

Module SLAM 3 TP4

Propriétés	Description
Intitulé	PROGRAMMATION AVEC POSTGRESQL
Outils	• A.G.L : WIN'DESIGN • SGBD : POSTGRESQL
Durée estimée en heures	6 Heures
Savoir-faire module SLAM3	• 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	• Modèles de représentation des données • Langage de programmation associé à un SGBD
Documents joints	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
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 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, veuillez à contrôler que l'adresse IP de celle-ci est dans un adressage IP **différent** de votre Hôte. Si ce n'est pas le cas, **procéder aux modifications** pour paramétrer l'adresse IP de la carte réseau de la machine virtuelle (172.16.xx.2) en fonction de l'adresse IP de votre machine hôte (192.168.xx.1)
 - o Exemple : VM : 172.24.123.254 (IP/DHCP géré par le commutateur virtuel d'Hyper-V) versus l'hôte : 192.168.1.100 (IP/DHCP fournit par la box)
- Vérifier l'accès à Internet à partir de la machine virtuelle pour effectuer les opérations nécessaires.

2. Accès au serveur de base de données distant :

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

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

A faire

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



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

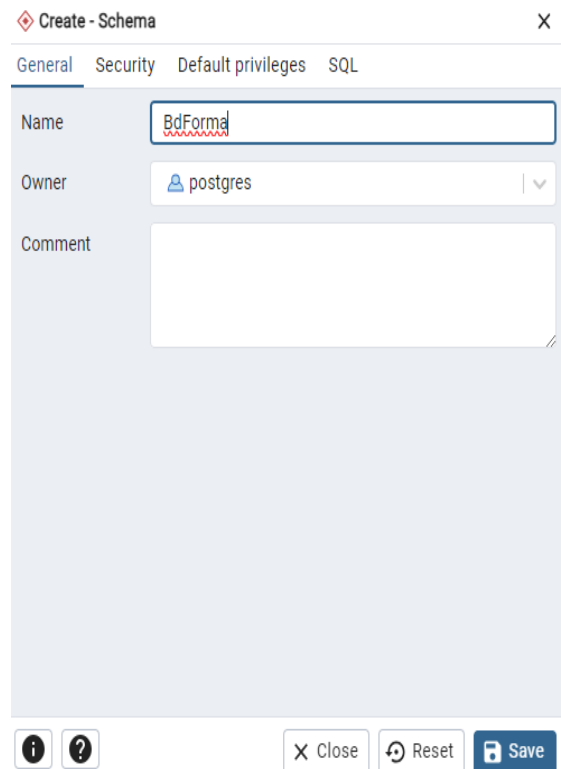
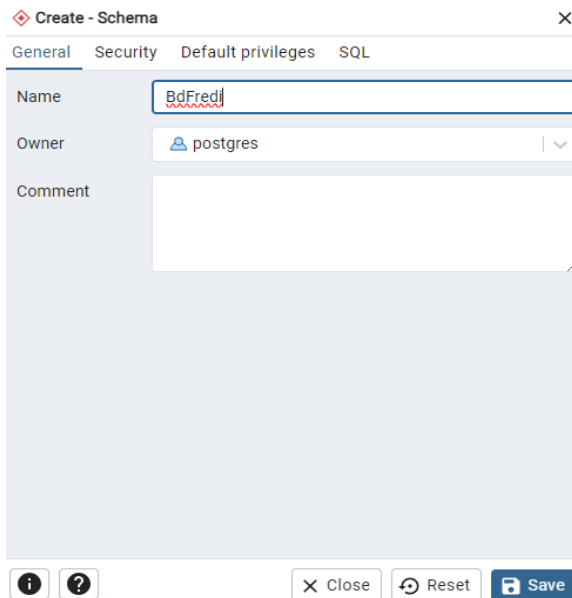
3. Installation de la base de données distante :

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

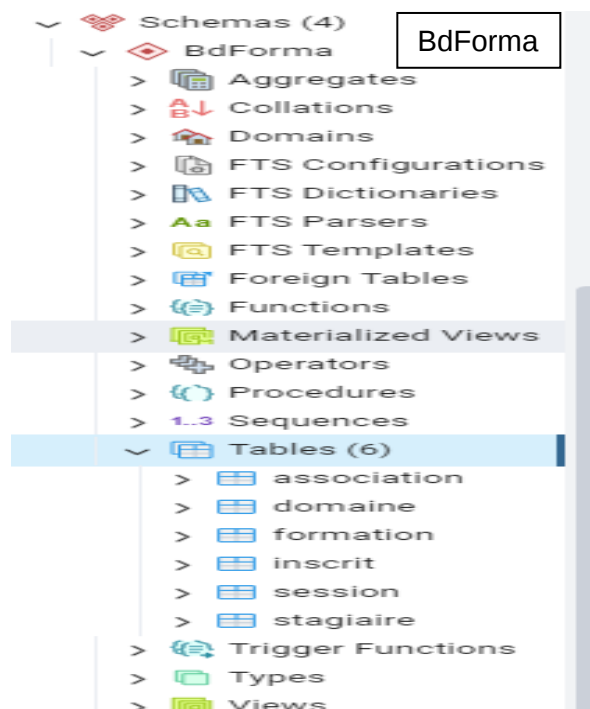
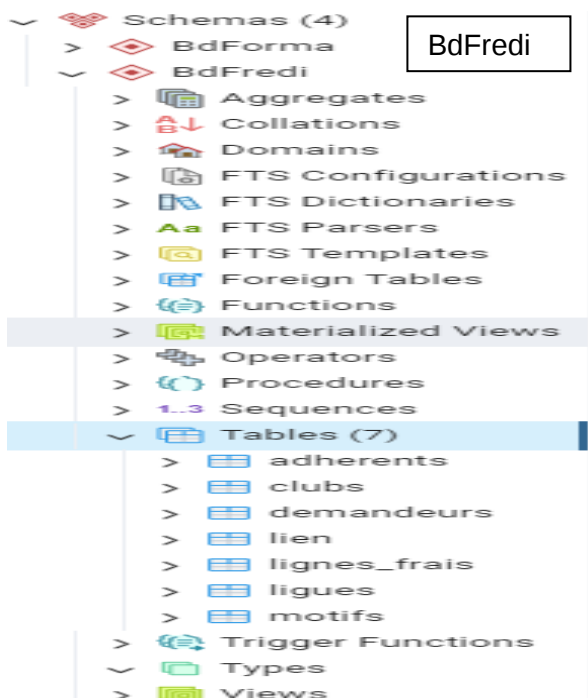
A faire

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

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



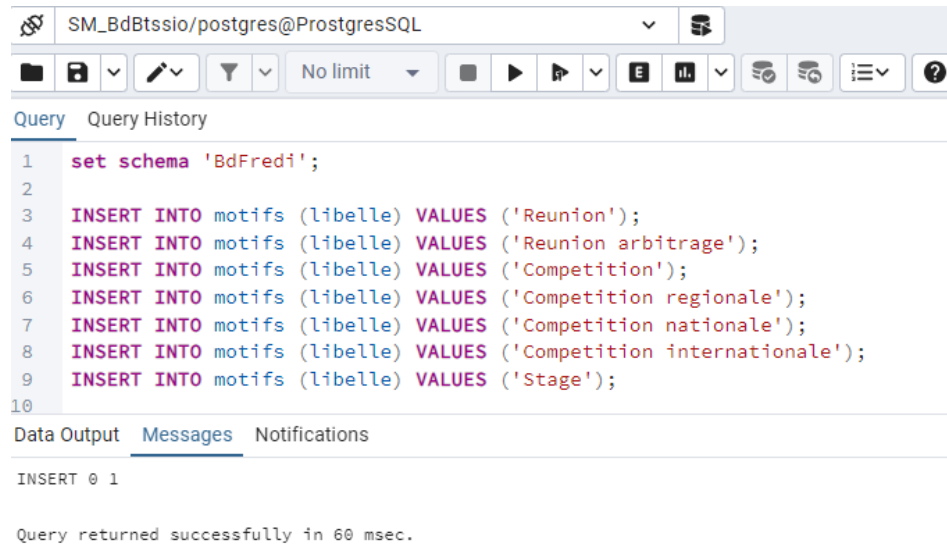
- Exécuter le script de création **BdpgFredri.sql** ou **BdpgForma.sql**



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

BDFREDI :

Insertion motifs



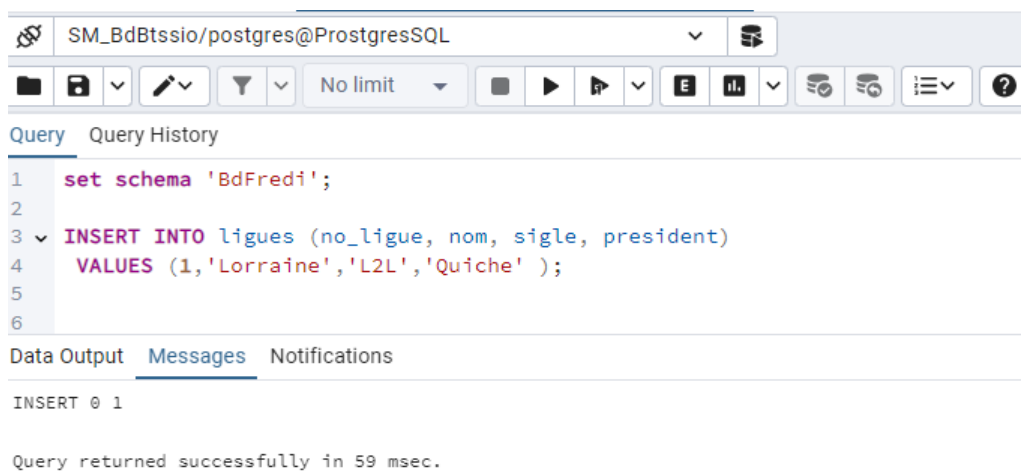
The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'SM_BdBtssio/postgres@ProstgresSQL'. The toolbar includes icons for file operations, query execution, and settings. The 'Query' tab is active, displaying a SQL script with 10 lines. The script sets the schema to 'BdFredI' and inserts seven records into the 'motifs' table. The 'Messages' tab is also visible, showing a confirmation message: 'INSERT 0 1'. A status message at the bottom states 'Query returned successfully in 60 msec.'

```
1 set schema 'BdFredI';
2
3 INSERT INTO motifs (libelle) VALUES ('Reunion');
4 INSERT INTO motifs (libelle) VALUES ('Reunion arbitrage');
5 INSERT INTO motifs (libelle) VALUES ('Competition');
6 INSERT INTO motifs (libelle) VALUES ('Competition regionale');
7 INSERT INTO motifs (libelle) VALUES ('Competition nationale');
8 INSERT INTO motifs (libelle) VALUES ('Competition internationale');
9 INSERT INTO motifs (libelle) VALUES ('Stage');
10
```

INSERT 0 1

Query returned successfully in 60 msec.

Insertion ligues



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'SM_BdBtssio/postgres@ProstgresSQL'. The toolbar includes icons for file operations, query execution, and settings. The 'Query' tab is active, displaying a SQL script with 6 lines. The script sets the schema to 'BdFredI' and inserts one record into the 'ligues' table. The 'Messages' tab is also visible, showing a confirmation message: 'INSERT 0 1'. A status message at the bottom states 'Query returned successfully in 59 msec.'

```
1 set schema 'BdFredI';
2
3 INSERT INTO ligues (no_ligue, nom, sigle, president)
4 VALUES (1, 'Lorraine', 'L2L', 'Quiche' );
5
6
```

INSERT 0 1

Query returned successfully in 59 msec.

Insertion clubs

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'SM_BdBTssio/postgres@ProstgresSQL'. Below the toolbar, the 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdFredI';
2
3  INSERT INTO clubs (num_club, no_ligue, nom_club) VALUES (1, 1, 'Salle Armes de Villers les Nancy' );
4
5
6

```

Below the query, the 'Data Output' tab shows the result: 'INSERT 0 1'. A message at the bottom states: 'Query returned successfully in 56 msec.'

Insertion demandeurs

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'SM_BdBTssio/postgres@ProstgresSQL'. Below the toolbar, the 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdFredI';
2
3  INSERT INTO demandeurs(
4      adresse_mail, nom, prenom, rue, cp, ville, num_recu, motdepasse)
5      VALUES ('jc.berbier@gmail.com', 'Berbier', 'Jean-Christophe', '12 rue des marrons', '54600', 'Villers-Les-Nancy', 0, 'jcb');
6  INSERT INTO demandeurs(
7      adresse_mail, nom, prenom, rue, cp, ville, num_recu, motdepasse)
8      VALUES ('r.becker@gmail.com', 'Becker', 'Romain', '1 rue des mesanges', '54600', 'Villers-Les-Nancy', 0, 'rb');
9
10

```

Below the query, the 'Data Output' tab shows the result: 'INSERT 0 1'. A message at the bottom states: 'Query returned successfully in 88 msec.'

Insertion lignes frais

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'SM_BdBTssio/postgres@ProstgresSQL'. Below the toolbar, the 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdFredI';
2
3  INSERT INTO lignes_frais VALUES ('r.becker@gmail.com', '2012-02-02', 'Reunion arbitrage', 'Nancy-Paris', 600, 0, 0, 0, 0, 0, 0);
4  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-01-12', 'Compétition', 'Nancy-Strasbourg', 280, 24, 0, 0, 0, 0, 0);
5  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-03-13', 'Compétition', 'Nancy-St-Avoid', 150, 0, 0, 0, 0, 0, 0);
6  INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '2012-06-24', 'Compétition', 'Nancy-Marseille', 1500, 180, 100, 120, 0, 0, 0);
7  );
8
9

```

Below the query, the 'Data Output' tab shows the result: 'INSERT 0 1'. A message at the bottom states: 'Query returned successfully in 56 msec.'

Insertion adherents

```

1  set schema 'BdFredf!';
2
3  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 443', 1, 'BANDILLELLA', 'CLEMENT', 'M
4  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 340', 1, 'BERBIER', 'LUCILLE', 'F
5  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 338', 1, 'BERBIER', 'THEO', 'M
6  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 309', 1, 'BECKER', 'ROMAIN', 'M
7  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 334', 1, 'BIAQUEL', 'VERONIQUE', 'F
8  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 399', 1, 'BIDELOT', 'BRIQUETTE', 'F
9  INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 442', 1, 'BIDELOT', 'JULIE', 'F
10 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 308', 1, 'BILLOT', 'DIDIER', 'M
11 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 329', 1, 'BILLOT', 'CLAIRE', 'F
12 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 254', 1, 'BILLOT', 'MARIANNE', 'F
13 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 407', 1, 'BINNET', 'MARIUS', 'M
14 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 444', 1, 'CALDI', 'THOMAS', 'M
15 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 431', 1, 'CASTEL', 'TINOTHE', 'M
16 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 428', 1, 'CHEOLLE', 'NICOLAS', 'M
17 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 414', 1, 'CHERRON', 'UGO', 'M
18 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 441', 1, 'CHEVOITINE', 'LOUIS', 'M
19 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 440', 1, 'CHOUARNO', 'TOM', 'M
20 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 402', 1, 'COTIN', 'FLORIAN', 'M
21 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 351', 1, 'DEPERRIN', 'ARNAUD', 'M
22 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 409', 1, 'DEPRETTE', 'BEATRICE', 'F
23 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 446', 1, 'DUCRICK', 'AUGUSTIN', 'M
24 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 395', 1, 'GARBILLON', 'GILLES', 'M
25 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 339', 1, 'GARBILLON', 'YANN', 'M
26 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 382', 1, 'HAGENBACH', 'CLEMENTINE', 'F
27 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 420', 1, 'HASFELD', 'AUXANE', 'F
28 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 341', 1, 'HUMERT', 'ISABELLE', 'F
29 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 432', 1, 'LAFIEGLON', 'CLEMENT', 'M
30 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 429', 1, 'LAMOINE', 'GREGOIRE', 'M
31 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 419', 1, 'LANIELLE', 'NICOLAS', 'M
32 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 401', 1, 'LIEVIN', 'NATHAN', 'M
33 INSERT INTO adherents (numero_licence, num_club, nom, prenom, sexe, date_naiss, rue, cp, ville) VALUES (' 17 05 40 010 439', 1, 'LOTANG', 'CYPRIEN', 'M

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 56 msec.

Insertion lien

```

1  set schema 'BdFredf!';
2
3  INSERT INTO lien VALUES (' 17 05 40 010 338', 'jc.berbier@gmail.com');
4  INSERT INTO lien VALUES (' 17 05 40 010 340', 'jc.berbier@gmail.com');
5  INSERT INTO lien VALUES (' 17 05 40 010 309', 'r.becker@gmail.com');
6
7

```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 94 msec.

BDFORMA :**Insertion domaine**

```

1  set schema 'BdForma';
2
3  INSERT INTO domaine (IDDOMAINE, NOM) VALUES
4  (1, 'Gestion'),
5  (2, 'Informatique'),
6  (3, 'Développement durable'),
7  (4, 'Secourisme'),
8  (5, 'Communication');
9

```

Data Output Messages Notifications

INSERT 0 5

Query returned successfully in 57 msec.

Insertion formation

```

1  set schema 'BdForma';
2
3  INSERT INTO formation (IDFORMATION, IDDOMAINE, TITRE, PUBLICFORM, CONTENU, COUT, OBJECTIFS) VALUES
4  (10, 1, 'Soirée d'information sur la convention collective nationale du sport', 'Bénévoles et salariés', 'La convention collective du sport règle, sur l'ensemble du territoire y compr
5  (11, 1, 'Actualisation des connaissances sur la convention collective nationale du sport et la responsabilité des dirigeants.', 'Bénévoles et salariés', 'L'avenant 88 de la CCNS', 70,
6  (12, 1, 'Comptabilité', 'Bénévoles et salariés', 'La comptabilité est une discipline pratique', 120, 'Apprendre les bases de la comptabilité en entreprise'),
7  (13, 1, 'Recherche de partenariat', 'Bénévoles et salariés', 'Le partenariat se définit comme une association active de différents intervenants', 60, 'Comprendre les rouages dans la re
8  (20, 2, 'Outlook Niveau 1', 'Bénévoles et salariés', 'Configurer Outlook, paramétrer les notifications dans Outlook', 70, 'Prendre en main l'outil Outlook'),
9  (21, 2, 'Outlook Niveau 2', 'Bénévoles et salariés', 'Initiation au VBA d'Outlook, configuration pour un serveur IMAP', 90, 'Perfectionnement sur l'outil Outlook'),
10 (22, 2, 'Power Point Niveau 1', 'Bénévoles et salariés', 'Introduction à powerpoint\r\nMasques de diapositive et arrière-plan', 50, 'Parfaire ses connaissances sur PowerPoint'),
11 (23, 2, 'Photoshop Niveau 1', 'Bénévoles et salariés', 'Rappel images numériques, modes colorimétriques, présentation et personnalisation', 80, 'Parfaire ses connaissances sur Photoshop
12 (24, 2, 'Photoshop Niveau 2', 'Bénévoles et salariés', 'Retouches photos, principes de bases d'impression', 120, 'Parfaire ses connaissances sur Photoshop'),
13 (30, 3, 'Organiser une manifestation éco responsable', 'Bénévoles et salariés', 'La responsabilité environnementale (écoresponsabilité ou la responsabilité humaine dans l'habitat)', 8
14 (40, 4, 'Prévention et secours civique (PSC)', 'Bénévoles et salariés', 'Les situations d'accident sont abordées en modules', 110, 'Initiation à la réduction des risques'),
15 (41, 4, 'Bonnes pratiques et premiers secours', 'Bénévoles et salariés', 'Au programme de cette initiation : \r\n\r\nPrise de conscience de l'existence des risques', 130, 'Apprentissage
16 (50, 5, 'Conduite en réunion', 'Bénévoles et salariés', 'Permettre à chacun de s'exprimer', 50, 'Savoir-être en réunion'),
17 (51, 5, 'Communiquer avec la presse', 'Bénévoles et salariés', 'Quel message véhiculer? Après de quels journalistes?', 70, 'Communiquer avec la presse c'est facile !'),
18 (52, 5, 'Langues étrangères', 'Bénévoles et salariés', 'Anglais, Chinois, Espagnol, Russe, Japonais', 30, 'Se perfectionner dans les langues étrangères');
19
20

```

Data Output Messages Notifications

INSERT 0 15

Query returned successfully in 78 msec.

Insertion session

```

1  set schema 'BdForma';
2
3  |
4  INSERT INTO session (IDSESSION, IDFORMATION, JOUR, DATESESSION, INTERVENANT, DATELIMITEINSCRIPTION, NBPLACESRESTANTES, NBPLACESMAX) VALUES
5  (1, 11, 'Mercredi', '2015-01-14', 'A Contador', '2014-12-31', 24, 25),
6  (1, 12, 'Samedi', '2015-01-24', 'C Froome', '2015-01-03', 28, 30),
7  (1, 21, 'Mardi', '2015-01-20', 'C Froome', '2015-01-05', 10, 10),
8  (1, 30, 'Mercredi', '2015-03-18', 'A Contador', '2015-03-04', 8, 8),
9  (2, 11, 'Vendredi', '2015-02-20', 'V Nibali', '2015-02-06', 8, 8),
10 (2, 12, 'Jeudi', '2015-03-26', 'A Contador', '2015-03-12', 7, 7),
11 (2, 21, 'Vendredi', '2015-02-20', 'V Nibali', '2015-02-06', 9, 9),
12 (3, 11, 'Lundi', '2015-01-30', 'V Nibali', '2015-01-10', 70, 70),
13 (3, 12, 'Mardi', '2015-01-25', 'C Froome', '2015-01-10', 80, 80),
14 (4, 11, 'Jeudi', '2015-02-07', 'A Contador', '2015-01-20', 40, 40),
15 (5, 11, 'Lundi', '2015-02-01', 'C Froome', '2015-01-15', 90, 90);

```

Data Output Messages Notifications

INSERT 0 11

Query returned successfully in 57 msec.

Insertion association

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is 'SM_BdBtssio/postgres@ProstgresSQL'. The 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdForma';
2
3  INSERT INTO association (IDASSOCIATION, NOMA, NOICOM, NOMI, PRENOMI) VALUES
4  (100, 'Cercle Escrime', 15484758, 'Bidart', 'Julien' ),
5  (101, 'Comite Regional Olympique et Sportif de Lorraine', 16584785, 'Giroux', 'Micheline');
6

```

Below the query, the 'Messages' tab is active, showing the execution result:

```

INSERT 0 2

Query returned successfully in 56 msec.

```

Insertion stagiaire

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is 'SM_BdBtssio/postgres@ProstgresSQL'. The 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdForma';
2
3  --
4  -- Contenu de la table `stagiaire`
5  --
6
7  INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS) VALUES
8  (1, 101, 'PINOT', 'Thibaut', 'Benevole', 'Interlocuteur', 'pinot@gmail.com', 1),
9  (2, 100, 'BARDET', 'Romain', 'Salarie', 'Administrateur', 'bardet@gmail.com', 1),
10 (3, 100, 'VUILLERMOZ', 'Alexis', 'Salarie', 'Directeur du pole formation', 'vuill@gmail.com', 2),
11 (4, 101, 'DEMARRE', 'Arnaud', 'Benevole', 'Développeur', 'demarre@gmail.com', 0 ),
12 (5, 101, 'ALAPHILIPPE', 'Julian', 'Benevole', 'Développeur', 'alaphil@gmail.com', 1),
13 (6, 100, 'POULIDOR', 'Raymond', 'Benevole', 'Développeur', 'poupou@gmail.com', 0);

```

Below the query, the 'Messages' tab is active, showing the execution result:

```

INSERT 0 6

Query returned successfully in 55 msec.

```

Insertion inscrit

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is 'SM_BdBtssio/postgres@ProstgresSQL'. The 'Query' tab is active, displaying the following SQL code:

```

1  set schema 'BdForma';
2
3  --
4  -- Contenu de la table `inscrit`
5  --
6
7  INSERT INTO inscrit (IDSTAGIAIRE, IDFORMATION, IDSESSION) VALUES
8  (1, 11, 1 ),
9  (2, 12, 1 ),
10 (3, 12, 1 );

```

Below the query, the 'Messages' tab is active, showing the execution result:

```

INSERT 0 3

Query returned successfully in 51 msec.

```

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

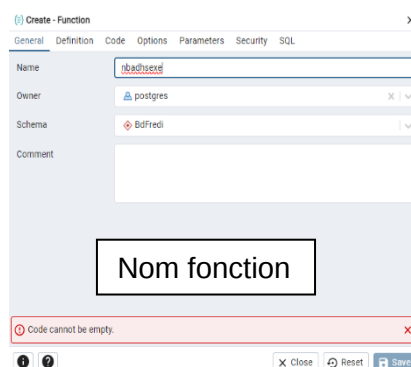
```
DROP FUNCTION nomfonction ();
```

A faire avec BdFredri :

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
```

nbadhsexe(lettre "char") X

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
```

X Close Reset Save

2. Utilisation de la fonction

Dans l'éditeur SQL exécuter la fonction par :

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

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdFred';
2
3 select nbadhsexe('F');
4
```

Data Output Messages Notifications

nbadhsexe integer	
1	12

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdFred';
2
3 select nbadhsexe('M');
4
```

Data Output Messages Notifications

nbadhsexe integer	
1	28

A faire avec BdForma:

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

1. Création de la fonction

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

The image shows two screenshots of the PgAdmin 3 'Create Function' dialog. The left screenshot shows the 'General' tab with fields for Name (nbstagstatut), Owner (postgres), and Schema (BdFred). The right screenshot shows the 'Definition' tab for the function 'nbstagstatut(etat character)', with Return type set to 'integer', Language set to 'plpgsql', and one argument named 'etat' of type 'character' in 'IN' mode.

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

Query Query History

```

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

```

Data Output Messages Notifications

nbstagstatut integer	
1	4

SM_BdBTssio/postgres@ProstgresSQL

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

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

A faire avec BdFredri :

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

nbkmdem(nom_demandeurs character) X

General Definition Code Options Parameters Security SQL

Name: nbkmdem

Owner: postgres X | v

Schema: BdBredi | v

Comment:

nbkmdem(nom_demandeurs character) X

General Definition Code Options Parameters Security SQL

Return type: integer

Language: plpgsql X | v

Arguments

Data type	Mode	Argument name	Default
character v	IN v	nom_demandeurs	

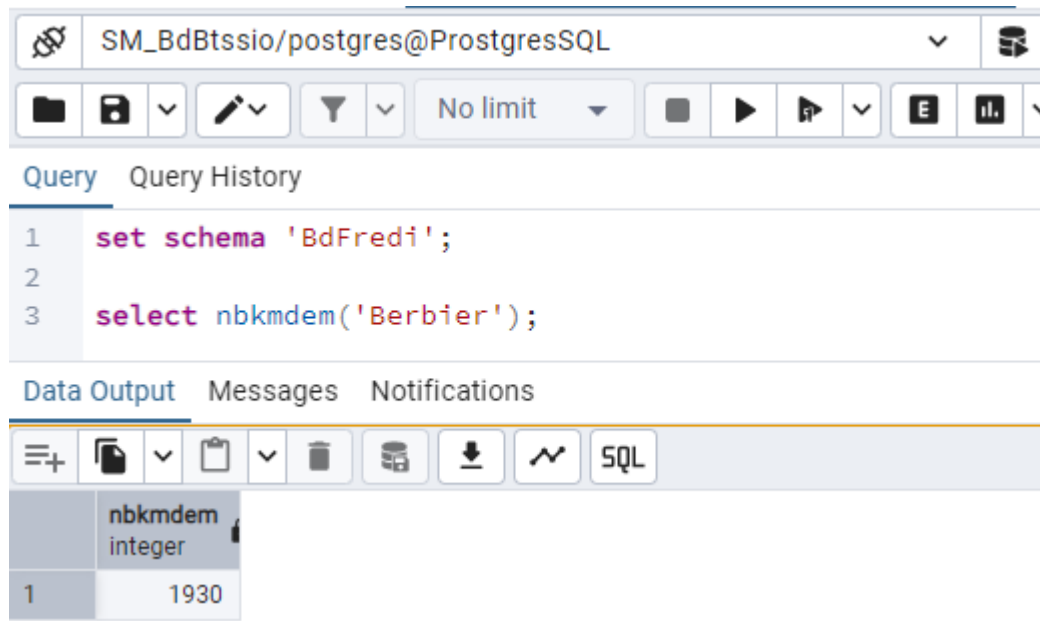
La fonction doit normalement avoir le contenu suivant :

```
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
```

nbkmdem(nom_demandeurs character) X

General Definition Code Options Parameters Security SQL

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

Résultats :


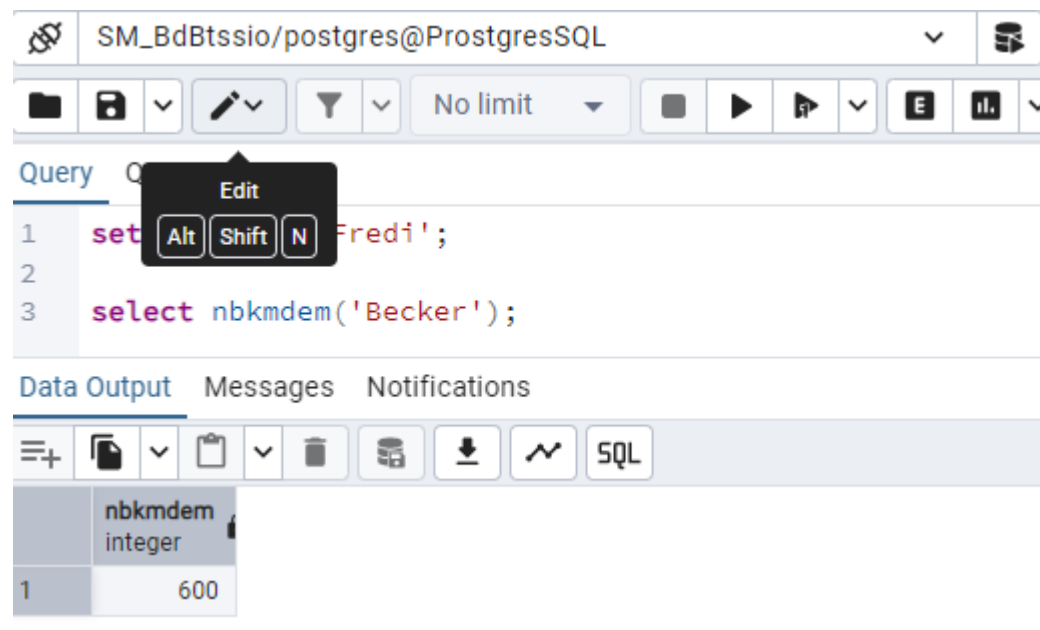
SM_BdBtssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdFredI';
2
3 select nbkmdem('Berbier');
```

Data Output Messages Notifications

	nbkmdem integer
1	1930



SM_BdBtssio/postgres@ProstgresSQL

Query Q

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

Data Output Messages Notifications

	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.

```

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;

```

coutkm(IN mail character, IN datef date)

General Definition Code Options Parameters Security SQL

```

1 DECLARE coutkm INT;
2 BEGIN
3     SELECT SUM(km*0.28) INTO coutkm FROM lignes_frais WHERE adresse_mail = mail AND date_frais = datef;
4     RAISE NOTICE 'TARIF : % ADRESSE : % DATE : %', coutkm, mail, datef;
5 RETURN coutkm;
6 END;

```

Test à faire :

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

SM_BdBtssio/postgres@ProstgreSQL

Query Query History

```

1 set schema 'BdFred';
2
3 select coutkm('r.becker@gmail.com','02/02/2012');

```

Data Output Messages Notifications

	coutkm integer
1	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 **nbseform** qui renverra le nombre de sessions de formation pour un domaine donné (on donne le nom en paramètre)

nbseform(IN nom_domaine character)

General Definition Code Options Parameters Security SQL

Name: nbseform

Owner: postgres

Schema: BdForma

Comment:

nbseform(IN nom_domaine character)

General Definition Code Options Parameters Security SQL

Return type: integer

Language: plpgsql

Arguments

Data type	Mode	Argument name	Default
character	IN	nom_domaine	

```

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;

```

nbseform(IN nom_domaine character)

General Definition Code Options Parameters Security SQL

```

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

```

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

caform(titref character, nbse integer)

General Definition Code Options Parameters Security SQL

Name: caform

Owner: postgres

Schema: BdForma

Comment:

caform(titref character, nbse integer)

General Definition Code Options Parameters Security SQL

Return type: integer

Language: plpgsql

Arguments

Data type	Mode	Argument name	Default
character	IN	titref	
integer	IN	nbse	

```

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 = nb ses;
    RAISE NOTICE 'CA TOTAL : % SESSION : % FORMATION : %', nbCa, nb ses,
    titref;
RETURN nbCa;
END;

```

 caform(titref character, nb ses integer)

General Definition **Code** Options Parameters Security SQL

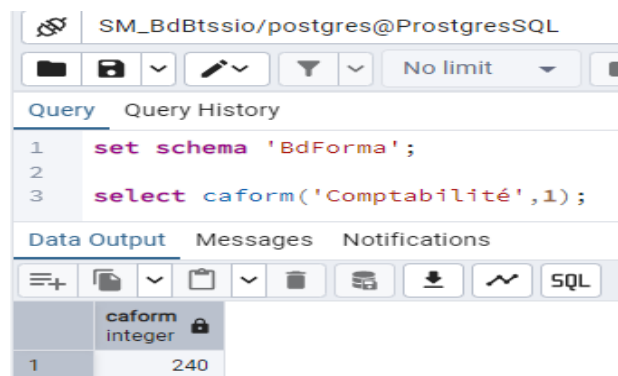
```

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 = nb ses;
6     RAISE NOTICE 'CA TOTAL : % SESSION : % FORMATION : %', nbCa, nb ses, 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 déclenchées 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 **BdFredri**
- 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;
```

```
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;
```

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

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

```
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 **BdFred**

Dashboard X Properties X SM_BdBTssio/post... X SM_BdBTssio/postgres@ProstgresSQL X BdFred.demandeu... X

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdFred';
2 INSERT INTO demandeurs VALUES ('e.gerardgmail.com','gerard','ethane','2 rue des mesanges','54600','Villers-Les-Nancys',0,'eg');
```

Data Output Messages Notifications

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

SQL state: P0001

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

Dashboard X Properties X SM_BdBTssio/post... X SM_BdBTssio/post... X BdFred.demandeu... X SM_BdBTssio/postgres@Prostgr

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdForma';
2 INSERT INTO stagiaire VALUES ('7','101','GERALD','White','Benevole','Acheteur','gerald-whitegmail.com','1')
```

Data Output Messages Notifications

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

SQL state: P0001

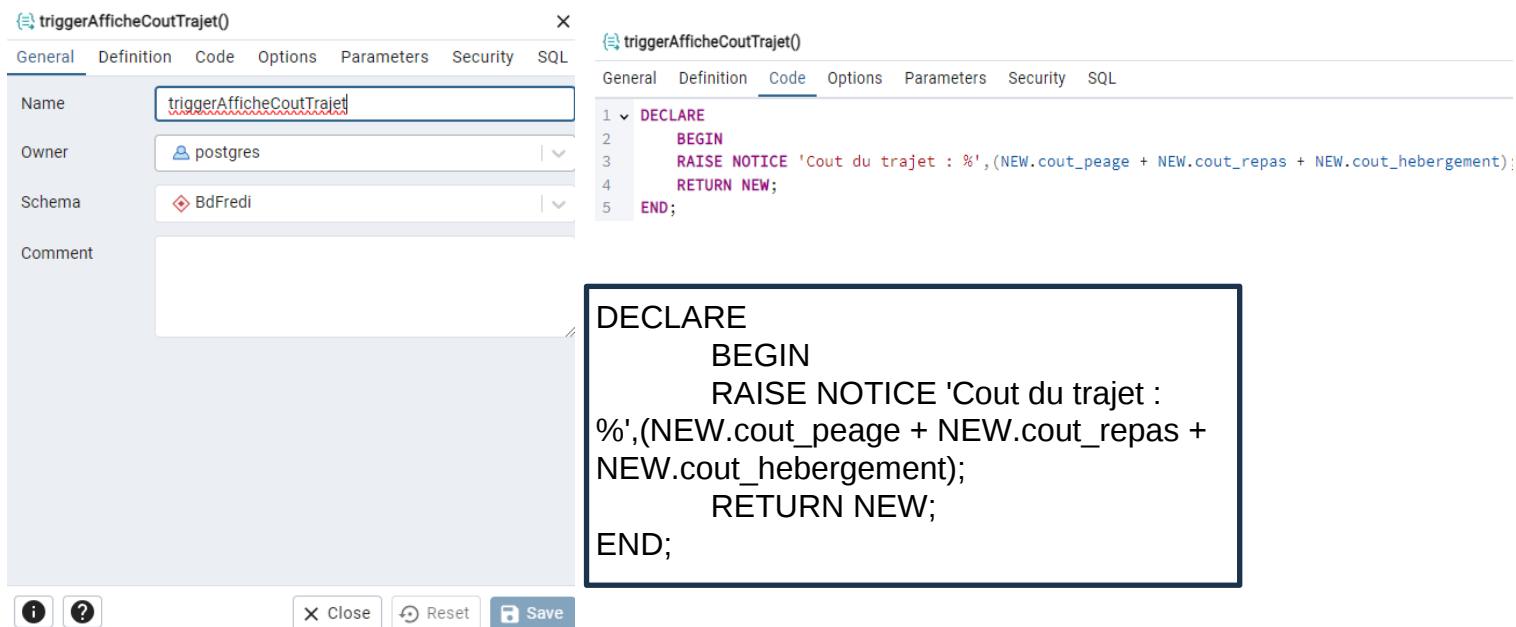
Etape 2 - Création d'un trigger

Vous pouvez créer les triggers suivants des 2 façons suivantes :

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

A faire avec BdBredi :

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



The screenshot shows the pgAdmin 4 interface for creating a trigger. The left pane shows the 'triggerAfficheCoutTrajet()' object with the following properties:

- Name: triggerAfficheCoutTrajet
- Owner: postgres
- Schema: BdBredi
- Comment: (empty)

The right pane shows the SQL code for the trigger function:

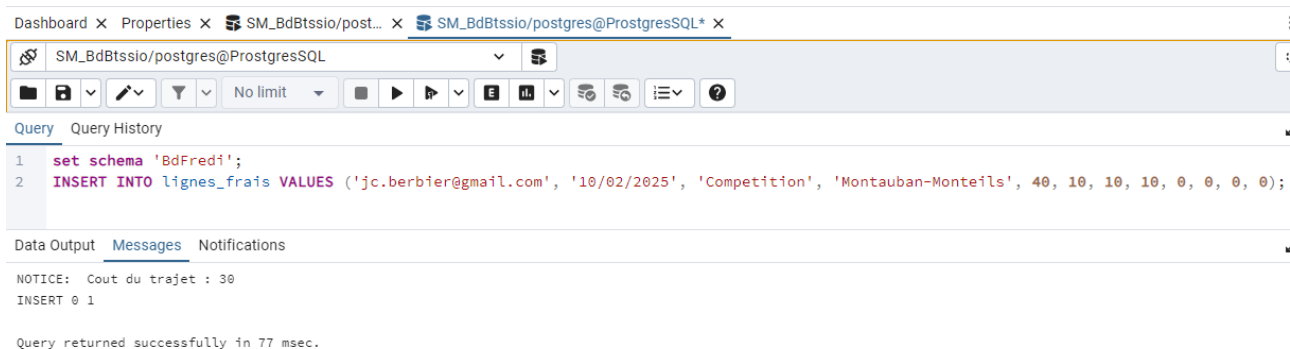
```
1 DECLARE
2 BEGIN
3 RAISE NOTICE 'Cout du trajet : %',(NEW.cout_peage + NEW.cout_repas + NEW.cout_hebergement);
4 RETURN NEW;
5 END;
```

A callout box highlights the function code:

```
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 "BdBredi".lignes_frais
FOR EACH ROW
EXECUTE FUNCTION "BdBredi"."triggerAfficheCoutTrajet"();
```

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



The screenshot shows the pgAdmin 4 query window with the following SQL code:

```
1 set schema 'BdBredi';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '10/02/2025', 'Compétition', 'Montauban-Montetils', 40, 10, 10, 10, 0, 0, 0, 0);
```

The output shows the following messages:

```
NOTICE: Cout du trajet : 30
INSERT 0 1
```

The query returned successfully in 77 msec.

A faire :

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

triggerVerifKm()

General Definition Code Options Parameters Security SQL

Name: triggerVerifKm

Owner: postgres

Schema: BdFred

Comment:

triggerVerifKm()

General Definition Code Options Parameters Security SQL

```

1 DECLARE
2 BEGIN
3 IF (NEW.km > 0 AND NEW.cout_peage > 0 ) THEN
4 RAISE NOTICE 'Votre kilométrage parcouru est supérieur 0';
5 ELSE
6 RAISE NOTICE 'Votre kilométrage parcouru est bien inférieur à 0';
7 END IF;
8 RETURN NEW;
9 END;
10

```

```

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 "BdFred".lignes_frais
FOR EACH ROW
EXECUTE FUNCTION "BdFred"."triggerVerifKm";

```

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

SM_BdBTssio/postgres@ProstgresSQL

Query History

```

1 set schema 'BdFred';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '10/02/2096', 'Compétition', 'Montauban-Montetils', 100, 10, 10, 10, 0, 0, 0, 0);

```

Data Output Messages Notifications

```

NOTICE: Votre kilométrage parcouru est supérieur 0
NOTICE: Cout du trajet : 30
INSERT 0 1

```

Query returned successfully in 78 msec.

```

1 set schema 'BdFred1';
2 INSERT INTO lignes_frais VALUES ('jc.berbier@gmail.com', '10/02/2016', 'Compétition', 'Montauban-Montetils', -1, 10, 10, 10, 0, 0, 0, 0);

```

Data Output Messages Notifications

NOTICE: Votre kilométrage parcouru est bien inférieur à 0
 NOTICE: Cout du trajet : 30
 INSERT 0 1

Query returned successfully in 78 msec.

En Bonus :

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

Create - Trigger function

General Definition Code Options Parameters Security

Name triggerVerifPlaces

Owner postgres

Schema BdForma

Comment

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

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
```

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```

1  set schema 'BdForma';
2
3  INSERT INTO session VALUES (6, 13, 'Mercredi', '2025-10-04', 'C Froome', '2025-10-18', 200, 100);

```

Data Output Messages Notifications

NOTICE: Il y a plus de places disponible.
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.

Create - Trigger function

General Definition Code Options Parameters Security SQL

Name: triggerAffichePlacesRest

Owner: postgres

Schema: BdForma

Comment:

triggerAfficheVerifPlacesRest()

General Definition Code Options Parameters Security SQL

```

1  DECLARE
2  placesnb INT;
3  BEGIN
4      SELECT session.nbplacesrestantes INTO placesnb FROM session
5      WHERE session.idsession = NEW.idsession AND session.idformation = NEW.idformation;
6  IF (placesnb = 0) THEN
7      RAISE EXCEPTION 'Il y a plus de places';
8  ELSE
9      RAISE NOTICE 'Nombre de places restantes : %',placesnb;
10  END IF;
11  RETURN NEW;
12  END;

```

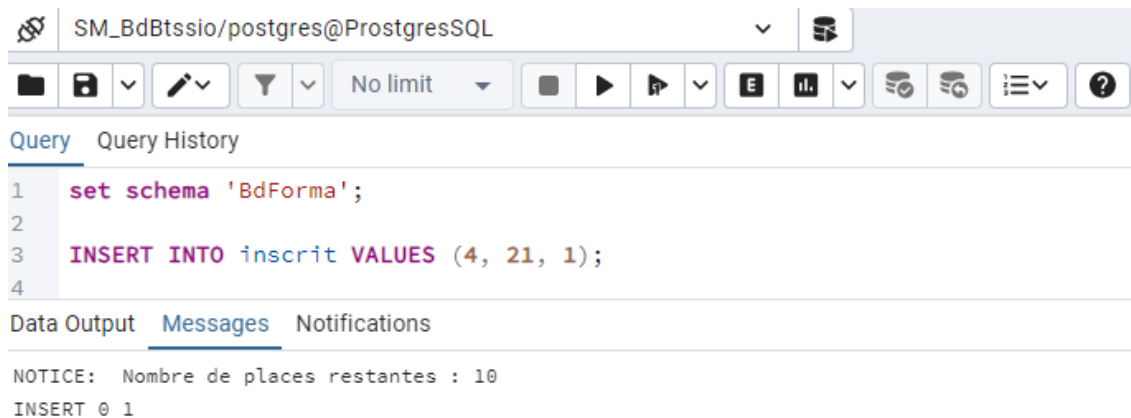
```

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.

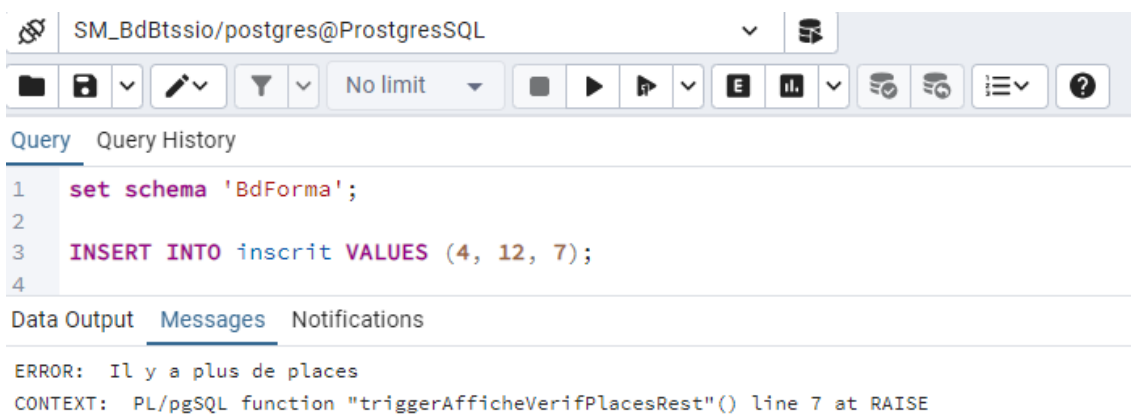


The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is 'SM_BdBtssio/postgres@ProstgresSQL'. The query editor contains the following SQL code:

```
1 set schema 'BdForma';
2
3 INSERT INTO inscrit VALUES (4, 21, 1);
4
```

Below the query editor, the 'Messages' tab is selected, showing the following output:

```
NOTICE: Nombre de places restantes : 10
INSERT 0 1
```



The screenshot shows the same PostgreSQL query editor interface. The query editor contains the following SQL code:

```
1 set schema 'BdForma';
2
3 INSERT INTO inscrit VALUES (4, 12, 7);
4
```

Below the query editor, the 'Messages' tab is selected, showing the following error message:

```
ERROR: Il y a plus de places
CONTEXT: PL/pgSQL function "triggerAfficheVerifPlacesRest"() line 7 at RAISE
```

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.

 triggerVerifNbForm() X

General Definition Code Options Parameters Security SQL

Name

Owner  postgres | v

Schema  BdForma | v

Comment

 Create - Trigger function

General Definition **Code** Options Parameters Security SQL

```

1  DECLARE
2      nbFormation int;
3      nbFormationStag int;
4
5  BEGIN
6      SELECT nbFormation INTO nbFormation FROM stagiaire WHERE idstagiaire = NEW.idstagiaire;
7      SELECT COUNT(*) INTO nbFormationStag FROM inscrit WHERE idstagiaire = NEW.idstagiaire;
8      IF nbFormationStag >= nbFormation THEN
9          RAISE EXCEPTION 'Le stagiaire a dépassé le nombres de formation';
10     END IF;
11     RETURN NEW;
12 END;
```

```

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	<code>set schema 'BdForma';</code>
2	
3	<code>INSERT INTO inscrit VALUES (2,13,1);</code>
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
Name	<u>triggerVerifDateIns</u>					
Owner	 postgres ▼					
Schema	 BdForma ▼					
Comment						

triggerVerifDateIns()

General Definition Code Options Parameters Security SQL

```

1  DECLARE
2  datelimit DATE;
3  dateauj DATE;
4  BEGIN
5  SELECT datelimitinscription INTO datelimit FROM session
6  WHERE NEW.idinformation = idinformation AND NEW.idsession = idsession;
7  SELECT CURRENT_DATE INTO dateauj FROM session;
8  IF ( dateauj > datelimit ) 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

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**

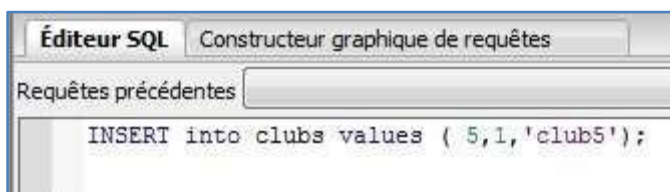
```
CREATE OR REPLACE RULE "insertClub" AS
  ON INSERT TO clubs
  DO
  SELECT ' Ajout du club ' || new.nom_club
  AS " Insertion club "
  FROM clubs
  WHERE clubs.num_club = new.num_club;
```

SM_BdBtssio/postgres@ProstgresSQL

Query Query History

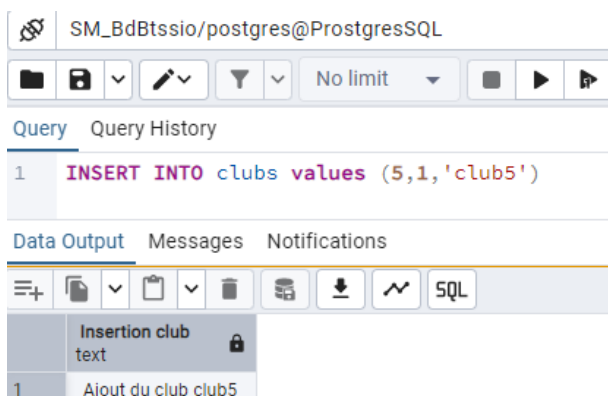
```
1 set schema 'BdFredI';
2
3 CREATE OR REPLACE RULE "insertClub" AS
4   ON INSERT TO clubs
5   DO
6   SELECT ' Ajout du club ' || new.nom_club
7   AS " Insertion club "
8   FROM clubs
9   WHERE clubs.num_club = new.num_club;
```

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



Panneau sortie

Sortie de données		Expliquer (Exp)
	Insertion club text	
1	Ajout du club club5	



- 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

SM_BdBtssio/postgres@ProstgresSQL

Query Query History

```
1 set schema 'BdFredI';
2
3 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 UFOLEP Lorraine

InsertClub

General Definition Condition Commands SQL

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

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;
```

```

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.

```

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	

```

1  set schema 'BdForma';
2
3  INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS) VALUES (7,101,'AMSTRONG','Lance', 'Benevole', 'Docteur', 'lance@gmail.com',1);

```

Insertion stagiaire text
1 Ajout du stagiaire AMSTRONG

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

```

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 1 association Cercle Escrime

InsertStagiaire

General Definition Condition Commands SQL

```

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

```

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```

1 set schema 'BdForma';
2
3 INSERT INTO stagiaire (IDSTAGIAIRE, IDASSOCIATION, NOM, PRENOM, STATUT, FONCTION, ADRESSE_MAIL, NBFORMATIONS) VALUES (8,100,'FIGNON','Laurent', 'Benevole', 'Soigneur', 'fignon@gmail.com',1);

```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

Insertion stagiaire
Ajout du stagiaire FIGNON association Cercle Escrime

Etape 2 - Création d'une règle

A faire sur BdFredri :

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

SM_BdBTssio/postgres@ProstgresSQL

Query Query History

```

1 set schema 'BdFredri';
2
3 CREATE OR REPLACE RULE "majLigueClub" AS
4 ON UPDATE TO clubs DO
5     SELECT 'Mise à jour du club ' || NEW.nom_club::text ||
6           ' ancienne ligue ' || OLD.no_ligue ||
7           ' nouvelle ligue ' || NEW.no_ligue AS "Nouveau club"
8 FROM clubs
9 WHERE clubs.num_club = NEW.num_club;

```

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 'BdFred';
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

Après 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

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

Query **Query History**

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

Data Output Messages Notifications

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**

```
1 set schema 'BdForma';
2
3 CREATE OR REPLACE RULE "majAssoStagiaire " AS
4 ON UPDATE TO stagiaire DO
5 SELECT 'Mise à jour du stagiaire : ' || NEW.nom::text ||
6 ' Ancienne Asso : ' || OLD.idassociation ||
7 ' Nouvelle asso : ' || NEW.idassociation AS "Changement d' association"
8 FROM stagiaire
9 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 character(25)
7	7	101	AMSTRONG	Lance	Benevo

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

Après 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

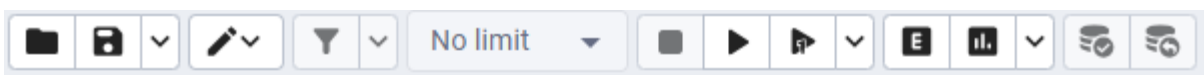
A faire : Modifier la règle **majAssoStagiaire** sur la table **stagiaire** qui à la place des numéros des associations affichera **le nom des associations** correspondantes comme dans l'exemple cidessous :

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

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



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 : Cercle Escrime Nouvelle asso : AMSTRONG