

**Module SLAM 3 TP4**

Propriétés	Description
<b>Intitulé</b>	<b>PROGRAMMATION AVEC POSTGRESQL</b>
<b>Outils</b>	<ul style="list-style-type: none"> <li>• A.G.L : WIN'DESIGN</li> <li>• SGBD : POSTGRESQL</li> </ul>
<b>Durée estimée en heures</b>	6 Heures
<b>Savoir-faire module SLAM3</b>	<ul style="list-style-type: none"> <li>• Concevoir une base de données</li> <li>• Valider un schéma de base de données</li> <li>• Programmer dans l'environnement de développement associé à un SGBD</li> </ul>
<b>Savoirs Module SLAM3</b>	<ul style="list-style-type: none"> <li>• Modèles de représentation des données</li> <li>• Langage de programmation associé à un SGBD</li> </ul>
<b>Documents joints</b>	Fiche d'exploitation pédagogique Annexe : Document de présentation de PL/PgSQL, fonctions et triggers Site de référence : <a href="https://docs.postgresql.fr/9.6/plpgsql.html">https://docs.postgresql.fr/9.6/plpgsql.html</a>
<b>Réception</b>	Développement de fonctions et de triggers
<b>Equipe</b>	Seul <input checked="" type="checkbox"/> Par équipe de ... <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4

## Présentation

PostgreSQL permet d'utiliser plusieurs langages de programmation qui permettent d'écrire des procédures, des fonctions et des triggers : les **PL-SQL**.

Ces langages sont SQL, PgSQL, Perl, C, Php, Java.

Les PL-SQL standard livrés avec PostgreSQL sont SQL et PgSQL

**Documentation** : Document de présentation de PL/PgSQL, fonctions et triggers

**Sites de référence** : <https://docs.postgresql.fr/9.6/plpgsql.html>

## Préalable :

Les manipulations sur la base de données devront s'effectuer à partir de la machine hôte selon 2 modes possibles :

- A partir de **Pgadmin** installé au TP 3 sur lequel on va installer un accès à distance
- A partir de **phpPgadmin**, outil à installer sur la machine hôte et accessible à partir du navigateur) **==> je vous conseille ce dernier** car pgadmin déjà fait...)

## Première partie :

# Mise en place de la base de données distante

### 1. Environnement matériel et logiciel à mettre en place :

#### 2 choix sont proposés :

- Utiliser votre machine virtuelle créée dans le TPinitPostgreSQL ou,
- Création d'une nouvelle VM Linux/Debian avec une nouvelle installation complète qui comprendra PostgreSQL seul.
  - Je vous suggère d'utiliser VirtualBOX et de regarder ce qu'il est possible de faire avec Vagrant : <https://fr.wikipedia.org/wiki/Vagrant>. Vagrant est à mi -chemin entre la virtualisation et Docker. Il permet de gagner du temps sur la création des ses environnements de travail (développement, test....)
  - <https://www.synbioz.com/blog/tech/vagrant-et-la-virtualisation-pour-faciliter-le-developpement>
  - L'incontournable : <https://www.grafikart.fr/tutoriels/vm-vagrant-chef-solo-482> ==> attention toutefois la vidéo date un peu et certains outils ont évolués depuis !
  - <https://www.supinfo.com/articles/single/6606-tutoriel-vagrant>

## A faire

- Si vous réutilisez votre 1<sup>ère</sup> VM, veuillez à contrôler que l'adresse IP de celle-ci est dans un adressage IP **différent** de votre Hôte. Si ce n'est pas le cas, **procéder aux modifications** pour paramétrer l'adresse IP de la carte réseau de la machine virtuelle (172.16.xx.2) en fonction de l'adresse IP de votre machine hôte (192.168.xx.1)
  - o Exemple : VM : 172.24.123.254 (IP/DHCP géré par le commutateur virtuel d'Hyper-V) versus l'hôte : 192.168.1.100 (IP/DHCP fournit par la box)
- Vérifier l'accès à Internet à partir de la machine virtuelle pour effectuer les opérations nécessaires.

### 2. Accès au serveur de base de données distant :

Suite à la configuration de la machine virtuelle, il faut tester les accès possibles à Postgresql à partir de la machine hôte, avec **PhppgAdmin**, via votre navigateur.

- Pour ceux qui réutilisent leur VM, l'utilisateur et mot de passe sont identiques à ceux que vous avez utilisé pour vous connecter à PostgreSQL avec pgadmin à partir de la VM.
- Pour ceux qui refont une installation, prenez le soin de les saisir dans un fichier .txt pour mémoire. Ce seront ceux que vous aurez paramétrer à l'installation de postgresQL.

## A faire

- Télécharger et installer PhppgAdmin (dernière version 7.1....)
  - o **Pré requis** : avoir un WAMP ou XAMP pour Windows
- Avec le navigateur, accéder au serveur avec **phppgadmin** en saisissant le nom de l'utilisateur et son mot de passe
  - o Précéder **phppgadmin** de l'adresse IP de votre VM (attention de bien vérifier les adresses entre votre commutateur virtuel de la VM vs l'hôte).



Il se peut que certains fichiers de paramétrage de Postgresql soient à configurer pour autoriser la machine distante à utiliser les bases de données. A vous de chercher sur Internet, si vous êtes « bloqué » me solliciter.

### 3. Installation de la base de données distante :

Une fois la connexion effectuée avec PhppgAdmin, il est possible d'installer et de manipuler des bases de données. Vous pouvez choisir d'installer la base de données correspondante ( Fredi ou Forma ).

## A faire

Les manipulations suivantes peuvent se faire à partir de **pgadmin** ou **phppgadmin**

- Implanter la base de données **BdFredri** ou **BdForma** à partir des scripts contenus dans le dossier **DOC TP4** de la façon suivante :

- Créer la base **BdFredri** ou **BdForma** (codage **UTF8**)

- Exécuter le script de création **BdpgFredri.sql** ou **BdpgForma.sql**

```

1  SET SCHEMA 'BdFredri';
2
3  CREATE TABLE DEMANDEURS
4  (
5      ADRESSE_MAIL text NOT NULL ,
6      NOM text NULL ,
7      PRENOM text NULL ,
8      RUE text NULL ,
9      CP text NULL ,
10     VILLE text NULL ,
11     NUM_RECU integer NULL
12     DEFAULT 0 ,
13     MOTDEPASSE text NULL
14 ,   CONSTRAINT PK_DEMANDEURS PRIMARY KEY (ADRESSE_MAIL)
15 );
16
17 -----
18 --      TABLE : MOTIFS
19 -----
20
21 CREATE TABLE MOTIFS
22 (

```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 246 msec.

Tables (7)	
>	adherents
>	clubs
>	demandeurs
>	lien
>	lignes_frais
>	ligues
>	motifs

- Alimenter la base avec les données à partir des scripts fournis

### BdFredri :

```
1 SET SCHEMA 'BdFredri';
2 INSERT INTO ligues (no_ligue, nom, sigle, president)
3 VALUES (1, 'Lorraine', 'L2L', 'Quiche' );
4
```

```
1 SET SCHEMA 'BdFredri';
2
3 INSERT INTO clubs (num_club, no_ligue, nom_club) VALUES (1, 1, 'Salle Armes de Villers les Nancy' );
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 56 msec.

```
1 SET SCHEMA 'BdFredri';
2
3 INSERT INTO demandeurs(
4     adresse_mail, nom, prenom, rue, cp, ville, num_recu, motd
5     VALUES ('jc.berbier@gmail.com', 'Berbier', 'Jean-Christophe', '12
6 INSERT INTO demandeurs(
7     adresse_mail, nom, prenom, rue, cp, ville, num_recu, motd
8     VALUES ('r.becker@gmail.com', 'Becker', 'Romain', '1 rue des mesai
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 95 msec.

```
1 SET SCHEMA 'BdFredri';
2
3 INSERT INTO lien VALUES (' 17 05 40 010 338', 'jc.berbier@gmail.com');
4 INSERT INTO lien VALUES (' 17 05 40 010 340', 'jc.berbier@gmail.com');
5 INSERT INTO lien VALUES (' 17 05 40 010 309', 'r.becker@gmail.com');
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 53 msec.

Query Query History

Query	Query History
23	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
24	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
25	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
26	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
27	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
28	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
29	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
30	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
31	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
32	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
33	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
34	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
35	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
36	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
37	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
38	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
39	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
40	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
41	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
42	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)
43	INSERT INTO adherents (numero_licence, num_club, nom, prenom, adresse_mail, num_recu, motd)

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 63 msec.

```

3  INSERT INTO motifs (libelle) VALUES ('Reunion');
4  INSERT INTO motifs (libelle) VALUES ('Reunion arbitrage');
5  INSERT INTO motifs (libelle) VALUES ('Competition');
6  INSERT INTO motifs (libelle) VALUES ('Competition regionale');
7  INSERT INTO motifs (libelle) VALUES ('Competition nationale');
8  INSERT INTO motifs (libelle) VALUES ('Competition internationale');
9  INSERT INTO motifs (libelle) VALUES ('Stage');
10
11
3  INSERT INTO lignes_frais VALUES ('r.becker@gmail.com', '2012-02-02', 'R
4  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-01-12',
5  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-03-13',
6  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-06-24',
7  );
8
9
10
11

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 54 msec.

BdForma :

```

1  SET SCHEMA 'BdForma';
2
3  CREATE TABLE ASSOCIATION
4  (
5      IDASSOCIATION int4 NOT NULL ,
6      NOMA char(50) NULL ,
7      NOICOM int4 NULL ,
8      NOMI char(25) NULL ,
9      PRENOMI char(50) NULL
10 ,   CONSTRAINT PK_ASSOCIATION PRIMARY KEY (IDASSOCIATION)
11 );
12
13 -----
14 --      TABLE : STAGIAIRE
15 -----
16
17 CREATE TABLE STAGIAIRE
18 (
19     IDSTAGIAIRE int4 NOT NULL ,
20     IDASSOCIATION int4 NOT NULL ,
21     NOM char(50) NULL ,
22     PRENOM char(50) NULL ,

```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 63 msec.

```

1 SET SCHEMA 'BdForma';
2
3 v INSERT INTO association (IDASSOCIATION, NOMA, NOICOM, NOMI, PRENOMI) VALUES
4 (100, 'Cercle Escrime', 15484758, 'Bidart', 'Julien' ),
5 (101, 'Comite Regional Olympique et Sportif de Lorraine', 16584785, 'Giroux',

```

```

1 SET SCHEMA 'BdForma';
2
3 v INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, I
4 (1, 101, 'PINOT', 'Thibaut', 'Benevole', 'Interlocuteur'
5 (2, 100, 'BARDET', 'Romain', 'Salarie', 'Administrateur'
6 (3, 100, 'VUILLERMOZ', 'Alexis', 'Salarie', 'Directeur du
7 (4, 101, 'DEMARRE', 'Arnaud', 'Benevole', 'Développeur',
8 (5, 101, 'ALAPHILIPPE', 'Julian', 'Benevole', 'Développeur
9 (6, 100, 'POULIDOR', 'Raymond', 'Benevole', 'Développeur

```

```

1 v INSERT INTO domaine (IDDOMAINE, NOM) VALUES
2 (1, 'Gestion'),
3 (2, 'Informatique'),
4 (3, 'Développement durable'),
5 (4, 'Secourisme'),
6 (5, 'Communication');
7
8 v INSERT INTO formation (IDFORMATION, IDDOMAINE, TITRE, PUBLICFORM, CONTEN
9 (10, 1, 'Soirée d'information sur la convention collective nationale du
10 (11, 1, 'Actualisation des connaissances sur la convention collective na
11 (12, 1, 'Comptabilité', 'Bénévoles et salariés', 'La comptabilité est un
12 (13, 1, 'Recherche de partenariat', 'Bénévoles et salariés', 'Le partena
13 (20, 2, 'Outlook Niveau 1', 'Bénévoles et salariés', 'Configurer Outlook
14 (21, 2, 'Outlook Niveau 2', 'Bénévoles et salariés', 'Initiation au VBA
15 (22, 2, 'Power Point Niveau 1', 'Bénévoles et salariés', 'Introduction à
16 (23, 2, 'Photoshop Niveau 1', 'Bénévoles et salariés', 'Rappel images nu
17 (24, 2, 'Photoshop Niveau 2', 'Bénévoles et salariés', 'Retouches photos
18 (30, 3, 'Organiser une manifestation éco responsable', 'Bénévoles et sal
19 (40, 4, 'Prévention et secours civique (PSC)', 'Bénévoles et salariés',
20 (41, 4, 'Bonnes pratiques et premiers secours', 'Bénévoles et salariés',
21 (50, 5, 'Conduite en réunion', 'Bénévoles et salariés', 'Permettre à cha

```

Data Output Messages Notifications

```

19 (40, 4, 'Prévention et secours civique (PSC)', 'Bénévoles et salariés', 'Le
20 (41, 4, 'Bonnes pratiques et premiers secours', 'Bénévoles et salariés', 'Al
21 (50, 5, 'Conduite en réunion', 'Bénévoles et salariés', 'Permettre à chacun
22 (51, 5, 'Communiquer avec la presse', 'Bénévoles et salariés', 'Quel message
23 (52, 5, 'Langues étrangères', 'Bénévoles et salariés', 'Anglais, Chinois, Es
24
25 v INSERT INTO session (IDSESSION, IDFORMATION, JOUR, DATESESSION, INTERVENANT,
26 (1, 11, 'Mercredi', '2015-01-14', 'A Contador', '2014-12-31', 24, 25),
27 (1, 12, 'Samedi', '2015-01-24', 'C Froome', '2015-01-03', 28, 30),
28 (1, 21, 'Mardi', '2015-01-20', 'C Froome', '2015-01-05', 10, 10),
29 (1, 30, 'Mercredi', '2015-03-18', 'A Contador', '2015-03-04', 8, 8),
30 (2, 11, 'Vendredi', '2015-02-20', 'V Nibali', '2015-02-06', 8, 8),
31 (2, 12, 'Jeudi', '2015-03-26', 'A Contador', '2015-03-12', 7, 7),
32 (2, 21, 'Vendredi', '2015-02-20', 'V Nibali', '2015-02-06', 9, 9),
33 (3, 11, 'Lundi', '2015-01-30', 'V Nibali', '2015-01-10', 70, 70),
34 (3, 12, 'Mardi', '2015-01-25', 'C Froome', '2015-01-10', 80, 80),
35 (4, 11, 'Jeudi', '2015-02-07', 'A Contador', '2015-01-20', 40, 40),
36 (5, 11, 'Lundi', '2015-02-01', 'C Froome', '2015-01-15', 90, 90);

```

Data Output Messages Notifications

INSERT 0 11

Query returned successfully in 62 msec.

```

1 SET SCHEMA 'BdForma';
2
3 v INSERT INTO inscrit (IDSTAGIAIRE, IDFORMATION, IDSESSION) VALUES
4 (1, 11, 1 ),
5 (2, 12, 1 ),
6 (3, 12, 1 );

```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 57 msec.

- Créer les utilisateurs (rôles) suivants :
  - adminsio** avec tous les droits ( mdp : **asio**)
  - usersio** avec la possibilité connexion seulement ( mdp : **usio** )
- Vérifier la connexion à la base de données **BdFred** ou **BdForma** avec les utilisateurs **adminsio** et **usersio**.

# Deuxième partie : Les fonctions en PL/PgSQL

Le langage *plpgsql* permet d'écrire des procédures, des fonctions, des triggers.

## Etape 1 - Mise en place d'une fonction

### Structure d'une fonction

```
CREATE OR REPLACE
  FUNCTION nomFonction (paramètres) RETURNS type AS $$
[DECLARE]
  -- bloc de déclaration des variables locales
[BEGIN]
  -- bloc des instructions de la fonction
[END] $$ LANGUAGE nomLangage ;
```

### Suppression d'une fonction

```
DROP FUNCTION nomfonction ();
```

## A faire avec BdFred :

Tester avec Pgadmin, la fonction **nbadhsexe** ci-dessous qui renverra le nombre d'adhérents de sexe masculin ou féminin en fonction de la valeur entrée.

### 1. Création de la fonction

- ✓ Ajouter la fonction en donnant son nom ( **nbadhsexe** )

The screenshot shows the 'Create - Function' dialog box in PgAdmin. The 'General' tab is selected. The 'Name' field contains 'nbadhsexe'. The 'Owner' field is set to 'postgres' and the 'Schema' field is set to 'BdFred'. The 'Comment' field is empty. At the bottom, a red error message box states 'Return type cannot be empty.' The dialog has tabs for 'General', 'Definition', 'Code', 'Options', 'Parameters', 'Security', and 'SQL'. At the bottom right, there are buttons for 'Close', 'Reset', and 'Save'.



- ✓ Choisir le type renvoyé( **Integer** ) et le langage ( **plpgsql** )

Create - Function

General Definition Code Options Parameters Security SQL

Custom return type? ☐

Return type integer

Language plpgsql

Arguments

Data type	Mode	Argument name	Default
-----------	------	---------------	---------

Code cannot be empty.

Close Reset Save

- ✓ Donner le paramètre ( nom de l'argument et type )

Arguments

Data type	Mode	Argument name	Default
character	INOUT	lettre	

- ✓ Saisir le code de la fonction de DECLARE ,..., BEGIN,... jusqu'à END comme ceci :

```

DECLARE
NbADH INT;
BEGIN
  SELECT COUNT(*) INTO NbADH FROM ADHERENTS WHERE SEXE = LETTRE ;
  RAISE NOTICE 'NOMBRE : % SEXE : % ',NbADH, LETTRE ;
RETURN NbADH ;
END

```

Create - Function

General Definition Code Options Parameters Security SQL

```

1 DECLARE
2   NbAdh INT;
3 BEGIN
4   SELECT COUNT(*) INTO NbAdh FROM adherents WHERE sexe = lettre;
5   RAISE NOTICE 'Nombre : % sexe : %', NbAdh, lettre;
6   return NbAdh;
7 END

```

Close Reset Save

## 2. Utilisation de la fonction

Dans l'éditeur SQL exécuter la fonction par :

**select nbadhsexe('F') ou select nbadhsexe('M')**

**nbadhsexe('F') :**

```

1 SET SCHEMA 'BdFredI';
2
3 SELECT nbadhsexe('F');

```

Data Output Messages Notifications

	nbadhsexe integer
1	12

**nbadhsexe('M') :**

```

1 SET SCHEMA 'BdFredI';
2
3 SELECT nbadhsexe('M');

```

Data Output Messages Notifications

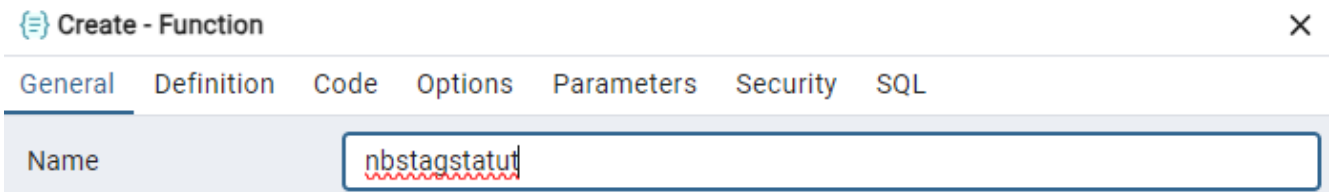
	nbadhsexe integer
1	28

## A faire avec BdForma:

Tester avec Pgadmin3, la fonction **nbstagstatut** ci-dessous qui renverra le nombre de stagiaires en fonction du statut entré ( Benevole ou Salarie )

### 1. Création de la fonction

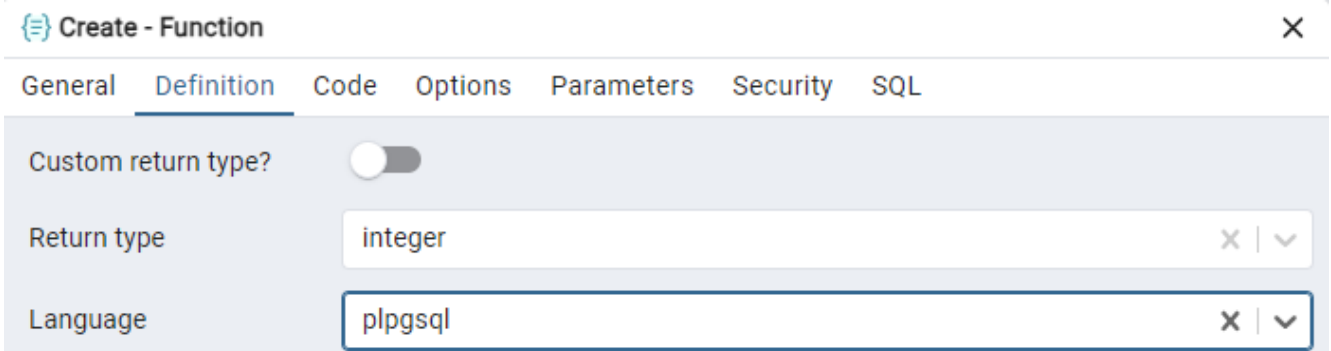
- ✓ Ajouter la fonction en donnant son nom (**nbstagstatut** )



General Definition Code Options Parameters Security SQL

Name

- ✓ Choisir le type renvoyé( **Integer** ) et le langage ( **plpgsql** )



General Definition Code Options Parameters Security SQL

Custom return type? ☐

Return type

Language

- ✓ Donner le paramètre ( nom de l'argument et type )



Arguments				
	Data type	Mode	Argument name	Default
	<input type="text" value="character"/>	<input type="text" value="IN"/>	<input type="text" value="stagiairestat"/>	<input type="text"/>

- ✓ Saisir le code de la fonction de DECLARE ,..., BEGIN,... jusqu'à END comme ceci :

```

DECLARE
NbSTAG INT;
BEGIN
SELECT COUNT(*) INTO NbSTAG FROM STAGIAIRE WHERE STATUT = ETAT;
RAISE NOTICE 'NOMBRE : % STATUT : %', NbSTAG, ETAT;
RETURN NbSTAG;
END

```

## 2. Utilisation de la fonction

Dans l'éditeur SQL exécuter la fonction par :

**Select nbstagstatut('Benevole') ou select nbstagstatut('Salarie')**

```
1 SET SCHEMA 'BdForma';
2
3 SELECT nbstagstatut('Benevole');
```

Data Output Messages Notifications

	nbstagstatut integer	
1	4	

```
3 SELECT nbstagstatut('Salarie');
```

Data Output Messages Notifications

	nbstagstatut integer	
1	2	

## Etape 2 - Création d'une fonction

**Remarque :** L'écriture de ces fonctions peut se faire avec pgadmin3 ou phpPgadmin

### A faire avec BdFredri :

1. Créer et tester la fonction **nbkmdem** qui renverra le nombre total de kilomètres parcourus pour un demandeur donné (on donne le nom en paramètre).

Create - Function

General

Definition

Code

Options

Parameters

Security

SQL

Name

nbkmdem

nbkmdem(nomdemandeur character)

General

Definition

Code

Options

Parameters

Security

SQL

Return type

integer

Language

plpgsql

Arguments

	Data type	Mode	Argument name	Default
	character	IN	nomdemandeur	

nbkmdem(nomdemandeur character)

General

Definition

Code

Options

Parameters

Security

SQL

```

1  DECLARE
2  NbKm INT;
3  BEGIN
4      SELECT SUM(km) INTO NbKm FROM lignes_frais
5      WHERE adresse_mail = (SELECT adresse_mail FROM demandeurs WHERE nom = nomdemandeur);
6      RAISE NOTICE 'NombreKm : % Demandeur : %', NbKm, nomdemandeur;
7      RETURN NbKm;
8  END
9

```

Test avec SELECT nbkmdem('Berbier') :

```
3 SELECT nbkmdem('Berbier');
```

Data Output Messages Notifications

nbkmdem integer

1	1930
---	------

- Créer et tester la fonction **coutkm** qui calcule le cout d'un trajet pour un demandeur donné et une date donnée.

Pour cela il faut rechercher le nombre de km effectués pour une ligne de frais que l'on multipliera par le cout kilométrique est fixé à 0,28 euros.

`coutkm(mailedem character, datetrajet date)` ✕

General Definition Code Options Parameters Security SQL

Name

Owner  ✕ ▼

Schema  ▼

Return type

Language  ✕ ▼

Arguments

	Data type	Mode	Argument name	Default
	character <span>▼</span>	IN <span>▼</span>	mailedem	<input type="text"/>
	date <span>▼</span>	IN <span>▼</span>	datetrajet	<input type="text"/>

```
1  ✓ DECLARE
2    coutKm double precision;
3  ✓ BEGIN
4      SELECT SUM(km * 0.28) INTO coutKm FROM lignes_frais
5      WHERE adresse_mail = maildem AND date_frais = datetrajet;
6      RAISE NOTICE 'Total : % Demandeur : %', coutKm, maildem;
7      RETURN coutKm;
8  END
```

**Test à faire** : `select coutkm('r.becker@gmail.com','02/02/2012');` ) donnera **168 €**.

```
1 SET SCHEMA 'BdFredI';
2
3 SELECT coutkm('r.becker@gmail.com', '02/02/2012');
```

**Remarque** : La condition **if not found then..** placée après la requête permet de savoir si l'enregistrement sollicité a été récupéré.

## A faire avec BdForma :

1. Créer et tester la fonction **nbseform** qui renverra le nombre de sessions de formation pour un domaine donné (on donne le nom en paramètre)

Create - Function

General

Definition

Code

Options

Parameters

Security

SQL

Name

nbseform

Owner

postgres

Schema

BdForma

Return type

integer

Language

plpgsql

Arguments

+

	Data type	Mode	Argument name	Default
	character	IN	nomdomaine	

Create - Function

General

Definition

Code

Options

Parameters

Security

SQL

```

1 DECLARE
2   nbSession INT;
3 BEGIN
4   SELECT COUNT(idformation) INTO nbSession FROM formation
5   WHERE iddomaine = (SELECT iddomaine FROM domaine WHERE nom = nomdomaine);
6   RAISE NOTICE 'Nombre de Sessions : % Nom Domaine : %', nbSession, nomdomaine;
7   RETURN nbSession;
8 END
9

```



```
1 SET SCHEMA 'BdForma';  
2  
3 SELECT nbstesform('Gestion')
```

Data Output			Messages	Notifications
	nbstesform			
	integer			
1		4		

2. Créer et tester la fonction **caform** qui calcule le chiffre d'affaires réalisé pour une session d'une formation donnée.

Create - Function ×

General

Definition

Code

Options

Parameters

Security

SQL

Name

caform

Owner

postgres

×

▼

Schema

BdForma

▼

**caform(nomform character, nbdesession integer)**
✕

General
Definition
Code
Options
Parameters
Security
SQL

Return type

integer

Language

plpgsql

✕ | ▼

**Arguments**

	Data type	Mode	Argument name	Default
	character   ▼	IN   ▼	nomform	
	integer   ▼	IN   ▼	nbdesession	

```

1  DECLARE
2  totalCa INT;
3  BEGIN
4      SELECT SUM(formation.cout * (session.nbplacesmax - session.nbplacesrestantes)) INTO totalCa FROM session
5      NATURAL JOIN formation
6      WHERE session.idsession = nbdesession AND formation.titre = nomform;
7      RAISE NOTICE 'Total CA : % Nom Formation : %', totalCa, nomform;
8      RETURN totalCa;
9  END

```

**Test à faire :** `select caform('Comptabilité',1)` donnera **240 €**.

```

1  SET SCHEMA 'BdForma';
2
3  SELECT caform('Comptabilité', 1)

```

Data Output
Messages
Notifications

	caform
1	integer
1	240

**Remarque:** La condition **if not found then..** placée après la requête permet de savoir si l'enregistrement sollicité a été récupéré.

# Troisième partie : Les Triggers en PL/PgSQL

## Présentation :

Les triggers sont des actions déclenchés par une requête action ( insert,delete,update)

- Les triggers se composent de deux éléments : le déclencheur et la fonction exécutée par le déclencheur. Cette fonction est sans argument et renvoie un résultat de type **trigger**.
- Plusieurs déclencheurs peuvent appeler la même fonction.
- OLD et NEW désignent la ligne à l'origine du déclenchement : OLD désigne les anciennes valeurs de cette ligne. OLD n'existe que pour un UPDATE ou un DELETE. NEW désigne les nouvelles valeurs de cette ligne. NEW n'existe que pour un INSERT ou un UPDATE. ☐☐☐ Les déclencheurs peuvent se déclencher en cascade

## Etape 1 - Mise en place d'un trigger

Exemple classique: mise en place d'un trigger déclenchant la fonction `verifMail()` avant l'insertion d'une occurrence dans une table

Mise en œuvre :

- Sur l'insertion d'un nouveau demandeur dans la base **BdFred**
- Sur l'insertion d'un nouveau stagiaire dans la base **BdForma**

Première étape :Création de la fonction Trigger

- ☐ la fonction trigger **verifMail()**

The screenshot shows the PostgreSQL function editor for a trigger named `verifMail()`. The interface has tabs for General, Definition, Code, Options, Parameters, Security, and SQL. The 'General' tab is selected, showing the function name 'verifMail' with a red squiggly underline, the owner 'postgres', and the schema 'BdForma'. Below this, the 'Definition' tab is selected, showing the arguments field (empty), the return type 'trigger', and the language 'plpgsql'.



```
verifMail()
General Definition Code Options Parameters Security SQL
1 BEGIN
2 IF (NEW.adresse_mail NOT LIKE '%@%') THEN
3     RAISE EXCEPTION 'email erroné';
4 END IF;
5 RETURN new;
6 END
```

Deuxième étape : création du Trigger (déclencheur) dans l'objet concerné

- Création sur la table **demandeurs** du trigger **verifDemandeur** pour la base **BdFred**

```
3 CREATE TRIGGER "verifDemandeur"
4     BEFORE INSERT ON demandeurs
5     FOR EACH ROW
6     EXECUTE PROCEDURE "BdFred"."verifMail"();
```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 82 msec.

- Création sur la table **stagiaire** du trigger **verifStagiaire** pour la base **BdForma**

```

3 CREATE TRIGGER "verifStagiaire"
4 BEFORE INSERT ON stagiaire
5 FOR EACH ROW
6 EXECUTE PROCEDURE "BdForma"."verifMail"();

```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 78 msec.

Test : Tester par insertion d'occurrences sans @ ou avec @ dans l'adresse mail

- dans la table **demandeurs** pour la base **BdFred**

**Sans @ :**

```

1 SET SCHEMA 'BdFred';
2
3 INSERT INTO demandeurs VALUES('mathisperotgmail.com', 'PEROT', 'Mathis', '68 Avenue Léon Gambetta', 82000, 'Montauban', '168764', 'mpp')

```

Data Output Messages Notifications

ERROR: email erroné  
CONTEXT: PL/pgSQL function "verifMail"() line 3 at RAISE

SQL state: P0001

**Avec @ :**

Query Query History

```

1 SET SCHEMA 'BdFred';
2
3 INSERT INTO demandeurs VALUES('mathisperot@gmail.com', 'PEROT', 'Mathis', '68 Avenue Léon Gambetta', 82000, 'Montauban', '168764', 'mpp')

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 79 msec.

- dans la table **stagiaire** pour la base **BdForma**

**Sans @ :**

```
1 SET SCHEMA 'BdForma';
2
3 INSERT INTO stagiaire VALUES(7, 101, 'PEROT', 'Mathis', 'Salarie', 'Développeur', 'mathisperotgmail.com', 3)
```

Data Output [Messages](#) Notifications

ERROR: email erroné  
CONTEXT: PL/pgSQL function "verifMail"() line 3 at RAISE  
  
SQL state: P0001

**Avec @ :**

```
1 SET SCHEMA 'BdForma';
2
3 INSERT INTO stagiaire VALUES(7, 101, 'PEROT', 'Mathis', 'Salarie', 'Développeur', 'mathisperot@gmail.com', 3)
```

Data Output [Messages](#) Notifications

INSERT 0 1

Query returned successfully in 84 msec.

## Etape 2 - Création d'un trigger

Vous pouvez créer les triggers suivants des 2 façons suivantes :

- ☐ soit avec **pgadmin en local** ☐ soit avec **phppgadmin à distance**

### A faire avec BdFred :

- Créer le trigger **afficheCoutTrajet** qui affichera le cout du trajet après une insertion d'une ligne de frais par un demandeur.

```
1 DECLARE
2 BEGIN
3     RAISE NOTICE 'Cout Total : %', (NEW.cout_peage + NEW.cout_repas + NEW.cout_hebergement);
4     RETURN NEW;
5 END;
```

```
3 CREATE TRIGGER "afficheCoutTrajet"
4 AFTER INSERT ON lignes_frais
5 FOR EACH ROW
6 EXECUTE PROCEDURE "BdFred"."calculCoutTrajet"();
```

- Tester par insertion d'occurrences dans la table **ligne\_frais** en utilisant le fichier de test **TestTriggerCoutTrajet** fourni.

```

1 SET SCHEMA 'BdFred';
2
3 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '11/02/2025', 'Compétition', 'Montauban-Montels', 40, 2, 4, 15, 0, 0, 0, 0);

```

Data Output Messages Notifications

NOTICE: Cout Total : 21  
 INSERT 0 1

Query returned successfully in 82 msec.

### A faire :

- Créer le trigger **verifKm** qui vérifiera avant l'insertion d'une ligne de frais que le kilométrage parcouru est > 0 s'il y a des frais de péage.

```

3 CREATE TRIGGER "verifKm"
4 BEFORE INSERT ON lignes_frais
5 FOR EACH ROW
6 EXECUTE PROCEDURE "BdFred"."verifKm"();

```

verifKm()

×

General Definition Code Options Parameters Security SQL

```

1 DECLARE
2 BEGIN
3 IF(NEW.km <= 0 AND NEW.cout_peage > 0) THEN
4 RAISE EXCEPTION 'Le kilométrage est inférieur ou égal à 0 alors que il y a des frais de péage';
5 END IF;
6 RETURN NEW;
7 END

```

- Tester par insertion d'occurrences dans la table **ligne\_frais** en utilisant le fichier de test **TestTriggerVerifKm** fourni.

Test si 0 Km alors que il y a des frais de péage :

```
1 SET SCHEMA 'BdFred';
2
3 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2025-06-28', 'Compétition', 'Montauban-Montetils', 0, 80, 4, 0, 0, 0, 0, 0);
```

Data Output Messages Notifications

ERROR: Le kilométrage est inférieur ou égal à 0 alors que il y a des frais de péage  
CONTEXT: PL/pgSQL function "verifKm"() line 4 at RAISE

SQL state: P0001

Test avec 80 Km alors que il y a pas de frais de péage :

Query Query History

```
1 SET SCHEMA 'BdFred';
2
3 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2025-06-29', 'Compétition', 'Montauban-Montetils', 80, 0, 4, 0, 0, 0, 0, 0);
```

Data Output Messages Notifications

NOTICE: Cout Total : 4  
INSERT 0 1

Query returned successfully in 80 msec.



**En Bonus :**

- Créer un trigger **verifLien** qui contrôlera qu'un demandeur est lié à un adhérent (on affichera le numéro, le nom et le prénom du ou des adhérents liés) avant l'insertion d'une ligne de frais. S'il n'y a pas d'adhérent lié au demandeur on refusera l'insertion dans la table **ligne\_frais**.

Quelques indications :

- ✓ Utiliser une variable de type record pour renvoyer une structure

Par exemple, l'instruction **select into adh \* from adherents where...** renvoie dans la variable **adh** les données correspondantes comme par exemple **adh.nom**, **adh.prenom**,...

- ✓ Utiliser **FOUND** pour savoir si le **select into** a renvoyé au moins un résultat
- ✓ Utiliser une boucle **for** pour afficher les numéros d'adhérents éventuels
- Tester en utilisant le fichier **TestTriggerVerifLien** fourni.

**A faire avec BdForma :**

- Créer le trigger **verifPlaces** qui vérifiera avant l'insertion d'une session d'une formation que le nombre de places proposées n'est pas supérieur au nombre de places maxi prévu.

```

3  CREATE TRIGGER "verifPlaces"
4      BEFORE INSERT ON session
5      FOR EACH ROW
6      EXECUTE PROCEDURE "BdForma"."verifPlaces"();

```

verifPlaces()

General Definition **Code** Options Parameters Security SQL

```

1  DECLARE
2      BEGIN
3          IF (NEW.nbplacesrestantes > NEW.nbplacesmax) THEN
4              RAISE EXCEPTION 'Le nombre de places proposé est supérieur au nombres de places totale';
5          END IF;
6          RETURN new;
7      END

```

- Tester par insertion d'occurrences dans la table **session** en utilisant le fichier de test **TestTriggerVerifPlaces** fourni.

Test avec INSERT INTO session (5, 12, 'Mercredi', '2025-10-04', 'C Froome', '2025-10-18', 100, 90) :

Data Output Messages Notifications

```
ERROR:  Le nombre de places proposé est supérieur au nombres de places totale  
CONTEXT:  PL/pgSQL function "verifPlaces"() line 4 at RAISE
```

SQL state: P0001

Test avec INSERT INTO session VALUES (5, 12, 'Mercredi', '2025-10-04', 'C Froome', '2025-10-18', 90, 100); :

```
3 INSERT INTO session VALUES (5, 12, 'Mercredi', '2025-10-04', 'C Froome', '2025-10-18', 90, 100);
```

Data Output Messages Notifications

```
INSERT 0 1
```

Query returned successfully in 55 msec.

**A faire :**

- Créer le trigger **affichePlacesRest** qui affichera le nombre de places restantes pour la session avant une insertion d'une inscription d'un stagiaire. Si le nombre de places restantes est nul, ne pas insérer l'occurrence.

```

3  CREATE TRIGGER "affichePlacesRest"
4      BEFORE INSERT ON inscrit
5      FOR EACH ROW
6      EXECUTE PROCEDURE "BdForma"."affichePlacesRest"();

```

 affichePlacesRest()

General Definition Code Options Parameters Security SQL

```

1  DECLARE
2      nbPlacesRestantes integer;
3  BEGIN
4      SELECT session.nbplacesrestantes INTO nbPlacesRestantes FROM session
5      WHERE session.idsession = NEW.idsession AND session.idformation = NEW.idformation;
6  IF (nbPlacesRestantes = 0) THEN
7      RAISE EXCEPTION 'Le nombre de places est nul. (nbplacesrestantes = %)', nbPlacesRestantes;
8  ELSE
9      RAISE NOTICE 'Le nombre de places restantes est de %', nbPlacesRestantes;
10 END IF;
11 RETURN new;
12 END

```

- Tester par insertion d'occurrences dans la table **inscrit** en utilisant le fichier de test **TestTriggerPlacesRest** fourni.

Test avec 0 places (nul) :

```

3  INSERT INTO inscrit VALUES (4, 11, 8);
4

```

Data Output Messages Notifications

ERROR: Le nombre de places est nul. (nbplacesrestantes = 0)  
CONTEXT: PL/pgSQL function "affichePlacesRest"() line 7 at RAISE

Test avec la session 1 et la formation 21 ou il reste 10 places :

```

1 SET SCHEMA 'BdForma';
2
3 INSERT INTO inscrit VALUES (4, 21, 1);
4

```

Data Output Messages Notifications

NOTICE: Le nombre de places restantes est de 10  
INSERT 0 1

Query returned successfully in 77 msec.

### En bonus:

- Créer le trigger **verifNbForm** qui vérifiera avant l'insertion d'une inscription que le stagiaire n'atteint pas la limite du nombre d'inscriptions.

(=) verifNbForm()

General Definition Code Options Parameters Security SQL

```

1 DECLARE
2     nbFormations INT;
3     nbStag INT;
4 BEGIN
5     SELECT stagiaire.nbformations INTO nbFormations FROM stagiaire WHERE idstagiaire = NEW.idstagiaire;
6     SELECT count(*) INTO nbStag FROM inscrit WHERE idstagiaire = NEW.idstagiaire;
7     IF (nbStag >= nbFormations) THEN
8         RAISE EXCEPTION 'Le stagiaire a atteint le nombre de formations maximum';
9     END IF;
10    RETURN new;
11 END;

```

```

1 SET SCHEMA 'BdForma';
2
3 CREATE TRIGGER "verifNbForm"
4     BEFORE INSERT ON inscrit
5     FOR EACH ROW
6     EXECUTE PROCEDURE "BdForma"."verifNbForm"();

```

- Tester par insertion d'occurrences dans la table **inscrit** en utilisant le fichier de test **TestTriggerNbformfourni**.

Stagiaire que je souhaite insérer n'ayant aucune formation disponible:

	idstagiaire [PK] integer	idassociation integer	nom character (50)	prenom character (50)	statut character (25)	fonction character (100)	adresse_mail character (32)	nbformations integer
1	6	100	POULIDOR	Raymond	Benevole	Développeur	poupou@gmail.com	0

Session dans laquelle j'essaie d'insérer le stagiaire ayant plus de formations disponibles :

	idsession [PK] integer	idformation [PK] integer	jour character (32)	datesession date	intervenant character (32)	datelimiteinscription date	nbplacesrestantes integer	nbplacesmax integer
1	8	12	Mercredi	2025-10-04	C Froome	2025-10-18	0	20

Test de l'insertion :

```
1 SET SCHEMA 'BdForma';
2
3 INSERT INTO inscrit VALUES(6, 12, 8)
```

```
ERROR: Le stagiaire a atteint le nombre de formations maximum
CONTEXT: PL/pgSQL function "verifNbForm"() line 8 at RAISE
```

- Créer le trigger **verifDateIns** qui vérifiera avant l'insertion d'une inscription que le stagiaire ne dépasse pas la date limite d'inscription.

```
3 CREATE TRIGGER "verifDateIns"
4 BEFORE INSERT ON inscrit
5 FOR EACH ROW
6 EXECUTE PROCEDURE "BdForma"."verifDateIns"();
7
8 verifDateIns()
```

General Definition Code Options Parameters Security SQL

```
1 DECLARE
2 dateLimite DATE;
3 BEGIN
4 SELECT datelimiteinscription INTO dateLimite FROM session;
5 IF (CURRENT_DATE > dateLimite) THEN
6 RAISE EXCEPTION 'La date saisie est supérieure à la date limite';
7 END IF;
8 RETURN new;
9 END;
10
```

- Tester par insertion d'occurrences dans la table **inscrit** en utilisant le fichier de test **TestTriggerVerifDatefourni**.

```
3 INSERT INTO inscrit VALUES (4, 12, 1);
4
```

Data Output Messages Notifications

NOTICE: Le nombre de places restantes est de 28

ERROR: La date saisie est supérieure à la date limite

CONTEXT: PL/pgSQL function "verifDateIns"() line 6 at RAISE

SQL state: P0001

Data Output Messages Notifications

	idstagiaire [PK] integer	idinformation [PK] integer	idsession [PK] integer
1	1	11	1
2	2	12	1
3	3	12	1
4	4	11	1
5	4	11	2
6	4	21	1

## Quatrième partie : Utilisation des règles

Les règles sont des actions déclenchées par une action du LMD select inclus. Ces actions peuvent remplacer l'action du déclencheur ou s'y ajouter.

### Etape 1 - Mise en place d'une règle

Il y a un exemple développé pour chaque mission ( FREDI ou FORMA )

### Exemple sur BdFredI :

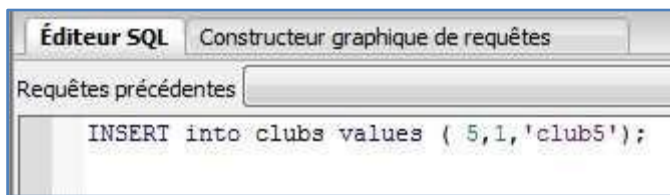
Création d'une règle **insertClub** qui, à chaque insertion dans la table Clubs affichera le nom du club ajouté.

□ . Création de la règle **insertClub**

```
CREATE OR REPLACE RULE "insertClub" AS
  ON INSERT TO clubs
  DO
  SELECT ' Ajout du club ' || new.nom_club
  AS " Insertion club "
  FROM clubs
  WHERE clubs.num_club = new.num_club;
```

```
1 SET SCHEMA 'BdFredI';
2
3 CREATE OR REPLACE RULE "insertClub" AS
4   ON INSERT TO clubs
5   DO
6   SELECT 'Ajout du club' || NEW.nom_club
7   AS "Insertion club"
8   FROM clubs
9   WHERE clubs.num_club = NEW.num_club;
```

- Tester par insertion d'occurrences dans la table **clubs** avec l'éditeur sql en vérifiant l'affichage en sortie de données.



Panneau sortie	
Sortie de données	
Insertion club text	
1	Ajout du club club5

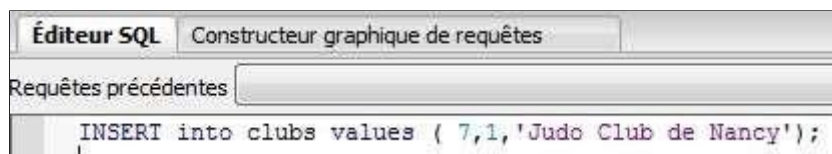
```

1 SET SCHEMA 'BdFredI';
2
3 INSERT INTO clubs VALUES (5,1, 'club5');

```

Data Output	
Insertion club text	
1	Ajout du clubclub5

- Modifier la règle **insertClub** en ajoutant dans l'affichage le nom de la ligue du club inséré comme dans l'exemple ci-dessous :



Sortie de données	
Insertion club text	
1	Ajout du club Judo Club de Nancy ligue UFOLEP Lorraine

Nouvelle règle :

```

insertClub
General Definition Condition Commands SQL
1 SELECT (((('Ajout du club '::text || new.nom_club) || ' ligue '::text) || ' '::text) || ligues.nom) AS "Insertion club"
2 FROM ("BdFredI".clubs
3 JOIN "BdFredI".ligues USING (no_ligue))
4 WHERE ((clubs.num_club = new.num_club) AND (ligues.no_ligue = new.no_ligue))

```

Test avec l'insertion :

```

3 INSERT INTO clubs VALUES (7, 1, 'Judo Club de Nancy');
4

```

	Insertion club text	🔒
1	Ajout du club Judo Club de Nancy ligue Lorraine	

## Exemple sur BdForma :

Création d'une règle **insertStagiaire** qui, à chaque insertion dans la table **stagiaire** affichera le nom du stagiaire ajouté.

- . Création de la règle **insertStagiaire**

```
CREATE OR REPLACE RULE "insertStagiaire" AS
ON INSERT TO stagiaire DO
SELECT ' Ajout du stagiaire ' || new.nom::text AS " Insertion stagiaire "
FROM stagiaire
WHERE stagiaire.idstagiaire = new.idstagiaire;
```

```
3 v CREATE OR REPLACE RULE "insertStagiaire" AS
4     ON INSERT TO stagiaire DO
5     SELECT ' Ajout du stagiaire ' || new.nom::text AS " Insertion stagiaire "
6     FROM stagiaire
7     WHERE stagiaire.idstagiaire = new.idstagiaire;
```

- Tester par insertion d'occurrences dans la table **stagiaire** avec l'éditeur sql en vérifiant l'affichage en sortie de données.

Éditeur SQL
INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS) VALUES (7, 101, 'AMSTRONG', 'Lance', 'Benevole', 'Docteur', 'lance@gmail.com', 1);

Sortie de données	Expliquer (Explain)	Messages
Insertion stagiaire text		
1	Ajout du stagiaire AMSTRONG	

```
3 v INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS)
4     VALUES (7, 101, 'AMSTRONG', 'Lance', 'Benevole', 'Docteur', 'lance@gmail.com', 1);
```

	Insertion stagiaire text	🔒
1	Ajout du stagiaire AMSTRONG	

- Modifier la règle **insertStagiaire** en ajoutant dans l'affichage le nom de l'association du stagiaire inséré comme dans l'exemple ci-dessous :

Nouvelle règle :



InsertStagiaire

General Definition Condition Commands SQL

```

1 SELECT ((( ' Ajout du stagiaire '::text || (new.nom)::text || ' de l association '::text) || (association.noma)::text) AS " Insertion stagiaire "
2 FROM "BdForma".stagiaire
3 JOIN "BdForma".association USING (idassociation))
4 WHERE ((stagiaire.idstagiaire = new.idstagiaire) AND (association.idassociation = new.idassociation))

```

Éditeur SQL

```

INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM,
STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS) VALUES
(8, 100, 'FIGNON', 'Laurent', 'Benevole', 'Soigneur', 'fignon@gmail.com', 1);

```

Sortie de données	
	Insertion stagiaire text
1	Ajout du stagiaire FIGNON de l association Cercle Escrime

Test avec nouvelle insertion :

```

3 INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS)
4 VALUES (8, 100, 'FIGNON', 'Laurent', 'Benevole', 'Soigneur', 'fignon@gmail.com', 1);

```

	Insertion stagiaire text
1	Ajout du stagiaire FIGNON de l association Cercle Escrime

## Etape 2 - Création d'une règle

### A faire sur BdFredri :

**A faire** : Créer une règle **majLigueClub** sur la table **club** qui affichera l'ancienne et la nouvelle ligue dans le cas d'une mise à jour de la ligue pour un club donné.

Exemple : Le club de Football de Laxou change de ligue. Il passe de la ligue de football de lorraine à la ligue UFOLEP de Lorraine

Les LIGUES :

	no_ligue [PK] integer	nom text	sigle text	president text
1	1	UFOLEP Lorraine	UFL	Quichet
2	2	FOOTBALL Lorraine	LFL	Parentin

Mise à jour du club de Football de Laxou :

Avant la mise à jour

	num_club [PK] integer	no_ligue integer	nom_club text	rue_ text
1	1	2	Laxou Football Club	

Mise à jour avec la règle **majLigueClub**

Sortie de données	Expliquer (Explain)	Messages	Historique
Changement de ligue text			
1	Mise a jour du club Laxou Football Club ancienne ligue 2 nouvelle ligue 1		

Après la mise à jour

	num_club [PK] integer	no_ligue integer	nom_club text
1	1	1	Laxou Football Club

Avant la mise à jour :

	num_club [PK] integer	no_ligue integer	nom_club text	rue_club text	cp_club text	ville_club text
1	1	1	Salle Armes de Villers les Nancy	[null]	[null]	[null]
2	2	2	Laxou Football Club	[null]	[null]	[null]

Ligues :

	no_ligue [PK] integer	nom text	sigle text	president text
1	1	UFOLEP Lorraine	L2L	Quiche
2	2	FOOTBALL Lorraine	LFL	Parentin

Mise à jour :

```
3 UPDATE clubs SET no_ligue = 1
4 WHERE num_club = 2
```

	Changement de ligue text
1	Mise a jour du club Laxou Football Club ancienne ligue 2 nouvelle ligue 1

Après mise à jour :

	num_club [PK] integer	no_ligue integer	nom_club text	rue_club text	cp_club text	ville_club text
1	1	1	Salle Armes de Villers les Nancy	[null]	[null]	[null]
2	2	1	Laxou Football Club	[null]	[null]	[null]

**A faire** : Modifier la règle **majLigueClub** sur la table **club** qui affichera à la place du numéro des ligues **le nom des ligues** correspondantes comme dans l'exemple ci-dessous :

Sortie de données	Expliquer (Explain)	Messages	Historique
Changement de ligue text			
1	Mise a jour du club : Laxou Football Club    Ancienne ligue : FOOTBALL Lorraine    Nouvelle ligue : UFOLEP Lorraine		

Règle mise à jour :

**majLigueClub**

General Definition Condition **Commands** SQL

```

1 SELECT ((((' Mise a jour du club '::text || clubs.nom_club) || ' Ancienne ligue : '::text) || (SELECT ligues.nom
2 FROM "BdFredri".ligues
3 WHERE (ligues.no_ligue = old.no_ligue))) || ' Nouvelle ligue : '::text) || (SELECT ligues.nom
4 FROM "BdFredri".ligues
5 WHERE (ligues.no_ligue = new.no_ligue))) AS " Changement de ligue "
6 FROM "BdFredri".clubs
7 WHERE (clubs.num_club = new.num_club)

```

Query Query History

```

1 SET SCHEMA 'BdFredri';
2
3 UPDATE clubs SET no_ligue = 1 WHERE num_club = 2;

```

Data Output Messages Notifications

Changement de ligue  
text

1 Mise a jour du club Laxou Football Club Ancienne ligue : FOOTBALL Lorraine Nouvelle ligue : UFOLEP Lorraine

	num_club [PK] integer	no_ligue integer	nom_club text	rue_club text	cp_club text	ville_club text
1	1	1	Salle Armes de Villers les Nancy	[null]	[null]	[null]
2	2	1	Laxou Football Club	[null]	[null]	[null]

## A faire sur BdForma :

**A faire** : Créer une règle **majAssoStagiaire** sur la table **stagiaire** qui affichera l'ancien et le nouveau club dans le cas d'une mise à jour de l'association pour un stagiaire donné.

Exemple : Le stagiaire AMSTRONG change d'association.

Il passe de l'association **Comité Régional** à l'association **Cercle d'escrime**

Les  
ASSOCIATIONS

	idassociation [PK] integer	noma character(50)	noicom integer	nomi character(25)
1	100	Cercle Escrime	15484758	Bidart
2	101	Comite Regional	16584785	Giroux

Mise à jour du club du stagiaire AMSTRONG :

Avant la mise à jour

	idstagiaire [PK] integer	idassociation integer	nom character(50)	prenom character(50)	statut charact
7	7	101	AMSTRONG	Lance	Benevo

Mise à jour avec la règle **majAssoStagiaire**



## Règle :

majAssoStagiaire

General Definition Condition Commands SQL

```

1 SELECT ((((' Mise a jour du stagiaire '::text || (stagiaire.nom)::text) || ' Ancienne Asso : '::text) || (( SELECT association.noma
2 FROM "BdForma".association
3 WHERE (association.idassociation = old.idassociation))))::text) || ' Nouvelle Asso : '::text) || (( SELECT association.noma
4 FROM "BdForma".association
5 WHERE (association.idassociation = new.idassociation))))::text AS " Changement d'association "
6 FROM "BdForma".stagiaire
7 WHERE (stagiaire.idstagiaire = new.idstagiaire)

```

## Avant la mise à jour :

7	7	100	AMSTRONG	...	La
---	---	-----	----------	-----	----

```

1 SET SCHEMA 'BdForma';
2
3 UPDATE stagiaire SET idassociation = 101 WHERE idstagiaire = 7

```

Data Output Messages Notifications

Changement d'association  
text

1	Mise a jour du stagiaire AMSTRONG Ancienne Asso : Cercle Escrime Nouvelle Asso : Comite Regional Olympique et Sportif de Lorraine
---	---

## Après la mise à jour :

7	7	101	AMSTRONG	...	L
---	---	-----	----------	-----	---