# Module SLAM 3 TP4

| Propriétés | Description |
|---|---|
| **Intitulé** | **PROGRAMMATION AVEC POSTGRESQL** |
| **Outils** | • A.G.L : WIN'DESIGN<br>• SGBD : POSTGRESQL |
| **Durée estimée en heures** | 6 Heures |
| **Savoir-faire module SLAM3** | • Concevoir une base de données<br>• Valider un schéma de base de données<br>• Programmer dans l'environnement de développement associé à un SGBD |
| **Savoirs Module SLAM3** | • Modèles de représentation des données<br>• Langage de programmation associé à un SGBD |
| **Documents joints** | Fiche d'exploitation pédagogique<br>Annexe : Document de présentation de PL/PgSQL, fonctions et triggers Site de référence : https://docs.postgresql.fr/9.6/plpgsql.html |
| **Réception** | Développement de fonctions et de triggers |
| **Equipe** | Seul × Par équipe de ... □ 2□ 3 □ 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 **phppgaadmin**, outil a 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 1ère 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 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)
    - 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….)
    - **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 :

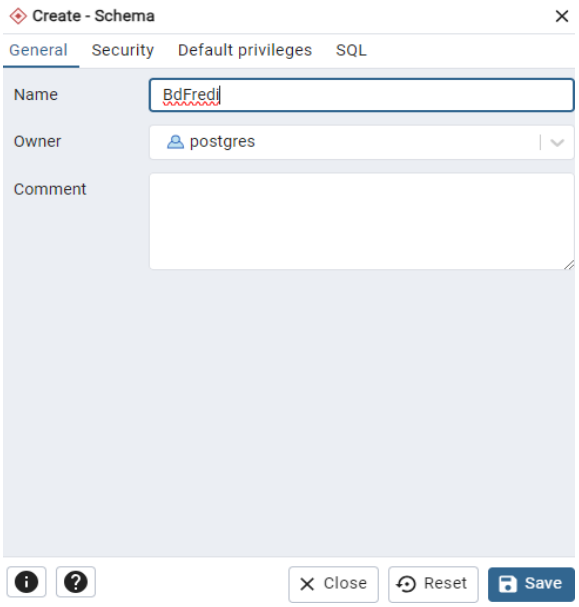
Une fois la connexion effectuée avec PhppgAdmin, il est possible d'installer et de manipuler des bases de données. Vous pouvez choisir d'installer la base de données correspondante ( Fredi ou Forma ).

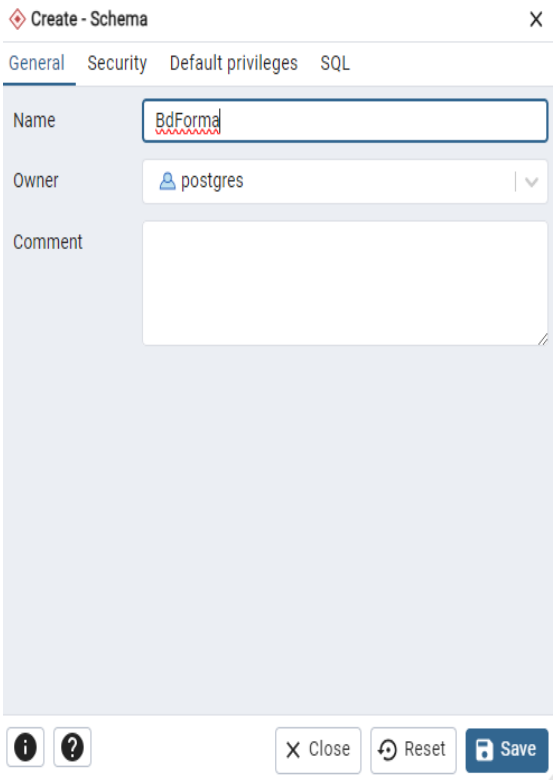
# A faire

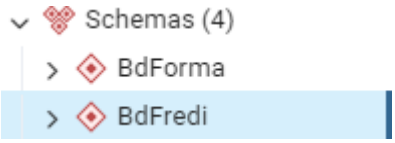
Les manipulations suivantes peuvent se faire à partir de **pgadmin** ou **phppgadmin**

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

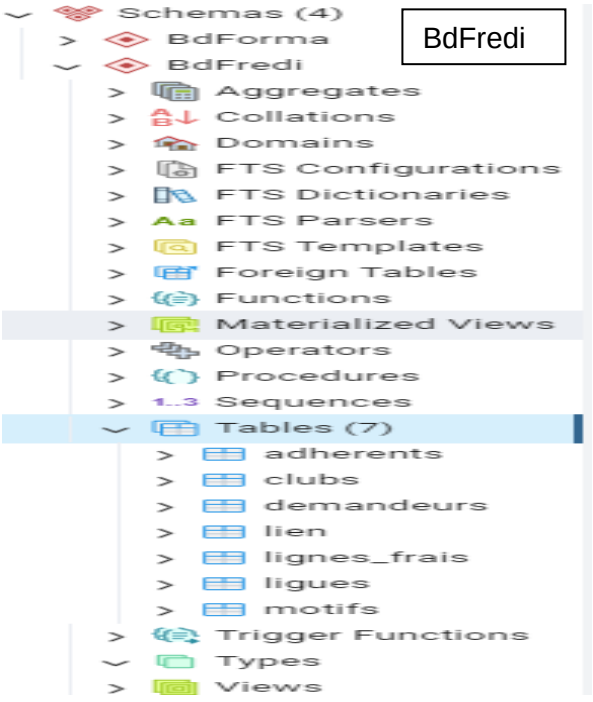
  - Créer la base **BdFredi** ou **BdForma** (codage **UTF8**)

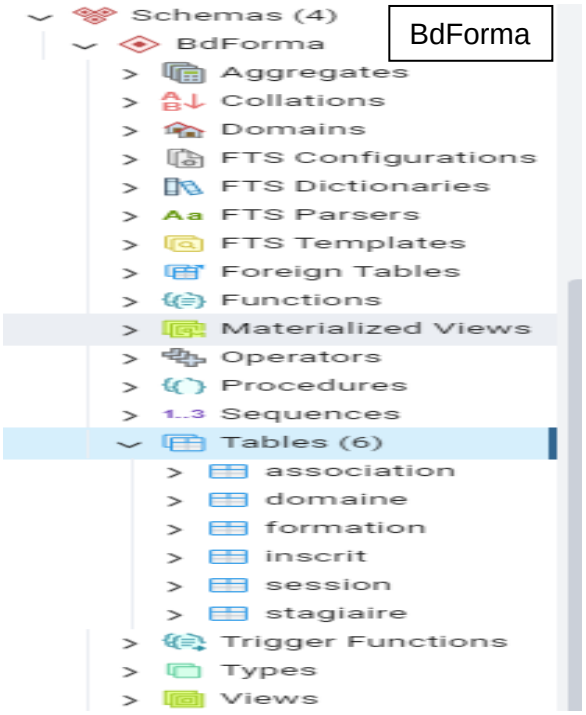



  - Exécuter le script de création **BdpgFredi.sql ou BdpgForma.sql**

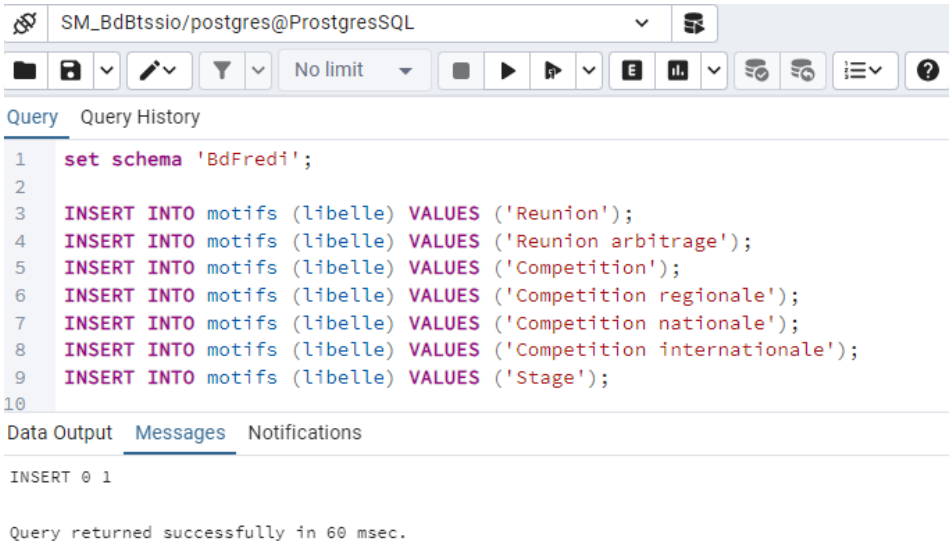


- Alimenter la base avec les données à partir des scripts fournis

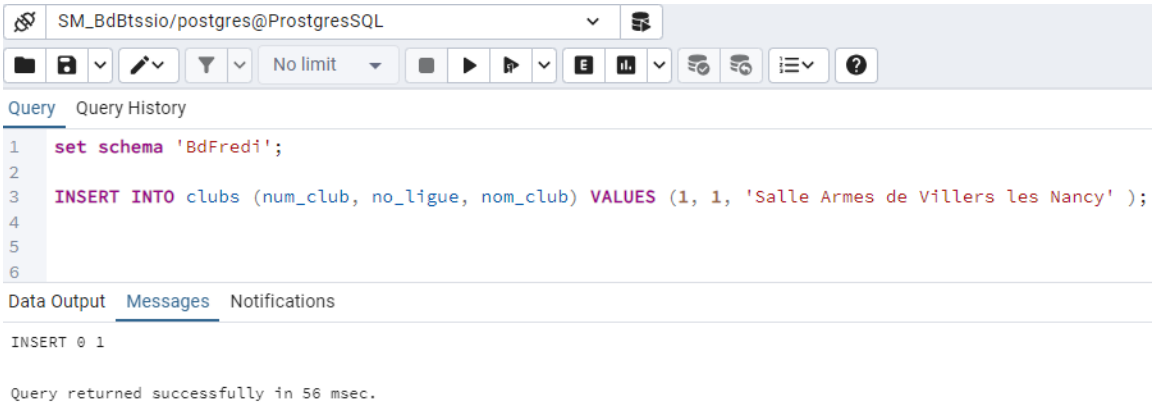
**BDFREDI :**

## Insertion motifs

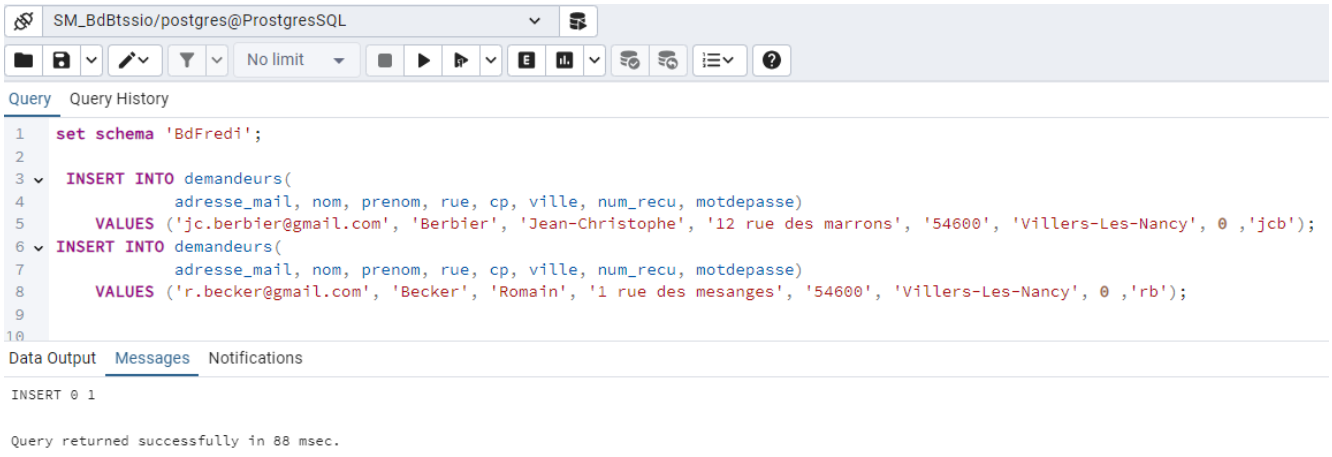



## Insertion ligues

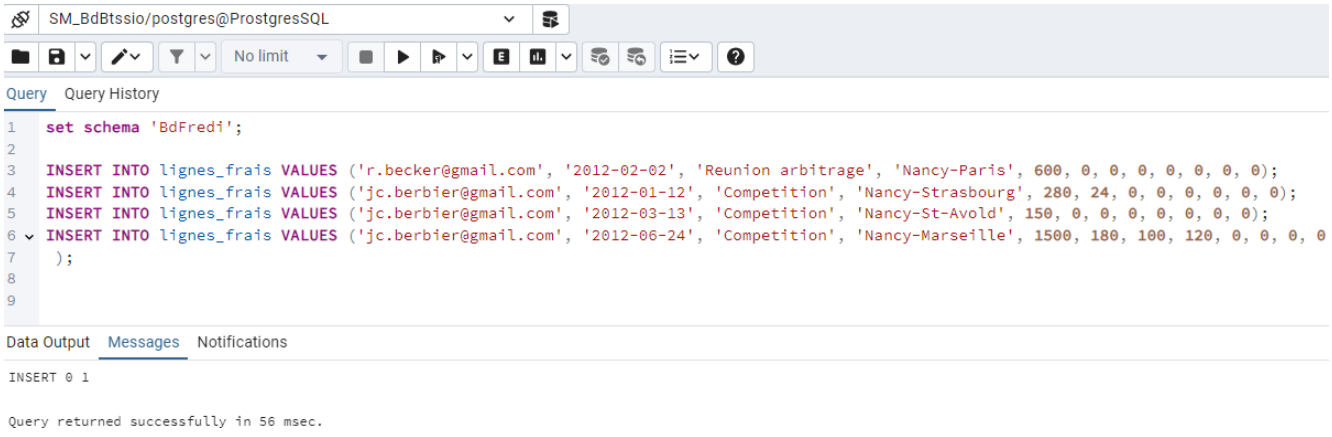
## Insertion clubs



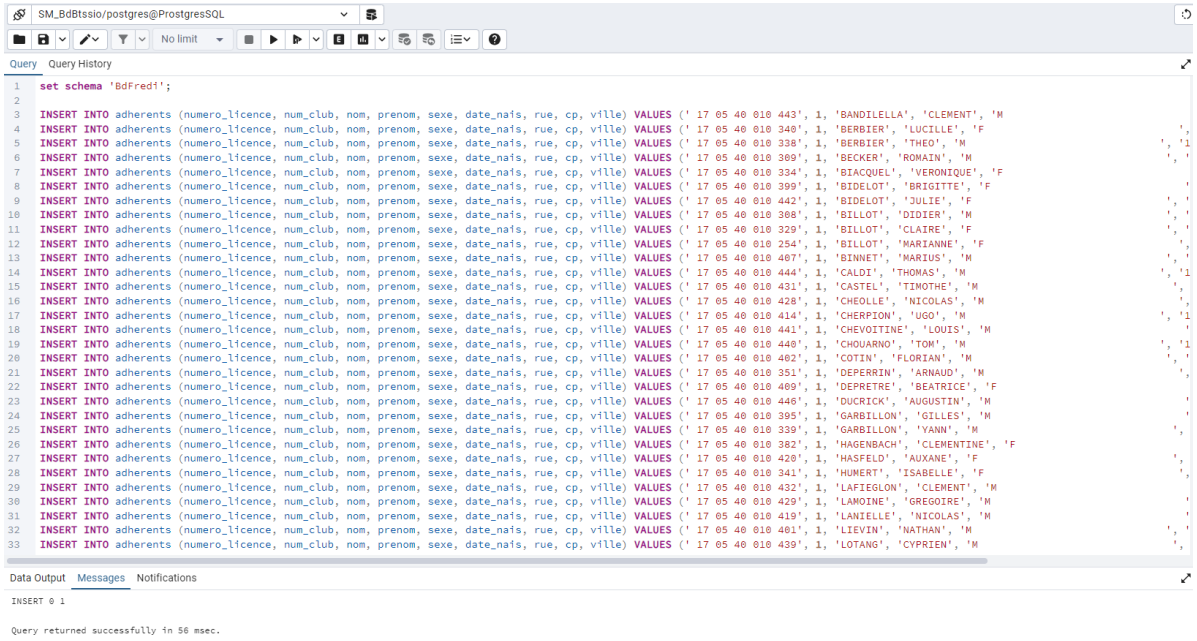

## Insertion demandeurs



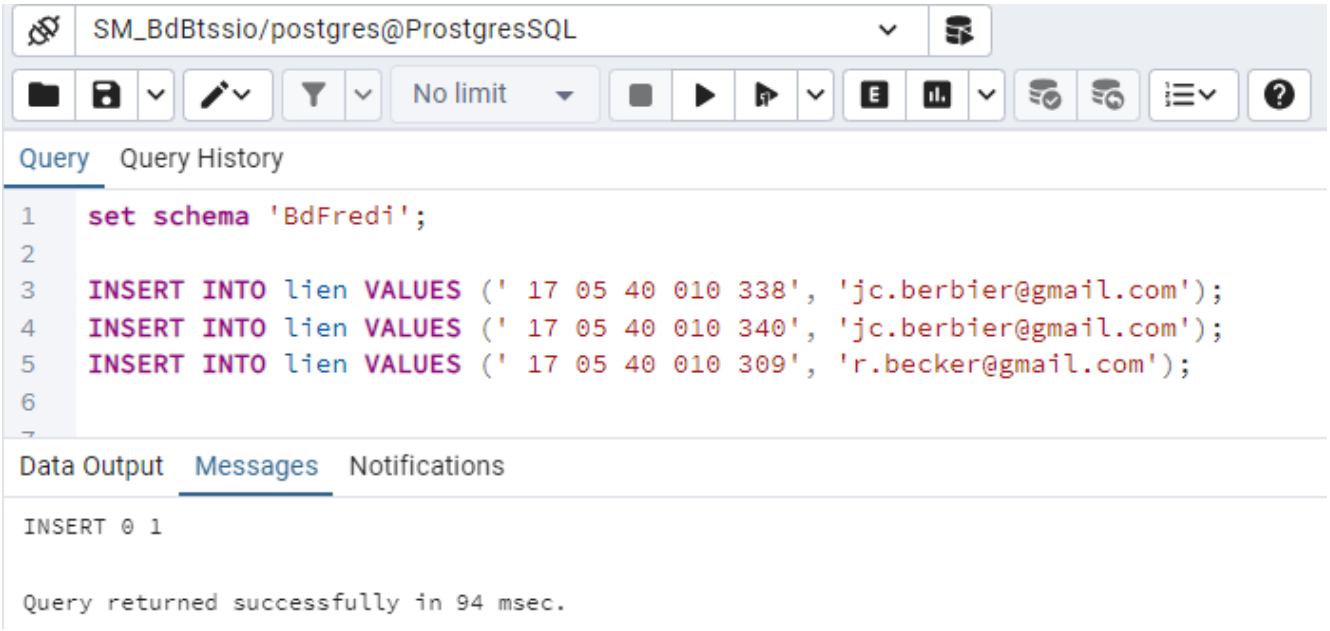

## Insertion lignes_frais



## Insertion adherents

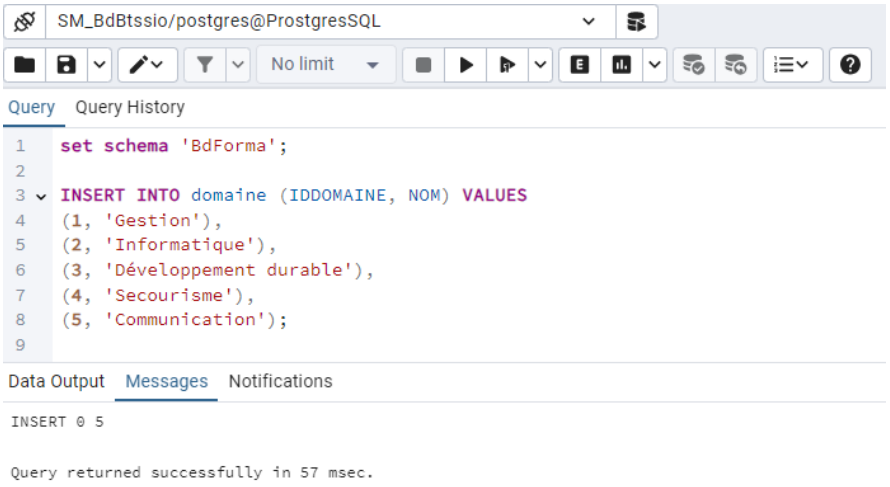



## Insertion lien

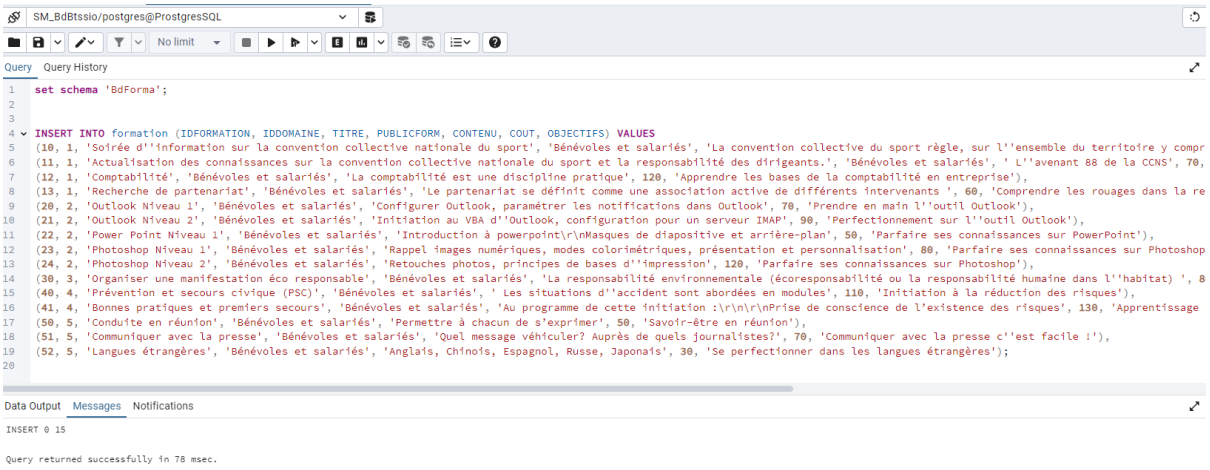


# BDFORMA :

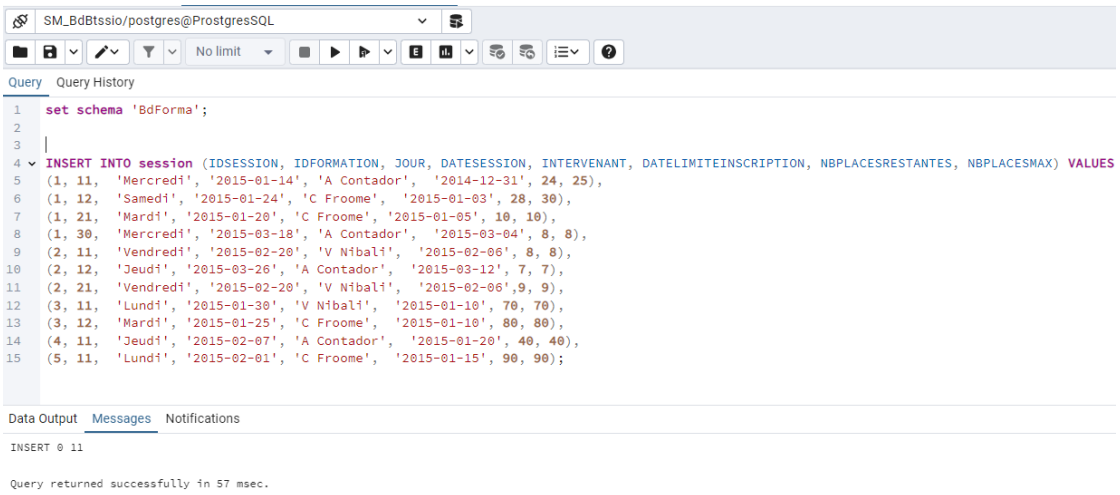
## Insertion domaine



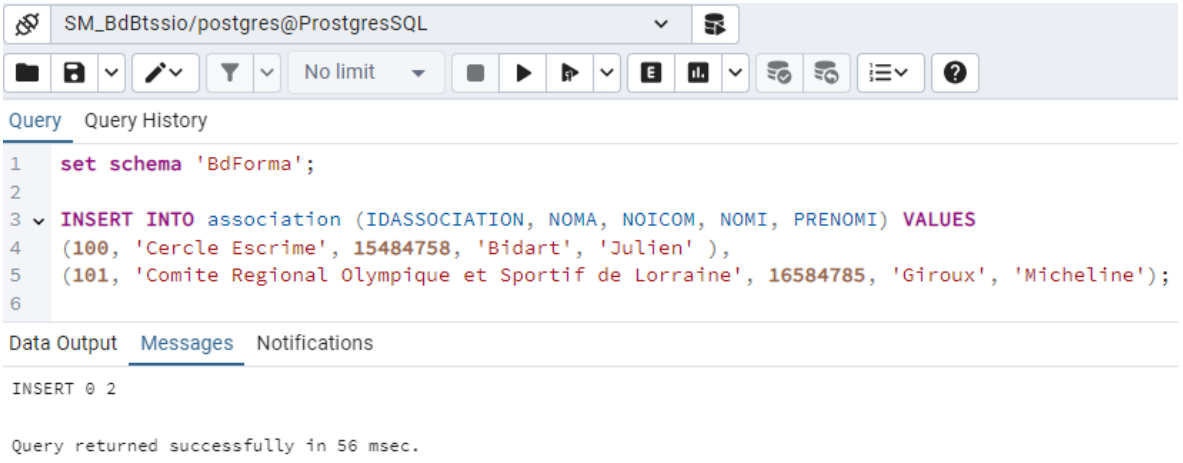

## Insertion formation



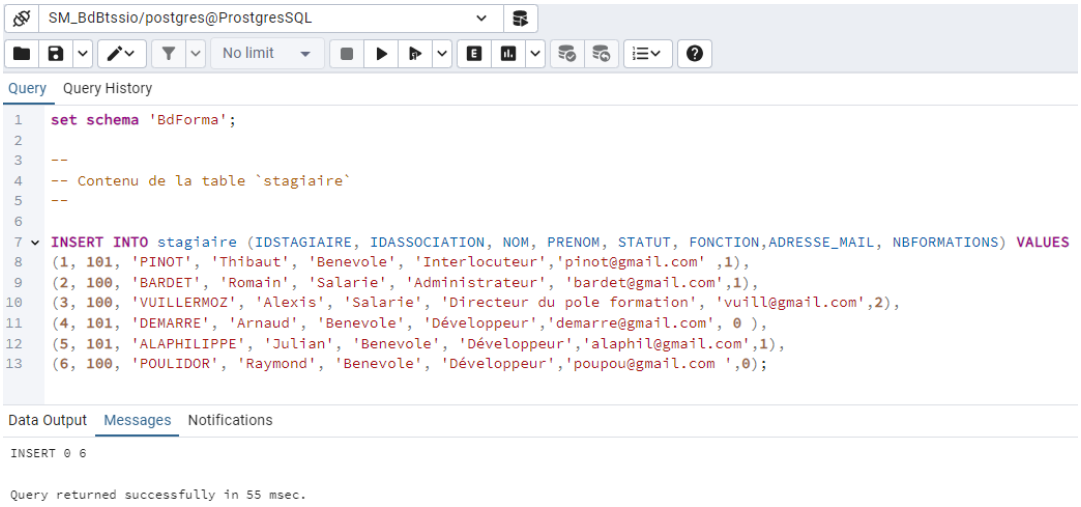

## Insertion session



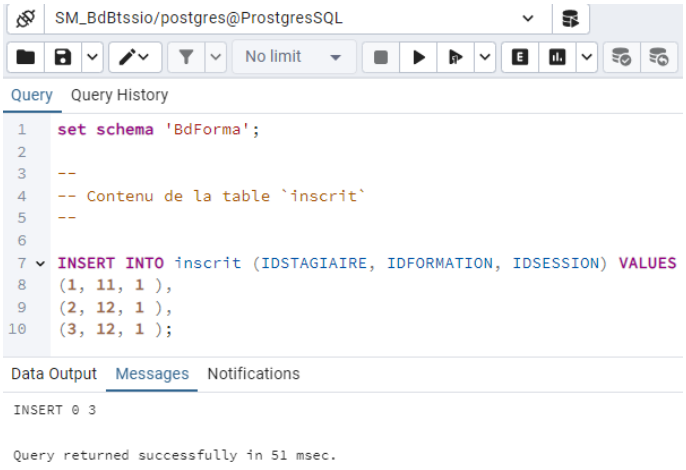
## Insertion association




## Insertion stagiaire




## Insertion inscrit



- 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 ***plpqsql*** 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 ;
```

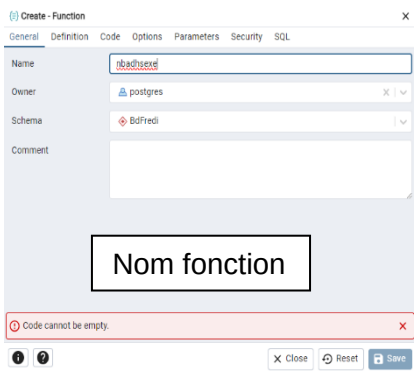
**Suppression d'une fonction**

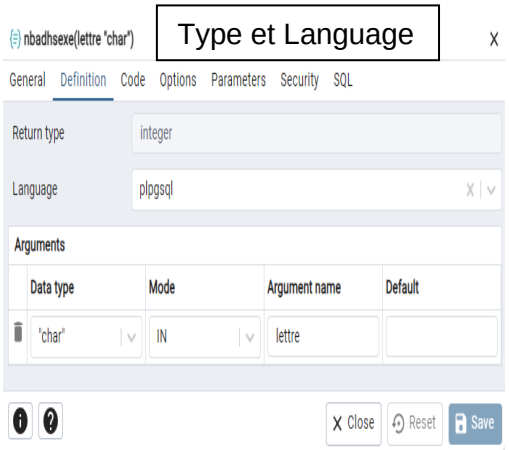
```
DROP FUNCTION nomfonction ();
```

# A faire avec BdFredi :

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

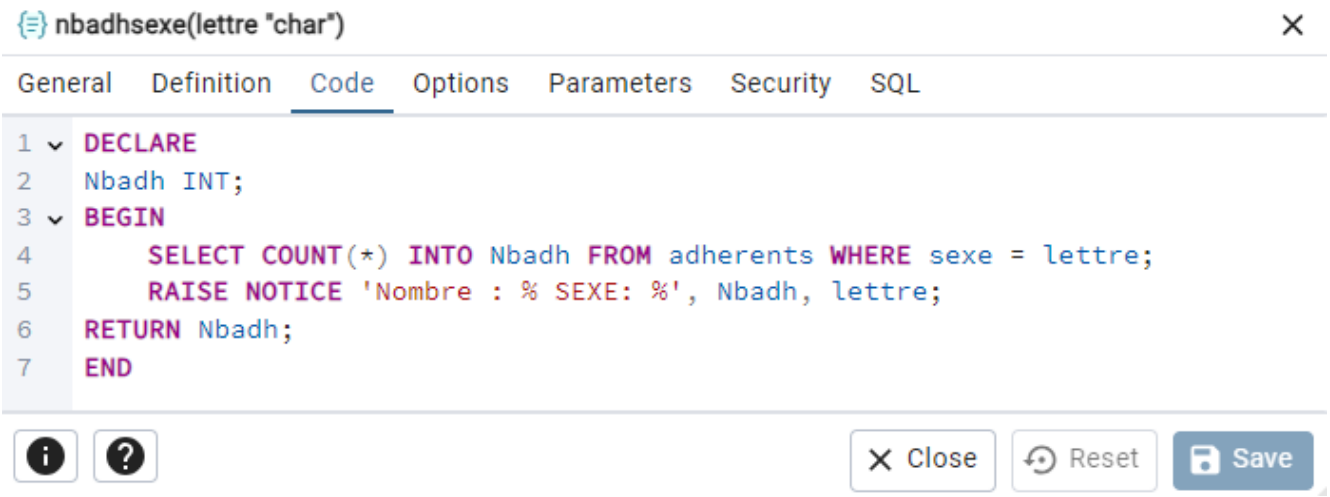
1. **Création de la fonction**
    - ✓ Ajouter la fonction en donnant son nom **( nbadhsexe )**
    - ✓ Choisir le type renvoyé**( Integer )** et le langage **( plpgsql )**
    - ✓ Donner le paramètre ( nom de l'argument et type )
    - ✓ Saisir le code de la fonction de DECLARE ,..., BEGIN,... jusqu'à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



La fonction doit normalement avoir le contenu suivant :

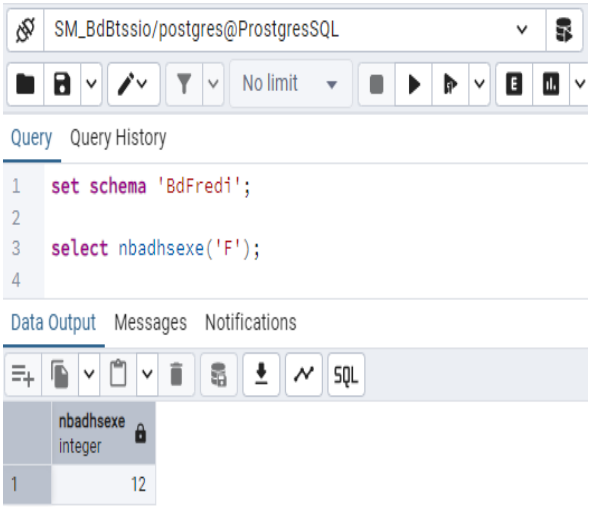
```
DECLARE
Nbadh INT;
BEGIN
        SELECT COUNT(*) INTO Nbadh FROM adherents WHERE sexe = lettre;
        RAISE NOTICE 'Nombre : % SEXE: %', Nbadh, lettre;
RETURN Nbadh;
END
```

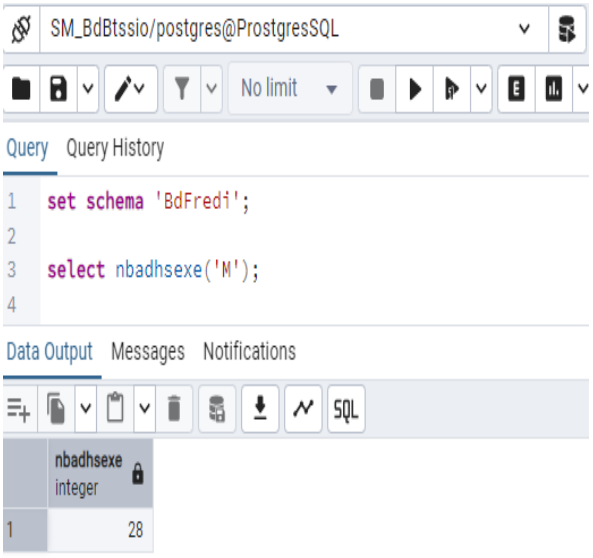



**2. Utilisation de la fonction**

**Dans l'éditeur SQL exécuter la fonction par :**

```
SELECT nbadhsexe('F') ou SELECT nbadhsexe('M')
```

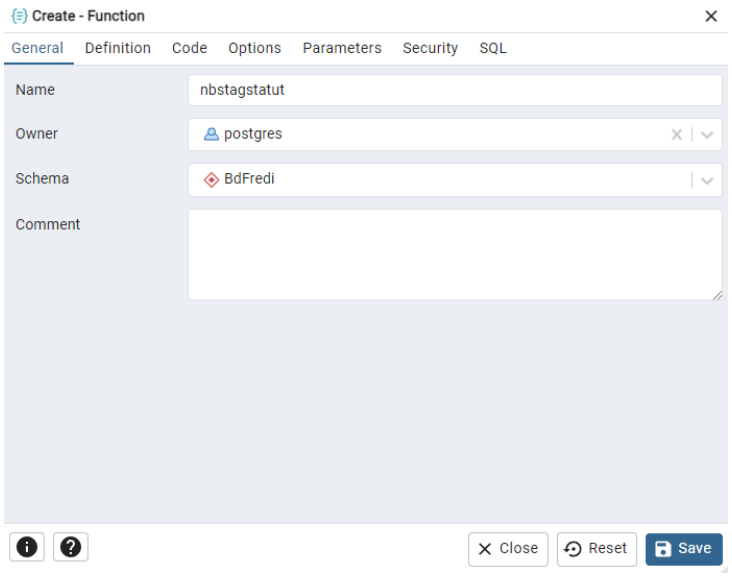


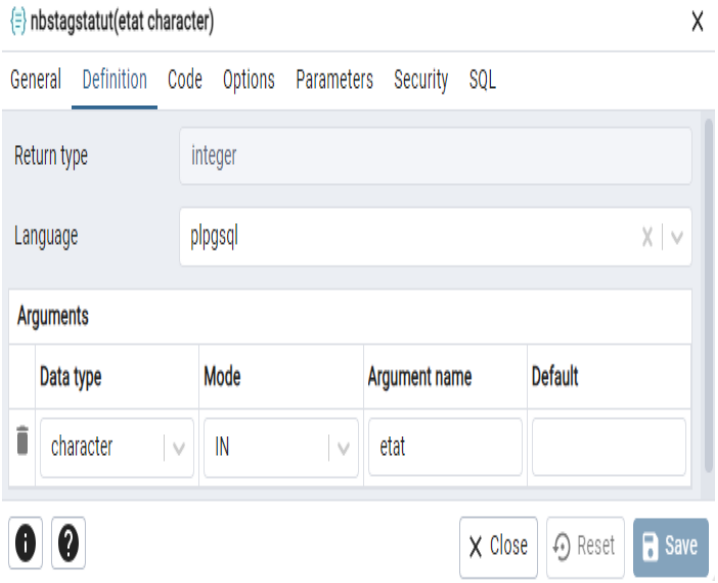



# A faire avec BdForma:

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 :




La fonction doit normalement avoir le contenu suivant :

```
DECLARE Nbstag   INT;
BEGIN
        SELECT COUNT(*) INTO Nbstag FROM stagiaire WHERE statut = etat;
        RAISE NOTICE 'NOMBRE : % STATUT : %', Nbstag, etat;
RETURN Nbstag;
END
```

nbstagstatut(etat character)

General Definition Code Options Parameters Security SQL

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

Close Reset Save

**2. Utilisation de la fonction**

Dans l'éditeur SQL exécuter la fonction par :

SELECT nbstagstatut('Benevole') ou SELECT nbstagstatut('Salarie')

SM_BdBtssio/postgres@ProstgresSQL

No limit

Query Query History

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

Data Output Messages Notifications

| | nbstagstatut integer |
|---|---|
| 1 | 4 |

SM_BdBtssio/postgres@ProstgresSQL

No limit

Query Query History

```
1  set schema 'BdForma';
2
3  select nbstagstatut('Salarie');
```

Data Output Messages Notifications

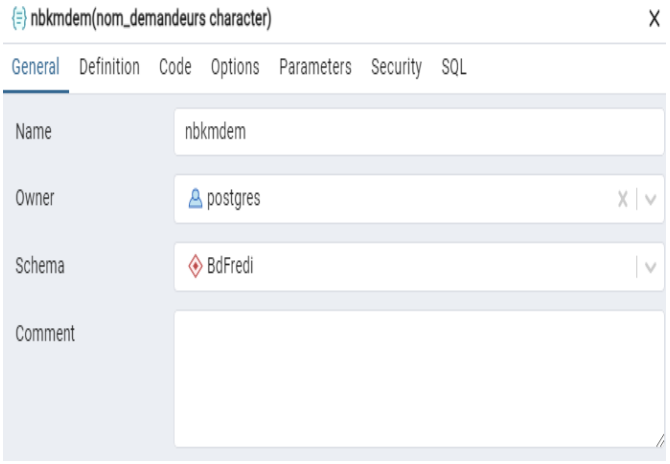
| | nbstagstatut integer |
|---|---|
| 1 | 2 |

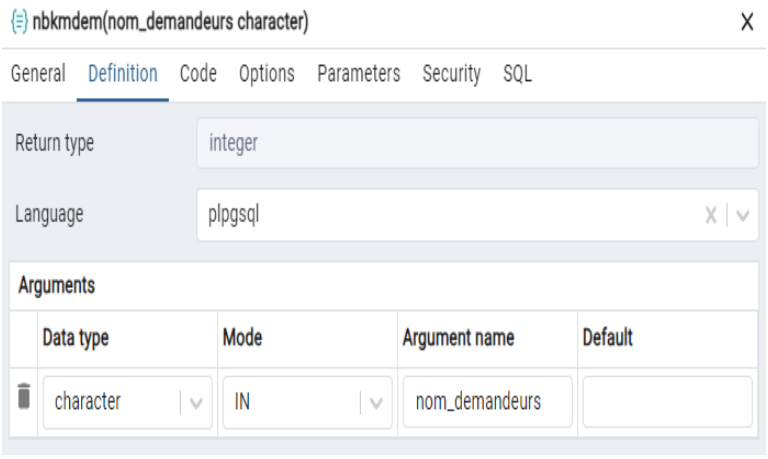
## Etape 2 - Création d'une fonction

**<u>Remarque</u> : L'écriture de ces fonctions peut se faire avec pgadmin3 ou phppgadmin**

## <u>A faire avec BdFredi</u> :

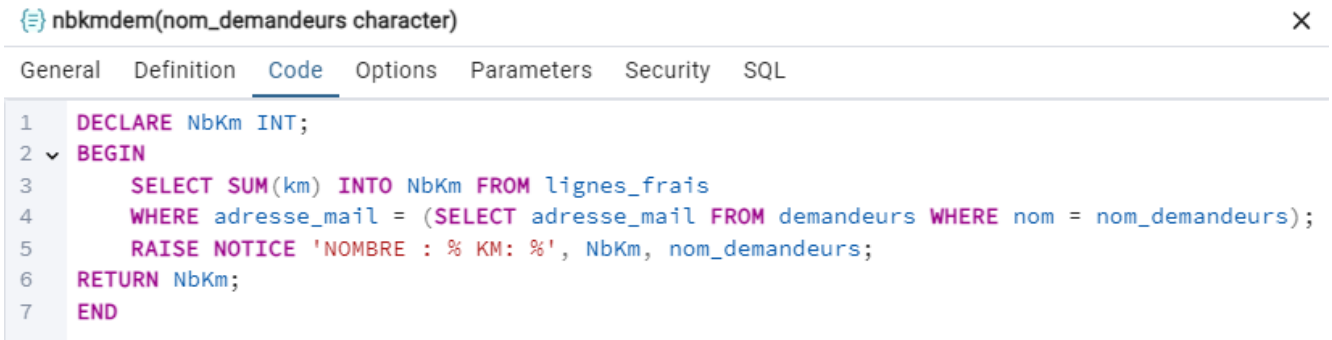
1. Créer et tester la fonction **nbkmdem** qui renverra le nombre total de kilomètres parcourus pour un demandeur donné (on donne le nom en paramètre).



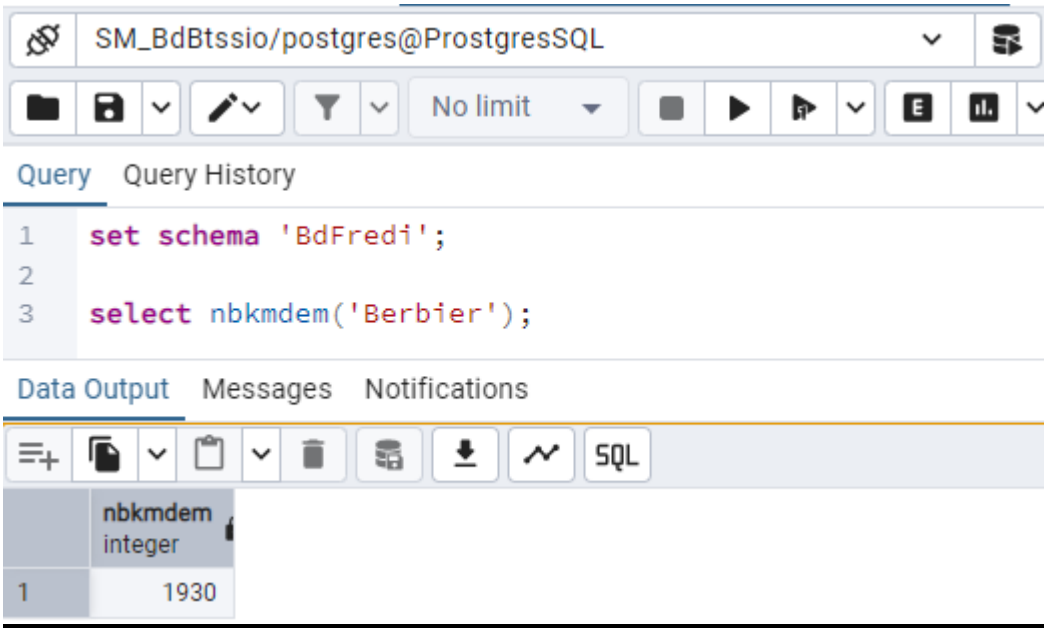



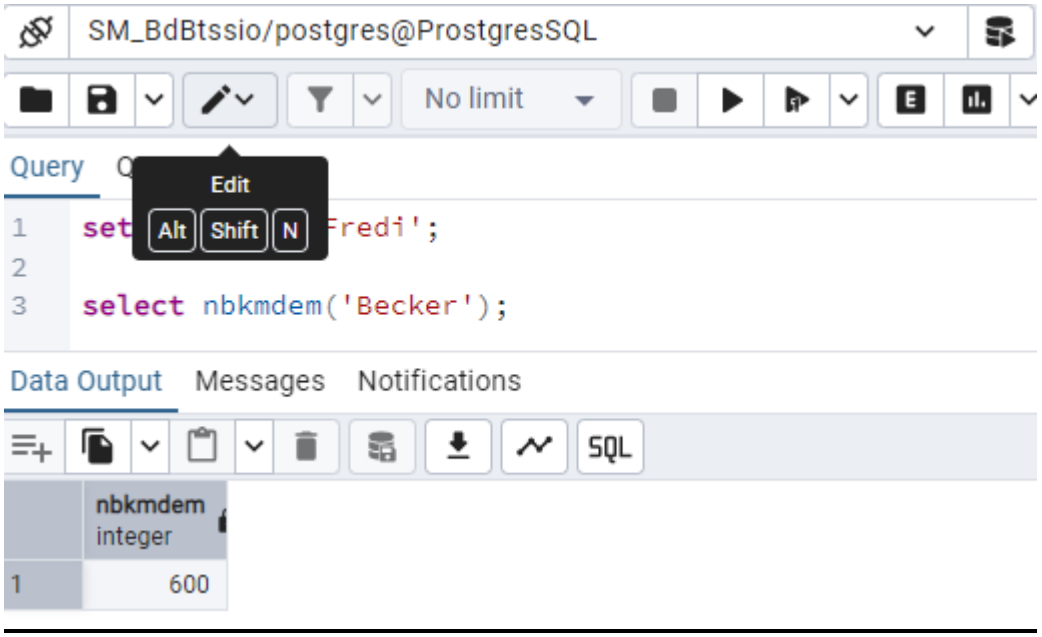

La fonction doit normalement avoir le contenu suivant :

```
DECLARE NbKm INT;
BEGIN
        SELECT SUM(km) INTO NbKm FROM lignes_frais
        WHERE adresse_mail = (SELECT adresse_mail FROM demandeurs WHERE
nom = nom_demandeurs);
        RAISE NOTICE 'NOMBRE : % KM: %', NbKm, nom_demandeurs;
RETURN NbKm;
END
```



**Résultats :**



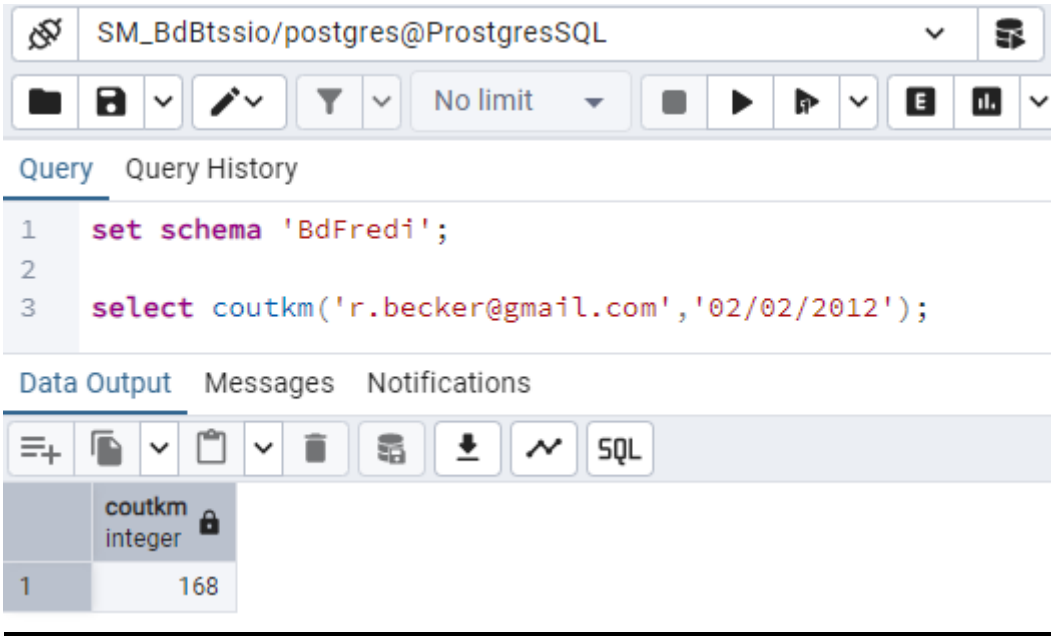




2. Créer et tester la fonction **coutkm** qui calcule le cout d'un trajet pour un demandeur donné et une date donnée.
   Pour cela il faut rechercher le nombre de km effectués pour une ligne de frais que l'on multipliera par le cout kilométrique est fixé à 0,28 euros.

```
DECLARE coutkm INT;
BEGIN
        SELECT SUM(km*0.28) INTO coutkm FROM ligne_frais WHERE
        adresse_mail = mail AND date_frais = datef;
        RAISE NOTICE 'TARIF : % ADRESSE : % DATE : %', coutkm, mail, datef;
RETURN coutkm;
END;
```


**Test à faire** :

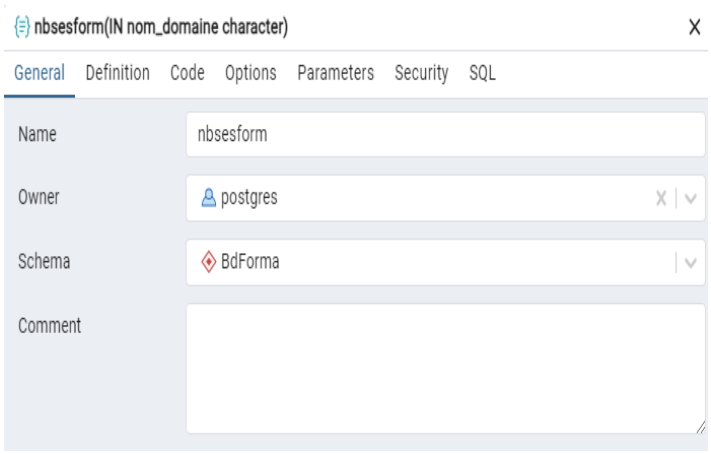
SELECT coutkm('r.becker@gmail.com','02/02/2012'); ) donnera 168 €.

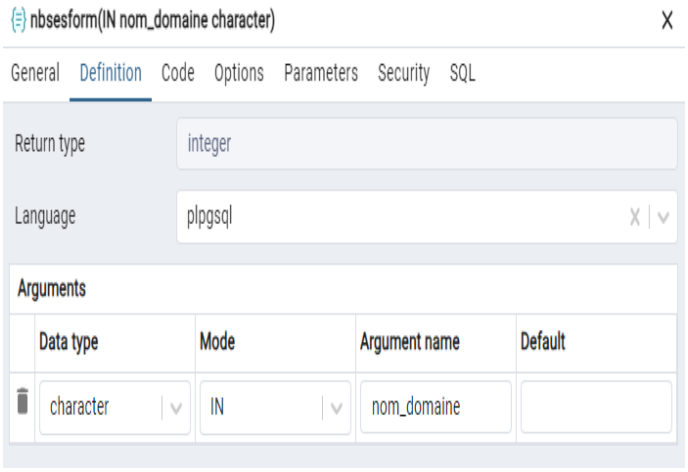



**Remarque** : La condition **if not found then..** placée après la requête permet de savoir si l'enregistrement sollicité a été récupéré.

# A faire avec BdForma :

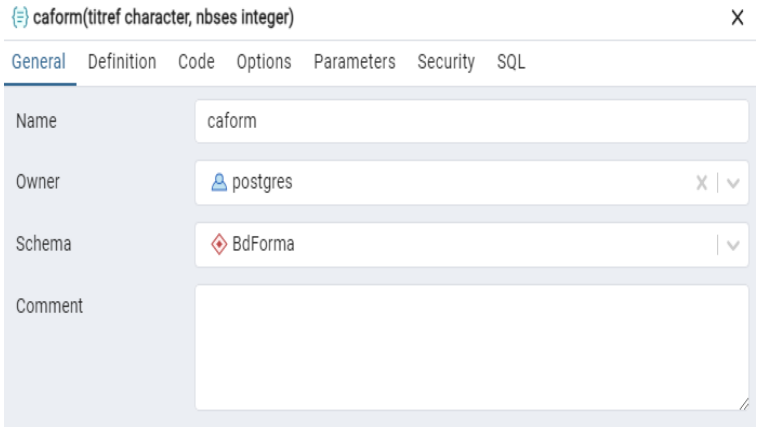
1. Créer et tester la fonction **nbsesform** qui renverra le nombre de sessions de formation pour un domaine donné (on donne le nom en paramètre)

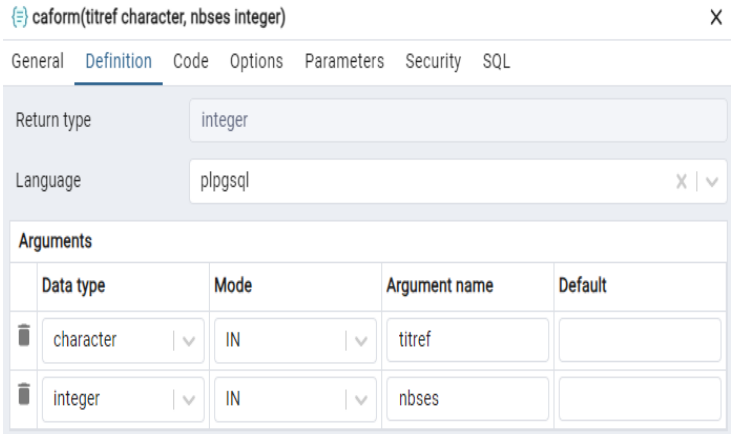



```
DECLARE Nbsession INT;
BEGIN
    SELECT COUNT(idformation) INTO Nbsession FROM formation WHERE iddomaine = (SELECT iddomaine FROM domaine WHERE nom = nom_domaine);
    RAISE NOTICE 'SESSION : % DOMAINE : %', Nbsession, nom_domaine;
RETURN Nbsession;
END;
```


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



```
DECLARE nbCa INT;
BEGIN
        SELECT (session.nbplacesmax - session.nbplacesrestantes) * formation.cout
INTO nbCa FROM session
        NATURAL JOIN formation
        WHERE formation.titre = titref AND session.idsession = nbses;
        RAISE NOTICE 'CA TOTAL : % SESSION : % FORMATION : %', nbCa, nbses,
titref;
RETURN nbCa;
END;
```

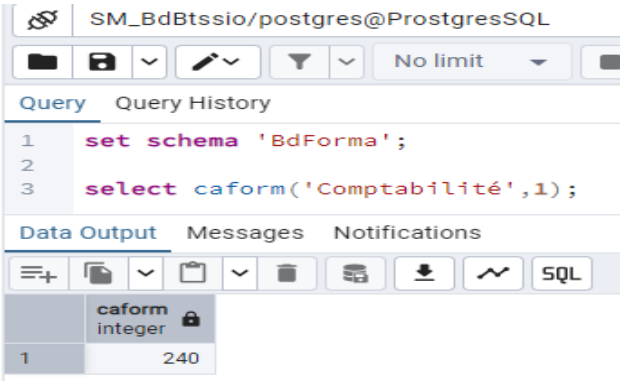
caform(titref character, nbses integer)

General Definition Code Options Parameters Security SQL

```
1  DECLARE nbCa INT;
2  BEGIN
3      SELECT (session.nbplacesmax - session.nbplacesrestantes) * formation.cout INTO nbCa FROM session
4      NATURAL JOIN formation
5      WHERE formation.titre = titref AND session.idsession = nbses;
6      RAISE NOTICE 'CA TOTAL : % SESSION : % FORMATION : %', nbCa, nbses, titref;
7  RETURN nbCa;
8  END;
```

**Test à faire :**

SELECT caform('Comptabilité',1) donnera 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

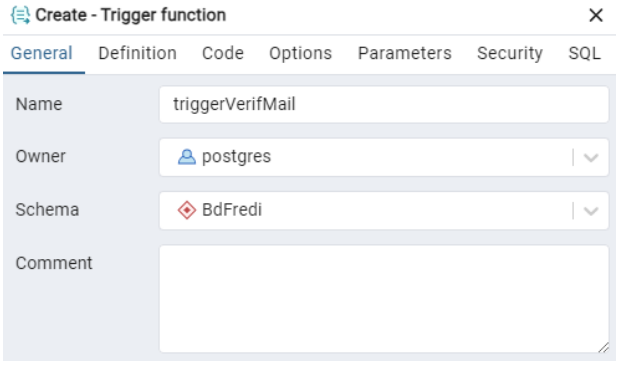
Exemple classique: mise en place d'un trigger déclenchant la fonction verifMail()avant l'insertion d'une occurrence dans une table

Mise en œuvre :

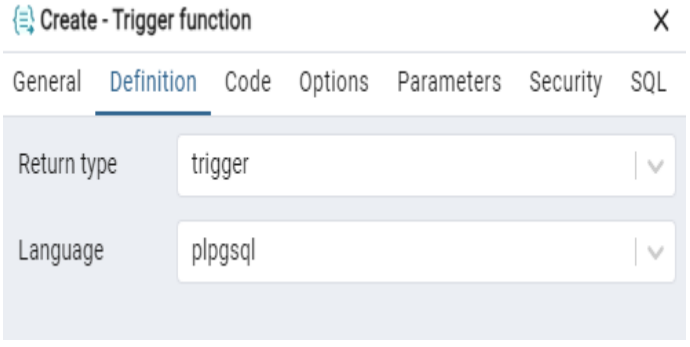
- Sur l'insertion d'un nouveau demandeur dans la base **BdFredi**
- Sur l'insertion d'un nouveau stagiaire dans la base **BdForma**

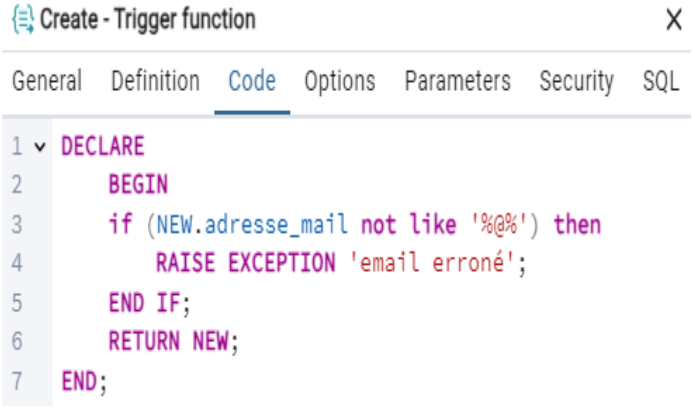
**Première étape** : Création de la fonction Trigger

□ la fonction trigger**verifMail()**




```
DECLARE
        BEGIN
        if (NEW.adresse_mail not like '%@%') then
                RAISE EXCEPTION 'email erroné';
        END IF;
        RETURN NEW;
END;
```



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

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

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

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

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

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

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


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

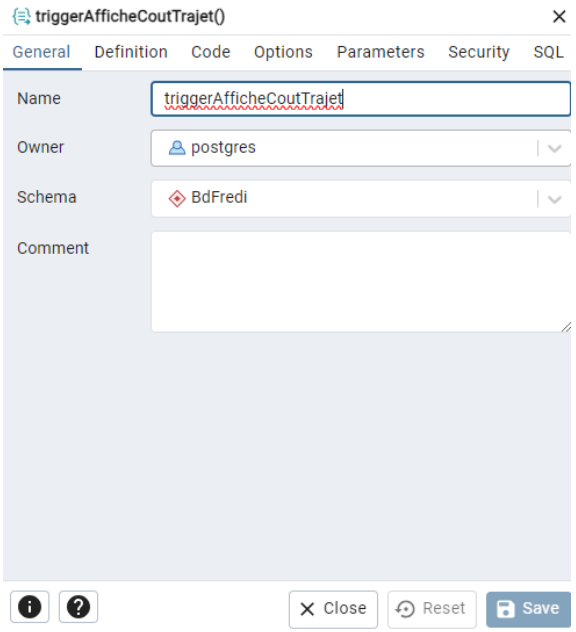
## Etape 2 - Création d'un trigger

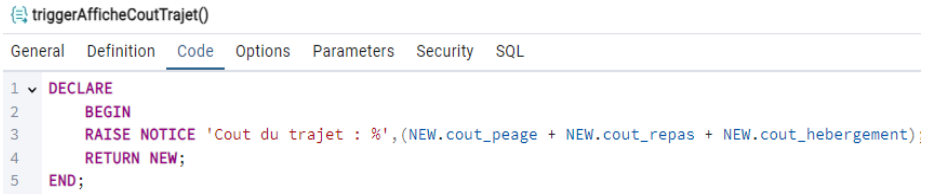
Vous pouvez créer les triggers suivants des 2 façons suivantes :

 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.

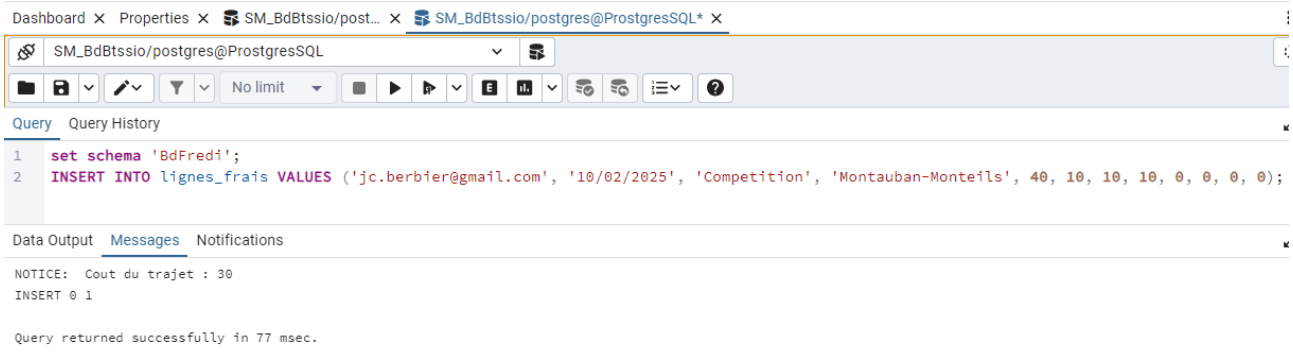



```
DECLARE
        BEGIN
        RAISE NOTICE 'Cout du trajet :
%',(NEW.cout_peage + NEW.cout_repas +
NEW.cout_hebergement);
        RETURN NEW;
END;
```

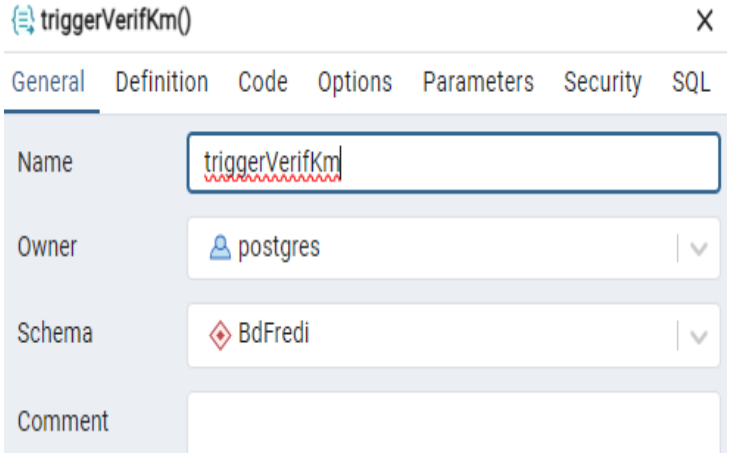
```
CREATE TRIGGER "afficheCoutTrajet"
   AFTER INSERT
   ON "BdFredi".lignes_frais
   FOR EACH ROW
   EXECUTE FUNCTION "BdFredi"."triggerAfficheCoutTrajet"();
```

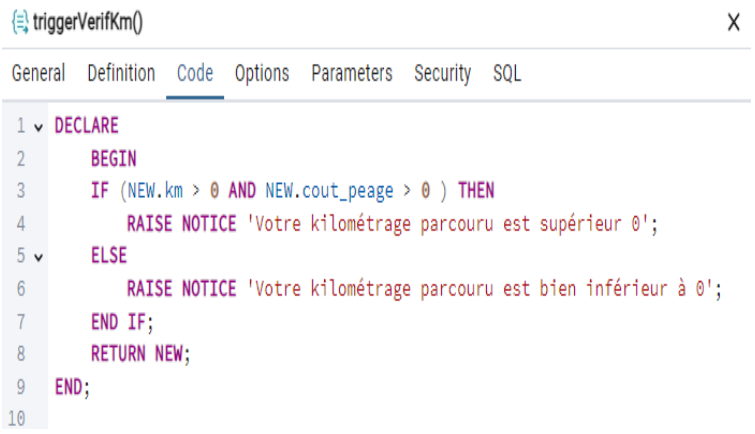
- Tester par insertion d'occurrences dans la table **ligne_frais** en utilisant le fichier de test **TestTriggerCoutTrajet** fourni.



**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.

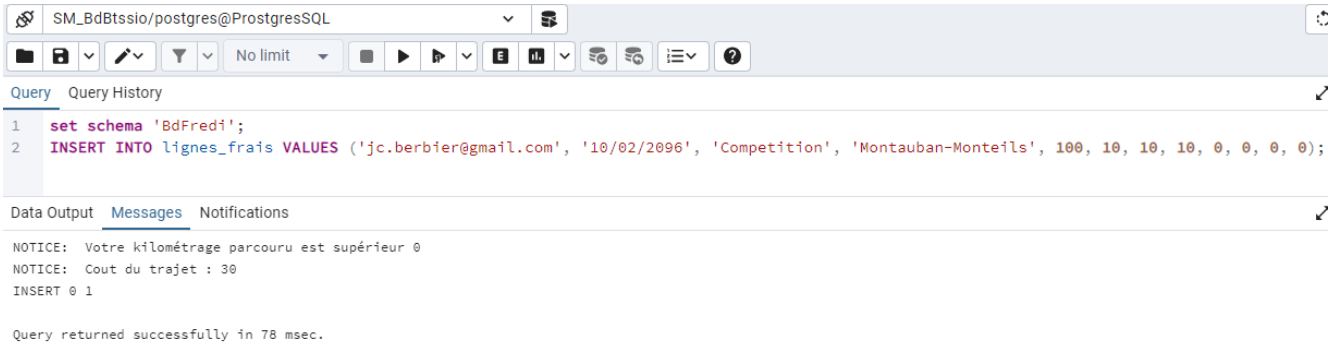


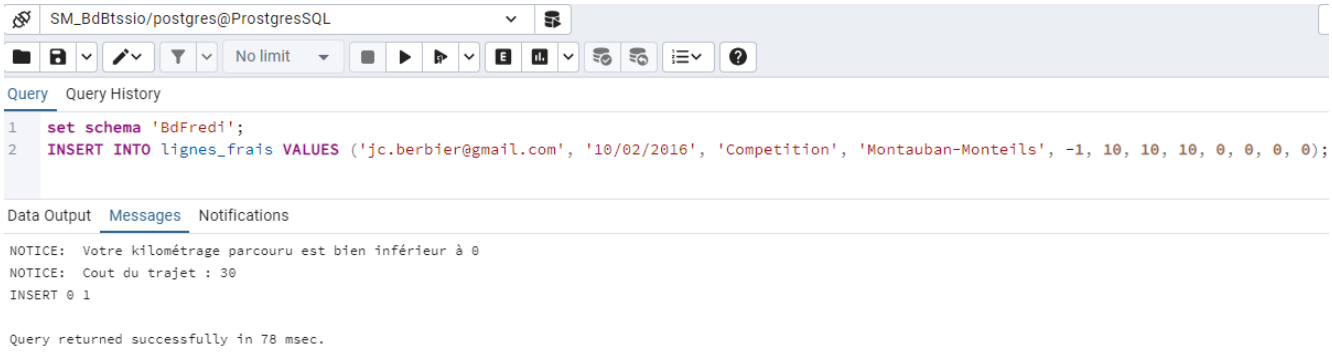




```
DECLARE
        BEGIN
        IF (NEW.km > 0 AND NEW.cout_peage > 0 ) THEN
            RAISE NOTICE 'Votre kilométrage parcouru est supérieur 0';
        ELSE
            RAISE NOTICE 'Votre kilométrage parcouru est bien inférieur à 0';
        END IF;
        RETURN NEW;
END;
```

```
CREATE TRIGGER "verifKm"
   BEFORE INSERT
   ON "BdFredi".lignes_frais
   FOR EACH ROW
   EXECUTE FUNCTION "BdFredi"."triggerVerifKm"();
```

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






**<u>En Bonus</u>** :

- 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 intoadh * from adherents where…** renvoie dans la variable **adh** les données correspondantes comme par exemple **adh.nom**, **adh.prenom**,…

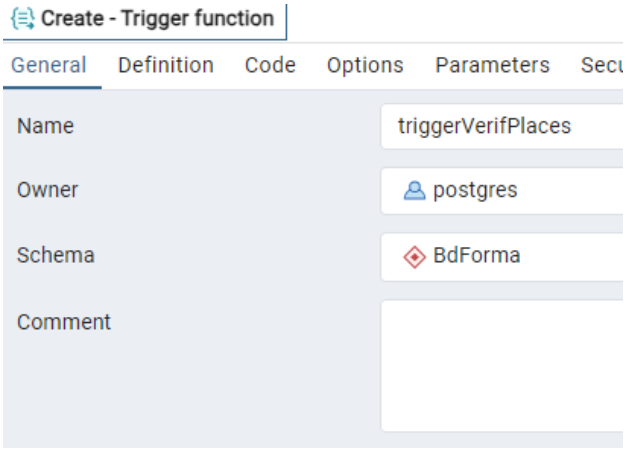
  ✓ Utiliser **FOUND** pour savoir si le **select into** a renvoyé au moins un résultat

  ✓ Utiliser une boucle **for** pour afficher les numéros d'adhérents éventuels

- Tester en utilisant le fichier **TestTriggerVerifLien** fourni.

## <u>A faire avec BdForma</u> :

- 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.



triggerVerifPlaces()

General Definition Code Options Parameters Security SQL

```
1  DECLARE
2      BEGIN
3      IF (NEW.nbplacesrestantes <= NEW.nbplacesmax) THEN
4          RAISE NOTICE 'Les places restantes sont de : % ',NEW.nbplacesrestantes;
5      END IF;
6      RETURN NEW;
7  END;
```

```
DECLARE
        BEGIN
        IF (NEW.nbplacesrestantes <= NEW.nbplacesmax) THEN
              RAISE NOTICE 'Les places restantes sont de : %.',NEW.nbplacesrestantes;
        ELSE
              RAISE NOTICE 'Il y a plus de places disponible.';
        END IF;
        RETURN NEW;
END;
```

```
CREATE TRIGGER "verifPlaces"
   BEFORE INSERT
   ON "BdForma".session
   FOR EACH ROW
   EXECUTE FUNCTION " BdForma "."triggerVerifPlaces"();
```

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

SM_BdBtssio/postgres@ProstgresSQL

No limit

Query Query History

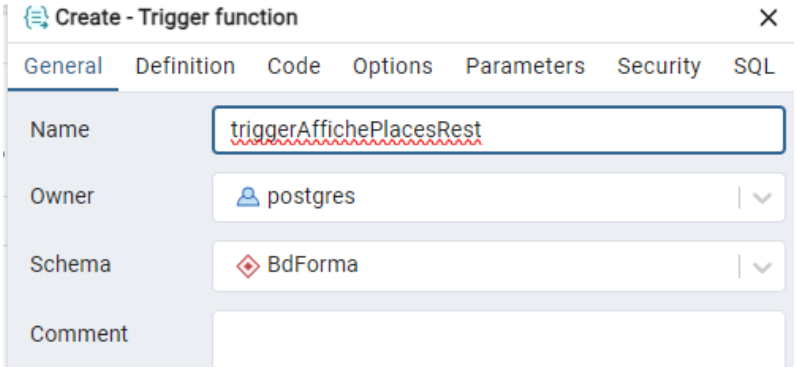
```
1  set schema 'BdForma';
2
3  INSERT INTO session VALUES (5, 12, 'Mercredi', '2025-10-04', 'C Froome', '2025-10-18', 90, 100);
```

Data Output Messages Notifications

```
NOTICE:  Les places restantes sont de : 90.
INSERT 0 1
```


**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.

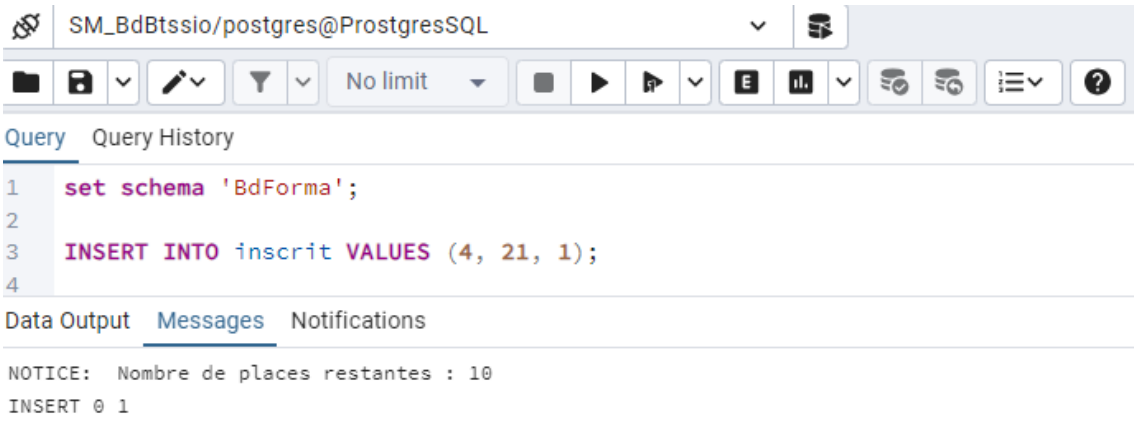


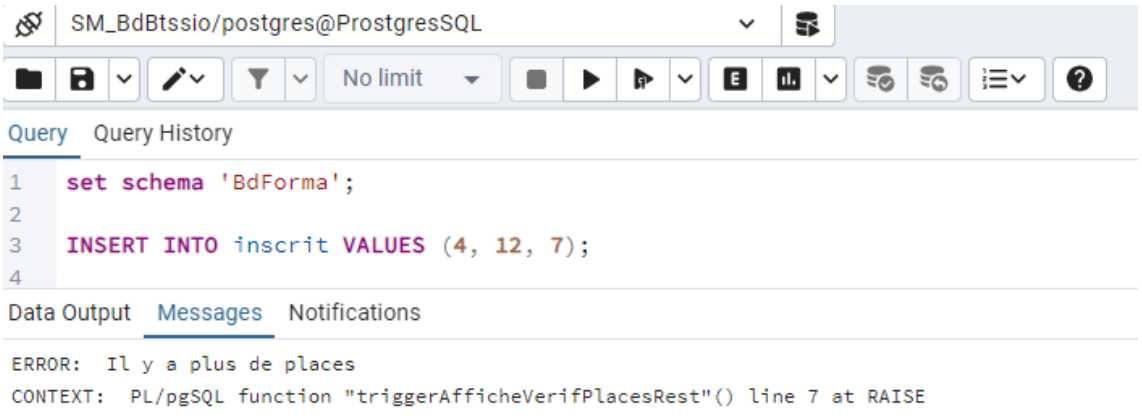


```
DECLARE
placesnb INT;
BEGIN
        SELECT session.nbplacesrestantes INTO placesnb FROM session
        WHERE session.idsession = NEW.idsession AND session.idformation =
        NEW.idformation;
        IF (placesnb = 0) THEN
           RAISE EXCEPTION 'Il y a plus de places';
        ELSE
                RAISE NOTICE 'Nombre de places restantes : %',placesnb;
        END IF;
        RETURN NEW;
END;
```

```
CREATE TRIGGER "affichePlacesRest"
   BEFORE INSERT
   ON "BdForma".inscrit
   FOR EACH ROW
   EXECUTE FUNCTION "BdForma"."triggerAfficheVerifPlacesRest"();
```

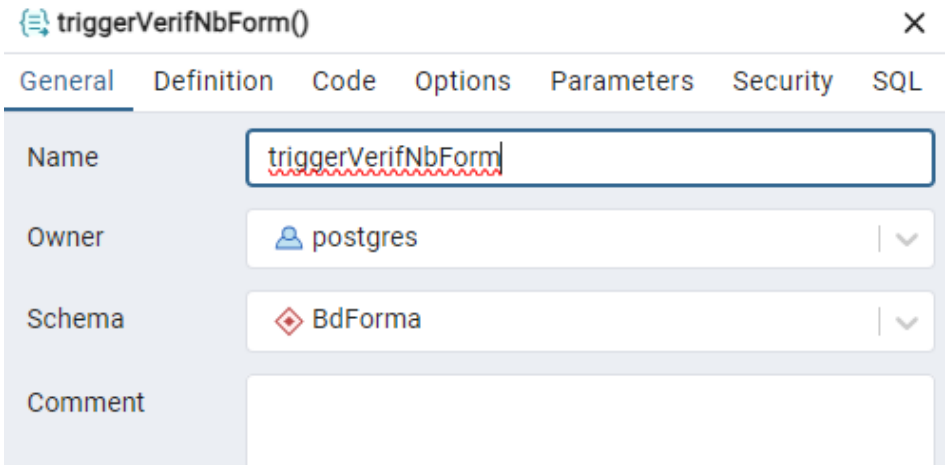
- Tester par insertion d'occurrences dans la table **inscrit** en utilisant le fichier de test **TestTriggerPlacesRest** fourni.

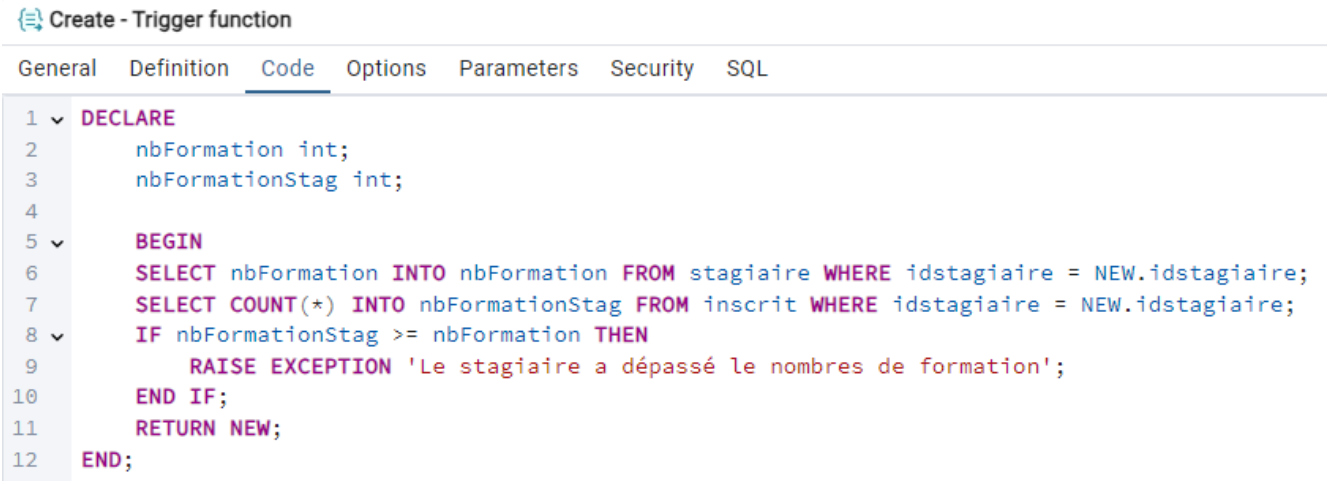



**<u>En bonus</u>**:

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




```
DECLARE
   nbFormation int;
   nbFormationStag int;

   BEGIN
   SELECT nbFormation INTO nbFormation FROM stagiaire WHERE idstagiaire =
NEW.idstagiaire;
   SELECT COUNT(*) INTO nbFormationStag FROM inscrit WHERE idstagiaire =
NEW.idstagiaire;
   IF nbFormationStag >= nbFormation THEN
      RAISE EXCEPTION 'Le stagiaire a dépassé le nombres de formation';
   END IF;
   RETURN NEW;
END;
```

```
CREATE TRIGGER "verifNbForm"
   BEFORE INSERT
   ON "BdForma".inscrit
   FOR EACH ROW
   EXECUTE FUNCTION "BdForma"."triggerVerifNbForm"();
```

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

Query Query History

```
1  set schema 'BdForma';
2
3  INSERT INTO inscrit VALUES (2,13,1);
4
```

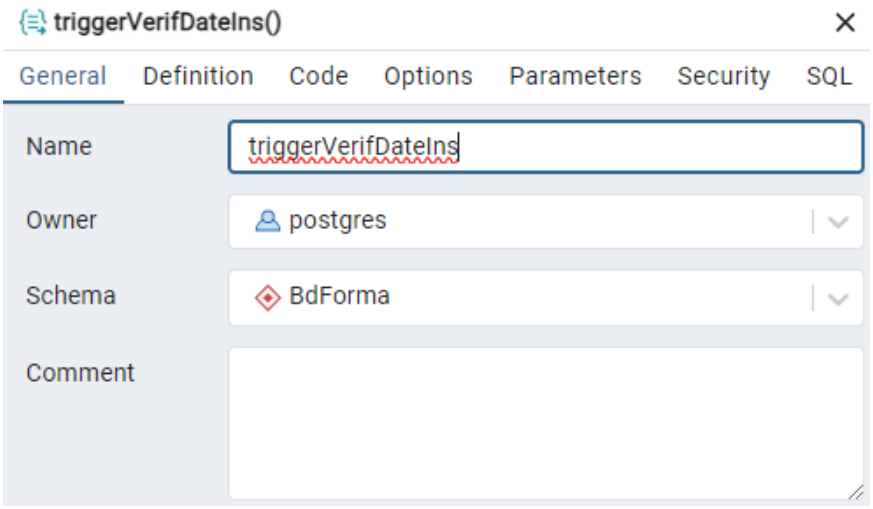
Data Output Messages Notifications

```
NOTICE:  Nombre de places restantes : <NULL>

ERROR:  Le stagiaire a dépassé le nombres de formation
CONTEXT:  PL/pgSQL function "triggerVerifNbForm"() line 9 at RAISE

SQL state: P0001
```

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



triggerVerifDateIns()

General Definition Code Options Parameters Security SQL

```
1  DECLARE
2  datelimite DATE;
3  dateauj DATE;
4      BEGIN
5      SELECT datelimiteinscription INTO datelimite FROM session
6      WHERE NEW.idformation = idformation AND NEW.idsession = idsession;
7      SELECT CURRENT_DATE INTO dateauj FROM session;
8      IF ( dateauj > datelimite ) THEN
9          RAISE EXCEPTION 'date limite atteinte';
10     END IF;
11     RETURN NEW;
12 END;
```

Create - Trigger

General Definition Events Transition Code SQL

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

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

SM_BdBtssio/postgres@ProstgresSQL

No limit

Query Query History

```
1  set schema 'BdForma';
2
3  INSERT INTO inscrit VALUES (4, 11, 1);
```

Data Output Messages Notifications

```
NOTICE:  Nombre de places restantes : 24

ERROR:  date limite atteinte
```

# 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

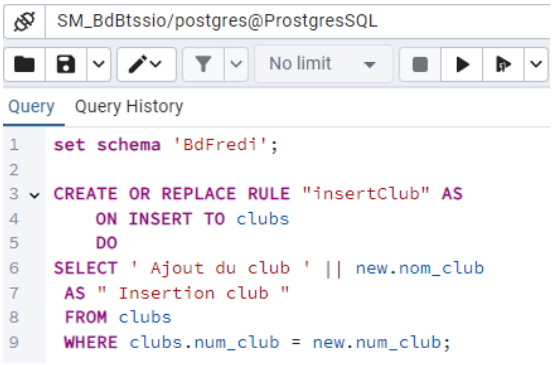
Il y a un exemple développé pour chaque mission ( FREDI ou FORMA )

## Exemple sur BdFredi :

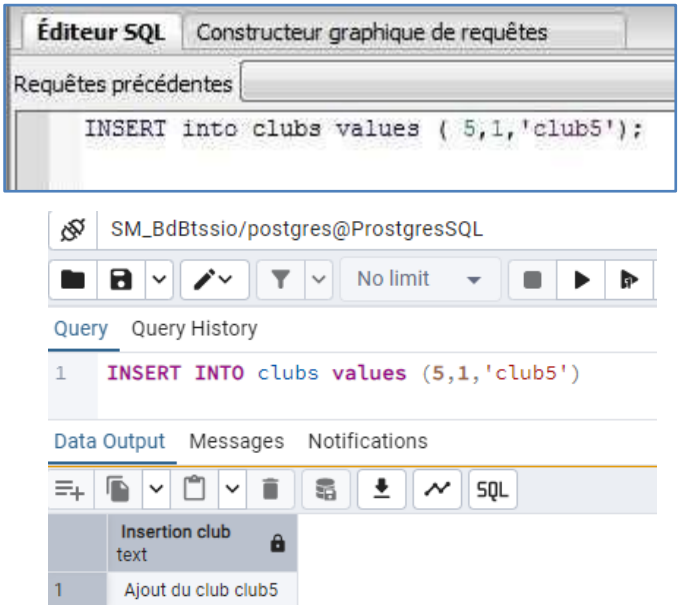
Création d'une règle **insertClub** qui, à chaque insertion dans la table Clubs affichera le nom du club ajouté.

- . Création de la règle **insertClub**

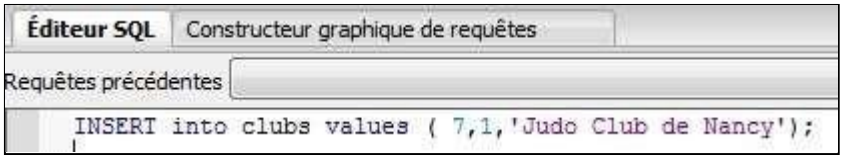
```
CREATE OR REPLACE RULE "insertClub" AS
    ON INSERT TO clubs
    DO
SELECT ' Ajout du club ' || new.nom_club
 AS " Insertion club "
 FROM clubs
 WHERE clubs.num_club = new.num_club;
```



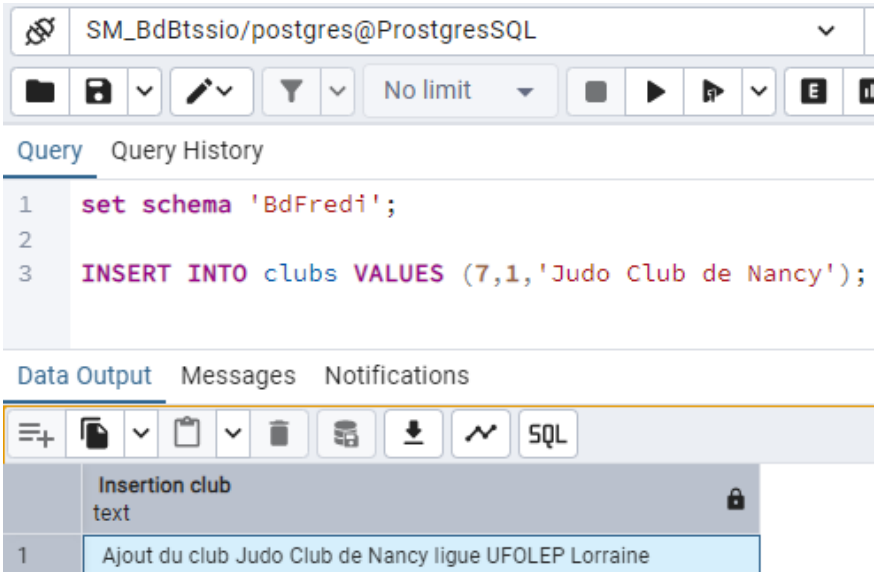

- Tester par insertion d'occurrences dans la table **clubs** avec l'éditeur sql en vérifiant l'affichage en sortie de données.

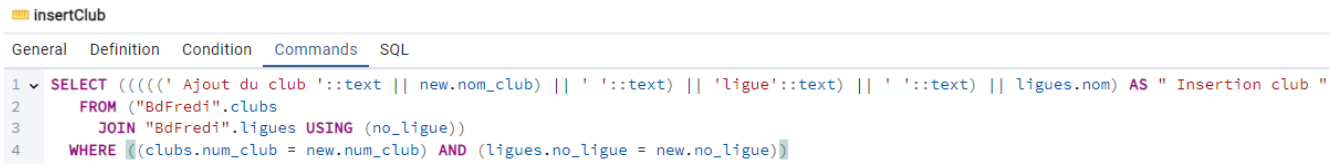



- Modifier la règle **insertClub** en ajoutant dans l'affichage le nom de la ligue du club inséré comme dans l'exemple ci-dessous :








## Exemple sur BdForma :

Création d'une règle **insertStagiaire** qui, à chaque insertion dans la table s**tagiaire** 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;
```

SM_BdBtssio/postgres@ProstgresSQL

No limit

Query Query History

```
1  set schema 'BdForma';
2
3  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;
```

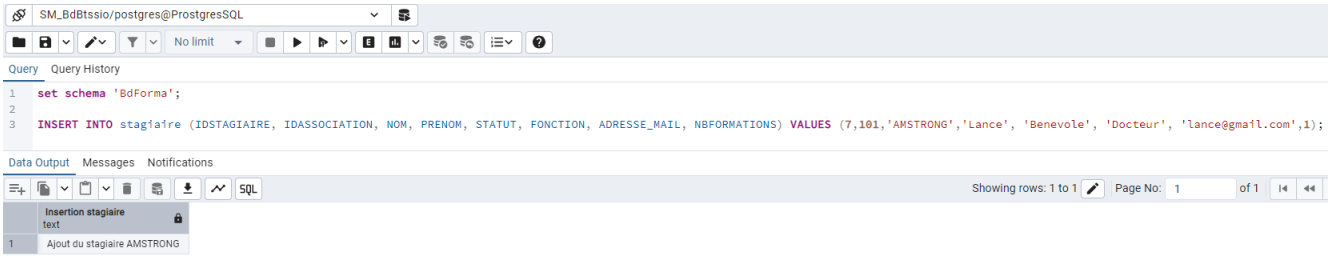
- Tester par insertion d'occurrences dans la table **stagiaire** avec l'éditeur sql en vérifiant l'affichage en sortie de données.

**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 |




- 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 |


# Etape 2 - Création 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 |

| | no_ligue [PK] integer | nom text | sigle text | president text |
|---|---|---|---|---|
| 1 | 1 | UFOLEP Lorraine | L2L | 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 | |

| | num_club [PK] integer | no_ligue integer | nom_club text | rue_club text | cp_club text | ville_club text |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Laxou Football Club | [null] | [null] | [null] |
| 2 | 5 | 1 | club5 | [null] | [null] | [null] |
| 3 | 7 | 1 | Judo Club de Nancy | [null] | [null] | [null] |

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 |

Query | Query History

```
1  set schema 'BdFredi';
2
3  UPDATE clubs SET no_ligue = 2 WHERE num_club = 1;
```

Data Output | Messages | Notifications

| | Nouveau club text |
|---|---|
| 1 | Mise à jour du club Laxou Football Club ancienne ligue 1 nouvelle ligue 2 |

Apres la mise à jour

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

| | num_club [PK] integer | no_ligue integer | nom_club text | rue_club text | cp_club text | ville_club text |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | Laxou Football Club | [null] | [null] | [null] |
| 2 | 5 | 1 | club5 | [null] | [null] | [null] |
| 3 | 7 | 1 | Judo Club de Nancy | [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 |

majLigueClub

General | Definition | Condition | Commands | SQL

```
SELECT (((((('Mise à jour du club : '::text || new.nom_club) || ' Ancienne ligue : '::text) || ( SELECT ligues.nom
            FROM "BdFredi".ligues
           WHERE (ligues.no_ligue = old.no_ligue))) || ' Nouvelle ligue : '::text) || ( SELECT ligues.nom
            FROM "BdFredi".ligues
           WHERE (ligues.no_ligue = new.no_ligue))) AS "Changement de ligue"
    FROM "BdFredi".clubs
   WHERE (clubs.num_club = new.num_club)
```

Query | Query History

```
set schema 'BdFredi';

UPDATE clubs SET no_ligue = 2 WHERE num_club = 1;
```

Data Output | Messages | Notifications

SQL

| | Changement de ligue text |
|---|---|
| 1 | Mise à jour du club : Laxou Football Club Ancienne ligue : UFOLEP Lorraine Nouvelle ligue : FOOTBALL 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.

Query | Query History

```
set schema 'BdForma';

CREATE OR REPLACE RULE "majAssoStagiaire " AS
ON UPDATE TO stagiaire DO
    SELECT 'Mise à jour du stagiaire : ' || NEW.nom::text ||
           ' Ancienne Asso :  ' || OLD.idassociation ||
           ' Nouvelle asso : ' || NEW.idassociation AS "Changement d' association"
    FROM stagiaire
    WHERE stagiaire.idstagiaire = NEW.idstagiaire;
```

Data Output | Messages | Notifications

```
CREATE RULE

Query returned successfully in 97 msec.
```

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 |

| | idassociation [PK] integer | noma character (50) | noicom integer | nomi character (25) | prenomi character (50) |
|---|---|---|---|---|---|
| 1 | 100 | Cercle Escrime | 15484758 | Bidart | Julien |
| 2 | 101 | Comite Regional Olympique et Sportif de Lorraine | 16584785 | Giroux | Micheline |

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 | Benevol |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 101 | AMSTRONG ... | Lance | Benevole | Docteur ... | lance@gmail.com | 1 |

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 |

Query | Query History

```
1  set schema 'BdForma';
2
3  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 : 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 |

| | idstagiaire [PK] integer | idassociation integer | nom character (50) | prenom character (50) |
|---|---|---|---|---|
| 1 | 7 | 100 | AMSTRONG ... | Lance ... |

**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 cidessous :

Sortie de données | Expliquer (Explain) | Messages | Historique

| | Changement d association |
|---|---|
| 1 | Mise a jour du stagiaire : AMSTRONG Ancienne Asso : Cercle Escrime Nouvelle asso : Comite Regional |

majAssoStagiaire

General Definition Condition Commands SQL

```
SELECT (((((('Mise à jour du stagiaire '::text || (stagiaire.nom)::text) || ' Ancienne Asso '::text) || (association.noma)::text) || ' Nouvelle asso '::text) || (( SELECT stagiaire.nom
          FROM "BdForma".association association_1
         WHERE (association_1.idassociation = new.idassociation)))::text) AS "Changement d'association"
  FROM ("BdForma".stagiaire
    JOIN "BdForma".association USING (idassociation))
 WHERE (stagiaire.idstagiaire = new.idstagiaire)
```

No limit

Query Query History

```
1  set schema 'BdForma';
2
3  UPDATE stagiaire SET idassociation = 100 WHERE idstagiaire = 7;
```

Data Output Messages Notifications

SQL

| | Changement d'association<br>text |
|---|---|
| 1 | Mise à jour du stagiaire : AMSTRONG Ancienne Asso : Cercle Escrime Nouvelle asso : AMSTRONG |