

Compte rendu TP : Initiation au langage PL/pgSQL
---

**Sommaire :**

1) Contexte .....	2
2) Exercices .....	2
3) Conclusion .....	6

## 1) Contexte

Ce TD consistait à nous initier au langage PL/pgSQL sur dbForge Studio for PostgreSQL.

## 2) Exercices

### Exercice 1 :

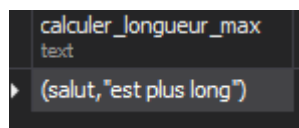
Écrire la fonction `calculer_longueur_max()` qui calcule la longueur de deux chaînes de caractères fournies en argument et qui retourne la longueur la plus longue.

- Arguments en entrée : 2 chaînes de caractères.
- Argument en sortie : longueur maximale des deux chaînes de caractères

```
CREATE OR REPLACE FUNCTION calculer_longueur_max(chaine1 VARCHAR, chaine2 VARCHAR)
RETURNS TEXT
LANGUAGE plpgsql
AS $plpgsql$
DECLARE a1 INT; a2 INT; x TEXT;
BEGIN
    a1=CHAR_LENGTH(chaine1);
    a2=CHAR_LENGTH(chaine2);
    IF a1>a2 THEN
        x = (chaine1,'est plus long');
    ELSIF a1<a2 THEN
        x = (chaine2,'est plus long');
    ELSE
        x = 'Ils sont de la même longueur';
    END IF;
    RETURN x;
END;
$plpgsql$;

SELECT calculer_longueur_max('salut', 'test');
```

On commence par déclarer les différentes chaînes de valeurs. Ensuite si la première chaîne est plus grande que la deuxième chaîne de valeurs alors on renvoie que la chaîne 1 est plus longue. Sinon on renvoie que la chaîne 2 est plus longue. Enfin on renvoie les deux chaînes de valeurs que l'on veut calculer.



calculer_longueur_max
(salut,"est plus long")

En exécutant la fonction, cela nous renvoie que salut est plus long.

## Exercice 2 :

Écrire la fonction nb\_occurrences() qui compte et retourne le nombre d'occurrences d'un caractère dans un intervalle d'une chaîne de caractères.

L'intervalle est indiqué par une position de départ et de fin dans la chaîne.

- Arguments en entrée :
  - paramètre 1 : un caractère ;
  - paramètre 2 : chaîne de caractères ;
  - paramètre 3 : indice de début de l'intervalle ;
  - paramètre 4 : indice de la fin de l'intervalle.
- Argument en sortie : nombre d'occurrences du caractère dans l'intervalle.

```
CREATE OR REPLACE FUNCTION nb_occurrence(carac TEXT, chaine TEXT, i_debut INT, i_fin INT)
RETURNS INT
LANGUAGE plpgsql
AS $plpgsql$
DECLARE a1 text;
        nb0 INT;
        chaine1 text;
BEGIN
    chaine1 = chaine;
    nb0 = 0;
    FOR i IN i_debut..i_fin LOOP
        a1=SUBSTR(chaine1, i, i_fin);
        IF a1=carac THEN
            nb0 = nb0+1;
        END IF;
    END LOOP;
    RETURN nb0;
END;
$plpgsql$;

SELECT nb_occurrences ('a', 'allo', '1', '4');
```

On commence par déclarer les différentes chaîne de valeurs. Ensuite si la première chaîne est plus grande que la deuxième chaîne de valeurs alors on renvoie que la chaîne 1 est plus longue. Sinon on renvoie que la chaîne 2 est plus longue. Enfin on renvoie les deux chaînes de valeurs que l'on veut calculer.

nb_occurrences
varchar
1

En exécutant la fonction, cela nous renvoie que salut est plus long.

### Exercice 3 :

Écrire la fonction getNbJoursParMois() qui calcule le nombre de jours dans un mois pour une date fournie en paramètre.

- Argument en entrée : une date (sous format SQL).
- Argument en sortie : nombre de jours du mois de la date

```
CREATE OR REPLACE FUNCTION getNbJourParMois(myDate DATE) RETURNS DATE AS
$$
    SELECT (date_trunc('MONTH', $1) + INTERVAL '1 MONTH - 1 day')::DATE;
$$
LANGUAGE 'sql';

SELECT DATE_PART('day', getNbJourParMois('2020-11-05'));
```

On commence par écrire la requête qui permet de calculer le nombre de jours dans un mois pour une date fournie en paramètre « SELECT (date\_trunc('MONTH', \$1) + INTERVAL '1 MONTH - 1 day')::DATE; ». Ensuite on entre la date choisi.

date_part	
double	
	30

En exécutant la fonction, cela nous renvoie qu'à la date choisi, le mois de novembre a 30 jour.

### Exercice 4 :

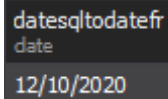
Écrire la fonction dateSqlToDatefr() qui vous permet de convertir la date fournie en paramètre au format SQL en une date au format JJ/MM/AA.

- Argument en entrée : une date au format AAAA-MM-JJ
- Argument en sortie : une date au format JJ/MM/AA.

```
CREATE OR REPLACE FUNCTION dateSqlToDatefr(myDate DATE)
RETURNS DATE
LANGUAGE plpgsql
AS $plpgsql$
BEGIN
    RETURN TO_CHAR(myDate, 'DD/MM/YYYY');
END;
$plpgsql$;

SELECT dateSqlToDatefr('2020-10-12');
```

On commence par entrer à la fonction que 'myDate' prend le format 'DD/MM/YYYY' puis on entre en paramètre une date au format SQL.



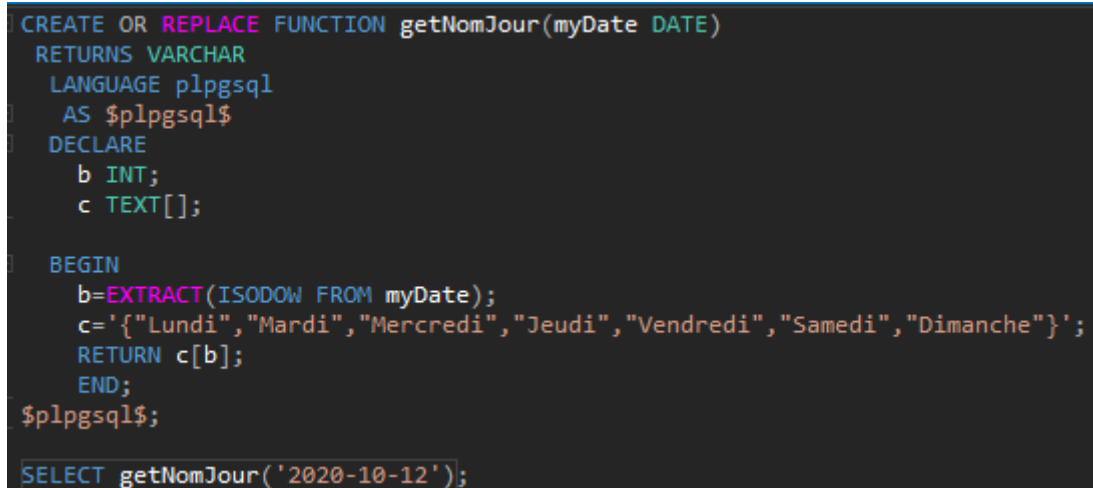
datesqltodatefr
date
12/10/2020

En exécutant la fonction, cela nous renvoie que '2020-10-12' au format SQL devient 12/10/2020 au format choisi.

#### Exercice 5 :

Écrire la fonction getNomJour() qui vous permet de retourner le nom du jour de la semaine correspondant à la date fournie en paramètre.

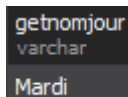
- Argument en entrée : une date au format SQL
- Argument en sortie : le nom du jour de la date.



```
CREATE OR REPLACE FUNCTION getNomJour(myDate DATE)
  RETURNS VARCHAR
  LANGUAGE plpgsql
  AS $plpgsql$
  DECLARE
    b INT;
    c TEXT[];
  BEGIN
    b=EXTRACT(ISODOW FROM myDate);
    c={'Lundi',"Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanche"};
    RETURN c[b];
  END;
$plpgsql$;

SELECT getNomJour('2020-10-12');
```

On commence par initialiser les différentes variables, puis on entre les différents jour de semaines puis on renvoie le nom du jour de la semaine correspondant à la date fournie en paramètre.



getnomjour
varchar
Mardi

En exécutant la fonction, cela nous renvoie que le '2020-10-12' est un Mardi.

### Exercice 6 :

Écrire une fonction qui retourne le nombre de clients débiteurs.

```
CREATE OR REPLACE FUNCTION countClient(count VARCHAR)
  RETURNS INT
  LANGUAGE plpgsql
  AS $plpgsql$

  BEGIN
    RETURN DISTINCT COUNT(DISTINCT client.num_client)
    FROM client
    WHERE client.adresse_client LIKE ('%' || count || '%');
  END;

$plpgsql$;

SELECT countClient('LANNIO');
```

countclient
int
2

En exécutant la fonction, cela nous renvoie qu'il y a 2 client débiteurs.

### **3) Conclusion**

Ce TP m'a permis de découvrir le langage PL/pgSQL, mais je n'ai pas réussi le dernier exercice.