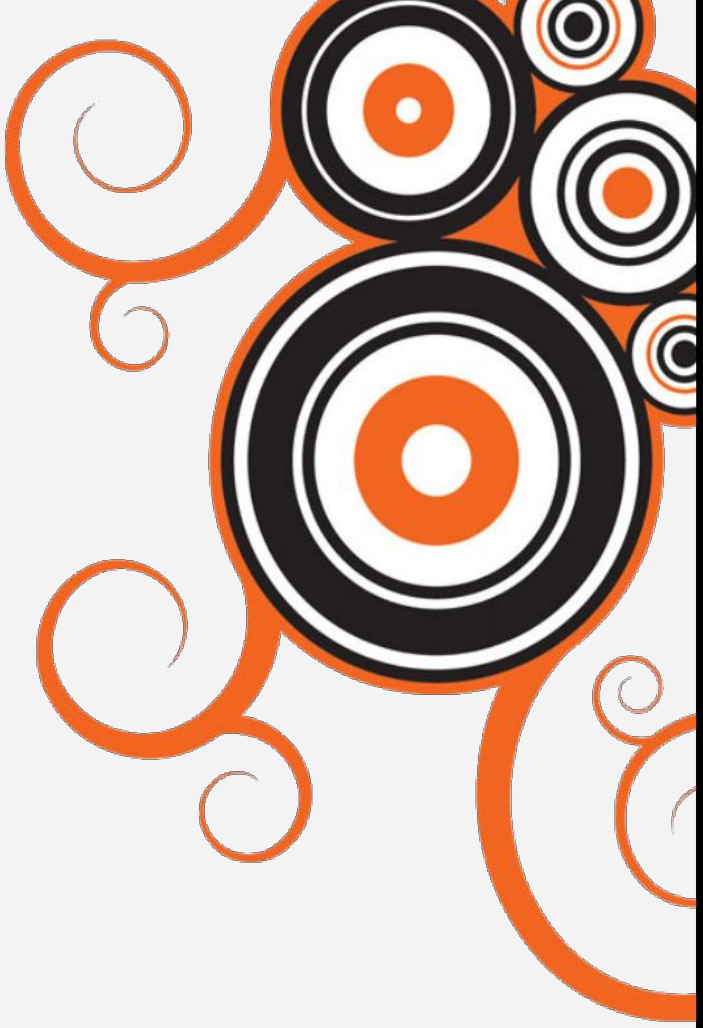
Langage pgSQL

Procédures stockées



par Nadine Giasson St-Amand

Attention

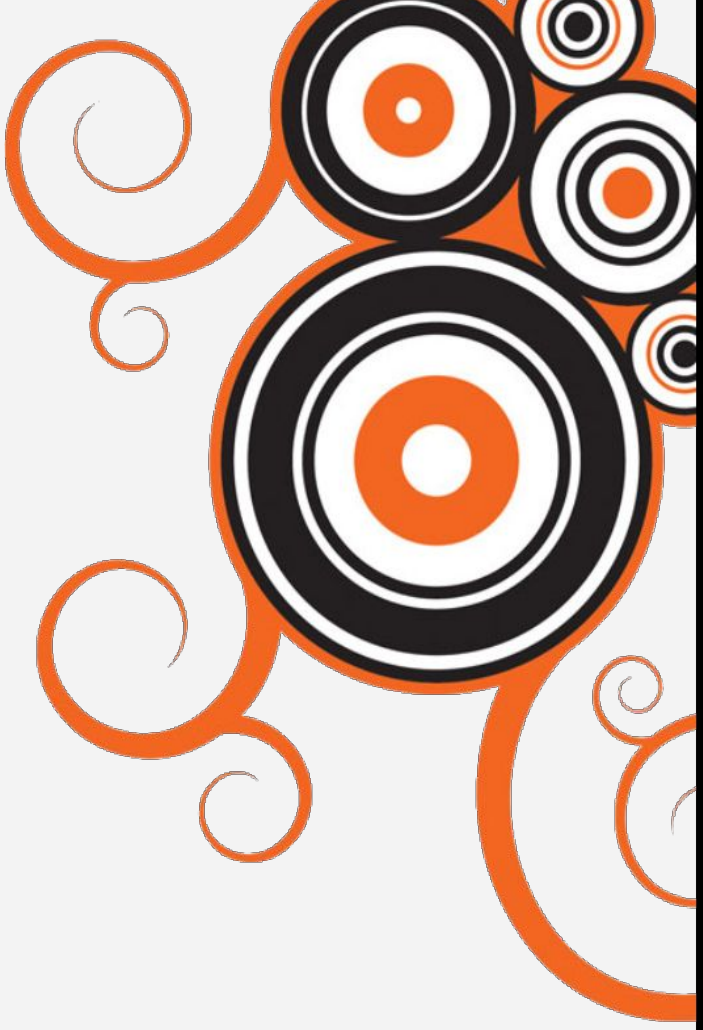
Ne pas utiliser de procédures stockées de manière systématique. Les DAO et les requêtes préparées sont là, dans l'application, pour prendre en charge la couche d'accès aux données avec les opérations CRUD et en assurant ACID.

- **CRUD = Create - Read - Update - Delete**
- **ACID = Atomicity - Consistency - Isolation - Durability**

Les procédures stockées ne devraient pas être utilisées pour gérer la logique applicative régulière mais seulement pour l'intégrité et des méta-opérations liées à la gestion de la base en elle-même. Par exemple :

- Intégrité complexe
  - **l'intégrité relationnelle complexe (avec des nombre d'éléments en one-to-many, etc) qui ne peut être exprimée en SQL**
  - **le caching (gestion de dénormalisations contrôlées)**
- Gestion de la base
  - **la journalisation**
  - **le backup**
  - **la surveillance : sécurité, données**

Squelettes pgSQL



en pgSQL pour PostGreSQL

Squelette sql

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

AS $$

      CODE SQL ICI

$$
```

Test:

```
select journaliser();
```

Squelette plpgsql

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

AS $$
BEGIN

END
$$
```

Test:

```
select journaliser();
```

plpgsql avec variables

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

AS $$
DECLARE

BEGIN

END
$$
```

Test:

```
select journaliser();
```

plpgsql avec retour

```
CREATE or REPLACE FUNCTION comparer()
      RETURNS boolean
   LANGUAGE 'plpgsql'

AS $$
DECLARE
      RETURN (une valeur booleenne);
BEGIN

END
$$
```

Test:

select journaliser();

Squelette fn avec trigger

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

AS $$
BEGIN
    return NEW;
END
$$
;
```

On return NEW si le trigger était un INSERT ou un UPDATE,

mais pour un DELETE on return OLD.

Test:

```
select journaliser();
```

Squelette du trigger

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

Test:

```
select journaliser();
```

Synthese (ressemble Ada)

Opérateur :

:= à la place de =

|| à la place de + ou . pour la concaténation

declaration : nomVariable TYPE;

Guillemets simples 'blabla' au lieu de double comme "blabla"

-- pour les commentaires

Types fréquents :

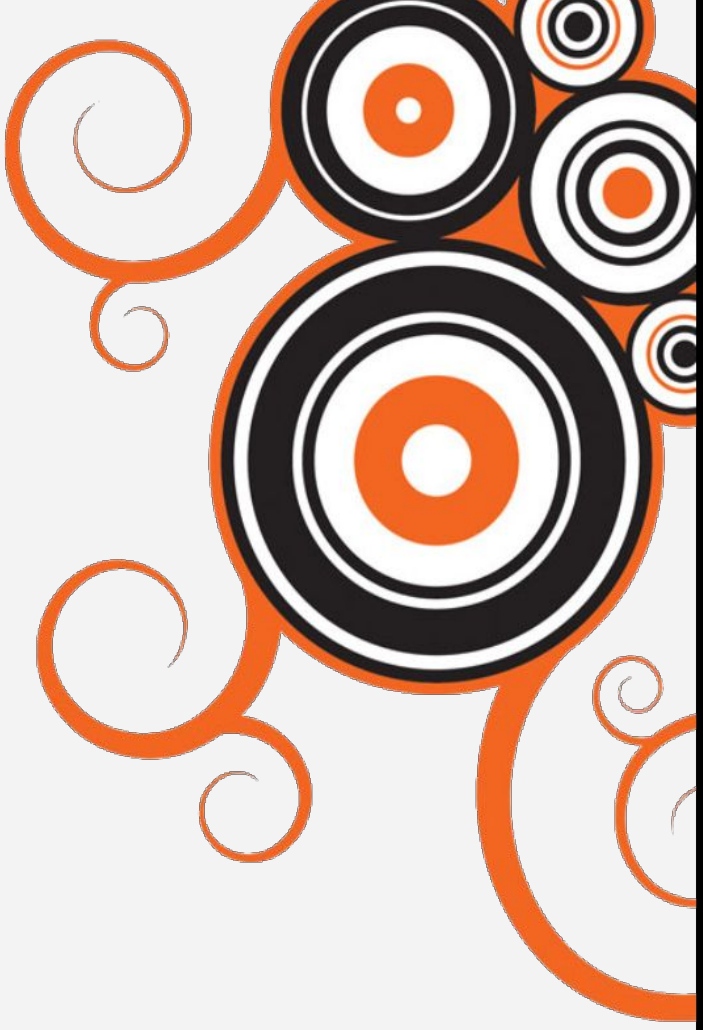
drapeau boolean

message text

nombre integer

Variables prédéfinies : NEW OLD PG_OP

Langage pgSQL



en pgSQL pour PostGreSQL

Conditionnelles

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

Boucles

- **FOR somevariable IN (1 ...someendnumber) LOOP .. END LOOP;**
- **FOR somevariable IN REVERSE someendnumber .. 1 BY somestep LOOP .. END LOOP;**
- **FOR somevariable IN EXECUTE(du_code_sql) LOOP ..RETURN NEXT; .. END LOOP;**
- **LOOP ..logic statements EXIT .. EXIT WHEN .. CONTINUE WHEN .. END LOOP;**
- **WHILE ... LOOP ... END LOOP;**
- **EXCEPTION WHEN .... WHEN ..**

Références

http://www.postgresonline.com/journal/index.php?/archives/58-Quick-Guide-to-writing-PLPGSQL-Functions-Part-1.html

http://www.postgresonline.com/journal/index.php?/archives/76-Quick-Guide-to-writing-PLPGSQL-Functions-Part-2.html

https://www.postgresql.org/docs/9.1/static/sql-createfunction.html

https://www.postgresql.org/docs/9.6/static/plpgsql.html

https://en.wikipedia.org/wiki/PL/pgSQL

SQL généré par une procédure stoquée

NEW

Data type RECORD; variable holding the new database row for INSERT/UPDATE operations in row-level triggers. This variable is NULL in statement-level triggers and for DELETEoperations.

OLD

Data type RECORD; variable holding the old database row for UPDATE/DELETE operations in row-level triggers. This variable is NULL in statement-level triggers and for INSERToperations.

TG_NAME

Data type name; variable that contains the name of the trigger actually fired.

TG_WHEN

Data type text; a string of either BEFORE or AFTER depending on the trigger's definition.

TG_LEVEL

Data type text; a string of either ROW or STATEMENT depending on the trigger's definition.

TG_OP

Data type text; a string of INSERT, UPDATE, DELETE, or TRUNCATE telling for which operation the trigger was fired.

SQL généré par une procédure stoquée

TG_RELID

Data type oid; the object ID of the table that caused the trigger invocation.

TG_RELNAME

Data type name; the name of the table that caused the trigger invocation. This is now deprecated, and could disappear in a future release. Use TG_TABLE_NAME instead.

TG_TABLE_NAME

Data type name; the name of the table that caused the trigger invocation.

TG_TABLE_SCHEMA

Data type name; the name of the schema of the table that caused the trigger invocation.

TG_NARGS

Data type integer; the number of arguments given to the trigger procedure in the CREATE TRIGGER statement.

TG_ARGV[]

Data type array of text; the arguments from the CREATE TRIGGER statement. The index counts from 0. Invalid indexes (less than 0 or greater than or equal to tg_nargs) result in a null value.

Parler au SQL en PL

- Pour exécuter du SQL sans résultat (caché)
  - **PERFORM le_code_sql;**
  - https://www.postgresql.org/docs/9.6/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-SQL-NORESULT
- Pour exécuter du SQL avec une ligne de résultat
  - **sql_directement**
  - https://www.postgresql.org/docs/9.6/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-SQL-ONEROW
- Pour exécuter du SQL dynamique (non-caché)
  - **EXECUTE le_code_sql**
  - **FOR somevariable IN EXECUTE(du_code_sql) LOOP ..RETURN NEXT; .. END LOOP;**
  - https://www.postgresql.org/docs/9.6/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-EXECUTING-DYN
- Obtenir des informations sur une commande SQL
  - **GET & GET DIAGNOSTICS**
  - https://www.postgresql.org/docs/9.6/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-DIAGNOSTICS

SQL généré par une procédure stoquée

```
EXECUTE format('UPDATE mouton SET nom = %S '
   'WHERE id_mouton = %L', mouton.getNom(),
mouton.getId());


EXECUTE format('UPDATE matable SET %I = %L '
   'WHERE id_matable = %L', nomcolonne, valeur, cle);
```

- Pour exécuter du SQL dynamique (non-caché)
  - **EXECUTE le_code_sql**
  - https://www.postgresql.org/docs/9.6/static/plpgsql-statements.html#PLPGSQL-STATEMENTS-EXECUTING-DYN