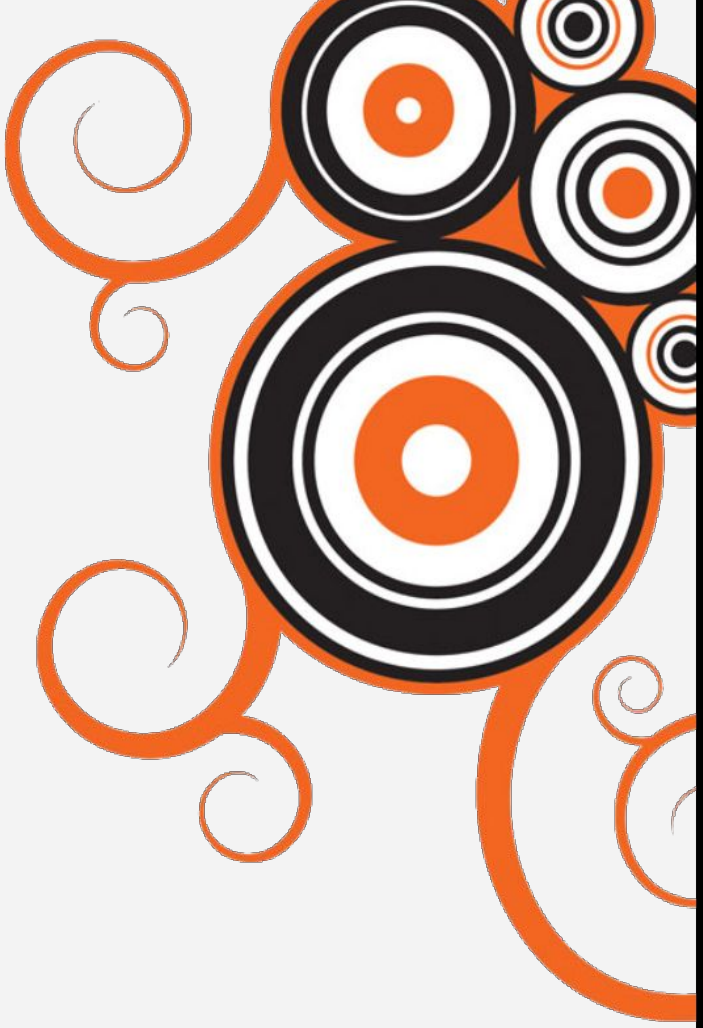
Exercices

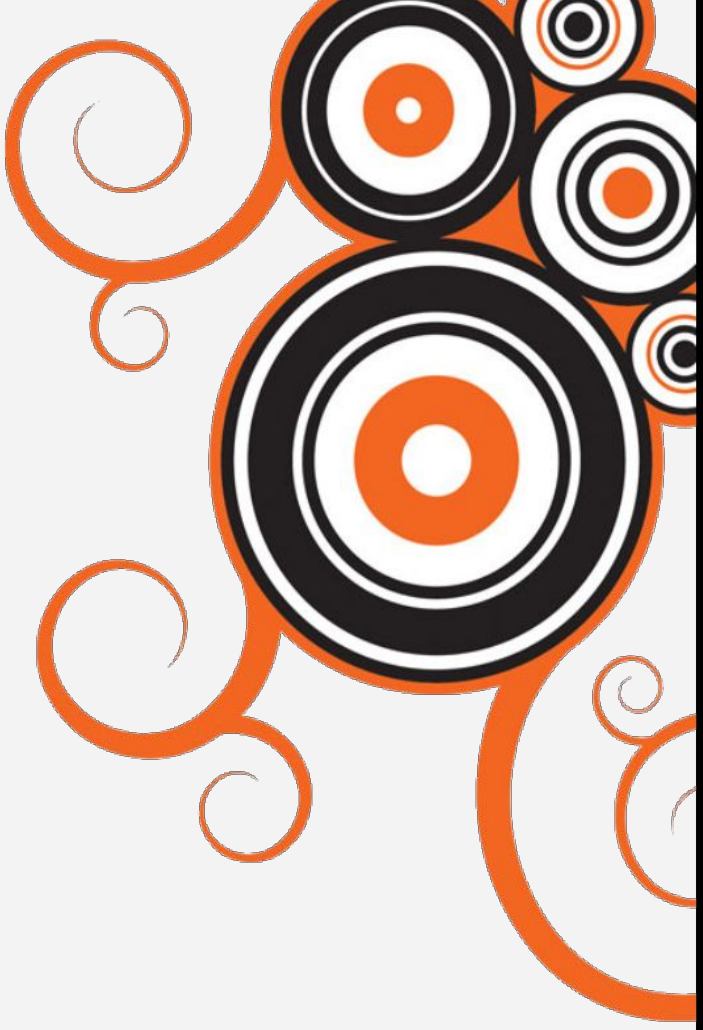
Procédures stockées



Exercices

par Nadine Giasson St-Amand

Fonctions & Procédures



en pgSQL pour PostGreSQL

Versionnement de procédure

Ne pas oublier de sauvegarder à mesure la procédure stockée.

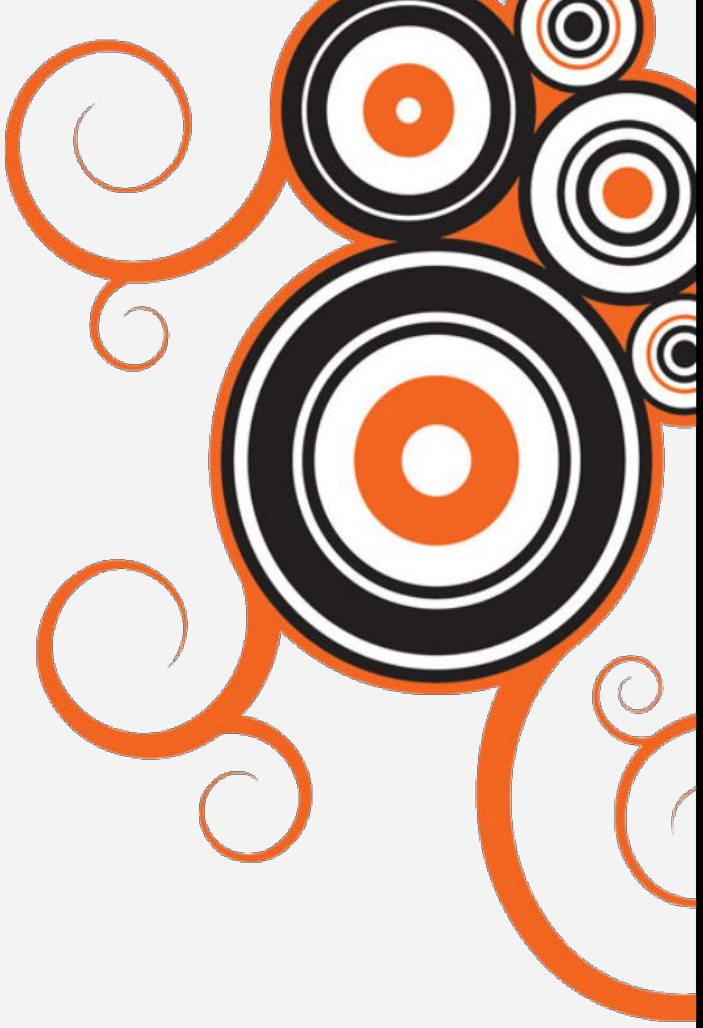
Truc :

- laisser la console ouverte
- pointer la redirection vers le répertoire du workspace

```
C:\Program Files\PostgreSQL\9.6\bin>pg_dump --username=postgres --inserts -C bergerie > c:/Users/stamandnadine/workspace-basededonnees/Bergerie/data/bergerie.sql
```

Plan

Exemples de base



en pgSQL pour PostGreSQL

Exemple sans retour

```
CREATE FUNCTION insererNombre()
RETURNS void
LANGUAGE 'sql'

AS $BODY$
insert into memoire(nombre) values(3)
$BODY$;
```

Test:

```
select insererNombre();
```

Exemple retour

```
CREATE FUNCTION estPlusPetit()
RETURNS boolean
LANGUAGE plpgsql


AS $$
BEGIN
    RETURN 4 < 10;
END;
$$

;
```

Pour tester :

SELECT estPlusPetit(3, 5);

Exemple retour + para

```
CREATE FUNCTION estPlusPetit(a integer, b integer)
RETURNS boolean
LANGUAGE plpgsql


AS $$
BEGIN
    RETURN a < b;
END;
$$

;
```

Pour tester :

SELECT estPlusPetit(3, 5);

Exemple paramètre + sql

```
CREATE FUNCTION ajouterRobot(nom text)
RETURNS void
LANGUAGE 'sql'

AS $$

    insert into robot(nom) values(nom);

$$

;
```

Pour tester :

SELECT ajouterRobot('Allo');

Exemple paramètre + sql

```
CREATE or REPLACE FUNCTION compterArticles()
      RETURNS integer
   LANGUAGE 'sql'

AS $$

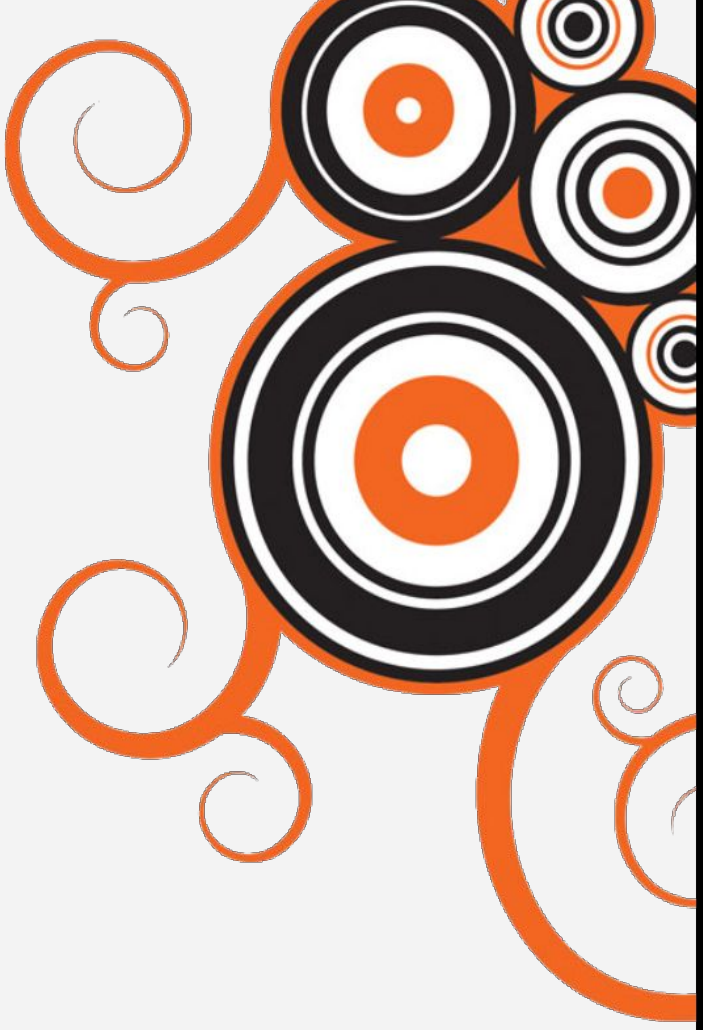
      select count(article) as nombre_article from article

$$
```

Pour tester :

```
select compterArticles();
```

Exemples de programmation



en pgSQL pour PostGreSQL

Exemple Variable

```
CREATE OR REPLACE FUNCTION direBonjour()
RETURNS text
LANGUAGE 'plpgsql' IMMUTABLE

    AS $$
    DECLARE
    message text;
    BEGIN
        message := 'Bonjour ';
        RETURN message;
    END;
    $$
```

Pour tester :

SELECT direBonjour();

Paramètre + concaténation

```
CREATE or REPLACE FUNCTION direBonjourA(prenom text)
RETURNS text
LANGUAGE 'plpgsql'

AS $BODY$
BEGIN

	return 'Bonjour ' || prenom;

END
$BODY$
```

Pour tester :

```
SELECT direBonjour('Nadine');
```

Exemple Paramètre

```
-- drop function direAlloMonde();
CREATE or REPLACE FUNCTION direBonjour() RETURNS text
LANGUAGE 'plpgsql'

AS $$
declare
        nom TEXT;
        message TEXT;
begin
        nom := 'Nadine';
        message := 'Bonjour ' || nom;
        return message;
end
$$
```

Pour tester :

```
SELECT direBonjour();
```

Exemple Paramètre + Var

```
-- drop function direBonjour();
CREATE or REPLACE FUNCTION direBonjour(prenom text)
RETURNS text
LANGUAGE 'plpgsql'

AS $BODY$
DECLARE
    message text;
BEGIN

        message := 'Bonjour ';
        return message || prenom;

END
$BODY$
```

Pour tester :

SELECT direBonjour('Nadine');

Exemple Paramètre + Var

```
-- DROP FUNCTION direAlloMonde()
CREATE or REPLACE FUNCTION direAlloMonde(nom TEXT) RETURNS
TEXT


AS $$
DECLARE
        message TEXT;
BEGIN
        message := 'Coucou ' || nom;
        return message;
END
$$


LANGUAGE 'plpgsql'
```

Pour tester :

SELECT direBonjour('Nadine');

Exemple Paramètre

```
CREATE OR REPLACE FUNCTION direBonjour(nom text)
RETURNS text
LANGUAGE 'plpgsql' IMMUTABLE

    AS $$
    DECLARE
    message text;
    BEGIN
        message := 'Bonjour ';
        message := message || nom || E'\r\n';
        RETURN message;
    END;
    $$
```

Pour tester :

SELECT direBonjour('Nadine');

direBonjour - Wizard

- nom : direBonjour
- type de retour : text
- language : plpgsql
- code

```
DECLARE
message text;
BEGIN
    message := 'Bonjour ';
    message := message || nom || E'\r\n';
    RETURN message;
END;
```

*** générer le script et retirer la ligne Row

repeterMessage sans para

```
DECLARE
    texte text;
    repetitions int;
    message text;
BEGIN
       texte := '';
    repetitions := 3;
    message := 'coucou';

    FOR i IN 1..repetitions LOOP
      texte := texte || ' ' || message;
    END LOOP;
    RETURN texte;
END
```

Pour tester :

```
SELECT repeterMessage();
```

repeterMessage avec para

```
CREATE or REPLACE FUNCTION repeterMessage(message text)
RETURNS text
LANGUAGE 'plpgsql'

AS $BODY$
DECLARE

	texte text;

BEGIN

	texte := '';



	return texte;
END
$BODY$
```

Pour tester :

SELECT repeterMessage();

repeterMessage avec para

```
CREATE or REPLACE FUNCTION repeterMessage(message text)
RETURNS text
LANGUAGE 'plpgsql'

AS $BODY$
DECLARE

	texte text;

BEGIN

	texte := '';
	FOR repetition IN 1..3 LOOP
	   messageRepete := texte || message || ' ';
	END LOOP;
	return texte;
END
$BODY$
```

Pour tester :

SELECT repeterMessage();

Exemple Boucle + retour

```
CREATE or REPLACE FUNCTION decompter() RETURNS TEXT

AS $$
DECLARE
    affichage TEXT;
BEGIN
    affichage := 'Decompte : ';
    for numero in 1..11 LOOP
     affichage := affichage || ' ' || (11-numero);
    END LOOP;
    return affichage;
END
$$

LANGUAGE 'plpgsql'
```

Exemple Boucle + retour

```
CREATE OR REPLACE FUNCTION decompter(maximum integer)
    RETURNS text
     LANGUAGE 'plpgsql'
AS $$
DECLARE
     message text;
BEGIN
    message := '';

    FOR tour IN 1..maximum LOOP
     message := message || (maximum + 1 - tour) || ' ';
    END LOOP;
    return message;
END
$$;
```

Exemple Boucle + SQL

```
CREATE or REPLACE FUNCTION insererSerie()
RETURNS void
LANGUAGE 'plpgsql'

AS $BODY$
    BEGIN
        FOR numero IN 1 .. 5 LOOP
            insert into memoire(nombre) values(numero);
        END LOOP;
    END;
$BODY$;
```

Pour tester :

```
select insererSerie();
```

Exemple Boucle + SQL + Para

```
CREATE or REPLACE FUNCTION insererSerie(premier integer, dernier
integer)
RETURNS void
LANGUAGE 'plpgsql'

AS $BODY$
      BEGIN
            FOR numero IN premier .. dernier LOOP
                  insert into memoire(nombre) values(numero);
            END LOOP;
      END;
$BODY$;
```

Pour tester :

select insererSerie(5,10);

Exemple Boucle + SQL + Para

```
CREATE or REPLACE FUNCTION insererSerie(premier integer, dernier
integer, pas integer)
    RETURNS void
    LANGUAGE 'plpgsql'

AS $BODY$
      BEGIN
            FOR numero IN premier .. dernier BY pas LOOP
                  insert into memoire(nombre) values(numero);
            END LOOP;
      END;
$BODY$;
```

Pour tester :

select insererSerie(5,10);

Exemple Boucle + SQL + Para

```
CREATE OR REPLACE FUNCTION listerDecennies() RETURNS text
   LANGUAGE 'plpgsql'
AS $$
DECLARE
  affichage text;
  depart integer;
  intervalle integer;
BEGIN
      affichage := '';
      depart := 1900;
      intervalle := 200;
      FOR annee IN depart..(depart+intervalle) BY 10 LOOP
            affichage := affichage || annee || ' ';
      END LOOP;
      return affichage;
END
```

Exemple Boucle + SQL

```
CREATE or REPLACE FUNCTION insererSerie()
RETURNS void
LANGUAGE 'plpgsql'
AS $BODY$
    DECLARE
        message TEXT;
    BEGIN
        FOR numero IN 1 .. 5 LOOP
            message := 'Numero ' || numero;
            insert into journal(moment, donnee) values(NOW(), message);
        END LOOP;
    END;
$BODY$;
```

Pour tester :

```
select insererSerie();
```

Autres exemples

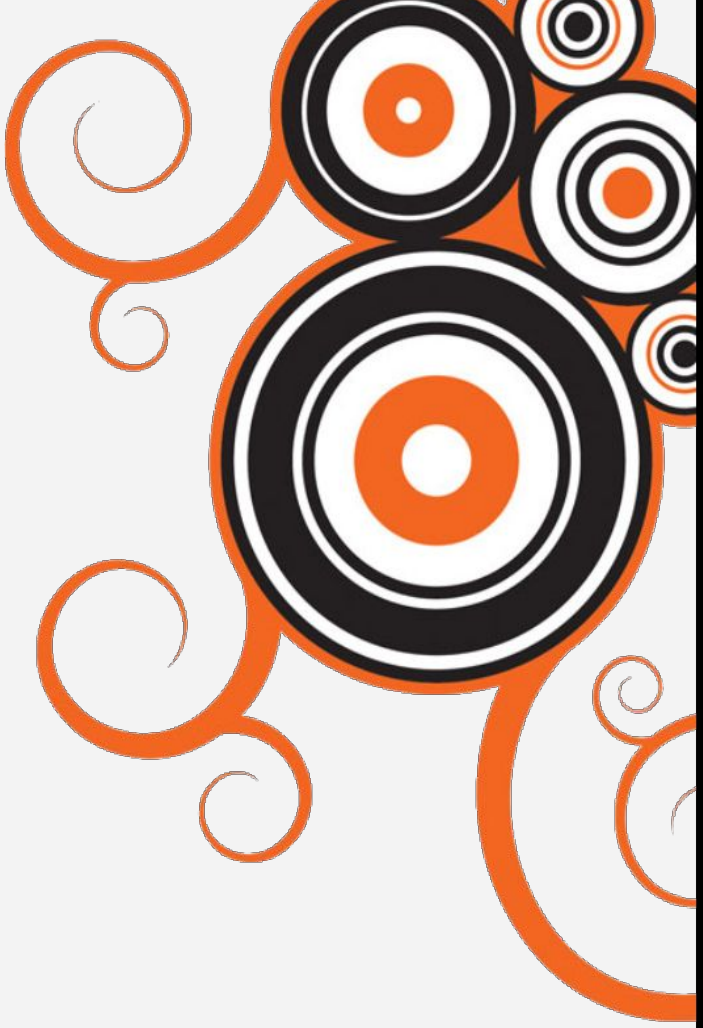
```
CREATE FUNCTION estPlusPetit(nombre1 integer, nombre2 integer)
        RETURNS boolean
    LANGUAGE 'plpgsql'

AS $$
BEGIN
        return nombre1 < nombre2;
END
$$

CREATE FUNCTION direBonjour(prenom text)
        RETURNS text
    LANGUAGE 'plpgsql'

AS $$
DECLARE
```

Démo du journal



en pgSQL pour PostGreSQL

Exemple Boucle + SQL

```
CREATE or REPLACE FUNCTION testerMoment()
RETURNS void
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
        description TEXT;
BEGIN
    description := 'test';
        insert into journal(moment, description) values(NOW(), description);
END;
$BODY$;
```

TEST
select insererMoment();
et regarder dans la table journal

Exemple avec trigger + sql

```
CREATE or REPLACE FUNCTION journaliserAjout()
RETURNS trigger
LANGUAGE 'plpgsql'

AS $$
DECLARE
      nouveauMouton text;
BEGIN

      nouveauMouton := NEW.nom || ' ' || NEW.couleur;
      INSERT into journal(moment, operation, description, objet)
VALUES(NOW(), 'AJOUT', nouveauMouton, 'mouton');

   return NEW;
END
$$
```

Exemple avec trigger + sql

```
CREATE TRIGGER evenementAjoutMouton
    BEFORE INSERT ON mouton
        FOR EACH ROW EXECUTE PROCEDURE
            journaliserAjout();
```

POUR TESTER

```
INSERT into mouton(nom, couleur)
VALUES('anonyme','roux');
```

Exemple avec trigger + sql

```
CREATE or REPLACE FUNCTION journaliserEdition()
RETURNS trigger
LANGUAGE 'plpgsql' AS $$
DECLARE
        nouveauMouton text;
        ancienMouton text;
        description text;
BEGIN
        nouveauMouton := NEW.nom || ' ' || NEW.couleur;
        ancienMouton := OLD.nom || ' ' || OLD.couleur;
        description := ancienMouton || ' => ' || nouveauMouton;
        INSERT into journal(moment, operation, description, objet)
VALUES(NOW(), 'EDITION', description, 'mouton');
    return NEW;
END
$$;
```

Exemple avec trigger + sql

```
CREATE TRIGGER evenementEditionMouton
    BEFORE UPDATE ON mouton
        FOR EACH ROW EXECUTE PROCEDURE
            journaliserEdition();

UPDATE mouton SET couleur = 'brun-roux' WHERE couleur = 'roux';
```

Point de départ projet

```
CREATE or REPLACE FUNCTION journaliserAjout()
      RETURNS TRIGGER
   LANGUAGE 'plpgsql'

AS $$
DECLARE



BEGIN

      return NEW;

END
$$
```

Point de départ projet

```
CREATE or REPLACE FUNCTION journaliserAjoutRobot()
      RETURNS TRIGGER
   LANGUAGE 'plpgsql'
AS $$
DECLARE

BEGIN

      INSERT INTO journal(moment, operation, objet, description) VALUES
(NOW(), 'ajout', 'robot', 'test');

      return NEW;

END
$$
```

TEST
insert into robot(nom) values('test trigger');
et regarder dans la table journal

Point de départ projet

```
CREATE or REPLACE FUNCTION journaliserAjoutRobot()
      RETURNS TRIGGER
   LANGUAGE 'plpgsql'
AS $$
DECLARE
      description text;
BEGIN
   description := 'var';

      INSERT INTO journal(moment, operation, objet, description) VALUES
(NOW(), 'ajout', 'robot', description);

      return NEW;
END
$$
```

TRIGGER:

```
CREATE TRIGGER evenementAjoutRobot
      BEFORE INSERT ON robot
      FOR EACH ROW EXECUTE PROCEDURE
            journaliserAjoutRobot();
```

TEST

insert into robot(nom) values('test trigger');

et regarder dans la table journal

Point de départ projet

```
CREATE or REPLACE FUNCTION journaliserAjoutRobot()
    RETURNS TRIGGER
  LANGUAGE 'plpgsql'

AS $$
DECLARE
    description text;
BEGIN
  description := NEW.nom;
    INSERT INTO journal(moment, operation, objet, description) VALUES
(NOW(), 'ajout', 'robot', description);
    return NEW;

END
$$
```

TRIGGER:

```
CREATE TRIGGER evenementAjoutRobot
      BEFORE INSERT ON robot
      FOR EACH ROW EXECUTE PROCEDURE
            journaliserAjoutRobot();
```

TEST

```
insert into robot(nom) values('test trigger');
```

et regarder dans la table journal

Point de départ projet

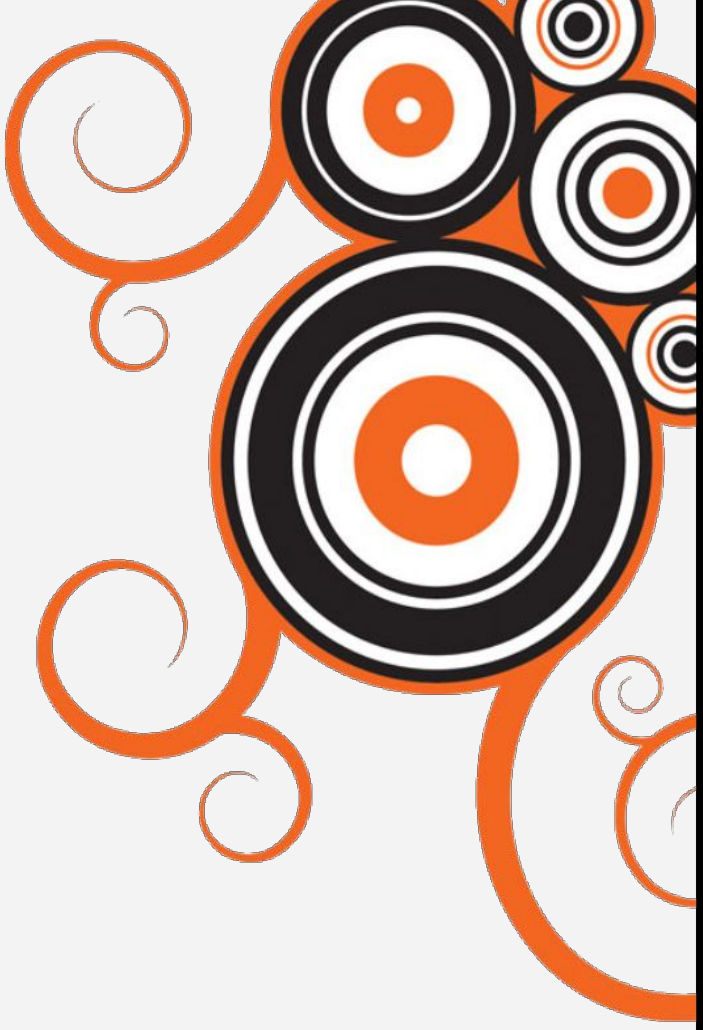
```
CREATE or REPLACE FUNCTION journaliserEditerRobot() RETURNS
TRIGGER LANGUAGE 'plpgsql'
AS $$
DECLARE
   descriptionAncien text;
   descriptionNouveau text;
   description text;
BEGIN
    descriptionAncien := OLD.nom || ' (' || OLD.couleur  || ',' || OLD.poids || ')';
    descriptionNouveau := NEW.nom || ' (' || NEW.couleur  || ',' || NEW.poids || ')';
    description := descriptionAncien || ' => ' || descriptionNouveau;
    INSERT INTO journal(moment, operation, objet, description) VALUES
(NOW(), 'edition', 'robot', description);
    return NEW;
END
$$
```

TEST
update robot set nom = 'toto', couleur = 'blanc', poids = 10 WHERE id = 7;

et regarder dans la table journal

```
CREATE TRIGGER evenementEditionMouton
   BEFORE UPDATE ON robot
      FOR EACH ROW EXECUTE PROCEDURE
         journaliserEditerRobot();
```

Exemples en format SQL



en pgSQL pour PostGreSQL

Exemple SQL

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliser(description text)
      RETURNS void
   LANGUAGE 'plpgsql'

AS $BODY$
BEGIN

INSERT into journal(message) VALUES(description);

END
$BODY$
```

POUR TESTER
select journaliser();

Exemple SQL

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliser(description text)
      RETURNS void
   LANGUAGE 'plpgsql'

AS $BODY$
BEGIN

INSERT into journal(moment, message) VALUES(NOW(), description);

END
$BODY$
```

POUR TESTER
select journaliser();

Exemple SQL

Code de la fonction :

```
CREATE FUNCTION journaliser()
      RETURNS void
   LANGUAGE 'plpgsql'

AS $BODY$
BEGIN

      INSERT into journal(operation) VALUES('test');

END
$BODY$
```

POUR TESTER
select journaliser();

Exemple SQL + timestamp

Table journal:

changer type 'moment' pour un timestamp

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliser()
      RETURNS void
   LANGUAGE 'plpgsql'

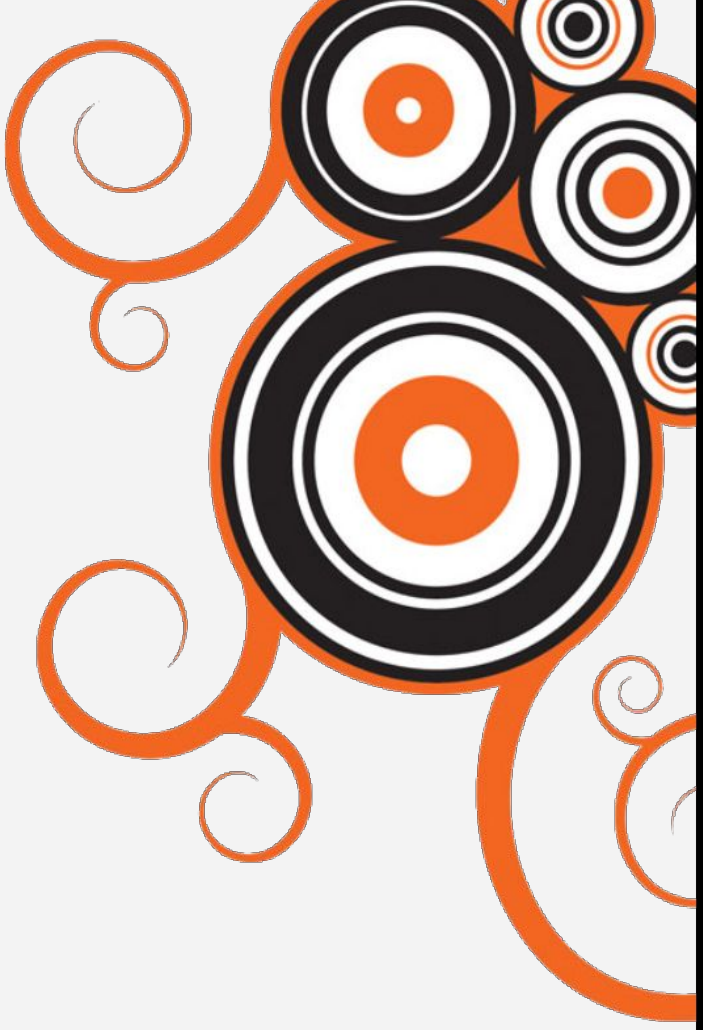
AS $BODY$
BEGIN

      INSERT into journal(moment, operation) VALUES(NOW(), 'test');

END
$BODY$
```

POUR TESTER
select journaliser();

Déclencheurs (trigger)



en pgSQL pour PostGreSQL

Déclencher après du SQL

```
CREATE TRIGGER evenementEtampeEmploye
    BEFORE INSERT OR UPDATE ON employe
        FOR EACH ROW EXECUTE PROCEDURE
            etamperEmploye();
```

https://www.postgresql.org/docs/9.6/static/plpgsql-trigger.html#PLPGSQL-DML-TRIGGER

Déclencher après du SQL

```
CREATE FUNCTION etamperEmploye()
    RETURNS trigger
AS $$
   BEGIN
      IF NEW.salaire < 0 THEN
      END IF;

      NEW.last_date := current_timestamp;
      NEW.last_user := current_user;
      RETURN NEW;
   END;
$$
LANGUAGE plpgsql;
```

Déclencher après du SQL

```
drop function journaliser();
CREATE or REPLACE FUNCTION journaliser()
RETURNS trigger
LANGUAGE 'plpgsql'


AS $$
BEGIN
    INSERT into journal(moment, operation, objet, description) VALUES(NOW(), 'AJOUTER', 'mouton', '{Dolly, 2016-06-01}');
END
$$
```

Valider des paramètress

```
CREATE FUNCTION etamperEmploye()
RETURNS trigger

AS $$
BEGIN
        IF NEW.nom IS NULL THEN
            RAISE EXCEPTION 'nom cannot be null';
        END IF;
        IF NEW.salaire IS NULL THEN
            RAISE EXCEPTION '% salaire 0', NEW.nom;
        END IF;
    END;
$$
LANGUAGE plpgsql;
```

Squelette avec trigger

```
CREATE or REPLACE FUNCTION journaliserMouton()
      RETURNS trigger
      LANGUAGE 'plpgsql'


AS $$
      DECLARE

      BEGIN
            return NEW;
      END
$$
```

POUR TESTER
insert into mouton(nom) values('jolie');

```
CREATE TRIGGER suiviMouton
BEFORE INSERT OR UPDATE ON mouton
            FOR EACH ROW EXECUTE
PROCEDURE
                  journaliserMouton();
```

Exemple prédéfini

```
CREATE or REPLACE FUNCTION journaliserMouton()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $suiviMouton$
    DECLARE
    BEGIN
        INSERT into journal(operation, objet, donnee, moment)
    VALUES(TG_OP, TG_TABLE_NAME, OLD.nom, NOW());
        return NEW;
    END
$suiviMouton$
```

POUR TESTER
insert into mouton(nom) values('jolie');

```
CREATE TRIGGER suiviMouton
BEFORE INSERT OR UPDATE OR DELETE ON mouton
        FOR EACH ROW EXECUTE PROCEDURE
            journaliserMouton();
```

Squelette avec trigger

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliserMouton()
    RETURNS trigger
    LANGUAGE 'plpgsql'
AS $suiviMouton$
    DECLARE
    BEGIN
        INSERT into journal(operation, objet, donnee, moment)
    VALUES(TG_OP, TG_TABLE_NAME, OLD.nom, NOW());
        return NEW;
    END
$suiviMouton$
```

POUR TESTER
insert into mouton(nom) values('jolie');

```
TRIGGER (drop cascade trigger suiviMouton)
CREATE TRIGGER suiviMouton
AFTER INSERT OR UPDATE ON mouton
        FOR EACH ROW EXECUTE PROCEDURE
            journaliserMouton();
```

Table mouton

```
CREATE TABLE public.mouton
(
    nom text COLLATE pg_catalog."default",
    naissance date,
    sexe character(1) COLLATE pg_catalog."default",
    couleur character(1) COLLATE pg_catalog."default",
    sousespece text COLLATE pg_catalog."default",
    description text COLLATE pg_catalog."default",
    id_troupeau integer,
    id_mouton integer NOT NULL DEFAULT nextval('mouton_id_mouton_seq'::regclass)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

Table journal

```
CREATE TABLE public.journal
(
    id_journal bigint NOT NULL DEFAULT nextval('journal_id_journal_seq'::regclass),
    operation text COLLATE pg_catalog."default",
    objet text COLLATE pg_catalog."default",
    donnee text COLLATE pg_catalog."default",
    moment timestamp without time zone,
    CONSTRAINT journal_pkey PRIMARY KEY (id_journal)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

Trigger et parametre

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliserMouton()
    RETURNS trigger AS $suiviMouton$
BEGIN
insert into journal(message) values(NEW.nom);

return ;
END;
$suiviMouton$

LANGUAGE plpgsql;
```

POUR TESTER
insert into mouton(nom) values('jolie');

```
CREATE TRIGGER suiviMouton
BEFORE INSERT OR UPDATE ON mouton
        FOR EACH ROW EXECUTE PROCEDURE
            journaliserMouton();
```

Trigger et parametre

Code de la fonction :

```
CREATE or REPLACE FUNCTION journaliserMouton()
    RETURNS trigger AS $suiviMouton$
DECLARE
        message TEXT;
BEGIN
message := NEW.nom || " " || NEW.description;
insert into journal(message) values(message);
return OLD;
END;
$suiviMouton$ LANGUAGE plpgsql;
```

POUR TESTER

```
insert into mouton(nom, description)
values('jolie', 'blabla');
```

TRIGGER:

```
CREATE TRIGGER suiviMouton
BEFORE INSERT OR UPDATE ON mouton
            FOR EACH ROW EXECUTE PROCEDURE
                journaliserMouton();
```

Exemple sans retour

Code de la fonction :

```
CREATE FUNCTION insererNombre()
    RETURNS void
    LANGUAGE 'sql'

  AS $BODY$
  insert into journal(numero) values(3)
  $BODY$;
```

POUR TESTER

```
select insererNombre();
```

Exemple sur une liste

Code de la fonction :

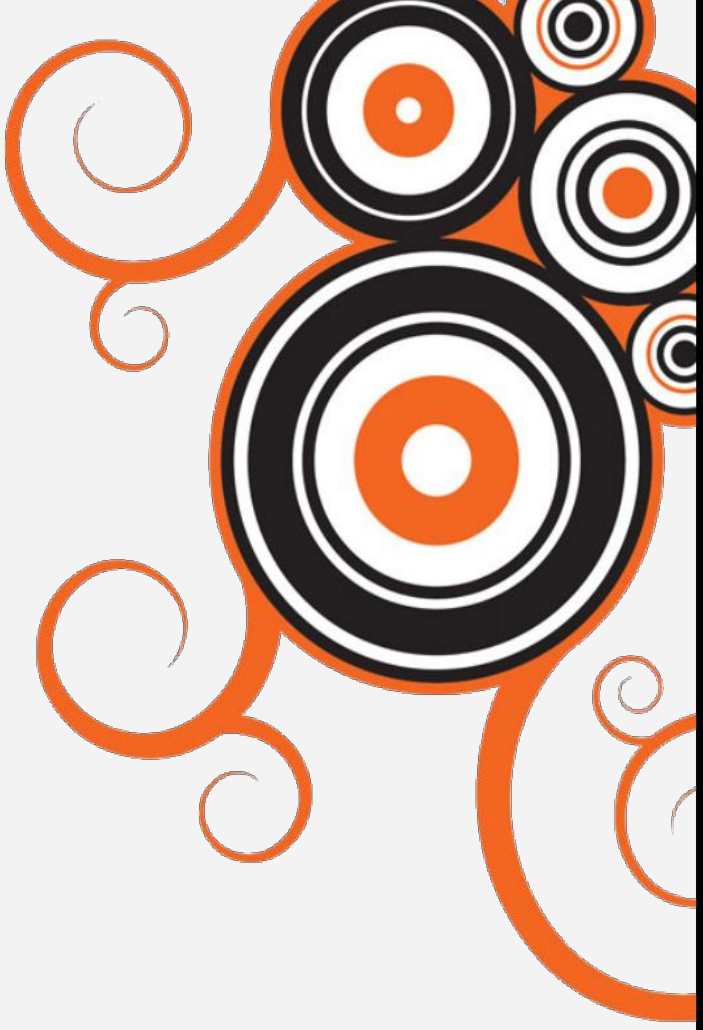
```
CREATE or REPLACE FUNCTION copierChamps()
  RETURNS void
  LANGUAGE 'plpgsql'

  AS $$
  DECLARE
    moutonCourant RECORD;
  BEGIN
     FOR moutonCourant IN
         SELECT * from mouton
     LOOP
         UPDATE mouton set poids = 2 WHERE id = moutonCourant.id ;
     END LOOP
  END
  $$
  ;
```

Exemple sur une liste

```
CREATE or REPLACE FUNCTION copierChamps()
RETURNS void
LANGUAGE 'plpgsql'
AS $$
DECLARE
  moutonCourant RECORD;
BEGIN
    FOR moutonCourant IN
          SELECT mouton.* from mouton
    LOOP
        -- https://www.postgresql.org/docs/8.2/static/functions-formatting.html
        -- requete testee : UPDATE mouton set poids =
  cast(mouton.ancienpoids as double precision) WHERE id = 7;
          UPDATE mouton set poids = cast(mouton.ancienpoids as double
  precision) WHERE id = moutonCourant.id ;
    END LOOP;
```

Fetch de données



possibilités en pgSQL

Selection dans une variable

```
drop function chercherIdentifiant()
CREATE or REPLACE FUNCTION chercherIdentifiant()
        RETURNS integer
    LANGUAGE 'sql'

AS $$

        select id from activite where titre = 'Triathlon';

$$
```

Selection dans une variable

```
drop function chercherIdentifiant();
CREATE or REPLACE FUNCTION chercherIdentifiant()
        RETURNS integer
    LANGUAGE 'plpgsql'

AS $$
declare
identifiant integer;
begin
        select id into identifiant from activite where titre = 'Triathlon';
        return identifiant;
end
$$
```

Selection dans une variable

```
-- DROP function compterLesRobots();
CREATE or REPLACE FUNCTION compterLesRobots()
        RETURNS integer
    LANGUAGE 'plpgsql'
AS $$
DECLARE
        nombreRobots integer;
BEGIN
        SELECT count(id) into nombreRobots FROM robot;
        return nombreRobots;
END
$$
```

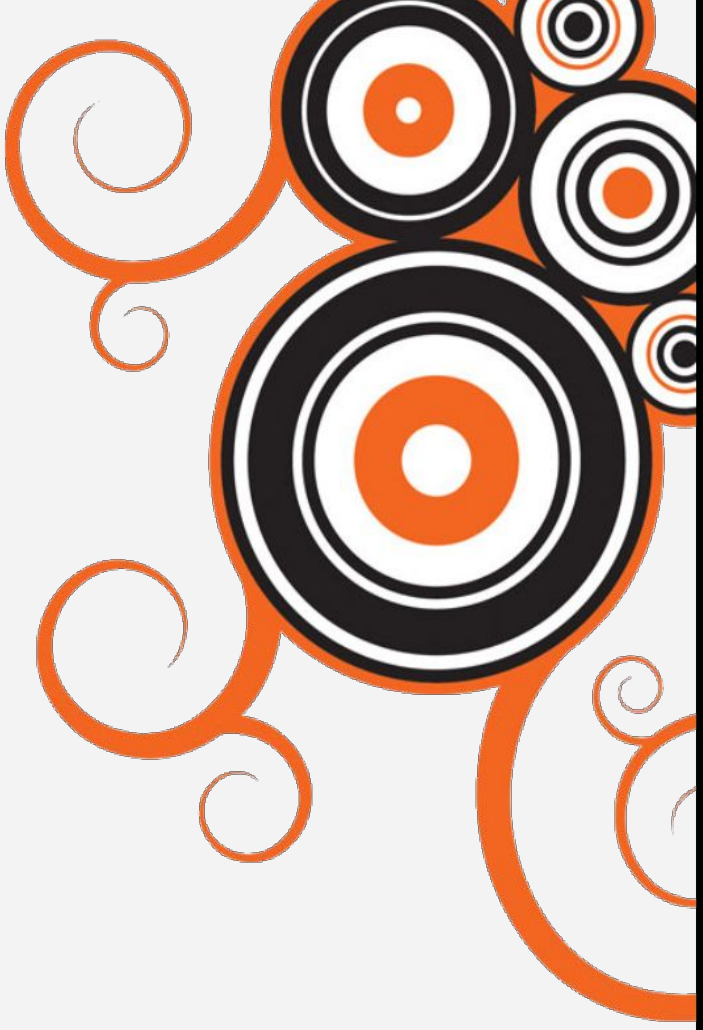
Curseur sur un résultat

```
CREATE or REPLACE FUNCTION retournerPoidsDuRobot(numero integer)
        RETURNS numeric
    LANGUAGE 'plpgsql'
AS $$
DECLARE
        curseurRobot SCROLL CURSOR FOR SELECT * FROM robot WHERE id = numero;
        robot robot%ROWTYPE;
BEGIN
        OPEN curseurRobot;
        FETCH curseurRobot INTO robot;
        return robot.poids;
        CLOSE curseurRobot;
END
$$
```

Curseur sur un résultat

```
CREATE or REPLACE FUNCTION calculerStatistiquesRobots()
        RETURNS numeric
    LANGUAGE 'plpgsql'
AS $$
DECLARE
        curseurRobot SCROLL CURSOR FOR SELECT count(*) as nombre
FROM robot;
        statistique RECORD;
BEGIN
        OPEN curseurRobot;
        FETCH curseurRobot INTO statistique;
        return statistique.nombre;
        CLOSE curseurRobot;
END
$$
```

Parler aux procédures stockées



en Procedural Language (PL)

Exemple PL PG SQL

```
CREATE OR REPLACE FUNCTION repeterMessage(repetitions integer,
message text)
RETURNS text AS
	$$
	DECLARE
	texte text;
	BEGIN
	texte := '';
	IF repetitions > 0 THEN
		FOR i IN 1 .. repetitions LOOP
			texte := texte || message || E'\r\n';
		END LOOP;
	END IF;
	RETURN texte;
	END;
	$$
LANGUAGE 'plpgsql' IMMUTABLE
SECURITY DEFINER
COST 10;
```

Pour tester :

```
SELECT repeterMessage(3, 'Coucou');
```

Exemple Variable

```
CREATE OR REPLACE FUNCTION direBonjour()
RETURNS text AS
	$$
	DECLARE
	message text;
	BEGIN
		message := 'Bonjour ';
		RETURN message;
	END;
	$$
LANGUAGE 'plpgsql' IMMUTABLE
SECURITY DEFINER
COST 10;
```

Pour tester :

SELECT direBonjour();