Stored Procedure (Function) Postgres

Banco de Dados II

Sintaxe

```
create [or replace] function <name> ([<param>])
  returns <type> as $$
declare
        -- variables
begin
        -- logic
end;
$$ language <plpgsql|SQL>;

chamada:
select <name> ([param]); -- or
select * from <name>([param]);
```

```
create [or replace] procedure <name> ([<param>])
 as $$
declare
        -- variables
begin
        -- logic
end;
$$ language <plpgsql|SQL>;

chamada:
call <name>([param]);
```

Exemplo - Procedure[1]

```
CREATE TABLE IF NOT EXISTS t_demo (id int);

CREATE OR REPLACE PROCEDURE sample_1(x int)
LANGUAGE SQL
AS $$
      INSERT INTO t_demo VALUES (x);
$$;
CALL sample_1(10);
```

```
SELECT xmin, * FROM t_demo;
```

```
CREATE OR REPLACE PROCEDURE sample_2(soma int)
LANGUAGE plpgsql
AS $$
DECLARE
  v_sum int8;
BEGIN
  CALL sample_1(1);
  CALL sample_1(2);
  CALL sample_1(3);
  COMMIT;
  CALL sample_1(4);
  CALL sample_1(5);
  SELECT sum(id) FROM t_demo
                 WHERE id < soma INTO v_sum;
  RAISE NOTICE 'debug info: %', v_sum;
END; $$;
call sample_2(20);
```

[1]https://www.cybertec-postgresql.com/en/stored-procedures-getting-started/

Mensagens

Comando raise:

raise info | log | notice | info | warning | exception 'texto <e variáveis>';

Exemplo:

```
do $$ -- bloco anônimo
begin
  raise info 'information message %', now() ;
  raise log 'log message %', now();
  raise debug 'debug message %', now();
  raise warning 'warning message %', now();
  raise notice 'notice message %', now();
end $$;
```

```
do $$ -- bloco anônimo
declare
  email varchar:='duplicate@com';
begin
  raise exception 'duplicate email %', email
        using hint='check email again';
end $$;
```

Variáveis

variable_name [constant] data_type [[default | :=] expression];

Tipos:

- todos vistos no create table (numeric, date, time, etc)
- int
- real

%data_type permite herdar o tipo data_type:

```
my_id customer.id%type;
my_customer customer%rowtype;
```

Exemplo:

```
do $$ declare tname train.nametr%type;
      begin     select nametr   from train
                                into tname
                                where idtr=2;
                raise notice ' Id: % Name: %' ,2,tname;
      end; $$;
```

Comando condicional

```
if      then        end if;
if      then        else        end if;
if      then        elsif       then        end if;
```

```
do $$
  declare
     train_id constant train.idtr%type := 2;
     train_name train.nametr%type;
  begin
        select nametr into train_name from train where idtr=train_id;
        if not found then
                raise notice 'Train id not found: %',train_id;
        else
                raise notice 'Train name: %',train_name;
        end if;
end; $$;
```

Comando condicional

```
do $$
  declare
     train_id constant train.idtr%type := 2;
     train_cap train.capac%type;
     msg varchar(100);
  begin
        select capac into train_cap from train where idtr=train_id;
        if train_cap > 100 then
                msg:= ' Long ' ;
        elsif train_cap > 50 then
                msg:= ' Medium ' ;
        else
               msg:= ' Short ' ;
        end if;
        raise notice  ' Size % ' , msg;
end; $$;
```

Comando condicional

```
case <variable>
        when <value1> then <bloco>
        when <value2> then <bloco>
        [else <bloco>]
end case;
do $$       declare
                mvar int := 2;
            begin
                case mvar
                    when 1 then raise notice 'One';
                    when 2 then raise notice 'Two';
                    else
                        raise notice 'Do not know';
                end case;
            end;
$$;
```

Comando laço

```
loop <bloco> end loop;
while <condition> loop <bloco> end loop;
for <var> in [reverse] <from> .. <to> loop <bloco> end loop;
        exit [when <condition>];
        continue [when <condition>];

do $$   declare
            counter int := 0;
        begin
            loop
                counter := counter +1;
                exit when counter > 10;
                continue when mod(counter,2) = 0;
                raise notice '%',counter;
            end loop;
        end; $$;
```

Modos dos parâmetros - In Out Inout

```
create function get_stats_train (out minc int, out maxc int, out avgc int)
as $$
begin
	select min(capac), max(capac), avg(capac)
	into    minc, maxc, avgc
	from train;
end;
$$ language plpgsql;

select get_stats_train(); -- Ou select * from get_stats_train();
```

Tratamento exceção

```
begin
        --  código que pode gerar o erro
        exception
                when <condition> then <handle exception>;
                when <condition> then <handle exception>;
                when others     then <handle exception>;
end;
create function get_name (id int) returns void
as $$ declare name train.nametr%type;
begin
        select nametr into strict name from train where idtr=id;
        exception
                when sqlstate 'P0002'  then raise exception 'Train id not found: %', id;
end; $$ language plpgsql;
```

lista erros: https://www.postgresql.org/docs/current/errcodes-appendix.html

Cursors

É uma estrutura de banco que permite navegar individualmente pelos resultados de uma consulta

Primeiro (bloco DECLARE): cursor_name CURSOR FOR query;
Segundo (no corpo): OPEN cursor_name;
Terceiro: FETCH NEXT FROM cursor_name INTO variable_list;
Finaliza: CLOSE cursor_name;

Geralmente:

```
LOOP
    FETCH NEXT FROM cursor_name INTO variable_list;
            :
    EXIT WHEN NOT FOUND;
            <rows processing>
END LOOP;
```

Cursors

```
$$
    DECLARE
        my_cursor CURSOR FOR select * from train_route;
        my_tuple train %rowtype;
    BEGIN
        OPEN my_cursor;
        LOOP
            FETCH NEXT FROM my_cursor INTO my_tuple;
            EXIT WHEN not found;
            RAISE NOTICE '% +1=%', my_tuple.datet, my_tuple.datet+1;
        END LOOP;
        CLOSE my_cursor;
    END;
$$;
```

Triggers

- É uma função que pode ser executada automaticamente quando os comandos INSERT, DELETE or UPDATE são invocados.
- Pode ser executada para cada tupla (FOR EACH ROW) ou quando o comando finaliza (FOR EACH STATEMENT).
- Pode ser executada antes (BEFORE) ou depois (AFTER) da execução do comando.
- Ordem:
  - Cria-se a função cujo o tipo de retorno deve ser TRIGGER.
  - Criar a trigger informando o função que será disparada automática

```
CREATE TRIGGER name
    {BEFORE | AFTER} {UPDATE OR SELECT OR DELETE}
    ON table
    FOR EACH {ROW | STATEMENT} EXECUTE PROCEDURE função_trigger;
```

Triggers

- Todos os atributos da tabela que invocou a trigger estão disponíveis dentro da função:
  - OLD: acessa os valores antes da operação realizada (não existe para INSERT)
  - NEW: acessa os valores após a operação realizada (não existe para DELETE)
- Existem uma série de outras variáveis prefixadas em TG_ que dão informações importantes para a função. Exemplos:
  - TG_WHEN: BEFORE ou AFTER
  - TG_OP: INSERT, UPDATE ou DELETE
  - TG_TABLE_NAME: nome da tabela que disparou a trigger

```
CREATE TRIGGER name
    {BEFORE | AFTER} {UPDATE|SELECT|DELETE}
    ON table
    FOR EACH {ROW | STATEMENT} EXECUTE PROCEDURE função_trigger;
```

Triggers

- Função associada a uma trigger:

```
CREATE FUNCTION trg_function()
    RETURNS TRIGGER
    AS $$
    DECLARE
    BEGIN
        -- lógica da trigger;
    END; $$;
```

- Uma trigger BEFORE, a lógica é executada antes da operação ser efetivada (o return tem que ser, em caso de sucesso, obrigatoriamente NEW)
- Já a AFTER, ocorre depois (geralmente, o retorno é NEW)

Exemplo

```
CREATE TABLE product (
        pid integer not null primary key,
        name varchar(30) not null,
        pqty  integer not null);
```

```
CREATE TABLE sale (
        sid integer not null primary key,
        sdate date not null
        address varchar(30) not null);
```

```
CREATE TABLE sale_item (
        sid integer not null,
        pid integer not null,
        sqty integer not null,
        CONSTRAINT pk_sale_item PRIMARY KEY (sid,pid),
        CONSTRAINT fk_sale_item_sale FOREIGN KEY (sid) REFERENCES sale(sid),
        CONSTRAINT fk_sale_item_product FOREIGN KEY (pid) REFERENCES product(pid)
);
```

Exemplo

- Ao ser vendido um item (produto), atualizar a quantidade ou bloquear a venda caso a quantidade vendida seja maior que a quantidade existente.

```
CREATE FUNCTION  check_stock () RETURNS trigger AS $check_stock$
DECLARE
    pqtty product.pqty%type;
BEGIN
      SELECT pqty INTO pqtty FROM product WHERE pid=NEW.pid;
      IF NEW.sqty > pqtty THEN
            RAISE EXCEPTION 'No Stock for product %', NEW.pid;
            RETURN OLD;
      END IF;
      UPDATE product SET pqty=pqty-NEW.sqty WHERE pid=NEW.pid;
      RETURN NEW;
END;
$check_stock$ LANGUAGE plpgsql;
```

Exemplo

- Ao ser vendido um item (produto), atualizar a quantidade ou bloquear a venda caso a quantidade vendida seja maior que a quantidade existente.

```
CREATE TRIGGER  check_stock BEFORE INSERT OR UPDATE ON sale_item
FOR EACH ROW EXECUTE PROCEDURE check_stock();
```