*Module M2106*

# Programmation
# et
# administration des bases de données

Semestre 2d - 2019/2020

IUT2
UNIVERSITÉ
Grenoble
Alpes
Département
INFO

# *Etude illustrée*

Cette étude illustrée :

- est réalisée en trinômes ;
- doit proposer la courte étude (2 pages environ) d'un sujet en rapport avec le module, à choisir parmi une liste présentée par l'enseignant ;
- cette étude doit être illustrée par des exemples que l'on peut appliquer facilement sur une base postgreSQL. Ces exemples doivent être originaux, mais peuvent être inspirés des exercices de TP s'il vous semble qu'ils illustrent bien votre propos ;
- Nous vous recommandons de lire attentivement les chapitres relatifs au sujet choisi, de la documentation officielle postgreSQL **https://docs.postgresql.fr/11/** La version anglaise peut être parfois plus claire **https://www.postgresql.org/docs/11/**
- Attention : vous devrez citer vos sources et ne pas faire de simples copier-coller.

Les documents à réaliser pour cette étude sont :

- un fichier PDF (tout autre format sera ignoré) comportant les noms des membres du trinôme, l'intitulé du sujet, une première partie présentant une étude du sujet qui cherchera à être la plus pédagogique possible (faites comme s'il s'agissait d'un court document à destination d'un de vos camarades), une deuxième partie illustrant cette étude par quelques exemples illustratifs ;
- des fichiers *drop.sql create.sql insert.sql* qui vont permettre à l'enseignant de créer la base de données utile pour les exemples illustratifs que vous avez imaginés (fournir ces fichiers même s'il s'agit des fichiers de TP *voile*);
- un fichier *demo.sql* qui contient les instructions SQL utilisées dans les exemples illustratifs, ainsi que quelques commentaires.

Le rendu se fera en suivant les instructions suivantes :

- créer un répertoire contenant les fichiers à rendre ;
- se positionner dans ce fichier (**`cd`**) ;
- exécuter la commande **`rendre-etude-m2106.`**

Si la commande est exécutée plusieurs fois, une confirmation vous sera demandée pour chaque fichier déjà présent. **Attention, l'accès au répertoire de rendu ne sera ouvert qu'en séance AA (ni avant, ni après)**.

La 1ère séance est consacrée à une première recherche personnelle : chaque étudiant doit rendre un seul fichier PDF (tout autre format sera ignoré) contenant une liste d'idées et de références sur le sujet choisi.

La 2ème séance est consacrée à la réalisation d'une première version d'étude illustrée. Les documents doivent être rendus en un seul exemplaire pour chaque trinôme, avant la fin de chaque séance. Seules les études complètes seront sélectionnées pour la phase suivante.

La 3ème séance est consacrée à la critique constructive d'une première version d'étude illustrée, rendue par un autre groupe à l'issue de la 2ème séance. A la fin de la 3ème séance, vous rendrez un fichier PDF d'1 page environ, composée de quatre parties ou paragraphes :

- le résumé de la notion présentée dans l'étude que vous analysez ;
- les qualités et défauts de la partie présentation de cette étude ;
- les qualités et défauts des exemples pédagogiques ;
- des conseils pour améliorer l'étude.

La note finale sera établie à partir de :

- l'avancée du travail rendu lors des deux premières séances ;
- la qualité de la critique argumentée et constructive rendue à la 3ème séance ;
- la qualité de la version finale de l'étude rendue à la 4ème séance.

# M2106 – Organisation du module

| Semaine | Cours | TD | TP | AA |
|---|---|---|---|---|
| 0 | Intro | Révisions | | |
| 1 | Procédures | | | |
| 2 | Curseurs | | | travail individuel |
| 3 | Curseurs (fin) + Triggers | | | 1ère version |
| 4 | Triggers (fin) | | | commentaires |
| 5 | User/Droits | | | version finale |
| 6 | | Révisions | | |

Evaluation : coefficient 25

- Etude illustrée: coefficient 8
  en trinômes, en séance de AA avec rendus électroniques.
- Contrôle final : coefficient 17
  en monômes, sur machine, vendredi 25 octobre.

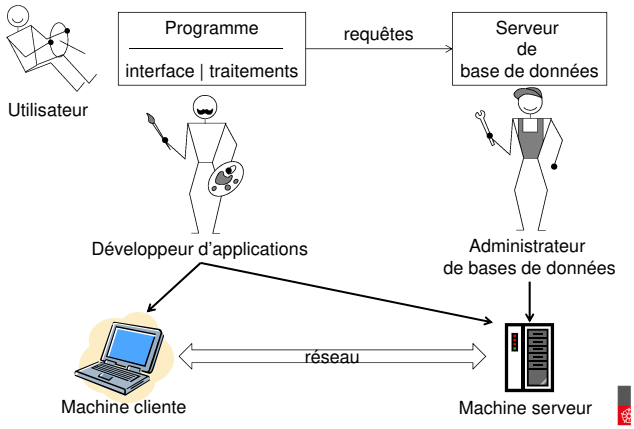
# M2106 – Plan du cours

I. Introduction

II. Programmation dans les bases de données
   1) Procédures stockées
   2) Curseurs
   3) Triggers

III. Administration des bases de données
   1) Utilisateurs
   2) Droits

# M2106 – Prog. et Admin. des BD




# M2106 – Prog. et Admin. des BD

Administrer une base de données : faire en sorte qu'elle puisse être utilisée dans de bonnes conditions.

1. Créer la base de données
   - ❖ Configurer et exécuter le programme « serveur de bases de données »
   - ❖ Définir les données grâce au SQL
2. Assurer une utilisation performante et sûre
   - ❖ performante : exécution rapide des requêtes
   - ❖ sûre

# M2106 – Prog. et Admin. des BD

Programme client : **`psql`**

- ✦**`-h`** hôte où s'exécute le programme serveur

Programme serveur : **`postgres`**

- ✦**`-D datadir`** : rép. contenant les données
- ✦**`-c config_file=`** fichier de configuration
  - o Paramètres physiques de la machine hôte
  - o Paramètres liés aux actions d'admin/maintenance
  - o Paramètres d'utilisation par les clients
- ✦1 processus père et des fils (1 par connexion), exécutés par le user unix postgres

# M2106 – Prog. et Admin. des BD

Bases définies par défaut :

- ✦template0 (modèle de sauvegarde)
- ✦template1 (modèle recopié pour créer une base)
- ✦postgres (pour une connexion par défaut)

Créer une nouvelle base :

```
CREATE DATABASE nom
                [with OWNER=owner];
```

## M2106 – Prog. et Admin. des BD

**CREATE** : définir l'organisation des données

- ✦En tables : quels noms?
- ✦Dans chaque table, quelles colonnes?
- ✦Pour chaque colonne, quelles contraintes?
  - o Type / contrôle de valeur
  - o Contrôle de la redondance (`PRIMARY/FOREIGN KEY`)

Pour définir d'autres contraintes externes

- ✦Triggers
- ✦Fonctions

## M2106 – Prog. et Admin. des BD

Sécurité d'un Système d'Information :

- ➢ Assurer la cohérence des informations (intégrité)
  - ❖ définition des données (`PRIMARY, FOREIGN KEY…`)
- ➢ Etre robuste face aux pannes
  - ❖ « service pérenne » sera vu en S3 (BD Avancées)
- ➢ Assurer la confidentialité des données
  - ❖ qui peut passer? (authentification)
  - ❖ que peut-il faire une fois à l'intérieur? (droits)

## M2106 – II – Programmation serveur

Etendre SQL par ajout de fonctions

- ✦ en langage de requêtes (SQL)
- ✦ en langage de procédure (PL/pgSQL, PL/Python…)
- ✦ en langage C (compilées en bibliothèques chargeables)

Pourquoi?

- ✦ Simplifier les requêtes en les décomposant (fonction = traitement intermédiaire)
- ✦ Factoriser des ensemble de requêtes similaires (via paramètres)
- ✦ Exécuter des traitements sur serveur plutôt que client

## M2106 – II – Programmation serveur

Exécuter des traitements sur serveur plutôt que client

- ✦ Performance
  - nombreux transferts client/serveur évités
  - bénéficier des données en cache
- ✦Facilité de maintenance
  - si multiples applications clientes
- ✦Sécurité
  - ne pas donner accès direct aux tables (ni nom, ni schéma)
  - assurer l'intégrité des données (cont. ext. ; trigger)
- ✦Administrer le serveur
  - fonctions d'audit…

## M2106 – II – Programmation serveur

Quand <u>ne pas</u> utiliser de procédures stockées?

- ✦Trop de contraintes
  - Un langage de plus à maîtriser
  - SGBD choisi en amont
- ✦Sans bénéfice
  - Gain pas évident en performance
  - Sécurité bien gérée au niveau applicatif

## M2106 – II - 1) PL/pgSQL

Langage procédural qui permet d'écrire :

- ✦des instructions SQL
- ✦des structures de contrôle (if, while, …)

Avantages :

- ✦Allier la puissance d'un langage de programmation à la facilité d'utilisation du SQL
- ✦Permettre des traitements complexes

## M2106 – II - 1) Procédures stockées

Fonction qui regroupe des instructions (SQL ou autres) stockée dans une table de catalogue système

```
CREATE FUNCTION nom
  ( [ [modearg] nomarg typearg [, ...] ] )
  [ RETURNS type_resultat ]
   AS $$definition$$
   LANGUAGE nom_lang ;

DROP FUNCTION nom
   ( [ [modearg] [nomarg] typearg [, ...] ] )
```

## M2106 – II - 1) PL/pgSQL - paramètres

```
CREATE FUNCTION nom
( [ [modearg] nomarg typearg [, ...] ] )
   [ RETURNS type_resultat ] ...
```

Une fonction est identifiée par sa signature :

son nom + le type de ses arguments **IN**.

On ne peut changer le type des paramètres **OUT**.

## M2106 – II - 1) PL/pgSQL - paramètres

```
CREATE FUNCTION nom
( [ [modearg] nomarg typearg [, ...] ] )
   [ RETURNS type_resultat ] ...
```

Les modes pour les paramètres sont :

**IN** (par défaut), **INOUT**, **OUT**

La fonction n'a pas de résultat :

```
RETURNS void
```

## M2106 – II - 1) PL/pgSQL - paramètres

```
CREATE FUNCTION nom
( [ [modearg] nomarg typearg [, ...] ] )
   [ RETURNS type_resultat ]
```

Les types utilisables sont :

- les mêmes qu'en SQL
- **nom_table.nom_att%TYPE** (est converti)
- **nom_table**
- **RECORD**

## M2106 – II - 1) PL/pgSQL - définition

La définition (le corps) d'une fonction est un bloc :

```
[DECLARE déclarations]
BEGIN
     instructions;
END;
```

Commentaires :

```
-- sur une ligne
/* bloc éventuellement sur
plusieurs lignes*/
```

## M2106 – II - 1) PL/pgSQL - déclarations

Toutes les variables doivent être déclarées, sauf les variables des boucles **FOR**

```
nom [CONSTANT] type [NOT NULL]
          [DEFAULT expression];
```

Les types utilisables sont :

- les mêmes qu'en SQL
- **nom_table.nom_att%TYPE**
- **nom_table%ROWTYPE**
- **RECORD**

## M2106 – II - 1) PL/pgSQL - instructions

- **NULL;**
- Affectation : **:=**
- Instruction conditionnelle :

```
IF … THEN … ELSIF … ELSE … END IF;
```

- Itérations

| | | |
|---|---|---|
| WHILE … LOOP … END LOOP; | LOOP … EXIT WHEN … END LOOP; | FOR … IN [REVERSE]… LOOP … END LOOP; |

- **RETURN …;**

## M2106 – II - 1) PL/pgSQL - instructions

Toutes les commandes SQL qui ne renvoient pas de résultat :

**INSERT, UPDATE, GRANT, CREATE TABLE, …**
pouvant utiliser les variables et les paramètres

Pour évaluer une requête d'interrogation :

- avec affectation du résultat dans une variable
  **SELECT … INTO … FROM …**
- sans récupération du résultat (pour effet de bord)
  **PERFORM … FROM … WHERE …**

```
PERFORM nom_fonction (…);
```

## M2106 – II - 1) PL/pgSQL- instructions

Fonction qui retourne le résultat d'une requête

```
CREATE FUNCTION nom (...)
   RETURNS SETOF type
AS $$
BEGIN
  …
  RETURN QUERY SELECT … FROM … ;
END $$ LANGUAGE 'plpgsql';
```

## M2106 – II - 1) PL/pgSQL - instructions

Statut du résultat d'une instruction : **FOUND**

Variable booléenne positionnée par :

- **SELECT INTO**
- **PERFORM**
- **UPDATE, INSERT, DELETE**
- **FOR**
- **RETURN QUERY**
- **FETCH**

## M2106 – II - 1) PL/pgSQL - instructions

Affichage d'un message ou d'une erreur

```
RAISE NOTICE 'message'[, …];
RAISE EXCEPTION 'message'[, …];
```

Le message est éventuellement suivi d'expressions qui remplacent les % du message.

## M2106 – II - 1) Appel de fonction

```
CREATE FUNCTION DoyenEspece
      (IN esp animal.espece%type,
       IN sex animal.sexe%type,
       OUT nom animal.noma%type,
       OUT age interval)…
```

- **SELECT DoyenEspece('koala','f');**
- **SELECT nom FROM DoyenEspece('koala','f') WHERE … ;**
- **SELECT (DoyenEspece('koala', 'f')).nom;**

Les paramètres effectifs doivent être donnés dans l'ordre (notation par position)

## M2106 – II - 2) Curseurs

Définition :

- ✦instruction **SELECT** prête à être exécutée,
- ✦dont on peut récupérer les lignes résultat pas à pas, via une position courante dans cet ensemble.

Utilisation :

- ✦Déclaration
- ✦Ouverture
- ✦Récupération(s) de lignes résultat
- ✦Fermeture

## M2106 – II - 2) Curseur - déclaration

Curseur non lié :

```
nom refcursor
```

Curseur lié à une requête :

```
nom CURSOR FOR SELECT ...
```

Curseur lié à une requête paramétrée :

```
nom CURSOR(nomarg typearg [,…]) FOR SELECT …
```

## M2106 – II - 2) Curseur - ouverture

Pour construire le plan de requête et le stocker dans une structure adaptée, prêt à être exécuté.

Curseur non lié :

```
OPEN nom FOR requete
```

Curseur lié :

```
OPEN nom [(paramètres)]
```

## M2106 – II - 2) Curseur - utilisation

Pour récupérer une ligne d'un curseur :

```
FETCH nom INTO cible
```

où ***cible*** est :

- ❖ une variable n-uplet (…**%ROWTYPE**)
- ❖ une variable record
- ❖ une liste de variables séparées par des virgules

Exemple:

```
LOOP
        FETCH nom INTO cible;
        EXIT WHEN not found;
        …
END LOOP;
```

## M2106 – II - 2) Curseur - fermeture

Pour libérer les ressources (important!) ou pour pouvoir ré-ouvrir le curseur.

```
CLOSE nom ;
```

Exemple complet :

```
create function filmsActeur(in nom varchar(20))returns void as $$
declare
  c1 cursor for select titre from acteur where nomact=nom;
  tit varchar(50);
begin
  raise notice '% joue dans :',nom;
  open c1;
  loop
    fetch c1 into tit;
    exit when not found;
    raise notice '              %',tit;
  end loop;
  close c1;
end $$ language 'plpgsql';
```

## M2106 – II - 2) Curseur - Variantes

Rendre **OPEN**, **FETCH**, **CLOSE** et déclaration de ***cible*** (de type ***record***) implicites.

```
FOR cible IN nom_curseur_lié [(paramètres)]
LOOP    …
END LOOP;
```

A chaque passage dans la boucle, une nouvelle ligne du curseur est affectée à ***cible***.

## M2106 – II - 2) Curseur - Variantes

Le curseur lui-même peut être implicite :

```
FOR cible IN requete
LOOP    …
END LOOP;
```

mais déclaration explicite de ***cible*** nécessaire :

- ❖une variable n-uplet (**…%ROWTYPE**)
- ❖ une variable record
- ❖ une liste de variables séparées par des virgules

## M2106 – II - 2) Curseur - Variantes

```
create function filmsActeur(in nom varchar(20))returns void as $$
declare
  c1 cursor for select titre from acteur where nomact=nom;
begin
  raise notice '% joue dans :',nom;
  for res in c1 loop -- res est un record implicite
    raise notice '                %',res.titre;
  end loop;
end $$ language 'plpgsql';
```

```
create function filmsActeur(in nom varchar(20))returns void as $$
declare
  tit varchar(50);
begin
  raise notice '% joue dans :',nom;
  for tit in select titre from acteur where nomact=nom loop
    raise notice '                %',tit;
  end loop;
end $$ language 'plpgsql';
```

## M2106 – II - 3) Triggers - Objectifs

Un trigger (déclencheur) permet d'exécuter automatiquement une fonction lorsque certains événements se produisent.

- ➢Pour vérifier des contraintes d'intégrités complexes (dynamiques, sur plusieurs relations, …)
- ➢Pour annuler ou propager des modifications (gestion de stocks…)
- ➢Pour tenir un historique des actions sur la base
- ➢Pour exécuter des actions sur une vue complexe

## M2106 – II - 3) Triggers sur relation

```
CREATE TRIGGER nom_trigger
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE [ OR ... ] }
ON nom_relation
FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE nom_fonction();

DROP TRIGGER nom_trigger
ON nom_relation ;
```

## M2106 – II - 3) Triggers sur relations

Description de l'événement déclencheur :

- ✦Sur quelle relation?
- ✦Pour quelle(s) modification(s)?
- ✦A quel moment? (avant ou après)
- ✦Pour chaque n-uplet ou instruction?

La fonction à exécuter :

- ✦Définie avant le trigger
- ✦Sans paramètre, retourne le type **trigger**
- ✦Écrite, par exemple, en **plpgsql**

## M2106 – II - 3) Trigger- exemple

a) Définir l'événement

Sur quelle relation?
→**animal**

Pour quelle(s) modification(s)?
→ **update**

A quel moment? (avant ou après)
→ **before**

Pour chaque n-uplet ou instruction?
→ **for each row**

## M2106 – II - 3) Trigger- exemple

b) Ecrire la fonction

```
CREATE FUNCTION f_animal_upd() RETURNS trigger
AS $$ DECLARE
  nb integer;
BEGIN
  select count(*) into nb
    from animal where pays='inconnu';
  raise notice 'Il reste % apatrides.',nb;
  return new;  -- pour confirmer l'insertion
END;
$$ LANGUAGE 'plpgsql' ;
```



## M2106 – II - 3) Trigger- exemple

c) Ecrire le trigger

```
CREATE TRIGGER t_animal_upd
BEFORE UPDATE ON animal
FOR EACH ROW
EXECUTE PROCEDURE f_animal_upd();
```



## M2106 – II - 3) Déclenchement

**FOR EACH ROW**

- fonction déclenchée sur chaque n-uplet
- **BEFORE** : avant chaque opération
- **AFTER** : après l'ensemble des opérations

**FOR EACH STATEMENT**

- fonction déclenchée une seule fois
- **BEFORE :** avant l'ensemble des opérations
- **AFTER :** après l'ensemble des opérations



## M2106 – II - 3) Fonction - instructions

**RETURN**

- valide l'opération.
- **FOR EACH STATEMENT** : **RETURN NULL;**
- **FOR EACH ROW + AFTER** : **RETURN NULL;**
- **FOR EACH ROW + BEFORE** : **RETURN rec;**

où **rec** est le n-uplet objet de la requête qui a déclenché.

**RAISE EXCEPTION**

- annule la requête et l'ensemble des actions prévues dans la fonction.
- que la fonction soit déclenchée **BEFORE** ou **AFTER**.



## M2106 – II - 3) Fonction - variables

**new** et **old** : n-uplets pré-définis

| opération | new | old |
|---|---|---|
| ON INSERT | le nouveau n-uplet à insérer | |
| ON UPDATE | la nouvelle valeur du n-uplet | le n-uplet à modifier |
| ON DELETE | | le n-uplet à supprimer |

## M2106 – II - 3) Fonction - variables

**new** et **old**

- définies seulement **FOR EACH ROW.**
- si **BEFORE** : est généralement le n-uplet renvoyé.

```
INSERT/UPDATE : RETURN new;
DELETE : RETURN old;
```

**TG_OP, TG_TABLE_NAME, TG_NAME …**

- variables caractérisant le trigger déclencheur
- ex : notifier chaque exécution de trigger

```
RAISE NOTICE '% ON % FIRES %',
      TG_OP, TG_TABLE_NAME, TG_NAME;
```

## M2106 – II - 3) Triggers - variables

| Nom | Type | Valeur |
|---|---|---|
| TG_WHEN | text | BEFORE, AFTER |
| TG_LEVEL | text | ROW, STATEMENT |
| TG_OP | text | INSERT, UPDATE, DELETE |
| TG_TABLE_NAME | name | Nom de la table qui a déclenché le trigger |
| TG_NAME | name | Nom du trigger |

## M2106 – II - 3) Triggers - résultat

| Trigger | | Pour valider la fonction | Pour annuler la fonction |
|---|---|---|---|
| FOR EACH ROW | AFTER | RETURN NULL; (valeur de retour ignorée) | RAISE EXCEPTION … |
| | BEFORE INSERT | RETURN new; * | RETURN NULL; ou RAISE EXCEPTION … |
| | BEFORE UPDATE | RETURN new; * | |
| | BEFORE DELETE | RETURN old; | |
| FOR EACH STATEMENT | | RETURN NULL; (valeur de retour ignorée) | RAISE EXCEPTION … |

* **new** initial ou modifié ou autre n-uplet de même schéma

## M2106 – II - 3) Trigger

Attention!

- Créer des triggers = créer des « effets de bords » peut devenir très difficile à gérer.
- Si plusieurs triggers sont définis pour la même action sur la même table, ils s'exécutent dans l'ordre suivant

```
FOR EACH STATEMENT, BEFORE
FOR EACH ROW, BEFORE
FOR EACH ROW, AFTER
FOR EACH STATEMENT, AFTER
```

puis, dans chaque catégorie, dans l'ordre alphabétique.



## M2106 – II - 3) Trigger

- Pour ne déclencher un trigger que lorsque cela est nécessaire :

```
CREATE TRIGGER t_animal_upd
BEFORE UPDATE ON animal
FOR EACH ROW
WHEN ((OLD.pays != NEW.pays)
        AND
       (OLD.pays = 'inconnu'
        OR NEW.pays = 'inconnu'))
EXECUTE PROCEDURE f_animal_upd();
```



## M2106 –Vues (Rappels M1104)

Une vue se crée comme résultat d'une requête

```
CREATE VIEW nom [(liste-attributs)]
    AS SELECT … FROM …
```

Suppression d'une vue : `DROP VIEW nom;`

Conseil d'utilisation :

```
CREATE OR REPLACE VIEW nom …
DROP VIEW IF EXISTS nom;
```

## M2106 –II - 3) Trigger sur vue

Une vue peut être:

- modifiable (répercussion sur tables d'origine sans ambiguïté ni perte d'intégrité)
- non-modifiable (sauf par un trigger `INSTEAD OF`)

Remarque pour postgreSQL :

- une vue est créée grâce à une règle
- une règle permet de modifier une vue

## M2106 – II - 3) Triggers sur vue

```
CREATE TRIGGER nom_trigger
INSTEAD OF
{ INSERT | UPDATE | DELETE [ OR ... ] }
ON nom_vue
FOR EACH ROW
EXECUTE PROCEDURE nom_fonction();

DROP TRIGGER nom_trigger
ON nom_vue ;
```

## M2106 – II - 3) Trigger sur vue

Principe :

- Lors d'une modification de la vue (événement **INSERT, UPDATE, DELETE**)
- Un trigger exécute une procédure qui la remplace par une ou plusieurs autres commandes SQL sur des relations (**SELECT, INSERT, UPDATE, DELETE**)

Le n-uplet renvoyé *informe* le contexte:

- **null** aucune modification n'a eu lieu
- **new/old** sinon

## M2106 – II - 3) Trigger sur vue

```
CREATE VIEW v_chat_males
AS
   SELECT noma, pays, datea, numc
    FROM animal
   WHERE espece = 'chat' AND sexe = 'm';


INSERT INTO v_chat_males(noma,pays,numc)
   VALUES ('Berlioz', 'France', 1);
```



## M2106 – II - 3) Trigger sur vue

```
CREATE FUNCTION f_VchatMales_ins()
RETURNS trigger
AS $$
BEGIN
  raise notice '% ON % FIRES %',
               TG_OP, TG_TABLE_NAME, TG_NAME;

   insert into animal values(new.noma, 'chat',
          'm',new.pays, current_date, new.numc);
   return new;
END; $$ LANGUAGE 'plpgsql' ;
```



## M2106 – II - 3) Triggers sur vue

```
CREATE TRIGGER t_VchatMales
INSTEAD OF
INSERT ON v_chat_males
FOR EACH ROW
EXECUTE PROCEDURE f_VchatMales_ins();
```

## M2106 – Prog. et Admin. des BD

Authentification sous postgreSQL:

contrôlée par fichier de configuration **pg_hba.conf**

**qui** peut se connecter, **à quelles bases**, **d'où**, et **comment**

```
TYPE      DATABASE      USER          ADDRESS           METHOD
# Unix domain socket connections (local)
local     all           postgres                        peer
# IPv4 local connections
host      all           all           127.0.0.1/32      reject
# IPv4 non-local connections
host      all           superman      0.0.0.0/0         scram-sha-256
host      ma_base       +invites      192.168.141.0/24  scram-sha-256
```

## M2106 – III - 1) Utilisateurs

```
psql -h host -U user ma_base
```

```
CREATE USER nom_user [WITH option […]];
```

*option: attributs spéciaux*

- **superuser** : peut tout faire
- **createrole** : peut créer de nouveaux utilisateurs
- **createdb** : peut créer de nouvelles bases
- **password 'xxx'**
- …

```
DROP USER nom_user;
```

```
ALTER USER nom_user option;
```

## M2106 – III - 1) Rôles

Rôle : nom + attributs spéciaux

- ❖ droit de connexion → utilisateur
- ❖ sinon → groupe

```
CREATE ROLE nom_role [WITH option […]];
```

**CREATE USER nom ≡ CREATE ROLE nom WITH LOGIN**

par défaut **CREATE ROLE** est **NOLOGIN**

Quel attribut spécial pour créer un rôle?

- ❖ **superuser**
- ❖ **createrole**

## M2106 – III - 1) Rôles

Inclure un rôle dans un groupe :

- ❖ à la création du rôle

```
CREATE nom_role IN ROLE nom_group
```

- ❖ après la création rôle

```
GRANT nom_group TO nom_role
              [WITH ADMIN OPTION]
```

Qui peut inclure un rôle dans un groupe :

- ❖ **superuser**
- ❖ **createrole**
- ❖ un membre qui a été inclus dans le groupe avec l'option **WITH ADMIN OPTION**

## M2106 – III - 2) Droits sur une base

```
GRANT CONNECT ON DATABASE nom_base
  TO {role|PUBLIC}[,…]
```

```
REVOKE CONNECT ON DATABASE nom_base
  FROM {role|PUBLIC}[,…]
```

Attention : à la création d'une base, le droit de connexion est accordé à **PUBLIC** par défaut!

➢ **REVOKE CONNECT ON DATABASE nom_base FROM PUBLIC**

## M2106 – III - 2) Droits sur les objets

```
GRANT {privilège, … | ALL}
                [(attribut, …)]
  ON {relation | vue} [,…]
  TO {role|PUBLIC}[,…]
```

*privilège* :

**SELECT, INSERT, UPDATE, DELETE**

```
REVOKE … ON … FROM …
```

✦Par défaut, aucun privilège sur les objets n'est accordé à **PUBLIC**.

## M2106 – III - 2) Droits et rôles

Principe :

➢ Rôle : ensemble de privilèges sur des objets

➢ Privilèges (**SELECT, INSERT, UPDATE, DELETE**) sont accordés ou retirés à un rôle sur une base ou un objet donné.

➢ Un rôle hérite des privilèges accordés :

+ à lui-même.
+ aux rôles auxquels il appartient.
+ à **PUBLIC**.

## M2106 – III - 2) Droits

Qui peut gérer un droit sur un objet/base :

- ❖ **superuser**
- ❖ le propriétaire de l'objet/base (ou tout rôle lui appartenant)
- ❖ un rôle à qui on a accordé ce droit avec la permission de le transmettre :

```
GRANT privilège ON objet TO role
        WITH GRANT OPTION
```

## M2106 – III - 2) Droits

Les droits de définition de données (**ALTER, DROP**) sont réservés :

- aux **superuser**.
- au rôle propriétaire et à tout rôle lui appartenant.

Ils ne se transmettent pas directement par **GRANT**.

## M2106 – III - 2) Vue pour définir Droits

✦Donner un droit sur une vue ne le donne pas sur les tables d'origine (ni ne l'exige).

✦Pour une vue complexe: **GRANT+TRIGGER+**

```
CREATE FUNCTION nom (...)
AS $$ BEGIN
…
END; $$ LANGUAGE'plpgsql' SECURITY DEFINER;
```

mais **getpgusername()** renvoie le **DEFINER.**

## M2106 – III - 2) Droits

Conseil :

➢ Définir les droits sur les objets dès leur création.

➢ Définir un groupe par rôle fonctionnel (admin, directeur, secrétaire,… définis par Règles d'Organisation).

➢ Accorder les droits à ces groupes (selon les RO).

➢ Inclure utilisateur dans les groupes selon ses rôles.