

SQL PROCÉDURAL

- **But : permettre de programmer un comportement directement au niveau de la base de données**
 - Assurer la cohérence de données
 - Abstraire la complexité en fournissant des fonctions de haut-niveau
- **Extension des instructions SQL**
 - Cas de PostgreSQL : PL/pgSQL

PROCÉDURAL ≠ ORIENTÉ OBJET

SQL procédural date des années 1970

- OO commence réellement dans les années 1990
- Pas de notion d'objet ni de classe
- Notion de procédure \approx méthode statique dans une unique classe
- Beaucoup de lourdeur

Etat

- Pas de variable globale
 - On a les tables par contre
- Les procédures sont globales (stockées dans une table système)

CREATE FUNCTION

- **Permet de déclarer une fonction**

- avec un nom
- avec des paramètres
- avec une valeur de retour

- **Invocation d'une fonction**

SELECT nom_procédure (arg1, arg2, ...)

CREATE FUNCTION

```
CREATE FUNCTION nomFonction(nomParam1 type1,  
    nomParam2 type2,..., nomParamX typeX)  
    RETURNS typeOut AS $$  
    DECLARE  
        nomVar1 typeVar1;  
        nomVar2 typeVar2;  
        ...  
        nomVarY typeVarY;  
    BEGIN  
        corpsDeLaFonction ;  
    END ;  
$$ LANGUAGE plpgsql;
```

} Variables locales

\$\$ sert de délimiteur entre SQL et PL/pgSQL

EXAMPLE FUNCTION

```
CREATE FUNCTION fib (fib_for integer) RETURNS integer AS $$  
DECLARE  
BEGIN  
    IF fib_for < 2 THEN  
        RETURN fib_for;  
    END IF;  
    RETURN fib(fib_for - 2) + fib(fib_for - 1);  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT fib(8);
```

AUTRE MANIÈRE DE FAIRE UNE FONCTION

```
CREATE FUNCTION nomFonction(type1, type2, ..., typeX)
  RETURNS typeOut AS $$
  DECLARE
    nomParam1 ALIAS FOR $1 ;
    nomParam2 ALIAS FOR $2 ;
    ...
    nomParamX ALIAS FOR $X ;
    nomVar1 typeVar1;
    nomVar2 typeVar2;
    ...
    nomVarY typeVarY;
  BEGIN
    corpsDeLaFonction ;
  END ;
$$ LANGUAGE plpgsql;
```

}} sert de délimiteur entre SQL et PL/pgSQL

Paramètres

Variables locales

EXAMPLE FUNCTION

```
CREATE FUNCTION fib (integer) RETURNS integer AS $$  
DECLARE  
    fib_for ALIAS FOR $1;  
BEGIN  
    IF fib_for < 2 THEN  
        RETURN fib_for;  
    END IF;  
    RETURN fib(fib_for - 2) + fib(fib_for - 1);  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT fib(8);
```

AFFECTATION

variable := expression ;

COMMENTAIRE

-- tout ce qui suit -- est ignoré jusqu'à la fin de la ligne

IF

IF condition THEN ... END IF;

IF condition THEN ... ELSE ... END IF;

FOR

```
FOR record IN instruction_select  
LOOP  
    instructions  
END LOOP
```

record doit être déclarée et typée dans la partie DECLARE de la procédure

- L'instruction FOR fera un parcours successif des tuples retournées par l'`instruction_select`

EXAMPLE FOR ET IF

```
CREATE FUNCTION compteSalesDetailQtyMin10() RETURNS INTEGER AS $$  
DECLARE  
    i INTEGER := 0;  
    record RECORD;  
BEGIN  
    FOR record IN SELECT * FROM Salesdetail LOOP  
        IF record.qty>=10 THEN  
            i := i + record.qty;  
        END IF;  
    END LOOP;  
    RETURN i;  
END;  
$$ LANGUAGE plpgsql;
```

FOR

La procédure précédente est équivalente à
SELECT SUM(qty) FROM Salesdetail WHERE qty>=10;

Remarque importante : il n'y a aucune valeur ajoutée à implémenter soi-même ce qui devrait en fait être une requête SQL. Le code ne sera jamais aussi performant que la requête équivalente. Cela prend toujours plus de temps d'écrire une implémentation plutôt que d'écrire la requête, et le risque d'erreur est plus élevé. Dans le cadre de ce cours, ceci est donc considéré comme une faute et sanctionné comme tel.

WHILE

WHILE condition LOOP

instructions

END LOOP

EXCEPTION

Lever une exception

- RAISE EXCEPTION condition_name
- Confer documentation PostgreSQL pour la liste des condition_name possibles

<http://www.postgresql.org/docs/9.3/static/errcodes-appendix.html>

EXCEPTIONS

```
CREATE FUNCTION cours.ajouterCoursAuProgramme(etudiant
    INTEGER, cours INTEGER) RETURNS integer AS $$

DECLARE

    nombreECTS INTEGER;

    id_programme INTEGER;

BEGIN

    SELECT sum(ects) FROM cours.programme p, cours.cours c
    WHERE p.etudiant_id=etudiant AND p.cours_id=c.cours_id
    INTO nombreECTS;

    IF(nombreECTS>180) THEN RAISE data_exception; END IF;

    INSERT into cours.programme(etudiant_id,cours_id)
    VALUES (etudiant,cours) RETURNING id INTO id_programme;

    RETURN id_programme;

END;

$$ LANGUAGE plpgsql;
```


INTO

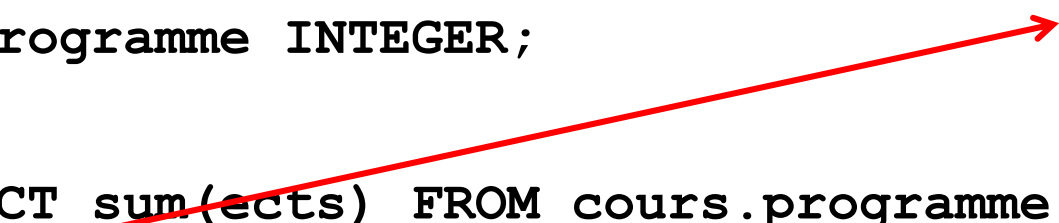
```
CREATE FUNCTION cours.ajouterCoursAuProgramme (etudiant  
    INTEGER, cours INTEGER) RETURNS integer AS $$
```

```
DECLARE
```

```
    nombreECTS INTEGER;
```

```
    id_programme INTEGER;
```

pour récupérer le résultat
d'un select dans une
variable locale



```
BEGIN
```

```
    SELECT sum(ects) FROM cours.programme p, cours.cours c  
    WHERE p.etudiant_id=etudiant AND p.cours_id=c.cours_id  
    INTO nombreECTS;
```

```
    IF (nombreECTS>180) THEN RAISE data_exception; END IF;
```


```
    INSERT into cours.programme(etudiant_id,cours_id)  
    VALUES (etudiant,cours) RETURNING id INTO id_programme;
```

```
    RETURN id_programme;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

pour récupérer la clé
auto-incrémentée
après un insert



PROCEDURE RENVOYANT UN TABLEAU

```
CREATE OR REPLACE FUNCTION listeAuthorsLivres() RETURNS SETOF RECORD AS $$  
  
DECLARE  
    sep VARCHAR;  
    texte VARCHAR;  
    sortie RECORD;  
    author RECORD;  
    title RECORD;  
  
BEGIN  
    FOR author IN SELECT * FROM authors LOOP  
        texte:='';sep:='';  
        FOR title IN SELECT * FROM titles t, titleauthor ta WHERE  
t.title_id=ta.title_id AND ta.au_id=author.au_id LOOP  
            texte:=texte || sep || title.title;  
            sep:=', ';  
        END LOOP;  
        SELECT author.au_fname, author.au_lname, texte INTO sortie;  
        RETURN NEXT sortie;  
    END LOOP;  
    RETURN;  
  
END;  
  
$$ LANGUAGE 'plpgsql';
```

PROCEDURE RENVOYANT UN TABLEAU

- Pour appeler une procédure renvoyant un tableau, il faut préciser la structure des colonnes ainsi que leurs noms:

```
SELECT * FROM listeAuthorsLivres() t(fname  
VARCHAR(20), lname VARCHAR(40), titles VARCHAR);
```

- On peut réutiliser cette requête

```
SELECT * FROM listeAuthorsLivres() t(fname  
VARCHAR(20), lname VARCHAR(40), titles VARCHAR)  
WHERE lname LIKE 'A%';
```