان مشخص کرد که قبل یا بعد از رخداد اجرا شود. در واقع یک آغازگر وابسته به یک جدول و رخدادهای آن است. برای ایجاد یک آغازگر ابتدا یک تابع تعریف می‌شود و سپس این تابع به یک جدول متصل می‌شود.

بصورت کلی دو نوع statement-level و row-level تعریف می‌شوند. تفاوت این دو در تعداد اجرای آن‌ها می‌باشد. به طور مثال زمانی که یک دستور بروزرسانی برای یک جدول اجرا می‌شود، ممکن است تعداد زیادی از ردیف‌های جدول را در بر بگیرد، در این حالت اگر نوع آغازگر استفاده شده بصورت row-level باشد، آنگاه به تعدادی که از ردیف های جدول بروزرسانی شده اند اجرا می‌شود ولی اگر بصورت statement-level باشد فقط یکبار اجرا می‌شود.

شاید بتوان گفت مهم ترین کارکرد آغازگرها نقش نظارتی آن‌ها بر روی جداول است، از آن جای که یک پایگاه داده ممکن است از طریق نرم افزار های مختلفی مورد استفاده قرار بگیرد، باید بتوان به گونه‌ای بر روی اعمالی که بر روی پایگاه داده انجام می‌شود نظارت داشت. بطور مثال یکی از نیازمندی های سیستم ممکن است وجود تاریخچه‌ای از تغییرات جداول باشد. و یا این که با استفاده از آغازگرها می‌توان جامعیت اطلاعات درون پایگاه داده را تضمین کرد، بطور مثال پس از اضافه شدن یک ردیف در جدول سبد خرید، باید از موجودی آن کالا کم شود و یا تغییراتی بر روی جداول دیگر صورت بگیرد.

ایجاد آغازگرها

ایجاد آغازگرها در دو مرحله انجام می‌شود. در مرحله اول باید یک تابع ایجاد شود و در مرحله دوم تابع ایجاد شده باید به جدل مورد نظر متصل شود.

---

[1] Trigger

برای ایجاد تابع، باید دقت شود که هیچ مقدار ورودی نباید داشته باشد و همچنین مقدار خروجی آن باید از نوع trigger باشد. به عنوان مثال پرس و جوی زیر تابعی را ایجاد می‌کند که قرار است قبل از هر بروزرسانی مقدار last_update رکورد جدید را برابر تاریخ آن لحظه قرار دهد. این تابع در مرحله بعدی به جدول یا جداولی می‌تواند متصل شود.

```
1.  CREATE OR REPLACE FUNCTION public.last_updated()
2.      RETURNS trigger
3.      LANGUAGE 'plpgsql'
4.      VOLATILE
5.      COST 100
6.  AS $BODY$
7.  BEGIN
8.      NEW.last_update = CURRENT_TIMESTAMP;
9.      RETURN NEW;
10. END
11. $BODY$;
```

با این حال که این تابع نباید ورودی داشته باشد، اما یکسری اطلاعات در یک قالب مشخص به نام TriggerData در لحظه اجرای آن، به تابع داده می‌شود. این ساختار داده حاوی تعدادی متغییر محلی می‌باشد. بطور مثال old و new دو متغیر کاربردی آن است، که مقدار اول بیانگر ردیف اطلاعات قبل از تغییر و مقدارم دوم پس از آن می‌باشد. متغیر های دیگری نیز وجود دارد که نام آن ها با عبارت _TG شروع می‌شود. ازقبیل TG_WHEN و TG_TABLE_NAME. برای درک کاربرد این متغیر ها کافی است به این نکته دقت کنیم که یک تابع ممکن است برای تعداد زیادی جدول و یا تعداد زیادی رخداد اجرا شود، در نتیجه لازم است در مواردی بدانیم که این کدام جدول و یا کدام رخداد بوده است.

حال پس از ایجاد تابع می توانیم یک آغازگر ایجاد کرد و آن را به تابع موردنظر متصل کرد. برای این کار از ساختار پرس و جوی زیر استفاده می‌کنیم.

```
1.  CREATE TRIGGER trigger_name
2.     {BEFORE | AFTER} { event }
3.     ON table_name
4.     [FOR [EACH] { ROW | STATEMENT }]
5.         EXECUTE PROCEDURE trigger_function
```

همانطور که مشخص است، پس از نام آغازگر باید مشخص شود که قبل یا بعد از چه رخدادی باید این آغازگر فعال شود. بطور مثال می‌توان گفت که بعد از هر insert در جدول table_name. و در انتهای هم می‌توان نوع آغازگر را که در ابتدا توضیح داده شد مشخص کرد.

پرس و جوی زیر در ابتدا یک تابع ایجاد می‌کند و سپس آن را به یک آغازگر متصل می‌کند. همانطور که مشخص است این تابع قرار است هنگام بروزرسانی یک جدول اجرا شود و مقادیر قدیمی را درون یک جدول دیگر ذخیره کند.

```
1.  CREATE OR REPLACE FUNCTION log_last_name_changes()
2.    RETURNS TRIGGER
3.    LANGUAGE PLPGSQL
4.    AS
```

```
$$
BEGIN
    IF NEW.last_name <> OLD.last_name THEN
        INSERT INTO employee_audits(employee_id,last_name,changed_on)
        VALUES(OLD.id,OLD.last_name,now());
    END IF;

    RETURN NEW;
END;
$$
CREATE TRIGGER last_name_changes
  BEFORE UPDATE
  ON employees
  FOR EACH ROW
  EXECUTE PROCEDURE log_last_name_changes();
```

ایجاد آغازگر با رابط کاربری

درست همانند ایجاد یک تابع، می‌توان آغازگر هارا هم با استفاده از رابط کاربری ایجاد کرد. با توجه به این نکته که ایجاد یک آغازگر دارای دو مرحله می‌باشد. مرحله اول باید در قسمت Trigger functions ها یک تابع ایجاد شود و سپس با توجه به این که کدام جداول مورد نظر است، در قسمت Triggers هر کدام یک آغازگر ساخته و آن را به تابع ایجاد شده متصل کرد.

حذف آغازگرها

برای حذف و یا تغییر یک آغازگر می‌توان از Drop و Alter همانند تابع استفاده کرد.

تمرین

۱. تابعی بنویسید که یک عدد ۱۱ رقمی به عنوان ورودی دریافت کند و تشخیص دهد مربوط به شماره تلفن همراه است یا شماره تلفن ثابت یا هیچکدام. در صورتی که شماره مربوط به تلفن همراه است عبارت"mobile phone number" را خروجی دهد. در غیر اینصورت ۳ رقم مربوط به کد شهر و ۸ رقم دیگر شماره را بصورت مجزا چاپ کند و مشخص کند شماره مربوط به اصفهان است یا تهران. در صورتی که خروجی مربوط به هیچ یک از موارد نبود عبارت " invalid phone number" را در خروجی نشان دهد.

مثال:

09135479645 خروجی: mobile phone number

03122222222 خروجی: city code =031 ,last 8 digits=22222222 city =Esfahan

33333333333 خروجی: invalid phone number

۲. تابعی بنویسید که دو تاریخ مشخص به عنوان ورودی دریافت کند و فیلم هایی که در بین این دوتاریخ به اجاره رفته اند را خروجی دهد.

۳. با استفاده از تابع سوال قبل، تابع دیگری بنویسید که مشخصات مشتریانی را در خروجی دهد که در بین دو تاریخ داده شده فیلمی را اجاره کرده‌اند ولی هنوز تحویل نداده اند.(مقدار return_date در صورت عدم بازگشت NULL می‌باشد).

۴. آغازگری برای جدول rental بنویسید که هنگام اضافه یا بروزرسانی هرکدام از ردیف های جدول، بررسی کند آیا زمان return_data با توجه به مقدار rental_duration در جدول film، معتبر است یا خیر (تاخیر داشته است یا خیر). به این صورت که اگر ردیفی اضافه یا تغییر کرد و مدت زمان اجاره فیلم بیشتر از مقدار تعیین شده در جدول film باشد یک ردیف در جدولی به نام Log ایجاد شود و مشخص شود که کدام مشتری، چند روز تاخیر داشته است. (جدول log را باید ایجاد کنید و حداقل id مشتری در آن وجود داشته باشد با تعداد روز های تاخیر)

۵. یکه رویه بنویسید که دو مقدار ورودی دریافت کند و مقدار اول را از یکی از ستون های جدول برای تمام سطر ها کم کند. بطور مثال فرض کنید این رویه قرار است مقدار rental_duration در جدول film را برای هر ردیفی به یک مقدار افزایش دهد.