

Module SLAM 3 TP4

| Propriétés | Description |
|----------------------------------|--|
| Intitulé | PROGRAMMATION AVEC POSTGRESQL |
| Outils | <ul style="list-style-type: none"> • A.G.L : WIN'DESIGN • SGBD : POSTGRESQL |
| Durée estimée en heures | 6 Heures |
| Savoir-faire module SLAM3 | <ul style="list-style-type: none"> • Concevoir une base de données • Valider un schéma de base de données • Programmer dans l'environnement de développement associé à un SGBD |
| Savoirs Module SLAM3 | <ul style="list-style-type: none"> • Modèles de représentation des données • Langage de programmation associé à un SGBD |
| Documents joints | <p>Fiche d'exploitation pédagogique Annexe : Document de présentation de PL/PgSQL, fonctions et triggers Site de référence : https://docs.postgresql.fr/9.6/plpgsql.html</p> |
| Réception | Développement de fonctions et de triggers |
| Equipe | 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^{ère} VM, veillez 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 paramétriser 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)
 - 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 fourni 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....)
 - **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
 - 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.

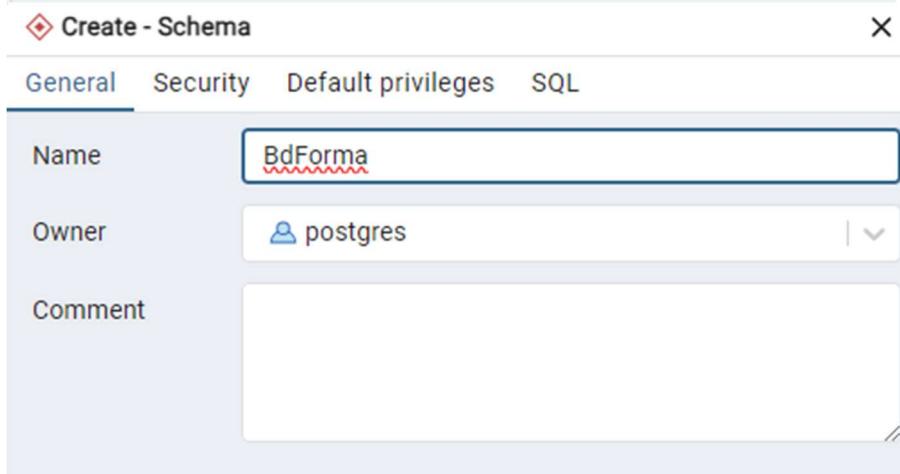
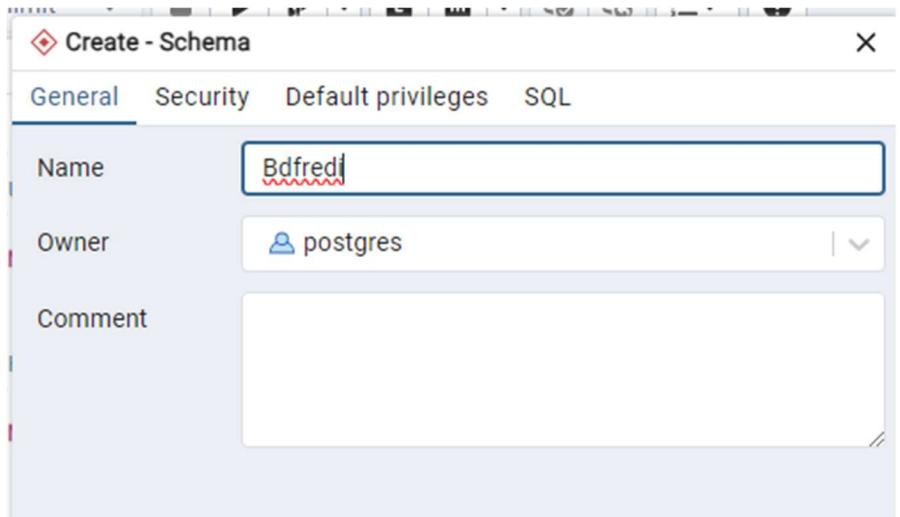
3. Installation de la base de données distante :

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 **BdFredi** ou **BdForma** à partir des scripts contenus dans le dossier **DOC TP4** de la façon suivante :
 - Créer la base **BdFredi** ou **BdForma** (codage **UTF8**)



- Exécuter le script de création **BdpfFredi.sql** ou **BdpfForma.sql**

Query Query History

```

1  -- Generation d'une base de données pour
2  -- PostgreSQL
3  -- (4/12/2012 18:12:27)
4
5
6  -- Nom de la base : MLR1
7  -- Projet : Espace de travail
8  -- Auteur : mh
9  -- Date de dernière modification : 4/12/2012 18:09:25
10
11
12 -- drop database MLR1;
13
14 -- CREATION DE LA BASE
15
16
17 -- CREATE DATABASE MLR1;
18
19
20 -- TABLE : DEMANDEURS
21
22
23 ▾ CREATE TABLE DEMANDEURS

```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 260 msec.

Création des tables dans la base Fredi

Query Query History

```

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 ,

```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 67 msec.

Création de la base Forma

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

```

Query  Query History
1 set schema 'BdFredi';
2
3 ✓ INSERT INTO ligues (no_ligue, nom, sigle, president)
4   VALUES (1,'Lorraine','L2L','Quiche' );
5
6 INSERT INTO clubs (num_club, no_ligue, nom_club) VALUES (1, 1, 'Salle Armes de Villers les Nancy' );
7
8 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 443', 1, 'BANDILELLA', 'CLEMENT', 'M
9 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 340', 1, 'BERBIER', 'LUCILLE', 'F
10 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 338', 1, 'BERBIER', 'THEO', 'M
11 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 309', 1, 'BECKER', 'ROMAIN', 'M
12 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 334', 1, 'BIACQUEL', 'VERONIQUE', 'F
13 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 399', 1, 'BIDELOT', 'BRIGITTE', 'F
14 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 442', 1, 'BIDELOT', 'JULIE', 'F
15 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 308', 1, 'BILLOT', 'DIDIER', 'M
16 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 329', 1, 'BILLOT', 'CLAIREE', 'F
17 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 254', 1, 'BILLOT', 'MARIANNE', 'F
18 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 407', 1, 'BINNET', 'MARIUS', 'M
19 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 444', 1, 'CALDI', 'THOMAS', 'M
20 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 431', 1, 'CASTEL', 'TIMOTHE', 'M
21 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 428', 1, 'CHEOLLE', 'NICOLAS', 'M
22 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 414', 1, 'CHERPION', 'UGO', 'M
23 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 441', 1, 'CHEVOITINE', 'LOUIS', 'M
24 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 440', 1, 'CHOURARNO', 'TOM', 'M
25 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 402', 1, 'COTIN', 'FLORIAN', 'M
26 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 351', 1, 'DEPERRIN', 'ARNAUD', 'M
27 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 409', 1, 'DEPRETRE', 'BEATRICE', 'F
28 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 446', 1, 'DURCRICK', 'AUGUSTIN', 'M
29 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 395', 1, 'GARBILLON', 'GILLES', 'M
30 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 339', 1, 'GARBILLON', 'YANN', 'M
31 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 382', 1, 'HAGENBACH', 'CLEMENTINE', 'F
32 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 420', 1, 'HASFELD', 'AUXANE', 'F
33 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 341', 1, 'HUMERT', 'ISABELLE', 'F
34 ✓ INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_nais, rue, cp, ville) VALUES (' 17 05 40 010 422', 1, 'LAETECLON', 'CLEMENT', 'M

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 85 msec.

Insertion des données dans la base Fredi

Query Query History

```

1 set schema 'BdForma';
2 ✓ INSERT INTO domaine (IDDOMAINE, NOM) VALUES
3   (1, 'Gestion'),
4   (2, 'Informatique'),
5   (3, 'Développement durable'),
6   (4, 'Secourisme'),
7   (5, 'Communication');
8 ✓ INSERT INTO formation (IDFORMATION, IDDOMAINE, TITRE, PUBLICFORM, CONTENU, COUT, OBJECTIFS) VALUES
9   (10, 1, 'Soirée d''information sur la convention collective nationale du sport', 'Bénévoles et salariés', 'La convention collective du sport règle, sur l''ensemble
10  (11, 1, 'Actualisation des connaissances sur la convention collective nationale du sport et la responsabilité des dirigeants.', 'Bénévoles et salariés', ' L''aven
11  (12, 1, 'Comptabilité', 'Bénévoles et salariés', 'La comptabilité est une discipline pratique', 120, 'Apprendre les bases de la comptabilité en entreprise'),
12  (13, 1, 'Recherche de partenariat', 'Bénévoles et salariés', 'Le partenariat se définit comme une association active de différents intervenants ', 60, 'Comprendre
13  (20, 2, 'Outlook Niveau 1', 'Bénévoles et salariés', 'Configurer Outlook, paramétriser les notifications dans Outlook', 70, 'Prendre en main l''outil Outlook'),
14  (21, 2, 'Outlook Niveau 2', 'Bénévoles et salariés', 'Initiation au VBA d''Outlook, configuration pour un serveur IMAP', 90, 'Perfecter son utilisation de l''outil Outloo
15  (22, 2, 'Power Point Niveau 1', 'Bénévoles et salariés', 'Introduction à powerpoint\nMasques de diapositive et arrière-plan', 50, 'Parfaire ses connaissances su
16  (23, 2, 'Photoshop Niveau 1', 'Bénévoles et salariés', 'Rappel images numériques, modes colorimétriques, présentation et personnalisation', 80, 'Parfaire ses conn
17  (24, 2, 'Photoshop Niveau 2', 'Bénévoles et salariés', 'Retouches photos, principes de bases d''impression', 120, 'Parfaire ses connaissances sur Photoshop'),
18  (30, 3, 'Organiser une manifestation éco responsable', 'Bénévoles et salariés', 'La responsabilité environnementale (écoresponsabilité ou la responsabilité humain
19  (40, 4, 'Prévention et secours civique (PSC)', 'Bénévoles et salariés', ' Les situations d''accident sont abordées en modules', 110, 'Initiation à la réduction de

```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 53 msec.

Insertion des données dans la table Forma

- 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 **BdFredi** 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 BdFredi :

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

```
DECLARE
NbAdh INT;
BEGIN
    Select count(*) INTO NbAdh From adherents where sexe = lettre;
    RAISE NOTICE 'Nombre : % sexe : % ', NbAdh, lettre ;
return NbAdh;
END;
```

1. Création de la fonction

- ✓ Ajouter la fonction en donnant son nom (**nbadhsex**e)
- ✓ Choisir le type renvoyé(**Integer**) et le langage (**plpgsql**)
- ✓ Donner le paramètre (nom de l'argument et type)
- ✓ Saisir le code de la fonction de DECLARE ,..., BEGIN,... jusqu'à END comme ceci :

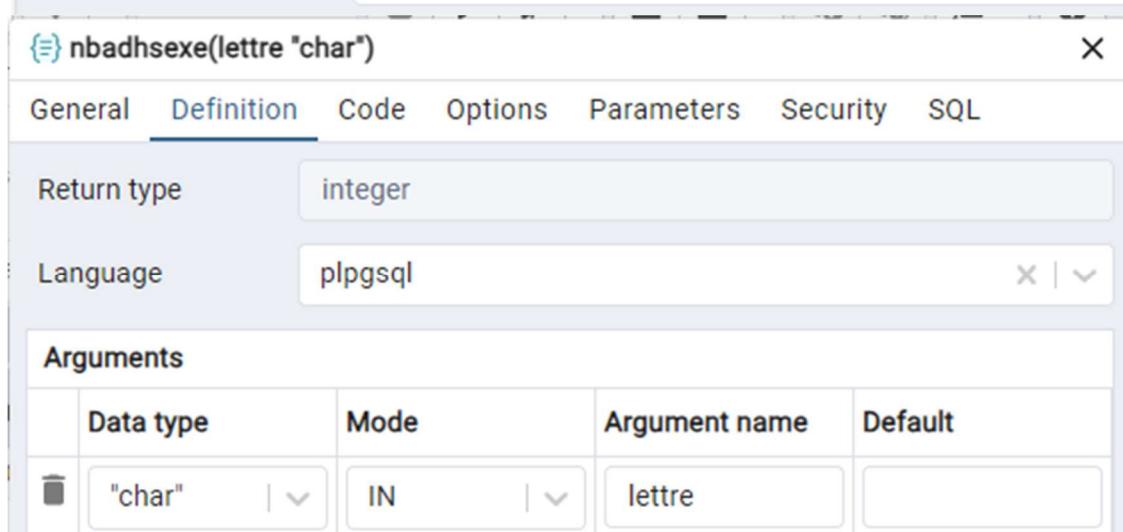
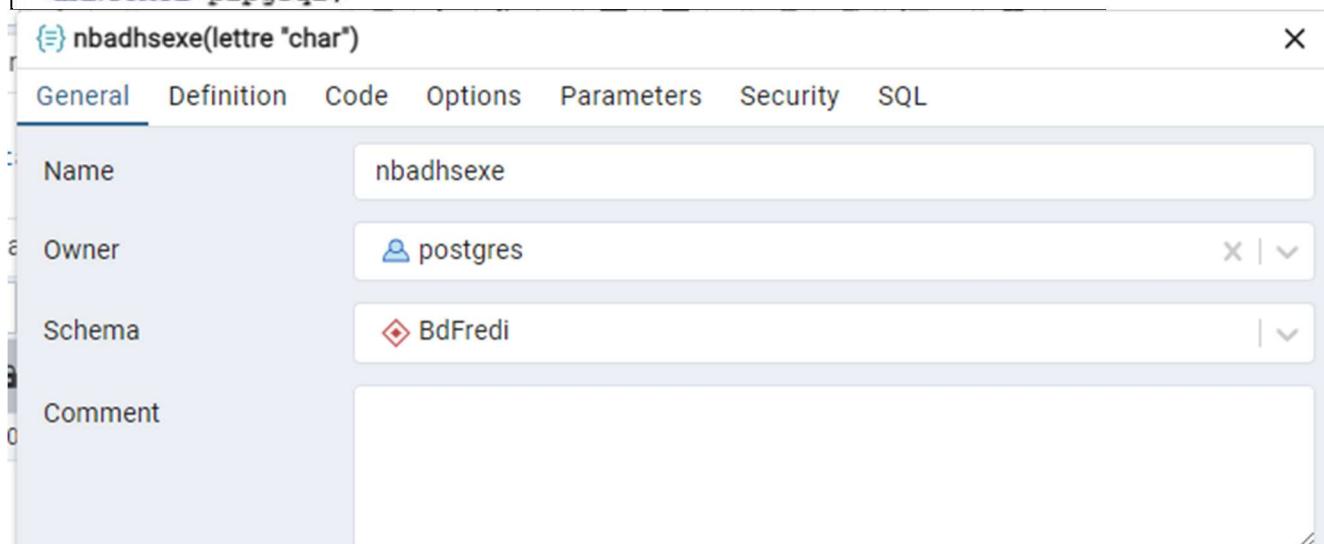
```
DECLARE
NBADH INT;
BEGIN
    SELECT COUNT(*) INTO NBADH FROM ADHERENTS WHERE SEXE = LETTRE ;
    RAISE NOTICE 'NOMBRE : % SEXE : % ',NBADH, LETTRE ;
RETURN NBADH ;
END
```

La fonction doit normalement avoir le contenu suivant :

```
-- Function: nbadhsex(text)

-- DROP FUNCTION nbadhsex(text);

CREATE OR REPLACE FUNCTION nbadhsex(lettre text)
RETURNS integer AS
$BODY$DECLARE
NbAdh int;
BEGIN
Select count(*) into NbAdh From adherents where sexe = lettre ;
RAISE NOTICE 'Nombre : % sexe : % ', NbAdh , lettre ;
return NbAdh ;
END$BODY$
LANGUAGE plpgsql;
```



nbadhsex()

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

```

i **?** **X Close** **↻ Reset** **Save**

2. Utilisation de la fonction

Dans l'éditeur SQL exécuter la fonction par :

select nbadhsex('F') ou select nbadhsex('M')

Query Query History

```

1 set schema 'BdFredi';
2 select nbadhsex('M');

```

Data Output Messages Notifications

+ **File** **▼** **Copy** **▼** **trash** **Download** **SQL**

| | nbadhsex | integer |
|---|----------|---------|
| 1 | | 28 |

Query Query History

```

1 set schema 'BdFredi';
2 select nbadhsex('F');

```

Data Output Messages Notifications

+ **File** **▼** **Copy** **▼** **trash** **Download**

| | nbadhsex | integer |
|---|----------|---------|
| 1 | | 12 |

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**)

- ✓ Choisir le type renvoyé(**Integer**) et le langage (**plpgsql**)
- ✓ Donner le paramètre (nom de l'argument et type)
- ✓ 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;
```

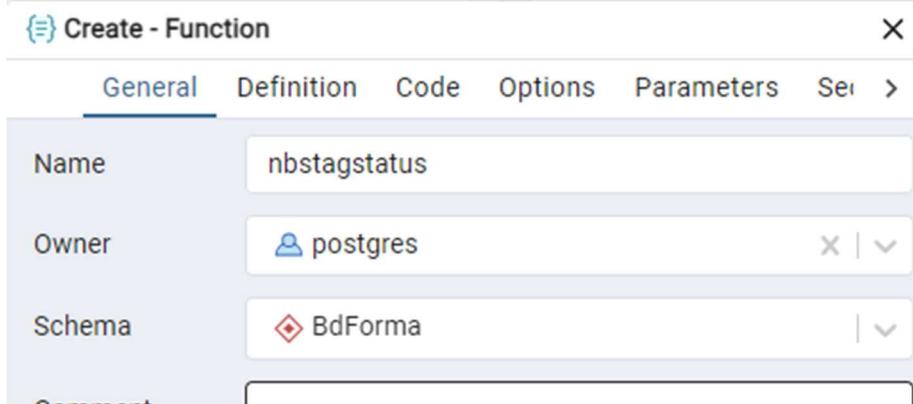
```
END
```

La fonction doit normalement avoir le contenu suivant

```
-- Function: nbstagstatut(text)

-- DROP FUNCTION nbstagstatut(text);

CREATE OR REPLACE FUNCTION nbstagstatut(etat text)
RETURNS integer AS
$BODY$DECLARE
NbStag int;
BEGIN
Select count(*) into NbStag From stagiaire where statut = etat;
RAISE NOTICE 'Nombre : % statut : %', NbStag, etat;
return NbStag;
END
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
```



The screenshot shows two windows from the pgAdmin interface. The top window is titled 'nbstagstatus(etat text)' and is under the 'Definition' tab. It shows the return type as 'integer' and the language as 'plpgsql'. The 'Arguments' section contains one argument named 'etat' with a data type of 'text' and a mode of 'IN'. The bottom window is also titled 'nbstagstatus(etat text)' and is under the 'Code' tab. It displays the following PL/pgSQL code:

```

1 DECLARE
2     NbStag INT;
3 BEGIN
4     SELECT COUNT(*) INTO NbStag FROM stagiaire WHERE statut = etat;
5     RAISE NOTICE 'Nombre : % statut : %', NbStag, etat;
6     RETURN NbStag;
7 END;

```

2. Utilisation de la fonction

Dans l'éditeur SQL exécuter la fonction par :

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

The screenshot shows the pgAdmin query editor. The 'Query' tab is active, containing the following SQL code:

```

1 set schema 'BdForma';
2 select nbstagstatus('Salarie');

```

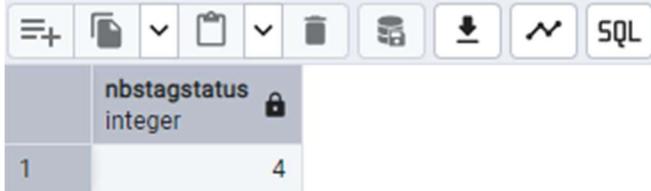
The 'Data Output' tab is active, showing the result of the query:

| | nbstagstatus | integer |
|---|--------------|---------|
| 1 | | 2 |

Query Query History

```
1 set schema 'BdForma';
2 select nbstagstatus('Benevole');
```

Data Output Messages Notifications

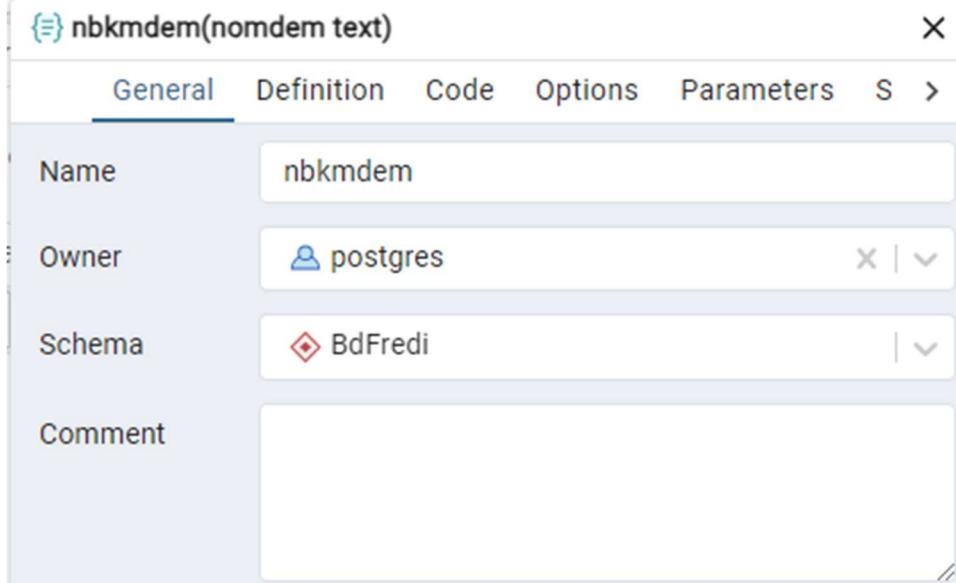


Etape 2 - Création d'une fonction

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

A faire avec BdFredi :

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



| General | Definition | Code | Options | Parameters | S > |
|---------|------------|------|---------|------------|-------|
| Name | nbkmdem | | | | X |
| Owner | postgres | | | | X ▾ |
| Schema | BdFredi | | | | ▾ |
| Comment | | | | | |

{ nbkmdem(nomdem text)

General Definition Code Options Parameters S >

Return type **integer**

Language **plpgsql**

Arguments

| | Data type | Mode | Argument na... | Default |
|--|-------------|-----------|----------------|---------|
| | text | IN | nomdem | |

{ nbkmdem(nomdem text)

General Definition **Code** Options Parameters Security SQL

```

1 ▼ DECLARE
2   NbKm INT;
3 ▼ BEGIN
4     Select SUM(lignes_frais.km) INTO NbKm FROM lignes_frais
5       INNER JOIN demandeurs ON demandeurs.adresse_mail = lignes_frais.adresse_mail
6       where demandeurs.nom = nomdem;
7       RAISE NOTICE 'Nombre de km : % Nom : %', NbKm, nomdem;
8   return NbKm;
9   END;

```

Close Reset Save

Query Query History

```

1 set schema 'BdFredi';
2 select nbkmdem('Berbier')

```

Data Output Messages Notifications

| | nbkmdem | integer |
|---|----------------|----------------|
| 1 | 1930 | |

The screenshot shows a PostgreSQL query interface. At the top, there are tabs for "Query" and "Query History". Below the tabs, a code editor contains the following SQL code:

```
1 set schema 'BdFredi';
2 select nbkmdem('Becker')
```

Below the code editor is a "Data Output" tab, which is currently selected. It displays the results of the query in a table format:

| | nbkmdem | integer |
|---|---------|---------|
| 1 | 600 | |

2. 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.

The screenshot shows a database management interface for creating a function. The title bar says "coutkm(mail text, date date)". Below the title bar, there are tabs for "General", "Definition", "Code", "Options", "Parameters", and "Security". The "General" tab is selected. The configuration fields are as follows:

- Name: coutkm
- Owner: postgres
- Schema: BdFredi
- Comment: (empty)

`{(coutkm(mail text, date date)`

General Definition Code Options Parameters Security >

Return type integer

Language plpgsql X | ▾

Arguments

| Data type | Mode | Argument name | Default |
|-----------|------|---------------|---------|
| text | IN | mail | |
| date | IN | date | |

i ? X Close Reset Save

`{(coutkm(mail text, date date)`

General Definition Code Options Parameters Security >

```

1 ✓ DECLARE
2   couttotal INT;
3 ✓ BEGIN
4     SELECT SUM(km)*0.28 INTO couttotal FROM lignes_frais
5       where adresse_mail = mail AND
6         date_frais = date;
7     RAISE NOTICE 'Mail : % cout : %',mail, couttotal;
8   return couttotal;
9 END;

```

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

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 :

- Créer et tester la fonction **nbsesform** 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 >

| | | |
|---------|-----------|--|
| Name | nbsesform | |
| Owner | postgres | |
| Schema | BdForma | |
| Comment | | |

Create - Function

General Definition Code Options Parameters Security >

| Custom return type? | <input checked="" type="checkbox"/> | | | | | | | | | | |
|--|-------------------------------------|------|---------------|---------|---------------|---------|--|------|----|------------|--|
| Return type | integer | | | | | | | | | | |
| Language | plpgsql | | | | | | | | | | |
| Arguments | | | | | | | | | | | |
| <table border="1"> <thead> <tr> <th></th> <th>Data type</th> <th>Mode</th> <th>Argument name</th> <th>Default</th> </tr> </thead> <tbody> <tr> <td></td> <td>text</td> <td>IN</td> <td>nomdomaine</td> <td></td> </tr> </tbody> </table> | | | Data type | Mode | Argument name | Default | | text | IN | nomdomaine | |
| | Data type | Mode | Argument name | Default | | | | | | | |
| | text | IN | nomdomaine | | | | | | | | |

{= nbsesform(nomdomaine text)}

General Definition Code Options Parameters Security SQL

```

1 ✓ DECLARE
2   nbsession INT;
3 ✓ BEGIN
4     select count(*) INTO nbsession from session
5     natural join formation
6     natural join domaine
7     where domaine.nom = nomdomaine;
8     RAISE NOTICE 'Nom domaine : % nombre session : %',nomdomaine ,nbsession;
9   RETURN nbsession;
10 END;

```

Query Query History

```

1 set schema 'BdForma';
2 select nbsesform('Gestion')

```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

| | nbsesform | integer |
|---|-----------|---------|
| 1 | | 8 |

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 Par

Name

Owner nostares

Create - Function

General Definition Code Options Parameters Security SQL

Custom return type?

Return type: integer

Language: plpgsql

Arguments

| | Data type | Mode | Argument name | Default |
|---|-----------|------|---------------|---------|
| 1 | text | IN | nomformation | |
| 2 | integer | IN | numsession | |

Create - Function

General Definition Code Options Parameters Security SQL

```

1 ✓ DECLARE
2   caformation INT;
3 ✓ BEGIN
4     select ((nbplacesmax - nbplacesrestantes)*formation.cout) INTO caformation from session
5       natural join formation
6       where formation.titre = nomformation AND session.idsession = numsession;
7       RAISE NOTICE 'Formation : % session : % CA : %', nomformation, numsession, caformation;
8   RETURN caformation;
9 END;

```

Buttons:

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

Query Query History

```

1 set schema 'BdForma';
2 select caform('Comptabilité',1)

```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: **SQL query of data**

| | caform | 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 **BdFredi**
- 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 pgAdmin interface for creating a function named 'verifMail()'. The code in the editor is:

```
-- Function: "verifMail"()
-- DROP FUNCTION "verifMail"();

CREATE OR REPLACE FUNCTION "verifMail"()
RETURNS trigger AS
$BODY$DECLARE
BEGIN
if (NEW.adresse_mail not like '%@%') then
RAISE EXCEPTION 'email erroné';
END IF;
Return new ;
$END$BODY$
LANGUAGE plpgsql VOLATILE;
```

The 'General' tab of the configuration dialog is selected, displaying the following details:

| Name | verifMail |
|---------|-----------|
| Owner | postgres |
| Schema | BdForma |
| Comment | (empty) |

At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

verifMail()

General Definition **Code** Options Parameters Sec >

```

1 ✓ DECLARE
2     BEGIN
3         if (NEW.adresse_mail not like '%@%') then
4             RAISE EXCEPTION 'email erroné';
5         END if;
6         return new;
7     END;

```

i **?** **X Close** **↻ Reset** **Save**

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

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

```
-- Trigger: verifDemandeur on demandeurs
-- DROP TRIGGER "verifDemandeur" ON demandeurs;
CREATE TRIGGER "verifDemandeur"
BEFORE INSERT
ON demandeurs
FOR EACH ROW
EXECUTE PROCEDURE "verifMail"();
```

verifDemandeur

General Definition Events Transition Code SQL

| | |
|---------|----------------|
| Name | verifDemandeur |
| Comment | |

i **?** **X Close** **↻ Reset** **Save**

→ verifDemandeur

General Definition Events Transition Code SQL

Fires BEFORE

Events

INSERT

UPDATE

DELETE

TRUNCATE

When 1

Columns

i ? X Close Reset Save

→ Create - Trigger

General Definition Events Transition Code SQL

```
1 CREATE TRIGGER "verifDemandeur"
2   BEFORE INSERT
3   ON "BdFredi".demandeurs
4   FOR EACH ROW
5   EXECUTE FUNCTION "BdFredi"."verifMail"();
```

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

```
-- Trigger: verifStagiaire on stagiaire
-- DROP TRIGGER "verifStagiaire" ON stagiaire;

CREATE TRIGGER "verifStagiaire"
BEFORE INSERT
ON stagiaire
FOR EACH ROW
EXECUTE PROCEDURE "verifMail"();
```

The screenshot displays four stacked 'Create - Trigger' dialog boxes, each with a different tab selected:

- Top Dialog (General Tab):** Shows 'Name' set to 'verifStagiaire' and an empty 'Comment' field.
- Second Dialog (Definition Tab):** Shows 'Row trigger?' as **ON**, 'Constraint trigger?' as **OFF**, 'Deferrable?' as **OFF**, and 'Deferred?' as **OFF**. The 'Trigger function' field contains "'BdForma"."verifMail'".
- Third Dialog (Events Tab):** Shows 'Fires' set to **BEFORE** and 'Events' set to **INSERT**.
- Bottom Dialog (SQL Tab):** Displays the generated SQL code:

```

1 CREATE TRIGGER "verifStagiaire"
2   BEFORE INSERT
3   ON "BdForma".stagiaire
4   FOR EACH ROW
5   EXECUTE FUNCTION "BdForma"."verifMail"();
    
```

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

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

[Query](#) [Query History](#)

```

1 set schema 'BdFredi';
2 v INSERT INTO demandeurs(
3     adresse_mail, nom, prenom, rue, cp, ville, num_recu, motdepasse)
4     VALUES ('etudiant@gmail.com', 'Becker', 'Romain', '1 rue des mesanges', '54600', 'Villers-Les-Nancy', 0 , 'rb');

```

[Data Output](#) [Messages](#) [Notifications](#)

ERROR: email erroné
CONTEXT: PL/pgSQL function "verifMail"() line 4 at RAISE

SQL state: P0001

[Insertion d'une donnée sans le '@'](#)[Query](#) [Query History](#)

```

1 set schema 'BdFredi';
2 v INSERT INTO demandeurs(
3     adresse_mail, nom, prenom, rue, cp, ville, num_recu, motdepasse)
4     VALUES ('etudiant@gmail.com', 'Becker', 'Romain', '1 rue des mesanges', '54600', 'Villers-Les-Nancy', 0 , 'rb');

```

[Data Output](#) [Messages](#) [Notifications](#)

INSERT 0 1

Query returned successfully in 78 msec.

[Insertion d'une donnée avec le '@'](#)

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

[Query](#) [Query History](#)

```

1 set schema 'BdForma';
2 v INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION,ADRESSE_MAIL, NBFORMATIONS) VALUES
3 (7, 100, 'POULIDOR', 'Raymond', 'Benevole', 'Développeur','etudiant@gmail.com ',0);

```

[Data Output](#) [Messages](#) [Notifications](#)

ERROR: email erroné
CONTEXT: PL/pgSQL function "verifMail"() line 4 at RAISE

SQL state: P0001

[Insertion d'une donnée sans le '@'](#)[Query](#) [Query History](#)

```

1 set schema 'BdForma';
2 v INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION,ADRESSE_MAIL, NBFORMATIONS) VALUES
3 (7, 100, 'POULIDOR', 'Raymond', 'Benevole', 'Développeur','etudiant@gmail.com ',0);

```

[Data Output](#) [Messages](#) [Notifications](#)

INSERT 0 1

Query returned successfully in 86 msec.

[Insertion d'une donnée avec le '@'](#)

Etape 2 - Création d'un trigger

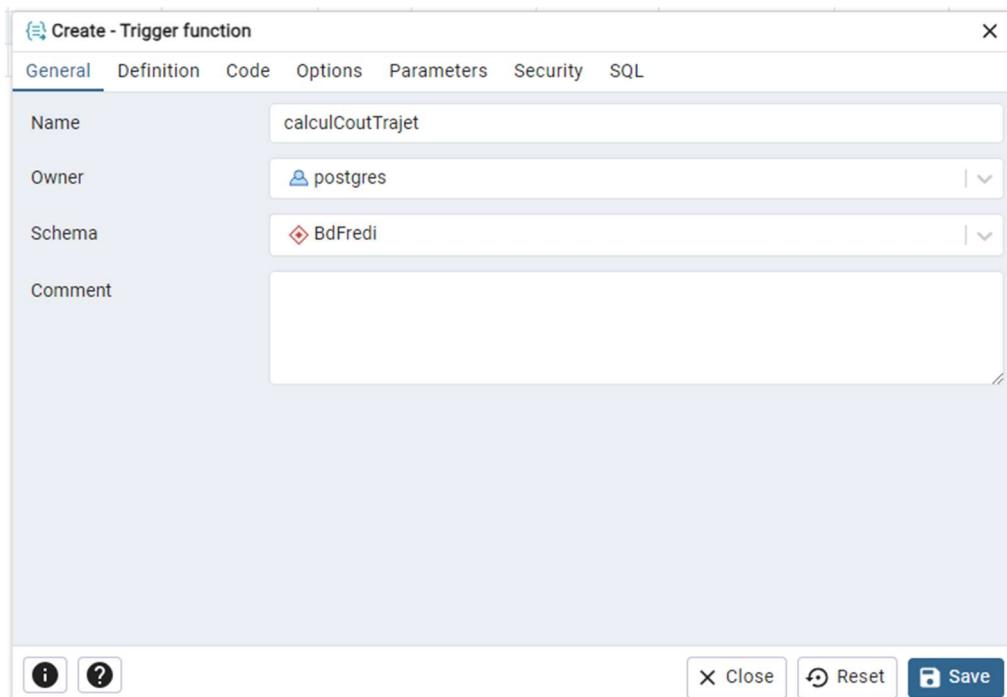
Vous pouvez créer les triggers suivants des 2 façons suivantes :

soit avec pgadmin en local soit avec phppgadmin à distance

A faire avec BdFredi :

- Créer le trigger **afficheCoutTrajet** qui affichera le cout du trajet après une insertion d'une ligne de frais par un demandeur.
- Tester par insertion d'occurrences dans la table **ligne_frais** en utilisant le fichier de test **TestTriggerCoutTrajet** fourni.

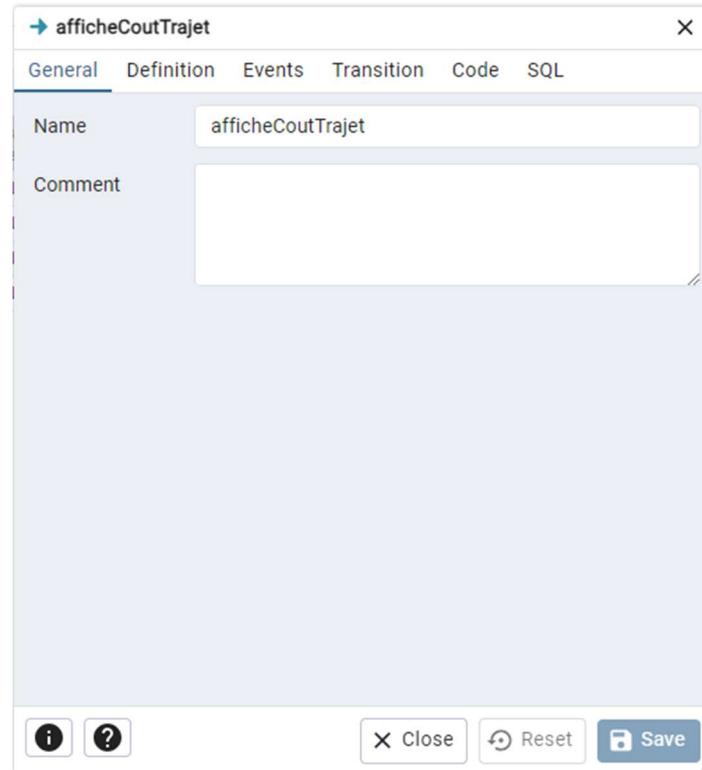
Creation de la fonction trigger calculCoutTrajet()



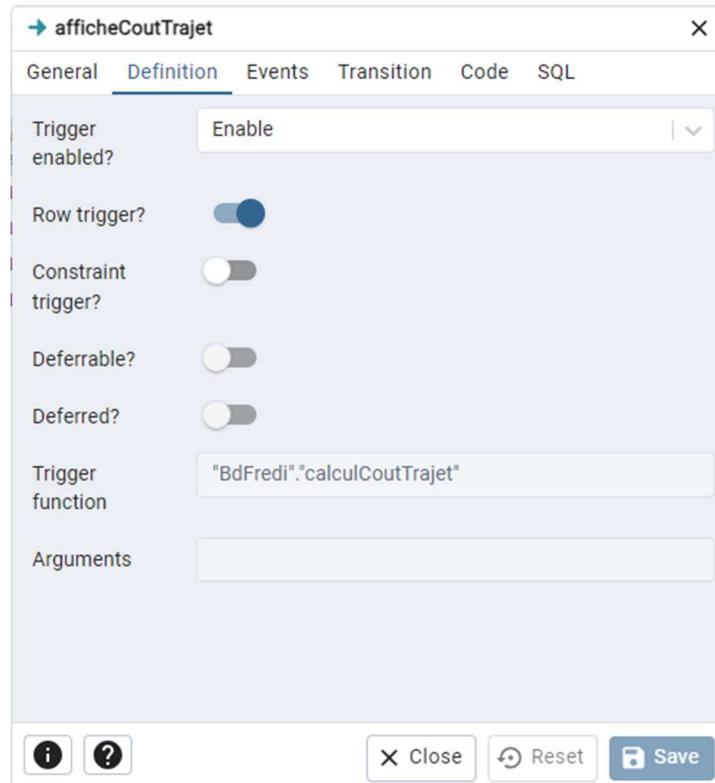
Code de la fonction trigger

```
>Create - Trigger function
General Definition Code Options Parameters Security SQL
1 ✓ DECLARE
2     BEGIN
3         return "Coût total du trajet"+(NEW.cout_peage + New.cout_repas + New.cout_hebergement);
4     END;
```

Création du trigger afficheCoutTrajet()



Définition du trigger sur la fonction trigger calculCoutTrajet



→ afficheCoutTrajet X

General Definition Events Transition Code SQL

Fires AFTER

Events

INSERT

UPDATE

→ Create - Trigger X

General Definition Events Transition Code SQL

```
1 CREATE TRIGGER "afficheCoutTrajet"
2     AFTER INSERT
3     ON "BdFredi".lignes_frais
4     FOR EACH ROW
5     EXECUTE FUNCTION "BdFredi"."calculCoutTrajet"();
```

i ? X Close ↻ Reset Save

Teste d'insertion d'une ligne dans la table lignes_frais

The screenshot shows a PostgreSQL client interface with two tabs: 'TX_BdBtssio/postgres@postgresSQL*' and 'BdFredi.lignes_frais...'. The query tab contains the following SQL code:

```

1 SET SCHEMA 'BdFredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '12/02/2025', 'Competition', 'Montauban-Monteils', 40, 60, 10, 50, 0, 0, 0, 0);

```

The results tab shows the output of the query:

```

NOTICE: Coût total : 120
INSERT 0 1

Query returned successfully in 78 msec.

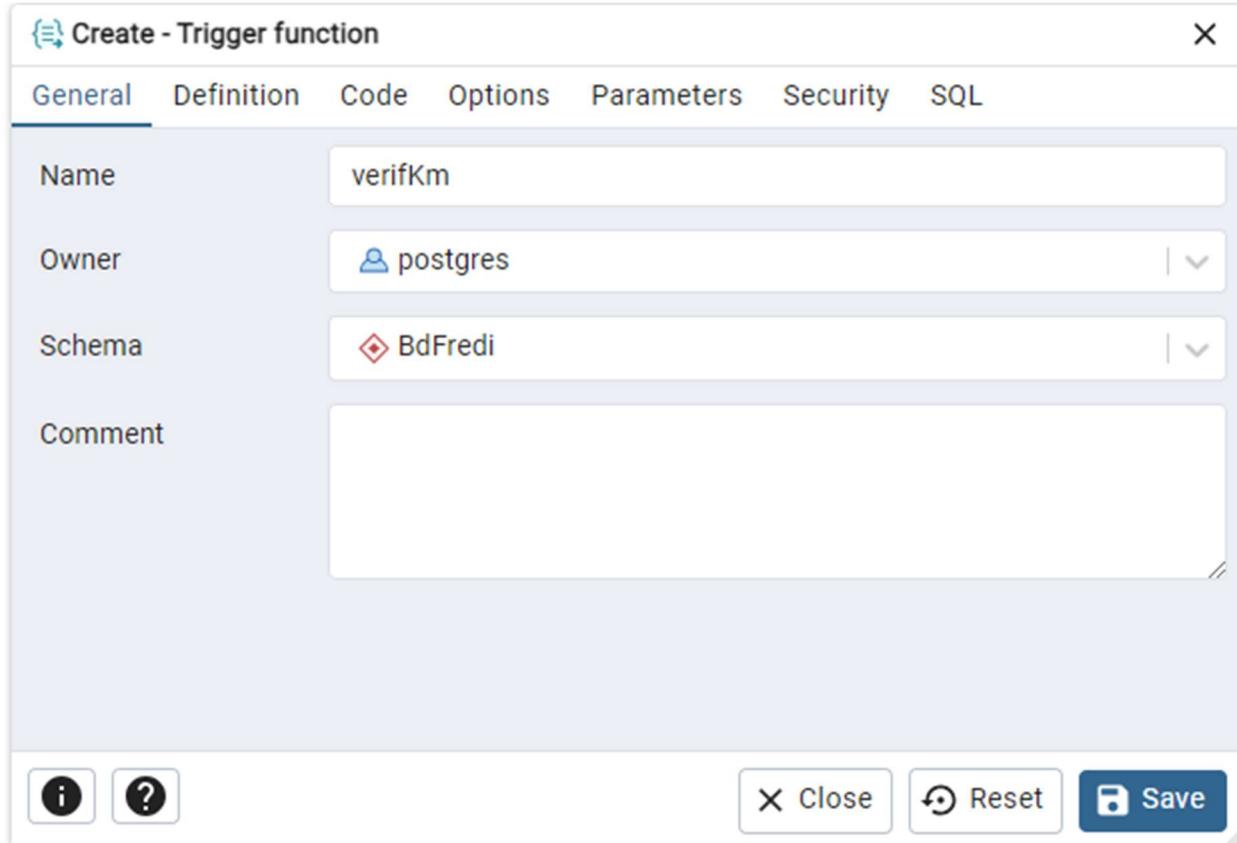
```

Affiche du cout total du trajet

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

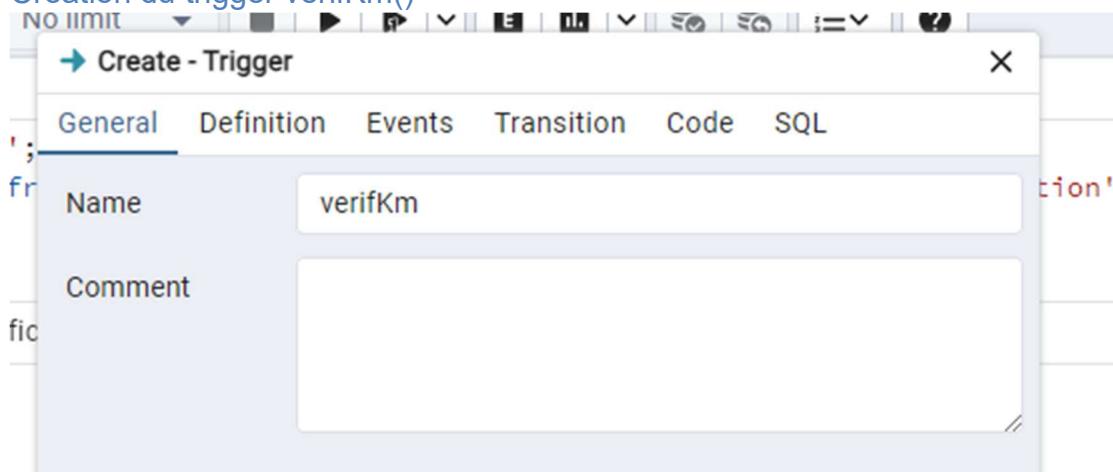
Création de la fonction trigger verifKm()



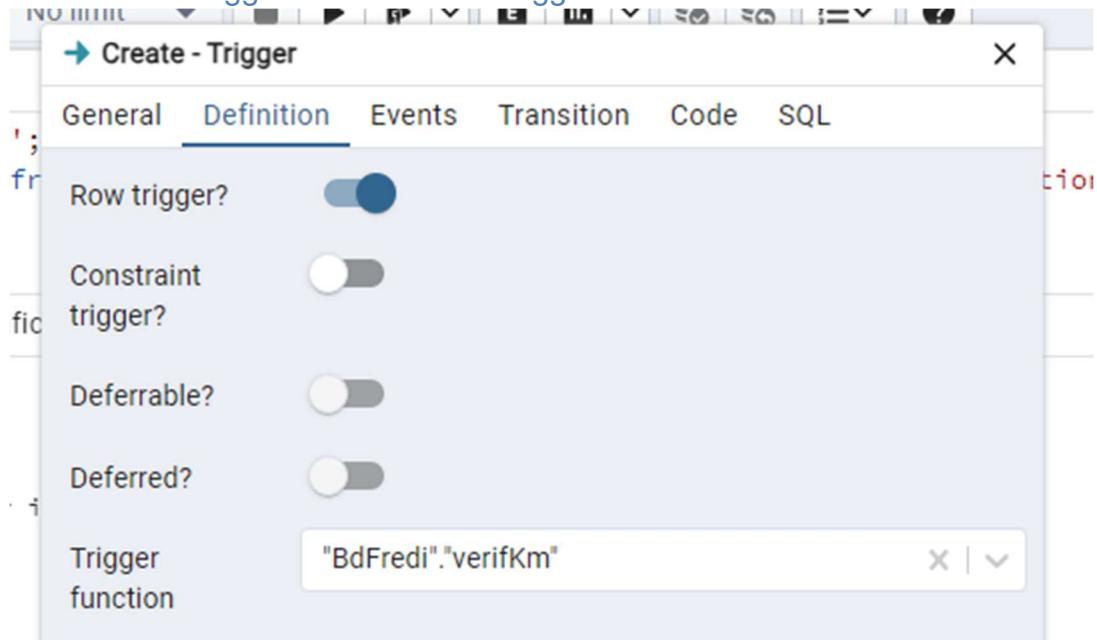
Code de la fonction trigger

```
verifKm()
General Definition Code Options Parameters Security SQL
1 1 ✓ DECLARE
2      BEGIN
3          IF (NEW.cout_peage > 0) AND (NEW.km != 0) then
4              RAISE EXCEPTION 'km invalide';
5          END IF;
6          RETURN NEW;
7      END;
```

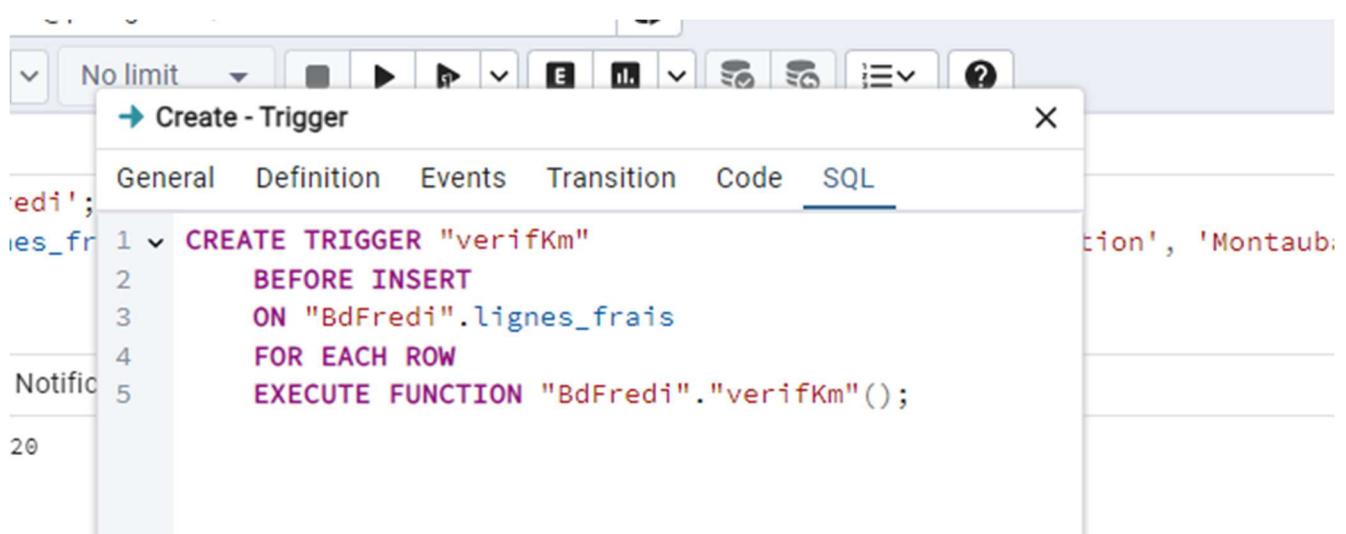
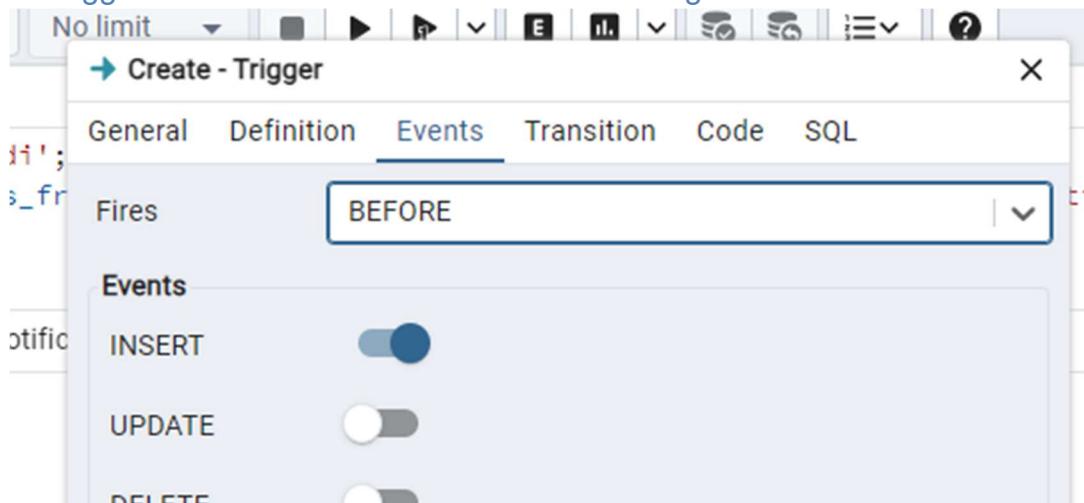
Création du trigger verifKm()



Définition du trigger sur la fonction trigger verifKm



Le trigger se déclenche avant l'insertion d'une ligne



Test de la fonction avec un km = 0 et cout_peage = 60

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '02/13/2025' , 'Competition', 'Montauban-Monteils', 0, 60, 10, 50, 0, 0, 0, 0);
```

Data Output Messages Notifications

ERROR: km invalide
CONTEXT: PL/pgSQL function "verifKm"() line 4 at RAISE

SQL state: P0001

Test de la fonction avec un km = 50 et cout_peage = 60

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '02/14/2025' , 'Competition', 'Montauban-Monteils', 50, 60, 10, 50, 0, 0, 0, 0);
```

Data Output Messages Notifications

NOTICE: Coût total : 120
INSERT 0 1

Query returned successfully in 75 msec.

Test de la fonction avec un km = 0 et cout_peage = 0

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '02/15/2025' , 'Competition', 'Montauban-Monteils', 0, 0, 10, 50, 0, 0, 0, 0);
```

Data Output Messages Notifications

NOTICE: Coût total : 60
INSERT 0 1

Query returned successfully in 81 msec.

Test de la fonction avec un km = 50 et cout_peage = 0

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '02/16/2025' , 'Competition', 'Montauban-Monteils', 50, 0, 10, 50, 0, 0, 0, 0);
```

Data Output Messages Notifications

NOTICE: Coût total : 60
INSERT 0 1

Query returned successfully in 114 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. Si l'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

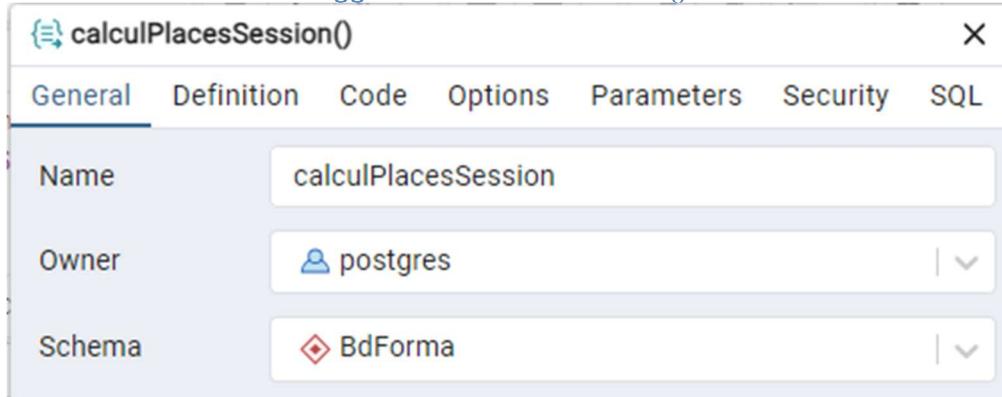
dans la variable **adh** les données correspondantes comme par exemple **adh.nom**, **adh.prenom**,...

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

Création de la fonction trigger calculPlacesSession()



Code de fonction trigger

`calculPlacesSession()`

General Definition **Code** Options Parameters Security SQL

```

1 ✓ DECLARE
2     BEGIN
3         IF (NEW.nbplacesrestantes > NEW.nbplacesmax) THEN
4             RAISE EXCEPTION 'nombre de places restantes invalide';
5         END IF;
6         RETURN NEW;
7     END;

```

Save

Création du trigger verifPlaces()

Create - Trigger

General Definition Events Transition Code SQL

Name: verifPlaces

Comment:

Définition du trigger sur la fonction trigger calculPlacesSession

Create - Trigger

General Definition Events Transition Code SQL

Row trigger?

Constraint trigger?

Deferrable?

Deferred?

Trigger function: "BdForma"."calculPlacesSession"

Arguments:

Save

Déclenchement du trigger après insertion

Create - Trigger

- Fires: BEFORE
- Events:
 - INSERT:
 - UPDATE:

Create - Trigger

- General Definition Events Transition Code SQL

```

1 CREATE TRIGGER "verifPlaces"
2   BEFORE INSERT
3   ON "BdForma".session
4   FOR EACH ROW
5   EXECUTE FUNCTION "BdForma"."calculPlacesSession"();

```

Test d'insertion d'une ligne sur la table session avec un nbplacesrestantes supérieure à nbplacesmax

Query Query History

```

1 SET SCHEMA 'BdForma';
2 INSERT INTO session VALUES (6 ,11, 'lundi', '02/13/2025', 'A Condator', '02/03/2025', 30, 25);

```

Data Output Messages Notifications

ERROR: nombre de places restantes invalide
CONTEXT: PL/pgSQL function "calculPlacesSession"() line 4 at RAISE

SQL state: P0001

Test d'insertion d'une ligne sur la table avec un nbplacesrestantes inférieure à nbplacesmax

Query Query History

```

1 SET SCHEMA 'BdForma';
2 INSERT INTO session VALUES (6 ,11, 'lundi', '02/13/2025', 'A Condator', '02/03/2025', 25, 30);

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 76 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.

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

[Création de la fonction trigger calculPlacesRest\(\)](#)

calculPlacesRest()

| General | Definition | Code | Options | Parameters | Security | SQL |
|---------|------------------|------|---------|------------|----------|-----|
| Name | calculPlacesRest | | | | | |
| Owner | postgres | | | | | |
| Schema | BdForma | | | | | |
| Comment | | | | | | |

[Code de la fonction trigger](#)

```

1 ✓ DECLARE
2   nbplaces INT;
3 ✓ BEGIN
4   SELECT session.nbplacesrestantes INTO nbplaces from session
5   WHERE session.idsession = new.idsession AND session.idformation = new.idformation;
6 ✓ IF (nbplaces > 0 ) then
7     RAISE NOTICE 'nombre de place(s) restante : %',nbplaces;
8 ✓ ELSE
9     RAISE EXCEPTION 'nombre de place indisponible';
10    END IF;
11    return new;
12 END;

```

(Modification : ajouter un update pour decremener le nombre de places restante dans la table session après une insertion)

[Création du trigger affichePlacesRest\(\)](#)

→ AffichePlacesRest

| General | Definition | Events | Transition | Code | SQL |
|---------|-------------------|--------|------------|------|-----|
| Name | AffichePlacesRest | | | | |
| Comment | | | | | |

Définition du trigger sur la fonction trigger calculPlacesRest()

→ affichePlacesRest

| General | Definition | Events | Transition | Code | SQL |
|---------------------|-------------------------------------|--------|------------|------|-----|
| Trigger enabled? | Enable | | | | |
| Row trigger? | <input checked="" type="checkbox"/> | | | | |
| Constraint trigger? | <input type="checkbox"/> | | | | |
| Deferrable? | <input type="checkbox"/> | | | | |
| Deferred? | <input type="checkbox"/> | | | | |
| Trigger function | 'BdForma"."calculPlacesRest" | | | | |

Déclenchement du trigger avant l'insertion

→ affichePlacesRest

| General | Definition | Events | Transition | Code | SQL |
|---------|-------------------------------------|--------|------------|------|-----|
| Fires | BEFORE | | | | |
| Events | | | | | |
| INSFRT | <input checked="" type="checkbox"/> | | | | |

Test d'insertion d'une ligne sur la table avec un inscrit pour 10 places restantes

Query Query History

```

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

```

Data Output Messages Notifications

```

NOTICE: nombre de place(s) restante : 10
INSERT 0 1

Query returned successfully in 76 msec.

```

| | | | | | | | | |
|---|---|----|-------|------------|----------|------------|----|----|
| 3 | 1 | 21 | Mardi | 2015-01-20 | C Froome | 2015-01-05 | 10 | 10 |
|---|---|----|-------|------------|----------|------------|----|----|

Test d'insertion d'une ligne sur la table avec un inscrit pour aucune place restante

Query Query History

```

1 SET SCHEMA 'BdForma';
2 INSERT INTO inscrit VALUES (5 ,12 ,7);

```

Data Output Messages Notifications

```

ERROR: nombre de place indisponible
CONTEXT: PL/pgSQL function "calculPlacesRest"() line 9 at RAISE
SQL state: P0001

```

| | | | | | | | | |
|----|---|----|----------|------------|----------|------------|---|----|
| 13 | 7 | 12 | Mercredi | 2025-10-04 | C Froome | 2025-10-18 | 0 | 20 |
|----|---|----|----------|------------|----------|------------|---|----|

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

Création de la fonction trigger **compteNbForm()**

compteNbForm()

General Definition Code Options Parameters Security SQL

| | |
|---------|--------------|
| Name | compteNbForm |
| Owner | postgres |
| Schema | BdForma |
| Comment | |

Code de la fonction trigger

compteNbForm()

General Definition Code Options Parameters Security SQL

```

1 ✓ DECLARE
2   nbform INT;
3   nbinscrit INT;
4 ✓ BEGIN
5     SELECT stagiaire.nbformations INTO nbform FROM stagiaire where NEW.idstagiaire = idstagiaire;
6     SELECT COUNT(*) INTO nbinscrit FROM inscrit where NEW.idstagiaire = idstagiaire;
7 ✓ IF (nbinscrit >= nbform) THEN
8     RAISE EXCEPTION 'le stagiaire a atteint la limite du nombre d inscription';
9   END IF;
10  return NEW;
11 END;

```

Création du trigger verifNbForm()

verifNbForm

General Definition Events Transition Code SQL

| | |
|---------|-------------|
| Name | verifNbForm |
| Comment | |

Définition du trigger sur la fonction trigger compteNbForm

→ verifNbForm

General Definition Events Transition Code SQL

Trigger enabled? Enable

Row trigger?

Constraint trigger?

Deferrable?

Deferred?

Trigger function "BdForma"."compteNbForm"

Arguments

Déclenchement avant une insertion

→ verifNbForm

General Definition Events Transition Code SQL

Fires BEFORE

Events

INSERT

→ Create - Trigger X

General Definition Events Transition Code SQL

```
1 ✓ CREATE TRIGGER "verifNbForm"
2     BEFORE INSERT
3     ON "BdForma".inscrit
4     FOR EACH ROW
5     EXECUTE FUNCTION "BdForma"."compteNbForm"();
```

Test d'insertion d'une ligne d'inscription pour le stagiaire 1

Query Query History

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

Data Output Messages Notifications

ERROR: nombre de place indisponible
 CONTEXT: PL/pgSQL function "calculPlacesRest"() line 9 at RAISE
 SQL state: P0001

Le nombre limite d'inscription pour le stagiaire 1 est de 1

Query Query History

```
1 SELECT idstagiaire, nbformations FROM "BdForma".stagiaire WHERE idstagiaire = 1
```

Data Output Messages Notifications

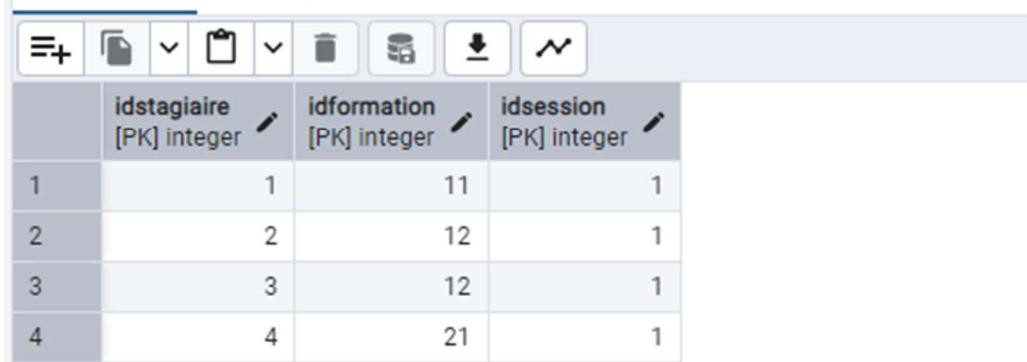


| | idstagiaire [PK] integer | nbformations integer |
|---|-----------------------------|-------------------------|
| 1 | 1 | 1 |

Query Query History

```
1 ▾ SELECT * FROM "BdForma".inscrit
2 ORDER BY idstagiaire ASC, idformation ASC, idsession ASC
```

Data Output Messages Notifications



| | idstagiaire [PK] integer | idformation [PK] integer | idsession [PK] integer |
|---|-----------------------------|-----------------------------|---------------------------|
| 1 | 1 | 11 | 1 |
| 2 | 2 | 12 | 1 |
| 3 | 3 | 12 | 1 |
| 4 | 4 | 21 | 1 |

Test d'insertion d'une ligne d'inscription pour le stagiaire 3

Query Query History

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

Data Output Messages Notifications

```
NOTICE: nombre de place(s) restante : 80
INSERT 0 1

Query returned successfully in 144 msec.
```

Le nombre limite d'inscription pour le stagiaire 3 est de 2

Query Query History

```
1 SET SCHEMA 'BdForma';
2 select idstagiaire, nbformations from "BdForma".stagiaire where idstagiaire = 3
```

Data Output Messages Notifications

| | idstagiaire [PK] integer | nbformations integer |
|---|-----------------------------|-------------------------|
| 1 | 3 | 2 |

Query Query History

```
1 ▾ SELECT * FROM "BdForma".inscrit
2 ORDER BY idstagiaire ASC, idformation ASC, idsession ASC
```

Data Output Messages Notifications

| | idstagiaire [PK] integer | idformation [PK] integer | idsession [PK] integer |
|---|-----------------------------|-----------------------------|---------------------------|
| 1 | 1 | 11 | 1 |
| 2 | 2 | 12 | 1 |
| 3 | 3 | 12 | 1 |
| 4 | 3 | 12 | 3 |
| 5 | 4 | 21 | 1 |

- Créer le trigger **verifDateIns** qui vérifiera avant l'insertion d'une inscription que le stagiaire ne dépasse pas la date limite d'inscription.
- Tester par insertion d'occurrences dans la table **inscrit** en utilisant le fichier de test **TestTriggerVerifDatefourni**.

Création de la fonction trigger verifDateInscription

The screenshot shows a database interface for creating a function trigger. The top window is titled 'verifdateInscription()' and has tabs for General, Definition, Code, Options, Parameters, Security, and SQL. The 'General' tab is selected, showing the function name 'verifdateInscription()', owner 'postgres', and schema 'BdForma'. Below this, the 'Code' tab is selected, showing the PL/pgSQL code for the trigger:

```

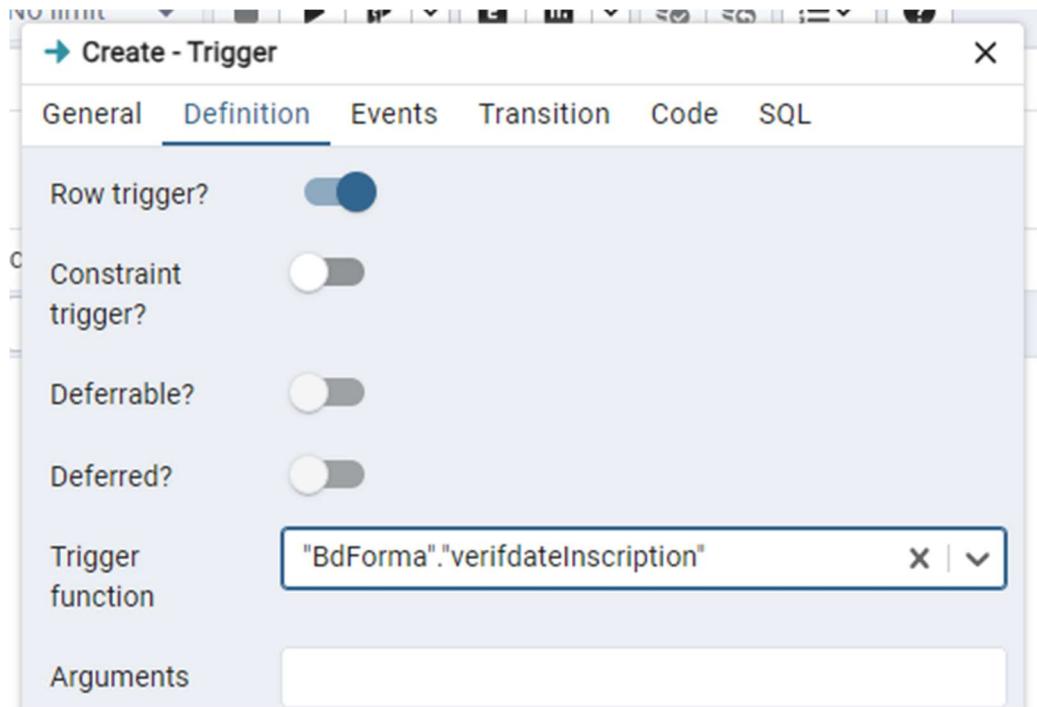
1  DECLARE
2      datelimit DATE;
3  BEGIN
4      SELECT datelimitinscription INTO datelimit FROM session
5      where New.idformation = idformation AND NEW.idsession = idsession;
6      IF (CURRENT_DATE > datelimit) THEN
7          RAISE EXCEPTION 'date limite dépassé';
8      END IF;
9      return NEW;
10 END;

```

Création du trigger verifDateIns()

The screenshot shows a database interface for creating a trigger. The top window is titled 'Create - Trigger' and has tabs for General, Definition, Events, Transition, Code, and SQL. The 'General' tab is selected, showing the trigger name 'verifDateIns()'. The 'Comment' field is empty.

Définition du trigger sur la fonction trigger verifDateInscription()



Déclenchement du trigger avant l'insertion

The screenshot shows the 'Create - Trigger' dialog with the 'Events' tab selected. It is set to fire 'BEFORE' on 'INSERT'. Below it, another 'Create - Trigger' dialog shows the generated SQL code:

```

1 CREATE TRIGGER "verifDateIns"
2   BEFORE INSERT
3   ON "BdForma".inscrit
4   FOR EACH ROW
5   EXECUTE FUNCTION "BdForma"."verifdateInscription"();
  
```

Test d'insertion d'une ligne dans inscrit à la date d'aujourd'hui avec une date limite dépassé

[Query](#) [Query History](#)

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

[Data Output](#) [Messages](#) [Notifications](#)

ERROR: date limite dépassé
CONTEXT: PL/pgSQL function "verifDateInscription"() line 6 at RAISE
SQL state: P0001

Test d'insertion d'une ligne dans inscrit à la date d'aujourd'hui avec une date limite non dépassé

[Query](#) [Query History](#)

```
1 SET SCHEMA 'BdForma';
2 INSERT INTO inscrit VALUES (4 ,11 ,7);
```

[Data Output](#) [Messages](#) [Notifications](#)

INSERT 0 1

Query returned successfully in 85 msec.

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;
```

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there are tabs for 'Query' and 'Query History'. Below them, the code for the rule is displayed:

```
5 | SELECT 'Ajout du club' || new.nom_club
6 | AS "Insertion club"
7 | FROM clubs
8 | where clubs.num_club = new.num_club;
```

Below the code, there are tabs for 'Data Output', 'Messages', and 'Notifications'. Under 'Data Output', the message 'CREATE RULE' is shown. At the bottom, it says 'Query returned successfully in 92 msec.'

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

The screenshot shows two windows from Oracle SQL Developer. On the left is the 'Éditeur SQL' window, which contains the following SQL command:

```
INSERT into clubs values ( 5,1,'club5');
```

On the right is the 'Panneau sortie' (Output Panel) window, which displays the results of the query:

| Sortie de données | |
|-------------------|------------------------|
| | Insertion club text |
| 1 | Ajout du club club5 |

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT into clubs values (5,1,'club5');
```

Data Output Messages Notifications

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 :

Éditeur SQL Constructeur graphique de requêtes

Requêtes précédentes

```
INSERT into clubs values ( 7,1,'Judo Club de Nancy');
```

Sortie de données Expliquer (Explain) Messages Historique

Insertion club
text

| | |
|---|--|
| 1 | Ajout du club Judo Club de Nancy ligue UFOLEP Lorraine |
|---|--|

insertClub

General Definition Condition Commands SQL

```
1 SELECT (((('Ajout du club '::text || new.nom_club) || ' ligue '::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)
```

Query Query History

```
1 SET SCHEMA 'BdFredi';
2 INSERT into clubs values (7,1,'Judo Club de Nancy');
```

Data Output Messages Notifications

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;
```

The screenshot shows the pgAdmin interface with the following details:

- Query History:** Shows the SQL code for creating the rule.
- Data Output:** Shows the message: "CREATE RULE".
- Messages:** Shows the message: "Query returned successfully in 106 msec."

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

The screenshot shows the PostgreSQL SQL editor with the following details:

- Éditeur SQL:** Contains the SQL query: `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:** Shows the output of the insert operation.

| | Insertion stagiaire |
|------|-----------------------------|
| text | |
| 1 | Ajout du stagiaire AMSTRONG |

The screenshot shows the pgAdmin interface with the following details:

- Query History:** Shows the SQL code for creating the rule and inserting data.
- Data Output:** Shows the message: "CREATE RULE".
- Messages:** Shows the message: "Query returned successfully in 106 msec."
- Sortie de données:** Shows the output of the insert operation.

| | 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 :

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 |

insertStagiaire

General Definition Condition Commands SQL

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

Query Query History

```
1 SET SCHEMA 'BdForma';
2 INSERT INTO stagiaire values (8,100,'FIGNON','Laurent','Benevole','Soigneur','fignon@gmail.com',1);
```

Data Output Messages Notifications

Insertion stagiaire
text

| | |
|---|---|
| 1 | Ajout du stagiaire FIGNON de l assosiation Cercle Escrime |
|---|---|

Etape 2 - Crédation d'une règle

A faire sur BdFredi :

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 | | | | |
| 1 | text | 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 |

Query History

```

1 SET SCHEMA 'BdFredi';
2 CREATE OR REPLACE RULE "majLigueClub" AS
3 ON UPDATE TO clubs DO
4     SELECT 'Mise a jour du club '::text || nom_club::text
5     || ' ancienne ligue'::text || clubs.no_ligue
6     || ' nouvelle ligue '::text
7     || new.no_ligue::text
8     AS " Changement de ligue "
9     FROM clubs
10    WHERE clubs.num_club = new.num_club;

```

Data Output Messages Notifications

CREATE RULE

Query returned successfully in 83 msec.

Query History

```
1 UPDATE clubs SET no_ligue = 1 WHERE num_club = 1;
```

Data Output Messages Notifications

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

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

```

SQL
majLigueClub
General Definition Condition Commands SQL
1 ✓ SELECT (((('Mise a jour du club '::text || clubs.nom_club)
2 || 'ancienne ligue ::text) || ligues.nom)
3 || 'nouvelle ligue ::text) || (SELECT ligues_1.nom
4 FROM "BdFredi".ligues ligues_1
5 WHERE (ligues_1.no_ligue = new.no_ligue)) AS "Changement de ligue"
6 FROM ("BdFredi".clubs
7 JOIN "BdFredi".ligues USING (no_ligue))
8 WHERE (clubs.num_club = new.num_club)

Query Query History
1 UPDATE clubs SET no_ligue = 1 WHERE num_club = 1;

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

```

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**

| | 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 character |
|---|-------------------------------------|----------------------------------|------------------------------|---------------------------------|-----------------------------|
| 7 | 7 | 101 | AMSTRONG | Lance | Benevo |

Mise à jour avec la règle majAssoStagiaire

| Sortie de données | | Expliquer (Explain) | Messages | Historique |
|---------------------------------|-------------------------------------|---------------------|---------------------|------------|
| Changement d association | | | | |
| 1 | Mise a jour du stagiaire : AMSTRONG | Ancienne Asso : 101 | Nouvelle asso : 100 | |

Apres la mise à jour

| | idstagiaire [PK] integer | idassociation integer | nom character(50) | prenom character(50) | statut character(25) |
|---|-------------------------------------|----------------------------------|------------------------------|---------------------------------|---------------------------------|
| 7 | 7 | 100 | AMSTRONG | Lance | Benevole |

Query History

```

1 SET SCHEMA 'BdForma';
2 CREATE OR REPLACE RULE "majAssoStagiaire" AS
3 ON UPDATE TO stagiaire DO
4     SELECT 'Mise à jour du stagiaire : '::text || stagiaire.nom::text ||
5         ' Ancienne Asso : '::text || stagiaire.idassociation::text ||
6         ' Nouvelle asso : '::text || new.idassociation::text
7
8     AS " Changement d association "
9     FROM stagiaire
10    WHERE stagiaire.idstagiaire = new.idstagiaire;

```

Data Output Messages Notifications

CREATE RULE

Query returned successfully in 105 msec.

Query History

```
1 UPDATE stagiaire SET idassociation = 100 WHERE idstagiaire = 7;
```

Data Output Messages Notifications



| | | | |
|---|-------------------------------------|---------------------|---------------------|
| | Changement d association | text | lock icon |
| 1 | Mise à jour du stagiaire : AMSTRONG | Ancienne Asso : 101 | Nouvelle asso : 100 |

A faire : Modifier la règle **majAssoStagiaire** sur la table **stagiaire** qui à la place des numéros des associations affichera le **nom des associations** correspondantes comme dans l'exemple ci-dessous :

| Sortie de données | | Expliquer (Explain) | Messages | Historique |
|---------------------------------|-------------------------------------|--------------------------------|---------------------------------|------------|
| Changement d association | | | | |
| 1 | Mise à jour du stagiaire : AMSTRONG | Ancienne Asso : Cercle Escrime | Nouvelle asso : Comite Regional | |

majAssoStagiaire

General Definition Condition Commands SQL

```

1 v  SELECT (((('Mise à jour du stagiaire : '::text ||
2   (stagiaire.nom)::text) ||
3   '      Ancienne Asso : '::text)|||
4   (association.noma)::text) ||
5   '      Nouvelle asso : '::text) ||
6   (( SELECT association_1.noma FROM "BdForma".association association_1
7           WHERE (association_1.idassociation = new.idassociation))::text)
8           AS " Changement d association "
9   FROM ("BdForma".stagiaire
10      JOIN "BdForma".association USING (idassociation))
11 WHERE (stagiaire.idstagiaire = new.idstagiaire)

```

Modification pour le stagiaire 7 à l'association 100 (Escrime)

Query Query History

1 UPDATE stagiaire SET idassociation = 100 WHERE idstagiaire = 7;

Data Output Messages Notifications

Changement d association
text

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

Inverse de la modification pour le stagiaire 7 à l'association 101(Comite Regional)

Query Query History

1 UPDATE "BdForma".stagiaire SET idassociation = 101 WHERE idstagiaire = 7;

Data Output Messages Notifications

Changement d association
text

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